Saarland University
Department of Computer Science

Enhancing the Security and Efficiency of Distributed Key Generation and Its Applications

Dissertation
zur Erlangung des Grades
des Doktors der Naturwissenschaften
der Fakultät für Mathematik und Informatik
der Universität des Saarlandes

von
Renas Bacho

Saarbrücken, 2025

Tag des Kolloquiums:	19. März 2026
Dekan:	Prof. Dr. Roland Speicher

Prüfungsausschuss:

Vorsitzender:	Prof. Dr. Ingmar Weber
Berichterstattende:	Prof. Dr. Julian Loss
	Prof. Dr. Markus Bläser
	Prof. Dr. Ling Ren
	Dr. Ittai Abraham
Akademischer Mitarbeiter:	Dr. Mahesh Sreekumar Rajasree

UNIVERSITÄT
DES
SAARLANDES

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Declaration of original authorship

I hereby declare that this dissertation is my own original work except where otherwise indicated. All data or concepts drawn directly or indirectly from other sources have been correctly acknowledged. This dissertation has not been submitted in its present or similar form to any other academic institution either in Germany or abroad for the award of any other degree.

Saarbrücken, October 2025

Abstract

Distributed Key Generation (DKG) is a fault-tolerant protocol in which n parties jointly generate a public–secret key pair. The secret key is distributed among the parties via a t-out-of-n threshold secret sharing scheme but is never reconstructed or stored in a single location. A DKG protocol is essential for bootstrapping threshold cryptosystems without relying on a trusted third party. This eliminates single points of failure and trust, which is crucial for decentralized systems such as blockchains. Key applications of DKG protocols include threshold encryption, threshold signatures, and distributed randomness generation. Indeed, many state-of-the-art Byzantine fault-tolerant (BFT) protocols employ threshold signatures and distributed randomness to enhance communication efficiency, or threshold encryption to prevent censorship.

In this thesis, we make significant progress in strengthening the security guarantees and improving the efficiency of distributed key generation and its applications, particularly threshold signatures and distributed randomness generation. Our main contributions are summarized as follows:

- **Part I:** We revisit the security of the threshold BLS signature scheme under adaptive corruptions, as used in many state-of-the-art Byzantine fault-tolerant (BFT) distributed protocols. First, we introduce a new security notion for DKG protocols and show that several existing protocols, previously proven only under static security, satisfy this notion under adaptive corruptions. Second, assuming any DKG protocol with this property, we provide a tight adaptive security proof for the threshold BLS signature scheme, thereby justifying real-world parameter choices.

- **Part II:** We revisit the security of efficient publicly verifiable secret sharing (PVSS) schemes under adaptive corruptions, which are employed in many state-of-the-art distributed randomness beacon and DKG protocols. First, we introduce a new security notion for (aggregatable) PVSS schemes and show that several existing schemes, previously proven only under static security, satisfy it under adaptive corruptions. Second, we demonstrate that this notion directly yields adaptive security for state-of-the-art distributed randomness beacon protocols in their respective network models, which build upon (aggregatable) PVSS schemes satisfying our new notion of unpredictability.

- **Part III:** We present a novel synchronous DKG protocol with near-quadratic communication cost, optimal corruption threshold, and adaptive security. It is the first DKG protocol in any network setting to achieve sub-cubic communication cost with an optimal corruption threshold. Furthermore, we propose a new distributed randomness beacon protocol with optimal corruption threshold, adaptive security, and quadratic communication cost per epoch in a single asynchronous round, resulting in a highly efficient randomness beacon that outperforms existing schemes.

- **Part IV:** We introduce a novel network-agnostic DKG protocol with an optimal corruption threshold under both synchrony and asynchrony, static security, and cubic communication cost. Our construction matches the best-known DKG protocols in their respective network models.

Finally, we conclude this thesis with a discussion of several open problems that may guide future research directions.

Zusammenfassung

Verteilte Schlüsselerzeugung (Distributed Key Generation, DKG) ist ein fehlertolerantes Protokoll, bei dem n Parteien gemeinsam ein öffentlich–geheimes Schlüsselpaar erzeugen. Der geheime Schlüssel wird mittels eines t-aus-n-Schwellwert-Geheimnisteilungsschemas unter den Parteien verteilt, jedoch niemals rekonstruiert oder an einem einzigen Ort gespeichert. Ein DKG-Protokoll ist wesentlich für die Initialisierung (Bootstrapping) von Schwellwert-Kryptosystemen, ohne auf eine vertrauenswürdige dritte Partei angewiesen zu sein. Dadurch werden einzelne Vertrauens- und Ausfallpunkte eliminiert, was für dezentrale Systeme wie Blockchains von entscheidender Bedeutung ist. Zu den wichtigsten Anwendungen von DKG-Protokollen zählen Schwellwert-Verschlüsselung, Schwellwert-Signaturen und verteilte Zufallsgenerierung. Tatsächlich nutzen viele moderne byzantinisch fehlertolerante (BFT) Protokolle Schwellwert-Signaturen und verteilte Zufälligkeit, um die Kommunikationseffizienz zu erhöhen, oder Schwellwert-Verschlüsselung, um Zensur zu verhindern.

In dieser Dissertation machen wir bedeutende Fortschritte bei der Stärkung der Sicherheitsgarantien und der Verbesserung der Effizienz der verteilten Schlüsselerzeugung und ihrer Anwendungen, insbesondere bei Schwellwert-Signaturen und der verteilten Zufallsgenerierung. Unsere wichtigsten Beiträge lassen sich wie folgt zusammenfassen:

- **Teil I:** Wir untersuchen die Sicherheit des Schwellwert-BLS-Signaturschemas unter adaptiven Korruptionen, das in vielen modernen byzantinisch fehlertoleranten (BFT) Protokollen verwendet wird. Zunächst führen wir eine neue Sicherheitsdefinition für DKG-Protokolle ein und zeigen, dass mehrere bestehende Protokolle, die bisher nur unter statischer Sicherheit bewiesen wurden, diese Definition auch unter adaptiven Korruptionen erfüllen. Anschließend zeigen wir, unter der Annahme eines DKG-Protokolls mit dieser Eigenschaft, einen engen adaptiven Sicherheitsbeweis für das Schwellwert-BLS-Signaturschema und rechtfertigen damit die Parameterwahl in realen Anwendungen.

- **Teil II:** Wir untersuchen die Sicherheit effizienter öffentlich verifizierbarer Geheimnisteilungsschemata (PVSS) unter adaptiven Korruptionen, die in vielen modernen Protokollen zur verteilten Zufallserzeugung und Schlüsselerzeugung eingesetzt werden. Zunächst führen wir eine neue Sicherheitsdefinition für (aggregierbare) PVSS-Schemata ein und zeigen, dass mehrere bestehende Schemata, die bisher nur unter statischer Sicherheit bewiesen wurden, diese auch unter adaptiven Korruptionen erfüllen. Anschließend zeigen wir, dass diese Definition direkt adaptive Sicherheit für moderne Protokolle zur verteilten Zufallserzeugung in ihren jeweiligen Netzwerkmodellen liefert, die auf (aggregierbaren) PVSS-Schemata beruhen, welche unsere neue Unvorhersagbarkeitsdefinition erfüllen.

- **Teil III:** Wir präsentieren ein neuartiges synchrones DKG-Protokoll mit nahezu quadratischem Kommunikationsaufwand, optimaler Korruptionsschwelle und adaptiver Sicherheit. Es ist das erste DKG-Protokoll in allen Netzwerkmodellen, das einen subkubischen Kommunikationsaufwand bei optimaler Korruptionsschwelle erreicht. Darüber hinaus schlagen wir ein neues Protokoll für verteilte Zufallserzeugung vor, das eine optimale Korruptionsschwelle, adaptive Sicherheit und einen quadratischen Kommunikationsaufwand pro Epoche in einer einzigen asynchronen Runde aufweist, was zu einem äußerst effizienten Zufallsbeacon führt, das bestehende Verfahren übertrifft.

- **Teil IV:** Wir stellen ein neuartiges netzwerkunabhängiges DKG-Protokoll mit optimaler Korruptionsschwelle unter sowohl synchronen als auch asynchronen Bedingungen, statischer Sicherheit und kubischem Kommunikationsaufwand vor. Unsere Konstruktion entspricht den besten derzeit bekannten DKG-Protokollen in ihren jeweiligen Netzwerkmodellen.

Abschließend stellen wir einige offene Probleme vor, die als Ausgangspunkt für zukünftige Forschung dienen können.

Background of this Dissertation

This dissertation is based on the publications listed below. Each chapter provides further details on how its content relates to the corresponding publication. Full citations are provided in the Bibliography.

[BL22a] **On the Adaptive Security of the Threshold BLS Signature Scheme.** *Renas Bacho, Julian Loss*. In ACM CCS 2022.

[Bac+23a] **Network-Agnostic Security Comes (Almost) for Free in DKG and MPC.** *Renas Bacho, Daniel Collins, Chen-Da Liu-Zhang, Julian Loss*. In IACR CRYPTO 2023.

[BL23a] **Adaptively Secure (Aggregatable) PVSS and Applications to Distributed Randomness Beacons.** *Renas Bacho, Julian Loss*. In ACM CCS 2023.

[Bac+24a] **GRandLine: Adaptively Secure DKG and Randomness Beacon with (Log-)Quadratic Communication Complexity.** *Renas Bacho, Christoph Lenzen, Julian Loss, Simon Ochsenreither, Dimitrios Papachristoudis*. In ACM CCS 2024.

Further Contributions of the Author

During my time as a Ph.D. student, I also made contributions to the following papers.

[BC26] **Earpicks: Tightly Secure Two-Round Multi- and Threshold Signatures.** *Renas Bacho, Yanbo Chen*. In IACR EUROCRYPT 2026.

[Bac+25a] **Adaptively Secure Partially Non-Interactive Threshold Schnorr Signatures in the AGM.** *Renas Bacho, Yanbo Chen, Julian Loss, Stefano Tessaro, Chenzhi Zhu*. In IACR EUROCRYPT 2026.

[BW26] **Tightly Secure Threshold Signatures over Pairing-Free Groups.** *Renas Bacho, Benedikt Wagner*. In IACR CiC 2026, 2(4).

[BW25a] **T-Spoon: Tightly Secure Two-Round Multi-Signatures with Key Aggregation.** *Renas Bacho, Benedikt Wagner*. In IACR CRYPTO 2025.

[Bac+25b] **Adaptively Secure Three-Round Threshold Schnorr Signatures from DDH.** *Renas Bacho, Sourav Das, Julian Loss, Ling Ren*. In IACR CRYPTO 2025.

[BK25] **SoK: Dlog-Based Distributed Key Generation.** *Renas Bacho, Alireza Kavousi*. In IEEE S&P 2025.

[Bac+25e] **Glacius: Threshold Schnorr Signatures from DDH with Full Adaptive Security.** *Renas Bacho, Sourav Das, Julian Loss, Ling Ren*. In IACR EUROCRYPT 2025.

[Bac+24d] **Twinkle: Threshold Signatures from DDH with Full Adaptive Security.** *Renas Bacho, Julian Loss, Stefano Tessaro, Benedikt Wagner, Chenzhi Zhu*. In IACR EUROCRYPT 2024.

[BW24a] **Tightly Secure Non-interactive BLS Multi-signatures.** *Renas Bacho, Benedikt Wagner*. In IACR ASIACRYPT 2024.

[Bac+24c] **HARTS: High-Threshold, Adaptively Secure, and Robust Threshold Schnorr Signatures.** *Renas Bacho, Julian Loss, Gilad Stern, Benedikt Wagner*. In IACR ASIACRYPT 2024.

Acknowledgments

I am deeply grateful to my advisor, Julian Loss, for his constant guidance and support. You are not only an outstanding researcher and advisor, but also a great friend and person. You have guided me through the world of research, and your continuous dedication has been central to my growth as a researcher. Your invaluable advice on writing papers, giving talks, preparing slides, conducting research, and much more has profoundly shaped me. I have always greatly enjoyed our discussions on open problems and the many opportunities to learn from you.

I would also like to thank Christoph Lenzen. It has been both joyful and enlightening to learn from your brilliant and sharp ideas, as well as from your clear and elegant way of formulating complex concepts. Working with you has always been a true pleasure and an inspiring experience.

I would also like to thank all the great people and friends I have met over the past years. My sincere thanks go to my brilliant friends and collaborators Sourav, Alireza, Benedikt, and Gilad, from whom I have learned so much about research and beyond. I am also grateful to my colleagues and friends at the institute, as well as to all my collaborators and our visitors, with special thanks to Daniel, Sravya, and Yanbo. I would also like to thank the members of the cryptography group of our institute for providing an enjoyable environment, with sincere thanks to Nico, Lucjan, Antoine, Kecheng, Xicheng, Omar, Simon, Clément, Jesko, Khue, Eugenio, Giacomo, Anne, and others. Furthermore, I would like to thank the members of my Ph.D. committee for their valuable time and effort.

I am also deeply grateful to all the teachers and professors who have supported me throughout my journey, with special thanks to Prof. Dorothee Schüth and Prof. Elmar Große-Klönne. I would also like to thank my close friends whom I have met along the way in life and whose presence always brings me joy: Markus, Daniel, Fabian, Evgueni, Onur, and Ferhat. I would also like to thank my cousins, with whom I grew up and share many wonderful memories, especially Guan, Glal, and Kaua.

My deepest thanks go to my family, whose unwavering support and unconditional love have carried me through every stage of life. To my parents, who have guided me throughout my life and always strived to provide the best for their children. To my elder brother, Aras, who first introduced me to mathematics and awakened my love for it, and for always being by my side, especially during my childhood. To my younger brothers, Kano and Delvin, who have always been there for me with their support and wise advice - your dedication and patience have been a constant source of inspiration to me. To my elder sisters, Jinda and Rodica, for their constant encouragement and support in life. It is always wonderful to be around you all.

Contents

1 Introduction

1.1 Background and Motivation

Distributed Key Generation (DKG) [Ped92] allows a set of n parties $P_1, \ldots, P_n$ to jointly generate a public-secret key pair (pk, sk), where the secret key is shared among the parties through a (t, n)-threshold secret sharing scheme. Specifically, each party P_i obtains a share sk_i of the secret key so that any subset of at least $t + 1$ shares uniquely determines the secret key, whereas t or fewer shares give no information about sk. Importantly, the secret key itself is never reconstructed or stored in a single location. Instead, parties can later use their secret key shares to jointly perform cryptographic operations in a distributed manner. The common public key pk then allows any external verifier to efficiently check the correctness of the group's outputs. Conceptually, a DKG shifts the generation of the public-secret key pair from a single authority to a collective of n parties: each holds only a share sk_i of the secret key, yet the group as a whole is represented by a single common public key pk. In this manner, the underlying secret key sk remains protected from an adversary that can corrupt up to t parties.

DKG protocols have become a fundamental building block for decentralized and fault-tolerant systems [Yin+19; BK25], because they enable operations such as signing, decryption, or randomness generation without introducing a single point of failure. In fact, a DKG protocol is essential for bootstrapping any threshold cryptosystem without relying on a trusted third party for key generation. Among the most widely used are DKG protocols for discrete-logarithm–based cryptosystems, which are the subject of our study. DKG protocols have a wide range of applications, such as threshold encryption and signing [Gen+07; SG98], distributed randomness beacons [KWJ23], secure multi-party computation [Gen+21a], and Byzantine consensus [Blu+20]. Abstractly, a DKG protocol typically consists of three core phases:

- **Sharing Phase.** Each party samples a uniformly random secret and shares it among all parties via a threshold secret sharing scheme. This phase is typically implemented using a (publicly) verifiable secret sharing scheme.
- **Agreement Phase.** The parties then agree on a subset of participants who have correctly shared their secrets, which will be included in the computation of the final secret key. This phase is typically implemented using some form of consensus protocol.
- **Key Derivation Phase.** The parties derive their secret key shares and the common public key, upon which they terminate the protocol. This phase is generally non-interactive and involves only local computation.

1.1.1 The Adversary in Threshold Cryptography

The literature on distributed cryptographic protocols traditionally considers an adversary who may corrupt up to t out of the n parties and cause them to deviate arbitrarily from the prescribed protocol specification. Such an adversary is also known as a *Byzantine* adversary [LSP82b].[1] It gives the adversary full control over the corrupted parties and allows it to take arbitrary (e.g., malicious) actions on their behalf, including collusion among them.

The literature further distinguishes between two types of Byzantine adversaries: *static* and *adaptive*. A static adversary must commit to which parties it will corrupt at the outset of the

[1] The literature on general secure multi-party computation (MPC) also refers to it as an *active* adversary.

protocol execution - i.e., before public keys or randomness are fixed. In contrast, an adaptive adversary may choose which parties to corrupt dynamically during the execution of the protocol, based on the messages it observes or on intermediate states. The adaptive adversary model is both stronger and more realistic in decentralized settings, where attackers may selectively compromise parties over time. Consequently, a substantial body of work has focused on designing distributed protocols that remain secure against adaptive adversaries [Can+96; Can+99; Dav+18].

Why is Adaptive Security Hard? Designing distributed protocols that remain secure against an adaptive adversary is highly challenging and introduces several complications. In cryptography, the security of a scheme is typically proved via a reduction argument. To this end, one constructs an algorithm R (the *reduction*) that simulates the execution environment of the scheme (public keys, corruptions, oracle responses, etc.) for an adversary A that is assumed to break the security of the scheme with non-negligible probability. The reduction then uses A as a subroutine to solve an underlying hard number-theoretic problem (e.g., the discrete logarithm problem) with non-negligible probability. By contraposition, this implies that the scheme is secure under the given hardness assumption.

Security proofs of distributed protocols in the static-adversary model are easier because the distribution of keys, commitments, and randomness can be conditioned on a *fixed* corruption set. These proofs often rely on embedding instances of a hardness assumption into the honest parties' randomness. With an adaptive adversary, however, one does not know in advance which parties will remain honest, so this technique breaks down. Moreover, when a party is corrupted, the adversary learns its entire internal state (secret keys, randomness, temporary variables). Upon that event of corruption, the simulator must therefore produce an internal state that is *consistent* with everything the adversary has already observed, including commitments to keys, randomness, and other protocol data. This is known as the famous *decommitment problem* in the MPC literature [ACS22].

As a consequence, many distributed protocols achieving adaptive security have to make major sacrifices, such as relying on parties to erase their internal states, resorting to computationally inefficient cryptographic tools such as non-committing encryption, introducing additional rounds of communication, and other costly modifications [Can+99; Nie02]. On the other hand, the generic approach of transforming a statically secure protocol into an adaptively secure one by guessing the set of parties an adaptive adversary may corrupt incurs an undesirable exponential security loss of $\binom{n}{t}$. While this approach is acceptable only for a small number of parties, many applications require n to be in the range of several thousands. Notably, the U.S. National Institute of Standards and Technology (NIST) recently published a call for threshold cryptography [BD22; BP25] and included adaptive security as a main goal, ideally supporting $n = 2^{10}$ or more parties.

1.1.2 Efficiency in Distributed Protocols

There are two network models relevant to this thesis. In a *synchronous network*, communication between parties proceeds in compute-send-receive rounds of known length $\Delta > 0$ (referred to as the *network delay*). When an honest party sends a message at the beginning of a round, the message is guaranteed to be received by the intended party by the end of that round. In an *asynchronous network*, messages can be delayed arbitrarily, under the condition that messages sent between honest parties must eventually be delivered. The difficulty in designing asynchronous protocols lies in the fact that it is, in general, not possible to distinguish a slow

honest party from a non-responding Byzantine party. Furthermore, it is typically assumed that a Byzantine adversary has full control over message delays, subject to the constraints specified by the respective network model.

When evaluating distributed cryptographic protocols such as DKG or distributed randomness beacons, the most critical efficiency metrics are typically communication complexity, round complexity, and computational cost per party. Roughly speaking, these notions can be defined as follows.

- **Communication Complexity.** We measure the per-party communication by counting the number of bits each party must send during an execution of the protocol. The total communication is then obtained by summing over all parties. Without assuming any form of strong trusted setup,[2] most distributed cryptographic protocols require $O(\lambda n^2)$ total communication, where λ denotes the length of the underlying cryptographic primitives, such as digital signatures or hash function outputs.

- **Round Complexity.** We measure the round complexity by counting the number of rounds required to complete the protocol. These may be either synchronous or asynchronous rounds, depending on the underlying network model. Without assuming shared randomness as part of the setup, most distributed cryptographic protocols require $O(n)$ rounds to terminate. However, there also exist randomized protocols that complete in a constant number of rounds, which is particularly valuable in practice.

- **Computational Cost.** We measure the per-party computational cost by counting the number of exponentiations, pairings, hash evaluations, or other cryptographic operations performed by each party. It is highly desirable to make use of lightweight cryptographic operations and avoid computationally heavy primitives such as succinct non-interactive arguments of knowledge (SNARKs) or similar constructions.

In many distributed cryptographic protocols, computation is no longer the main bottleneck. Instead, network bandwidth and latency have become the dominant performance constraints, especially in large-scale systems such as the Internet or blockchain networks, where parties (i.e., servers) are geographically distributed. In such settings, even modest increases in message size or the number of communication rounds can lead to substantial slowdowns due to network congestion, propagation delays, and queuing effects [Blu+20; Yin+19].

High communication cost directly slows down protocol execution and can exacerbate scalability limitations, as the overall time to complete an operation grows with the number of messages that need to be transmitted and verified. Moreover, increased communication overhead also enlarges the attack surface - for example, by providing more opportunities for an adversary to intercept, delay, or manipulate messages. Conversely, reducing communication complexity yields multiple benefits: it decreases latency and improves throughput, thereby enhancing the scalability and responsiveness of the protocol. It also reduces exposure to network-level attacks and facilitates deployment in heterogeneous environments, where bandwidth and connectivity may vary significantly across participants.

[2] By this, we typically refer to protocols that do not rely on any private setup assumptions.

1.1.2.1 Adaptive Adversary and Communication Complexity

There exist many distributed protocols that achieve low communication complexity in the presence of a static adversary [GK24; Bha+25]. This is possible because one can assign special roles to a small subset of parties (e.g., leaders, committees, or dealers) under the assumption that these parties will not be corrupted during the protocol execution. This allows for efficient communication patterns, such as one-to-many or small-committee-to-all broadcasts.

However, employing these techniques becomes much more challenging in the presence of an adaptive adversary, as it can strategically corrupt leaders or committee members once they are known. For instance, many communication-efficient protocols proceed in multiple iterations, each with a randomly elected leader. In the static setting, the leader is expected to be honest with constant probability (e.g., every two iterations), allowing the protocol to make effective progress and terminate quickly, thus keeping communication overhead low. In contrast, under adaptive corruptions, the adversary can observe the leader election and immediately corrupt the elected leader in every iteration, effectively delaying progress. As a result, the protocol may require up to $t \in \Theta(n)$ additional iterations, leading to a proportional blow-up in communication cost.

To counter adaptive attacks, protocols typically employ secret leader or committee election mechanisms (e.g., using verifiable random functions), or increase redundancy through multiple leaders, overlapping committees, or all-to-all verification. These countermeasures, however, inevitably increase both the communication and round complexity. In fact, several impossibility results for distributed consensus protocols [Abr+19a] establish a fundamental separation between static and adaptive security. For example, Algorand [Gil+17a] achieves sub-quadratic communication under static security by electing small committees of size $O(\log n)$ for block proposal and signing. In contrast, any adaptively secure Byzantine consensus protocol tolerating $t \in \Theta(n)$ corruptions must incur at least $\Omega(n^2)$ communication.

1.2 Our Research Goal

In this thesis, our primary goal is to strengthen the security guarantees and improve the efficiency of distributed key generation (DKG) protocols and their applications, such as threshold signatures and distributed randomness beacons. More concretely, our work proceeds along two complementary directions.

On the one hand, we revisit the adaptive security of several distributed cryptographic protocols that were previously proven secure only against static adversaries. Our objective is to analyze and upgrade the most efficient existing schemes to achieve adaptive security, thereby enabling real-world systems to rely on practical, high-performance protocols rather than less efficient adaptively secure alternatives. In particular, we focus on state-of-the-art DKG protocols, publicly verifiable secret sharing (PVSS) schemes, distributed randomness beacon protocols, and the threshold BLS signature, which are core primitives in modern consensus and distributed system protocols. On the other hand, we aim to design distributed cryptographic protocols that achieve improved efficiency while offering stronger and more versatile security guarantees than existing constructions. Our objective is to develop practically efficient protocols suitable for deployment in real-world decentralized systems. Specifically, we focus on designing DKG and distributed randomness beacon protocols that achieve adaptive or network-agnostic security with near-optimal communication complexity.

1.3 Overview of Our Results

We summarize our results as follows. In the following, we reuse parts of the abstracts of [BL22a; Bac+23a; BL23a; Bac+24a].

1.3.1 Results in Chapter 3

Threshold signatures are a crucial tool for many distributed protocols. As shown by Cachin, Kursawe, and Shoup (ACM PODC 2000), schemes with *unique signatures* are of particular importance, as they allow the implementation of distributed coin flipping in a highly efficient manner and without requiring any timing assumptions. This makes them an ideal building block for inherently randomized asynchronous consensus protocols. The threshold BLS signature scheme proposed by Boldyreva (IACR PKC 2003) is both unique and very compact, but unfortunately lacks a formal security proof against adaptive adversaries. As a result, current consensus protocols either rely on less efficient alternatives or lack adaptive security.

In this work, we revisit the security of the threshold BLS signature by establishing the following results under the assumption of t adaptive corruptions:

- We give a modular security proof that follows a two-step approach: (1) We introduce a new security notion for distributed key generation (DKG) protocols and show that it is satisfied by several existing protocols that previously only had a *static security proof*. (2) Assuming *any* DKG protocol with this property, we then prove the unforgeability of the threshold BLS signature scheme. Our reductions are *tight* and can therefore be used to justify concrete real-world parameter choices.

- To justify our use of strong assumptions such as the algebraic group model (AGM) and the hardness of the one-more discrete logarithm (OMDL) problem, we prove two impossibility results. First, without the AGM, we rule out a natural class of tight security reductions from $(t + 1)$-OMDL. Second, *even within the AGM*, a strong interactive assumption is required in order to prove the scheme secure.

1.3.2 Results in Chapter 4

Publicly Verifiable Secret Sharing (PVSS) is a fundamental primitive that allows a dealer to share a secret S among n parties via a publicly verifiable transcript T. Existing efficient PVSS schemes are only proven secure against *static adversaries*, which must decide which parties to corrupt before the protocol execution begins. As a result, any protocol (e.g., a distributed randomness beacon) that builds on top of such a PVSS scheme inherits this limitation.

To overcome this limitation, we revisit the security of PVSS under *adaptive corruptions* and show that, perhaps surprisingly, many existing schemes from the literature already achieve meaningful levels of adaptive security.

- We propose a new security definition for *aggregatable PVSS* - i.e., schemes that allow multiple transcripts to be homomorphically combined into one compact aggregate transcript AT that shares the sum of their individual secrets. Our notion captures the requirement that, if the secret shared by AT contains at least one contribution from an honestly generated transcript, it should remain computationally unpredictable. We then

prove that several existing schemes satisfy this notion against adaptive corruptions in the algebraic group model (AGM).

- To motivate our new definition, we show that it implies the adaptive security of two recent randomness beacon protocols, SPURT (IEEE S&P 2022) and OptRand (NDSS 2023), which build upon aggregatable PVSS schemes satisfying our unpredictability notion. For a security parameter λ, our results improve the communication complexity of the best-known adaptively secure randomness beacon protocols to $O(\lambda n^2)$ in synchronous networks with $t < n/2$ adaptive corruptions and partially synchronous networks with $t < n/3$ adaptive corruptions.

1.3.3 Results in Chapter 5

A *randomness beacon* is a source of continuous and publicly verifiable randomness, which is crucial for numerous cryptographic and distributed applications. Existing works on randomness beacons suffer from at least one of the following drawbacks: (i) security only against static (i.e., non-adaptive) adversaries, (ii) many rounds of communication per epoch, or (iii) reliance on computationally expensive tools such as proof-of-work (PoW) or verifiable delay functions (VDFs).

In this work, we introduce GRandLine, the first adaptively secure randomness beacon protocol that overcomes all these limitations while preserving simplicity and achieving optimal resilience in the synchronous network setting. Our protocol has the following properties:

- *Adaptive Security.* It is secure in the presence of an adaptive adversary.
- *One-Round Epoch.* Each epoch requires only a single asynchronous round of communication and relies on synchrony solely during its pre-processing phase.
- *Communication Efficiency.* It achieves a communication cost of $O(\lambda n^2)$ bits per epoch, where λ is the security parameter.
- *Optimal Resilience.* It achieves an optimal resilience threshold of $t < n/2$ in the synchronous network setting.
- *Lightweight Tools.* After its pre-processing phase, it uses only lightweight cryptographic primitives such as hash functions and pairings. Notably, it does not rely on tools like proof-of-work or verifiable delay functions.
- *Quadratic Pre-Processing.* Its pre-processing phase incurs a communication cost of only $O(\lambda n^2 \log n)$ bits.

We achieve our result in two steps. First, we design a novel distributed key generation (DKG) protocol, GRand, with $O(\lambda n^2 \log n)$ bits of communication cost that, unlike most conventional DKG protocols, outputs both secret and public keys as group elements. It is the first DKG protocol in any network setting that achieves subcubic communication cost with an optimal resilience threshold. Second, we present a simple construction that allows the keys output by GRand to be used for a non-interactive and unique *locally verifiable* threshold signature,[3] from

[3] By that we mean that the final threshold signature does not have an efficient verification algorithm independent of n, but each partial signature does.

which we naturally derive a one-round randomness beacon using a final hash operation. This enables the parties to generate a sequence of randomness beacon values, where each value requires only a single asynchronous round and $O(\lambda n^2)$ bits of communication.

We implement GRandLine and evaluate it in a network of up to 64 parties deployed across geographically distributed AWS instances. Our evaluation shows that GRandLine can produce approximately two beacon outputs per second in a 64-party network. We compare our protocol against state-of-the-art randomness beacon schemes - OptRand (NDSS 2023), BRandPiper (ACM CCS 2021), and Drand - in the same setting and observe that it significantly outperforms all of them.

1.3.4 Results in Chapter 6

Distributed key generation (DKG) protocols are an essential building block for threshold cryptosystems. Many DKG protocols tolerate up to $t_s < n/2$ corruptions under the assumption of a well-behaved synchronous network, but become insecure as soon as the network delay becomes unstable. On the other hand, solutions in the asynchronous model operate under arbitrary network conditions but only tolerate $t_a < n/3$ corruptions, even when the network is well-behaved.

In this work, we ask whether one can design a protocol that achieves security guarantees in both scenarios. We give a complete characterization of *network-agnostic* DKG protocols, showing that the tight bound is given by the identity $t_a + 2t_s < n$. More specifically, we propose the first network-agnostic DKG protocol. Our protocol tolerates $n/3 < t_s < n/2$ corrupted parties in the synchronous model and $t_a < n/3$ corrupted parties in the asynchronous model, where t_a and t_s can be chosen arbitrarily subject to $t_a + 2t_s < n$. Our protocol is resilience-optimal, since we also prove that $t_a + 2t_s < n$ is *necessary* for any network-agnostic DKG. It operates within the plain PKI model and allows the parties to share a field element sk corresponding to the public key $pk = g^{sk}$ with only $O(\lambda n^3)$ communication complexity. This matches the best-known results in the synchronous setting (Shrestha et al., IACR CiC 2024) and the asynchronous setting (Das et al., IEEE S&P 2022). Thus, our DKG protocol can be used to efficiently bootstrap trusted key generation for network-agnostic consensus and MPC protocols. In summary, our construction incurs no additional setup assumptions or asymptotic overhead compared to state-of-the-art communication-efficient protocols for the synchronous and asynchronous network models. This demonstrates that network-agnostic security comes *(almost) for free* in DKG.

1.4 Structure of the Thesis

In Chapter 2, we introduce the general preliminaries used throughout the thesis. Chapters 3 through 6 present our main results, with each chapter corresponding to one of our publications. More concretely, in Chapter 3, we present our results on the adaptive security of the threshold BLS signature scheme. Chapter 4 discusses our results on the adaptive security of (aggregatable) PVSS schemes and their applications to distributed randomness beacons. In Chapter 5, we present our results on distributed key generation and distributed randomness beacons with (near-)quadratic communication complexity. In Chapter 6, we present our results on the network-agnostic DKG protocol. Finally, in Chapter 7, we summarize our findings and outline several open problems that may guide future research.

2
General Preliminaries

In this chapter, we introduce notation and conventions that are used throughout the thesis. Most of these are quite standard in the cryptography literature.

2.1 Notation

We denote the security parameter by λ. We denote the set of integers by $\mathbb{Z}$, and the set of positive integers by $\mathbb{N}$. For a positive integer n, we define $[n] := \{1, \ldots, n\}$ and $[\![n]\!] := \{0, \ldots, n\}$; and for two integers $a \leq b$, we define $[a, b] := \{a, \ldots, b\}$. Further, we denote the ring of integers modulo n by $\mathbb{Z}_n := \mathbb{Z}/n\mathbb{Z}$, and we identify it with the canonical set of representatives $\{0, \ldots, n-1\}$. Its multiplicative group of units is denoted by $\mathbb{Z}_n^*$. Further, we denote the set of all finite bit strings by $\{0,1\}^*$. For a finite set S, we use $s \leftarrow S$ (and sometimes $s \leftarrow_\$ S$) to denote that s is sampled uniformly at random from S, and $|S|$ denotes the size of S. Unless stated otherwise, we assume all algorithms to be probabilistic and written in uppercase serif letters. For an algorithm A, we write T_A (or simply T) to denote an upper bound on the running time of A, which is an implicit function of A's input length. Further, we write $y \leftarrow \mathsf{A}(x)$ to denote that A is run on input x and uniformly random coins, whose resulting output is then assigned to y. If A is deterministic, we instead write $y := \mathsf{A}(x)$ for that. We further write $y \in \mathsf{A}(x)$ to mean that there are some random coins such that y is the output of A (on the input x and these coins). If A has oracle access to some algorithm B during its execution, then we write A^B for that. Oracle access means that A can submit an input x to B and obtains the output $\mathsf{B}(x)$ in return. We define security notions via probabilistic experiments, also commonly known as games. We write $\mathbf{G}^\mathsf{A} = 1$ to denote the event that a security experiment $\mathbf{G}$ involving algorithm A outputs 1. And we write $\Pr[E]$ to denote the probability of an event E occurring. We measure the communication complexity of distributed protocols in bits, and typically assume that their security properties hold except with probability negligible in λ.

2.2 Our Model

We define the model relevant for this thesis. This includes the distributed systems and adversarial model, and also idealizations such as the random oracle model.

Communication Model. We consider a system of n parties $P_1, \ldots, P_n$ (modeled as probabilistic polynomial-time algorithms). We assume that the parties are connected by a complete network of pairwise authenticated channels, i.e., the receiver of a message is aware of the sender's identity. For mathematical formulations, we may also use the indices $\{1, \ldots, n\}$ for the parties $\{P_1, \ldots, P_n\}$. Further, we assume a plain public key infrastructure (PKI). That is, each party P_i has a locally generated public-secret key pair (pk_i, sk_i), where pk_i is known to all parties but sk_i is known only to P_i. Further, we assume that the pairs (pk_i, sk_i) also (implicitly) include verification-signing key pairs (vk_i, dk_i) for a digital signature scheme to provide authentication. In particular, we assume that parties sign each message before they send it to other parties.

Adversarial Model. We consider an adversary A (modeled as probabilistic polynomial-time algorithm) that can corrupt up to t out of all the n parties in the system. We further assume a malicious adversary that may cause corrupted parties to deviate from the protocol specification arbitrarily, also referred to as a *Byzantine* adversary. The adversary is either *static* or *adaptive*: A static adversary has to specify the set of corrupted parties before the protocol execution, while an

adaptive adversary can dynamically corrupt parties over time throughout the protocol execution. Note that once a party is corrupted, it remains corrupt. Further, we assume that the adversary is in full control over message delays, subject to the network conditions as specified below.

Network Model. There are two types of networks relevant for this thesis. In a *synchronous network*, communication between parties proceeds in compute-send-receive rounds of known length $\Delta > 0$, termed as the network delay. When an honest party sends a message at the beginning of a round, the message is guaranteed to be received by the receiving party by the end of that round. In an *asynchronous network*, any message can be delayed arbitrarily under the condition that messages sent between honest parties must eventually be delivered.

Random Oracle Model. We assume the random oracle model (ROM) [BR93]. In this model, a hash function H is treated as an idealized random function. Concretely, H is modeled as an oracle with the following properties. The oracle internally keeps a list H for bookkeeping purposes. At the beginning, all entries of H are set to a default value $\perp$. On input m from the domain of H, the oracle first checks whether $H[m] \neq \perp$. If so, it returns $H[m]$. Otherwise, it sets $H[m]$ to a uniformly random value in the codomain of H and then returns $H[m]$.

Algebraic Group Model. The algebraic group model (AGM) [FKL18] was introduced by Fuchsbauer, Kiltz, and Loss as a model in between the generic group model (GGM) and the standard model. In the AGM, all algorithms are treated as *algebraic*. This means that whenever an algorithm outputs a group element, it must also output a representation of that element relative to all of the inputs the algorithm has received up to that point. This captures the intuition that any reasonable algorithm should know how it computes its outputs from its inputs. Formally, an algorithm A is called *algebraic* (over group $\mathbb{G}$) if for all group elements $\zeta \in \mathbb{G}$ that A outputs, it additionally outputs a vector $\vec{z} = (z_1, \ldots, z_k)$ of integers such that $\zeta = \prod_{i \in [k]} g_i^{z_i}$, where $(g_1, \ldots, g_k)$ is the list of group elements A has seen so far. We will make use of the algebraic group model in some of our security proofs, assuming the adversary to behave algebraic.

2.3 Cryptographic Assumptions

We define the most relevant computational hardness assumptions for this thesis. For this, let $(\mathbb{G}, p, g)$ define group parameters, where $\mathbb{G}$ is a cyclic group of prime order p and g is a generator.

One-More Discrete Logarithm. A computational hardness assumption that finds wide-ranging application in modern cryptography is the *one-more discrete logarithm (OMDL) assumption* introduced in [Bel+03]. It is foundational for the security analysis of identification protocols, and various signature schemes such as blind signatures and multi-signatures. Henceforth, we denote by $\mathsf{DL}_g()$ an oracle that on input $h = g^x \in \mathbb{G}$ returns the discrete logarithm x of h to the base g.

Definition 2.3.1 (One-More Discrete Logarithm Problem)**.** Let $par := (\mathbb{G}, p, g)$ define group parameters as specified above. For $k \in \mathbb{N}$ and an algorithm A, we define the experiment $k\text{-}\mathsf{OMDL}^{\mathsf{A}}$ as follows:

- **Offline Phase.** Sample values $(z_1, \ldots, z_k) \leftarrow \mathbb{Z}_p^k$ uniformly at random and define the elements $\xi_i := g^{z_i} \in \mathbb{G}$ for all $i \in [k]$.

- **Online Phase.** Run A on input $par = (\mathbb{G}, p, g)$ and $(\xi_1, \ldots, \xi_k)$. In this phase, A gets access to the oracle $\mathsf{DL}_g()$.
- **Output Determination.** Let $(z'_1, \ldots, z'_k)$ denote the output of A. Then, return 1 if (i) $z'_i = z_i$ for all $i \in [k]$, and (ii) $\mathsf{DL}_g()$ was queried at most $k-1$ times during the online phase. Otherwise, return 0.

We say that the one-more discrete logarithm problem of degree k is (ε, T)-hard if for all algorithms A running in time at most T, $\Pr[k\text{-}\mathsf{OMDL}^{\mathsf{A}} = 1] \leq \varepsilon$. Conversely, we say that an algorithm A (ε, T)-solves the one-more discrete logarithm problem of degree k if it runs in time at most T and $\Pr[k\text{-}\mathsf{OMDL}^{\mathsf{A}} = 1] > \varepsilon$.

q-Discrete Logarithm. A computational hardness assumption that is similar to the preceding OMDL assumption, but with the crucial difference that it is non-interactive (i.e., there is no oracle for the adversary). A special feature of q-DLOG is the fact that it implies every conceivable hardness assumption in the AGM (and even some interactive assumptions) [BFL20].

Definition 2.3.2 (q-Discrete Logarithm Problem)**.** Let $par := (\mathbb{G}, p, g)$ define group parameters as specified above. For $q \in \mathbb{N}$ and an algorithm A, we define the experiment q-$\mathsf{DLOG}^{\mathsf{A}}$ as follows:

- **Offline Phase.** Sample a value $z \leftarrow \mathbb{Z}_p$ uniformly at random and define elements $\xi_i := g^{z^i} \in \mathbb{G}$ for all $i \in [q]$.
- **Online Phase.** Run A on input $par = (\mathbb{G}, p, g)$ and $(\xi_1, \ldots, \xi_q)$.
- **Output Determination.** Let z' denote the output of A. Then, return 1 if $z' = z$, and return 0 otherwise.

We say that the q-discrete logarithm problem is (ε, T)-hard if for all algorithms A running in time at most T, $\Pr[q\text{-}\mathsf{DLOG}^{\mathsf{A}} = 1] \leq \varepsilon$. Conversely, we say that an algorithm A (ε, T)-solves the q-discrete logarithm problem if it runs in time at most T and also $\Pr[q\text{-}\mathsf{DLOG}^{\mathsf{A}} = 1] > \varepsilon$.

2.4 Cryptographic Primitives

We define syntax and security of basic cryptographic primitives relevant for this thesis. For this, we assume that system parameters *par* are given (which may vary depending on the chapter). Typically, these include group parameters and sometimes additional data such as the description of used hash functions.

Digital Signature Scheme. A digital signature scheme provides a user with a verification-signing key pair (vk, dk), where the signing key is only known to the user but the verification key is public. The signing key allows the user to sign any message of its choice, while any third party that knows vk can verify that the message was indeed signed by that particular user.

Definition 2.4.1 (Digital Signature Scheme)**.** A digital signature scheme is a tuple of algorithms $\mathsf{DS} = (\mathsf{SKey}, \mathsf{Sign}, \mathsf{Ver})$ with the following properties:

- SKey: The randomized *key generation algorithm* takes as input system parameters *par*. It outputs a verification key vk and a signing key dk.

- Sign: The possibly randomized *signing algorithm* takes as input a signing key dk and a message m. It outputs a signature σ.
- Ver: The deterministic *verification algorithm* takes as input a verification key vk, a message m, and a signature σ. It outputs 1 (valid) or 0 (invalid).

For a secure digital signature, we require that an adversary cannot create a signature on a new message, even after obtaining many signatures on messages of its choice.

Definition 2.4.2 (Unforgeability Under Chosen Message Attack)**.** Let $\mathsf{DS} = (\mathsf{SKey}, \mathsf{Sign}, \mathsf{Ver})$ be a digital signature scheme as defined above. For an algorithm A, we define the experiment $\mathbf{UF\text{-}CMA}^{\mathsf{A}}_{\mathsf{DS}}$ as follows:

- *Offline Phase.* Run SKey on input par to generate a key pair (vk, dk). Run A on input (par, vk). Initialize a queried message set $\mathcal{M} := \emptyset$.
- *Signing Oracle Queries.* At any point of the experiment, A gets access to an oracle that answers queries of the following type: When A submits a message m, return $\sigma \leftarrow \mathsf{Sign}(dk, m)$ and update $\mathcal{M} := \mathcal{M} \cup \{m\}$.
- *Output Determination.* When A outputs a message m^* and a signature σ^*, do the following. If $\mathsf{Ver}(vk, m^*, \sigma^*) = 1$ and $m^* \notin \mathcal{M}$, return 1. Otherwise, return 0.

We say that DS is (ε, T, q_s)*-unforgeable under chosen message attacks (UF-CMA)* if for all algorithms A that run in time at most T and make at most q_s signing oracle queries, $\Pr[\mathbf{UF\text{-}CMA}^{\mathsf{A}}_{\mathsf{DS}} = 1] \leq \varepsilon$. Conversely, we say that A (ε, T, q_s)*-breaks unforgeability* of DS under chosen message attacks if it runs in time at most T, makes at most q_s signing oracle queries, and $\Pr[\mathbf{UF\text{-}CMA}^{\mathsf{A}}_{\mathsf{DS}} = 1] > \varepsilon$.

Linear Erasure and Error Correcting Codes. We use standard (q, b)-Reed-Solomon (RS) codes. This primitive allows to encode b data symbols into code words of q symbols such that b elements of the code word suffice to recover the original data.

Definition 2.4.3 (Reed-Solomon Code.)**.** A Reed-Solomon code is a tuple of deterministic algorithms $\Sigma = (\mathsf{Encode}, \mathsf{Decode})$ with the following properties:

- Encode: The deterministic *encoding algorithm* takes as input b data symbols $(m_1, \ldots, m_b)$. It outputs a code word $(s_1, \ldots, s_q)$ of length q. Knowledge of any b elements of the code word uniquely determines the input message and the remaining of the code word.
- Decode: The deterministic *decoding algorithm* takes as input a code word $(s_1, \ldots, s_q)$ of length q. It outputs b data symbols $(m_1, \ldots, m_b)$. This algorithm tolerates up to c errors and d erasures in a code word $(s_1, \ldots, s_q)$ if and only if $q - b \geq 2c + d$.

Cryptographic Accumulator. A cryptographic accumulator allows to accumulate several elements from some set D into an accumulated value z. Further, for each element in D it allows to generate a compact proof of membership in D.

Definition 2.4.4 (Cryptographic Accumulator)**.** A cryptographic accumulator scheme is a tuple of probabilistic polynomial-time algorithms $\Sigma = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Wit}, \mathsf{Ver})$ with the following properties:

- Gen: The randomized *accumulator key generation algorithm* takes as input the security parameter λ and an accumulation threshold n. It outputs a (public) accumulator key ak.
- Eval: The deterministic *evaluation algorithm* takes as input an accumulator key ak and a set $D = \{d_1, \ldots, d_n\}$ of elements. It outputs an accumulation value z for the set D.
- Wit: The possibly randomized *witness creation algorithm* takes as input an accumulator key ak, an accumulation value z for D, and an element d_i. It outputs $\perp$ if $d_i \notin D$, and a witness w_i otherwise.
- Ver: The deterministic *verification algorithm* takes as input an accumulator key ak, an accumulation value z for D, a witness w_i, and an element d_i. It outputs 1 if w_i is a valid proof for membership $d_i \in D$, and 0 otherwise.

We define security of an accumulator scheme as standard via collision-resistance. Intuitively, this notion prohibits an adversary to create invalid proofs of membership.

Definition 2.4.5 (Collision-Resistant Accumulator)**.** Let $\Sigma = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Wit}, \mathsf{Ver})$ be a cryptographic accumulator scheme as defined above. For an algorithm A and $k \in \mathbb{N}$, we define the experiment $\mathbf{CR}^{\mathsf{A}}_{\Sigma,k}$ as follows:

1. *Offline Phase.* Run the accumulator key generation algorithm on input λ and n to generate an accumulator key $ak \leftarrow \mathsf{Gen}(\lambda, k)$.
2. *Online Phase.* Run A on input *par* and (k, ak). At some point in the experiment, A returns a tuple $(\{d_1, \ldots, d_k\}, d', w')$.
3. *Output Determination.* Compute the accumulation value $z \leftarrow \mathsf{Eval}(ak, \{d_1, \ldots, d_k\})$. Return 1 if $\mathsf{Ver}(ak, z, w', d') = 1$ and $d' \notin \{d_1, \ldots, d'\}$. Return 0 otherwise.

We say that Σ is (ε, T, k)*-collision-resistant* if for all algorithms A that run in time at most T and all $\ell \in [k]$, we have $\Pr[\mathbf{CR}^{\mathsf{A}}_{\Sigma,\ell} = 1] \leq \varepsilon$. Conversely, we say that A (ε, T, k)*-breaks collision-resistance* of Σ if it runs in time at most T and $\Pr[\mathbf{CR}^{\mathsf{A}}_{\Sigma,\ell} = 1] > \varepsilon$ for some $\ell \in [k]$.

3

On the Adaptive Security of the Threshold BLS Signature Scheme

Details on this Chapter

This chapter is based on the conference publication [BL22a] and its full version [BL22b]. I contributed to it as the main author. The original publication has been partially reordered, and editorial improvements have been made throughout the text.

Threshold signatures are a crucial tool for many distributed protocols. As shown by Cachin, Kursawe, and Shoup (ACM PODC 2000), schemes with *unique signatures* are of particular importance, as they allow to implement distributed coin flipping very efficiently and without any timing assumptions. This makes them an ideal building block for (inherently randomized) asynchronous consensus protocols. The threshold BLS signature of Boldyreva (IACR PKC 2003) is both unique and very compact, but unfortunately lacks a security proof against adaptive adversaries. Thus, current consensus protocols either rely on less efficient alternatives or are not adaptively secure. In this work, we revisit the security of the threshold BLS signature by showing the following results, assuming t *adaptive* corruptions:

- We give a modular security proof that follows a two-step approach: (1) We introduce a new security notion for distributed key generation protocols (DKG). We show that it is satisfied by several protocols that previously only had a *static security proof*. (2) Assuming *any* DKG protocol with this property, we then prove unforgeability of the threshold BLS scheme. Our reductions are *tight* and can be used to substantiate real-world parameter choices.
- To justify our use of strong assumptions such as the algebraic group model (AGM) and the hardness of one-more discrete logarithm (OMDL), we prove two impossibility results: (1) Without the AGM, we rule out a natural class of tight security reductions from $(t + 1)$-OMDL. (2) *Even in the AGM*, a strong interactive assumption is required in order to prove the scheme secure.

3.1 Introduction

Threshold signatures are a special type of digital signature that allows a sufficiently large set of $t + 1$ signers to jointly create a compact signature σ on a message m. At the same time, it should be infeasible for t or less signers to create a signature on m. For this reason, t is usually referred to as the *threshold* of the scheme. In this manner, one can create a very size-efficient proof that at least $t + 1$ parties have signed m. This makes threshold signatures an important building block for storage-sensitive systems such as blockchain protocols. Another intriguing application of threshold signatures is *distributed coin tossing*. Using a threshold signature scheme with unique signatures (per message m and public key pk) and a non-interactive signing procedure, one can efficiently agree on an unpredictable and unbiasable coin $b \in \{0, 1\}$ among n parties $P_1, \ldots, P_n$ as follows:

- Each party P_i (non-interactively) creates a signature share σ_i for some predetermined message m and sends σ_i to all parties.
- Upon collecting $t + 1$ signature shares σ_i, a party locally reconstructs the full signature σ for m.
- All parties can now derive the coin via $b := \mathsf{LSB}(\mathsf{H}(\sigma))$, where H is a suitable randomness extractor, e.g., a hash function (modeled as a random oracle).

Note that in the above construction, the uniqueness property is crucially used in two places. First, it ensures that all parties agree on the *same signature* σ, and, by extension, on the same

coin b. Second, uniqueness prevents a malicious adversary from biasing the outcome of the coin b by sending or withholding particular signature shares. Finally, σ (and therefore b) remains unpredictable to an adversary controlling at most t parties up until the point where the first honest party P_i participates in the coin toss by sending its share σ_i. These combined features make (unique and non-interactive) threshold signatures a crucial tool for the design of efficient randomized consensus protocols [CKS00; Abr+19b; AMS19; Lu+20; Guo+20]. This applies particularly to the fully asynchronous network setting, where consensus is known to be *impossible* unless randomized protocols are used [FLP85].

Static vs. Adaptive Corruptions. Cachin et al. [CKS00] were the first to realize the enormous potential of unique threshold signatures for building efficient asynchronous consensus algorithms. Their signature of choice was the threshold version of the full-domain-hash RSA signature [Sho00]. However, this scheme is only secure against a *static adversary* who chooses all corrupted parties at the beginning of the protocol (after observing their public keys). Modern systems, on the other hand, often require security against a much more powerful *adaptive adversary* who dynamically corrupts parties over time by observing the flow of the protocol execution. In addition, RSA signatures are rather large, taking up an order of magnitude more storage space than schemes based on elliptic curve cryptography. Therefore, an appealing alternative is the much more size-efficient scheme of Boldyreva [Bol03], which is based on the BLS signature scheme. Unfortunately, however, Boldyreva's scheme also lacks an adaptive security proof. To overcome these limitations, Libert et al. [LJY14] proposed an adaptively secure construction based on Boldyreva's scheme, which, thus far, has served as the state-of-the-art for building adaptively secure consensus protocols [Abr+19b; AMS19; Lu+20; Guo+20]. While their construction is still far more size-efficient than an RSA signature, it is roughly twice as expensive to store and verify as signatures in Boldyreva's original scheme. In addition, while Boldyreva's signature is compatible with modern BLS libraries [Chi22], Libert et al.'s scheme lacks such a compatibility. (There is, however, an efficient implementation of their scheme available at [Mou18].) Motivated by the above discussion, we ask the following question: *What are the adaptive security guarantees of the threshold BLS signature scheme?*

3.1.1 Our Contributions

Our results can be summarized as follows:

- We give a modular security proof that follows a two-step approach: (1) We introduce a new security notion for distributed key generation protocols (DKG). We show that it is satisfied by several protocols that previously only had a *static security proof*. (2) Assuming *any* DKG protocol with this property, we then prove unforgeability of the threshold BLS scheme. Our reductions are *tight* and can be used to substantiate real-world parameter choices.

- To justify our use of strong assumptions such as the algebraic group model (AGM) and the hardness of one-more discrete logarithm (OMDL), we prove two impossibility results: (1) Without the AGM, we rule out a natural class of tight security reductions from $(t + 1)$-OMDL. (2) *Even in the AGM*, a strong interactive assumption is required in order to prove the scheme secure.

3.2 Technical Overview

In the following, we explain the challenges and our novel techniques.

3.2.1 Handling Adaptive Corruptions

Our starting point is the construction of Libert et al. who gave the first adaptively secure, non-interactive, and unique threshold signature scheme in the random oracle model. Their adaptive security argument also extends to the distributed key generation (DKG) phase that sets up the shared keys for parties in the system. The established way of proving security for a DKG protocol is to argue that the messages that are exchanged as part of the protocol reveal nothing further about the distributed secret key sk (beyond what is already revealed by pk) [Gen+07]. This can be done by providing an efficient *simulator* Sim that, on input pk, provides a properly distributed view of an execution of the DKG protocol in which parties agree on the public key pk. In case corruptions are static, Sim also gets the set of corrupted parties as input. Assuming that Sim provides a perfect simulation (as indeed is often the case), this technique works even against an information-theoretic adversary who can compute sk from pk by brute force. Clearly, such an adversary could easily forge a signature with respect to sk, so what does this mean?

The Challenge of Adaptive Corruptions. One of the main insights of Libert et al. is to prove unforgeability of their scheme directly by reducing from a computational assumption. In this manner, their proof bypasses many of the issues encountered in the DKG literature when having to deal with adaptive corruptions. However, one central issue still remains: Sim needs to simulate correctly distributed internal states (including secret keys) of parties upon corruption. Existing DKG protocols overcome this problem by relying on heavy tools such as non-committing encryption and erasures [Can+99], [JL00]. This is a common and often frustrating issue to deal with in the context of simulation based security proofs. Namely, even if a statically secure protocol can not be simulated, it is far from clear whether it would actually be *insecure* in the presence of adaptive corruptions. This issue is also quite prominent in the context of threshold BLS signatures, even when a trusted dealer distributes the keys. Recall that in order to create a signature share σ_i on message m, party P_i computes $\mathsf{H}(m)^{sk_i}$. Here, $\mathsf{H} : \{0,1\}^* \to \mathbb{G}$ is a hash function that is modeled as a random oracle, $\mathbb{G}$ is a cyclic group of known prime order p, and $sk_i \in \mathbb{Z}_p$ denotes P_i's secret key share. The corresponding public key shares of parties are g^{sk_i}, where g is a known generator of $\mathbb{G}$. Now, assume that keys have been set up in such a way that for all $i \in [n]$, $sk_i = f(i)$ and $sk = f(0)$ for some suitable polynomial $f \in \mathbb{Z}_p[X]$ of degree t. Then one can compute a signature σ that verifies relative to $pk = g^{sk}$ from $t+1$ shares $\sigma_1, \ldots, \sigma_{t+1}$ by interpolating f in the exponent of g. It can be verified by checking the symmetric pairing equation $e(\sigma, g) = e(\mathsf{H}(m), pk)$. When dealing with adaptive corruptions, the issue is now that by sending a share $\sigma_i = \mathsf{H}(m)^{sk_i}$ an honest party P_i *commits* itself to its secret key sk_i. Hence, the simulator Sim must output sk_i upon P_i being adaptively corrupted. This, however, is challenging: if Sim knew sk_i for all $i \in [n]$, then it would also know sk. On the other hand, if it *does not know* at least $n - t$ of the sk_i, then it might fail during simulation.

Libert et al.'s Approach. Libert et al. circumvent this problem as follows. Their scheme uses asymmetric pairing groups $(\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T)$ of order p. They define their secret keys as $sk_i = (f(i), g(i), h(i), j(i)) \in \mathbb{Z}_p^4$, where f, g, h, j are independent polynomials of degree t. A

signature share on m is then computed as $\sigma_i = (z_i, r_i) \in \mathbb{G}^2$, where

$$z_i = \mathsf{H}_1^{f(i)}(m) \cdot \mathsf{H}_2^{g(i)}(m), \quad r_i = \mathsf{H}_1^{h(i)}(m) \cdot \mathsf{H}_2^{j(i)}(m),$$

with $\mathsf{H}_1, \mathsf{H}_2$ being independent random oracles. Public key pk is correspondingly set as $(g_1, g_2) = \left(g_z^{f(0)} \cdot g_r^{g(0)}, g_z^{h(0)} \cdot g_r^{j(0)}\right) \in \hat{\mathbb{G}}^2$, where g_z and g_r are two random generators of the group $\hat{\mathbb{G}}$. Similar as for threshold BLS, one can compute a signature $\sigma = (z, r)$ from $t + 1$ shares by interpolation in the exponent and verify it by checking whether $e(z, g_z) \cdot e(r, g_r) \cdot e(\mathsf{H}_1(m), g_1) \cdot e(\mathsf{H}_2(m), g_2) = 1$. In this manner, Libert et al.'s signature is *computationally unique* under the so-called double-pairing assumption (see [LM18] for a proof). The latter implies that it should be hard to find $(z', r') \neq (z, r)$ which also satisfies the above equation for the same m. At the same time, this flexibility is what allows their reduction (to the symmetric external Diffie-Hellman assumption) to go through. Namely, as their signatures do not commit the signer to a secret key, they can efficiently simulate a secret key at the appropriate point in the simulation where a party becomes corrupted. Unfortunately, the technique of Libert et al. does not work in the context of the original threshold BLS signature. Hence, to deal with adaptive corruptions, a completely new approach is required.

3.2.2 Adaptive Security from Oracle-Aided Simulatability

We begin by describing our idea for the simplified case when a trusted dealer computes and distributes the keys according to Boldyreva's original description of the threshold BLS scheme. Let us briefly recall our desired security notion of *unforgeability under chosen message attacks* in the context of threshold signatures. In this game, the adversary first observes the public key shares pk_i of all parties P_i. (In case the keys are distributed via some DKG protocol, the adversary also observes its execution as part of the game). Next, it repeatedly gets access to a signing oracle, which takes in a pair (i, m) and returns a signature share σ_i that is valid under pk_i. The adversary can also adaptively corrupt any hitherto uncorrupted party P_i, upon which it learns P_i's secret key sk_i (and any other internal variables held by P_i at the point of corruption). The adversary is considered successful if it can produce a forgery on a message m^* for which it has observed a total of fewer than $t + 1$ shares from corruptions and signing queries.

Reducing from One-More Discrete Logarithm. As already explained, the critical difficulty is to simulate the values of sk_i upon an adaptive corruption. Our key idea is to aid the simulator by giving it t-time access to a discrete logarithm oracle $\mathsf{DL}_g()$ which, on input $h = g^x \in \mathbb{G}$, returns the discrete logarithm x of h to base g. To facilitate such an oracle in our simulation, we reduce from the *one-more discrete logarithm (OMDL) assumption of degree* $t + 1$. Recall that in the OMDL assumption of degree k, the adversary is given an instance $(g^{x_1}, \ldots, g^{x_k})$ and gets $(k - 1)$-time access to $\mathsf{DL}_g()$. It is considered successful if it can produce the values of $x_1, \ldots, x_k \in \mathbb{Z}_p$. Moreover, we rely on the algebraic group model (AGM) [FKL18] to obtain a suitable system of linear equations that allow to solve the OMDL instance. Both of these tools have recently been popular choices for proving involved cryptosystems [KLX22; FPS20; NRS21; TZ22].

Replacing the Trusted Dealer. We now turn our attention to the more realistic setting in which no trusted dealer is available for setting up keys. In this setting, existing DKG protocols are either only statically secure or can be used exclusively with signature schemes that do not commit

a party to her secret key sk_i whenever she uses it to issue a signature share σ_i. Fortunately, we can appropriately modify our ideas from above so as to handle adaptive corruptions in the unforgeability game even when it is extended with a DKG phase. In some more detail, we begin by proposing a new (and rather weak) security definition for DKG protocols that we refer to as *oracle-aided simulatability*. Informally, this definition asserts the existence of an efficient simulator Sim that can simulate an execution of the DKG protocol (with adaptive corruptions), given some number of queries to a discrete logarithm oracle $\mathsf{DL}_g()$. While this definition may seem somewhat artificial at first glance, we show that it is actually sufficient to prove unforgeability against chosen message attacks for the threshold BLS scheme with adaptive corruptions. For the proof, we show a reduction from the OMDL assumption of appropriate degree. The reduction runs Sim internally so as to provide a simulation of the DKG protocol as part of the broader simulation of the unforgeability experiment. To emulate the oracle $\mathsf{DL}_g()$ toward Sim, the reduction simply forwards any query Sim directs to $\mathsf{DL}_g()$ to its own discrete logarithm oracle.

We stress that oracle-aided simulatability is a security notion for DKG protocols and can be proven completely independently from the context of threshold BLS signatures. Thus, our definition adds a useful layer of modularity: it allows future DKG designers to build protocols that can be directly integrated with our (adaptive) security proofs. To motivate our new notion even further, we show that for several DKG protocols from the literature that do not satisfy full simulatability with adaptive corruptions, it is possible to show oracle-aided simulatability.

3.2.3 The Necessity of Strong Assumptions

One might wonder whether such strong assumptions are truly necessary for proving adaptive security of the threshold BLS signature. We answer this question in the affirmative. Concretely, we show that there is no algebraic, non-rewinding reduction from the OMDL assumption of degree 2 to the adaptive security of the threshold BLS scheme with corruption threshold t. Our impossibility result follows the common metareduction template [Cor02; KK18]: Assuming an efficient reduction R as above, we show that one can obtain an efficient solver M (the metareduction) for the OMDL problem of degree 2. As OMDL of degree 2 is assumed to be hard, it follows that such a reduction R can not exist. Our metareduction also applies to reductions which are not fully black-box. In particular, we rule out reductions which themselves may rely on the AGM. We combine this with a second impossibility result, which states that there is no algebraic, non-rewinding reduction from the q-discrete logarithm (q-DL) assumption to the adaptive security of the threshold BLS scheme with corruption threshold t. Recall that in the q-DL assumption, the adversary is given an instance $(g, g^z, \ldots, g^{z^q})$. It is considered successful if it can produce the value of $z \in \mathbb{Z}_p$. Now Bauer et al. [BFL20] show that in the AGM, any conceivable static assumption (and even some non-static assumptions) is implied by the q-DL assumption for a suitable degree $q \in \mathbb{N}$. Hence, our results show that even in the AGM, the OMDL assumption of some higher degree is both necessary and sufficient to prove security, unless one uses a rewinding reduction, and static assumptions also do not suffice to prove security. Rewinding, however, would have a devastating impact on the tightness of the reduction, and would require much larger parameters for concrete security than what is used in real-world implementations for BLS signatures [Chi22]. In contrast to this, our reduction in the AGM is tight and hence justifies parameters currently used in practice. To further justify

our reliance on the AGM, we also provide a metareduction along the lines of Coron [Cor02] to rule out a natural class of tight black-box reductions (we call such reductions *naive* and define them in Section 3.5.4) from the one-more discrete logarithm assumption of degree $t + 1$ in the plain random oracle model. We remark that Coron's original metareduction did not consider reductions from interactive assumptions such as OMDL. Unsurprisingly, giving a reduction R access to the oracle $\mathsf{DL}_g()$ complicates matters significantly, as we now have to simulate $\mathsf{DL}_g()$ to R as part of our metareductions.

3.2.4 More on Related Work

Threshold signatures were first conceived by Desmedt [Des88]. They have recently received significant attention [GGN16; BGG19; Lin17; LN18; GG18; Can+20; Dal+20; KG20; Ara+21; Abr+21b; CKM21; Kon+21; Can+21; Lin22], mainly in the context of blockchain systems and cryptocurrency wallets. Most of these works focus on the ECDSA and Schnorr threshold signatures, as these are the most widely used schemes in major cryptocurrencies. Note however, that these schemes do not have unique signatures and hence do not lend themselves to distributed coin tossing. A closely related (and also very active) line of research has also studied *multi-signatures* [BN06; BDN18; Dri+19; Nic+20; NRS21; CKM21; AB21]. These can be seen as a threshold signature scheme where the threshold t is always set to $n - 1$, i.e., signing always requires *all parties* to contribute. In contrast to threshold signatures, multi-signatures usually focus on obtaining compact n-out-of-n signatures using parties' native public keys for signing. Thus, no trusted dealer or DKG is necessary to run these protocols.

Distributed Key Generation. There are numerous DKG protocols when the underlying network is synchronous [Ped92; Can+99; JL00; Gen+07; Shr+21a]. Among these, only the protocols of Canetti et al. [Can+99] and Jarecki and Lysyanskaya [JL00] provide adaptive security. Both of these works rely on heavy assumptions and/or cryptographic tools such as non-committing encryption. All of these protocols (except that of Shrestha et al. [Shr+21a]) rely on public broadcast channels being available. On the other hand, the asynchronous setting has only recently been explored by works of Kokoris-Kogias et al. [KMS20], Abraham et al. [Abr+21a], and Das et al.[Das+22c]. Among these, only the work of Kokoris-Kogias et al. provides adaptive security, but is substantially less efficient than its statically secure alternatives. We also remark that the asynchronous DKG of Abraham et al. [Abr+21a] produces a group element as the shared secret key rather than a field element and hence can not be used for most conventional signature schemes. This drawback was resolved by Das et al. [Das+22c] without increasing the total communication cost of $O(\lambda n^3)$ bits.

VRFs and Distributed Coin Tossing. Distributed randomness generation is an integral component of many distributed protocols. This applies in particular to the asynchronous model, where most distributed protocols of interest are inherently randomized. Asynchronous coin tosses rely either on verifiable secret sharing [CR93; Cac+02] or threshold signatures [CKS00; Abr+19b; AMS19; Lu+20; Guo+20]. Synchronous protocols [Mic17; Abr+19b; Abr+19a] can rely on a simpler alternative of tossing coins via *verifiable random functions (VRF)* [MRV99]. On input a message m and a secret key sk, a VRF F produces a pseudorandom string $r = F(sk, m)$ along with a proof ϱ that can be used to verify correct generation of r. To agree on a single bit among n parties in the kth round, a party P_i computes $r_i = F(sk_i, k)$ along with a proof ϱ_i. It then samples $b_i \leftarrow \{0, 1\}$ uniformly and sends (b_i, r_i, ϱ_i) to everybody at the beginning of the

round. Parties wait to receive messages from other parties until the end of the round. All parties derive the coin as $b := b_j$ where $j = \min_i\{r_i\}$ and the minimum goes over all r_i for which ϱ_i correctly verified. If j belongs to an honest party, then all honest parties indeed agree on a random bit b. Moreover, since F produces pseudorandom outputs and there is a unique output per party and coin toss j, this happens with probability $p \geq \frac{1}{2}$ when the majority of the parties is honest, i.e., b can not be biased.[1] In an asynchronous network, however, this approach utterly fails, as there is no notion of a synchronous round. Note that in order to guarantee liveness of the protocol, every honest party can only wait for up to $2n/3 + 1$ messages from other parties. But in that case, the adversary simply delays the $n/3$ messages (b_i, r_i, ϱ_i) of honest parties with the lowest r_i values. The honest parties receive the remaining $n/3 + 1$ messages from honest parties and $n/3 - 1$ messages from the adversary. Now, with overwhelming probability in n, the smallest r_i always belong to a corrupt party and hence the coin is almost completely under the control of the adversary. Note that this issue does not occur for threshold signatures, as parties can all reconstruct the same coin after receiving only $t + 1$ messages from other parties. Some works such as [CKS20; Blu+20] have shown how to circumvent this issue by either relying on strong setup assumptions or assuming a non-standard version of the asynchronous model in which the adversary can not reorder messages of honest parties arbitrarily.

3.2.5 Outline of this Chapter

The rest of this chapter is structured as follows. In Section 3.3, we define the preliminaries that are only relevant for this chapter. In Section 3.4, we give (new) definitions for the primitives of interest, including non-interactive threshold signatures and distributed key generation. In Section 3.5, we provide our security proofs and the impossibility results. In Appendix 3.5.4, we present the relevant DKG protocols from the literature.

3.3 Preliminaries for this Chapter

In addition to the general preliminaries in Chapter 2, we introduce here preliminaries that are only relevant for this chapter.

3.3.1 General Notation

Let λ denote the security parameter. In this chapter, we assume that global parameters $par := (\mathbb{G}, \mathbb{G}_T, p, g, e)$ are fixed and known to all parties. Here, $\mathbb{G}$ is a cyclic group of prime order p generated by g and endowed with a symmetric pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. For concrete choices, we will assume $\lambda = 128$ and that $\mathbb{G}$ is instantiated with a 256-bit elliptic curve. We define the Vandermonde matrix $V(x_1, \ldots, x_k)$ for the $k \geq 1$ numbers $x_1, \ldots, x_k \in \mathbb{Z}_p$ as

$$V(x_1, \ldots, x_k) := \begin{pmatrix} 1 & x_1^1 & x_1^2 & \cdots & x_1^{k-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_k^1 & x_k^2 & \cdots & x_k^{k-1} \end{pmatrix},$$

which is known to be invertible if and only if the x_i are pairwise distinct.

[1] We remark that for most consensus protocols, it is sufficient to agree on the coin b with some constant probability. Threshold signatures let parties agree on coins with probably close to 1.

3.3.2 Assumptions and Model

We specify our assumptions and the model for this chapter. We also define a class of algorithms that will be relevant for our impossibility results.

Communication and Network Model. In this chapter, we assume a synchronous network as defined in Chapter 2. However, we note that we focus on synchronous protocols only in this work, since many of the most well-known DKG protocols are synchronous. Further, we stress that our methods are equally applicable to asynchronous DKG protocols with a similar structure.

Adversarial Model. We assume a Byzantine adversary that can corrupt up to $t < n/2$ parties. The adversary is *strongly adaptive*: When it corrupts a party adaptively, it can delete or substitute any undelivered messages that this party previously sent (while being honest). Further, the adversary is *rushing*: in any synchronous round, it can observe the messages of all honest parties and then decide what messages it wants to send to honest parties for that round.

Idealized Models and Computational Assumptions. We assume the random oracle model and the algebraic group model. Further, we will work with the one-more discrete logarithm assumption and the q-discrete logarithm assumption. These models and assumptions are defined in Chapter 2.

Algebraic Black-Box Reductions. An algorithm R is called a *black-box reduction* from problem $\mathbf{P}_2$ to problem $\mathbf{P}_1$ if for any algorithm A solving $\mathbf{P}_1$, algorithm R^{A} solves $\mathbf{P}_2$ with black-box/oracle access to A during its execution. If the algorithm A happens to be algebraic, then R (called an *algebraic black-box reduction*) additionally has access to the representation of the output elements of A (relative to all of the inputs A has received up to that point). We assume algebraic black-box reductions for our metareduction results. Such reductions have previously been introduced and studied [BFL20; KLX22].

3.4 Threshold Signatures

Threshold cryptography is a fundamental multiparty paradigm for enhancing the security and the availability of cryptographic schemes. It achieves this by dividing the secret key into n shares distributed across a network of n parties. In (t, n)-threshold cryptosystems, secret key operations require the cooperation of at least $t + 1$ out of n parties. In this way, the system remains secure against adversaries that corrupt up to t parties.

3.4.1 Distributed Key Generation

Distributed key generation (DKG) protocols are an essential component of threshold cryptosystems. The purpose of a DKG protocol is to distribute the shared keys of parties securely without relying on a trusted dealer. At the end of the protocol, the public key is output in the clear, whereas the secret key is kept as a virtual secret shared among all parties. The secret key is never explicitly computed, reconstructed or stored in any single location. This shared secret key can then be used later for threshold cryptosystems, such as threshold signatures or threshold encryption, without ever being explicitly reconstructed.

Definition 3.4.1 (Distributed Key Generation Protocol)**.** Let Π be a protocol executed among n parties $P_1, \ldots, P_n$, where P_i outputs a *secret key share* sk_i, a vector of *public key shares*

$(pk_1, \ldots, pk_n)$, a public key pk, and parties terminate upon generating output. We define the following security and correctness properties for Π:

- **Consistency:** Π is *t-consistent* if the following holds whenever at most t parties are corrupted: all honest parties output the same public key $y = g^x$ and the same vector of public key shares $(pk_1, \ldots, pk_n)$.
- **Correctness:** Π is *t-correct* if the following holds whenever at most t parties are corrupted: there exists a unique polynomial $f \in \mathbb{Z}_p[X]$ of degree t such that for all $i \in [n]$, $sk_i = f(i)$ and $pk_i = g^{sk_i}$. Moreover, it is $pk = g^{f(0)}$.
- **Unpredictability:** Π is *t-unpredictable* if for every algorithm A, the success probability in the following experiment is negligible:
  - *Offline Phase.* On input par, A returns a group element $y' \leftarrow \mathsf{A}(par)$.
  - *Online Phase.* Run protocol Π. A can make up to t adaptive corruption queries. Let $y = g^x \leftarrow \Pi$ be the public key output by Π.
  - *Output Determination.* A is considered successful if and only if $y' = y$.
- **Oracle-aided Algebraic Simulatability.** Π has $(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$*-oracle-aided algebraic simulatability* if for every algorithm A that runs in time at most T_{A} and corrupts at most t parties, there exists an algebraic simulator Sim that runs in time at most T_{Sim}, makes $k-1$ queries to oracle $\mathsf{DL}_g()$, and satisfies the following properties:
  - On input $\xi = (g^{z_1}, \ldots, g^{z_k}) \in \mathbb{G}^k$, Sim simulates the role of the honest parties in an execution of Π. At the end of the simulation, Sim outputs the public key $pk = g^x$. If corruptions are static, Sim gets a set of corrupted parties $C \subset \{1, \ldots, n\}$ of size at most t as an additional input.
  - On input $\xi = (g^{z_1}, \ldots, g^{z_k}) \in \mathbb{G}^k$ and for $i \in [k-1]$, let $g_i \in \mathbb{G}$ denote the ith query to $\mathsf{DL}_g()$. Let $(\hat{a}_i, a_{i,1}, \ldots, a_{i,k})$ denote the corresponding algebraic coefficients, i.e., $g_i = g^{\hat{a}_i} \cdot \prod_{j=1}^{k} (g^{z_j})^{a_{i,j}}$ and set $(\hat{a}, a_{0,1}, \ldots, a_{0,k})$ as the algebraic coefficients corresponding to pk. Then the following matrix over $\mathbb{Z}_p$ is invertible

$$L := \begin{pmatrix} a_{0,1} & a_{0,2} & \cdots & a_{0,k} \\ a_{1,1} & a_{1,2} & \cdots & a_{1,k} \\ \vdots & \vdots & & \vdots \\ a_{k-1,1} & a_{k-1,2} & \cdots & a_{k-1,k} \end{pmatrix}.$$

    Whenever Sim completes a simulation of an execution of Π, we call L the *simulatability matrix* of Sim (for this particular simulation).
  - Denote by $\mathsf{view}_{\mathsf{A},y,\Pi}$ the view of A in an execution of Π conditioned on all honest parties outputting $pk = y$. Similarly, denote by $\mathsf{view}_{\mathsf{A},\xi,y,\mathsf{Sim}}$ the view of A when interacting with Sim on input ξ, conditioned on Sim outputting $pk = y$. (For convenience, Sim's final output pk is omitted from $\mathsf{view}_{\mathsf{A},\xi,y,\mathsf{Sim}}$.) Here, we define $\mathsf{view}_{\mathsf{A},\xi,y,\mathsf{Sim}} := \perp$ in case $pk \neq y$ for all runs of Sim on input ξ. Then, for all y and all ξ, the views $\mathsf{view}_{\mathsf{A},\xi,y,\mathsf{Sim}}$ and $\mathsf{view}_{\mathsf{A},y,\Pi}$ are identically distributed.

> Let $k \in \mathbb{N}$ be the minimum k such that Π has $(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$-oracle-aided algebraic simulatability. Then we call k the $(t, T_{\mathsf{A}}, T_{\mathsf{Sim}})$-*simulatability factor* of Π.

We say that Π has $(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$-*oracle-aided algebraic security* if it is t-consistent, t-correct, and has $(t, T_{\mathsf{A}}, T_{\mathsf{Sim}})$-simulatability factor k.

For informal discussions, we sometimes abbreviate our notation by omitting T_{A} and T_{Sim} (we simply assume both A and Sim to be some PPT algorithms).

Discussion. We give a brief discussion of our security properties for distributed key generation protocols. Consistency and correctness notions are in line with what is achieved by most conventional DKG protocols. We note that we could easily weaken the requirement of Sim being fully algebraic to Sim behaving algebraic only with respect to the elements $pk, g_1, \ldots, g_{k-1}$. (In other words, only these elements come with an algebraic representation.) All our results remain true for this weaker notion of oracle-aided algebraic security. We stress that this weaker notion is a direct generalization of the usual notion of secrecy which requires a (not necessarily algebraic) simulator Sim that on input $y \in \mathbb{G}$ perfectly simulates an execution of Π in which y is determined as the public key pk. Setting $k = 1$ in our (relaxed) definition, we see that Sim queries $\mathsf{DL}_g()$ exactly $k - 1 = 0$ times, is trivially algebraic towards the output element pk (since the input is just pk itself) and the simulatability matrix is $L = (1)$, which is trivially invertible. Therefore, one can view the degree k of Π's oracle-aided algebraic security as a measure of how far away Π is from being fully secret.

As already observed in [LJY14], *full secrecy is not inherently required* in the context of threshold signing. This is not particularly surprising, as one might expect any reasonable signature scheme to remain unforgeable even when some information about the secret key is leaked. This observation is also the motivation behind our notion of oracle-aided algebraic simulatability. While this notion might look somewhat artificial at first glance, we will show that it is sufficient to provide unforgeability for the threshold BLS signature scheme against an adaptive adversary (in the AGM). Moreover, we prove in Section 3.5.3 that several well-known DKG protocols including $\mathsf{JF\text{-}DKG}$ (proposed by Pedersen [Ped92]) and $\mathsf{New\text{-}DKG}$ (proposed by Gennaro et al. [Gen+07]) have oracle-aided algebraic simulatability, also against adaptive adversaries. We stress that neither of these protocols satisfies secrecy against an adaptive adversary. In fact, it has been noted many times in the literature that $\mathsf{JF\text{-}DKG}$ does not even achieve full secrecy against a static adversary that corrupts a mere two parties! Finally, we remark that we require perfect simulatability only for convenience; it is straightforward to adjust our definition so as to allow for statistical or even computationally indistinguishable simulations.

3.4.2 Threshold Signature Scheme

In this section, we introduce the syntax and security notions for threshold signature schemes. We remark that we focus on *non-interactive* schemes for this work.

Definition 3.4.2 (Non-Interactive Threshold Signature)**.** A *non-interactive* (t, n)*-threshold signature scheme* is a tuple of efficient algorithms $\Sigma = (\mathsf{DKG}, \mathsf{SSign}, \mathsf{SVer}, \mathsf{Ver}, \mathsf{Comb})$ with the following properties:

- DKG: This is a distributed key generation protocol in the sense of Definition 3.4.1.

- SSign: The *share signing algorithm* is a possibly randomized algorithm that takes as input a message m and a secret key share sk_i. It outputs a signature share σ_i.
- SVer: The *signature share verification algorithm* is a deterministic algorithm that takes as input a message m, a public key share pk_i, and a signature share σ_i. It outputs 1 (accept) or 0 (reject).
- Comb: The *signature share combining algorithm* is a deterministic algorithm that takes as input the public key pk, a vector of public key shares $(pk_1, \ldots, pk_n)$, a message m, and a set $\mathcal{S}$ of $t+1$ signature shares (σ_i, i) (with corresponding indices). It outputs either a signature σ or $\perp$.
- Ver: The *signature verification algorithm* is a deterministic algorithm that takes as input a public key pk, a message m, and a signature σ. It outputs 1 (accept) or 0 (reject).

Let $\mathsf{H} : \{0,1\}^* \to \mathbb{G}$ be a cryptographic hash function (modeled as a random oracle). We define the security of a non-interactive threshold signature scheme in the adaptive corruption setting as follows.

Definition 3.4.3 (Unforgeability Under Chosen Message Attack)**.** Let $\Sigma = (\mathsf{DKG}, \mathsf{SSign}, \mathsf{SVer}, \mathsf{Ver}, \mathsf{Comb})$ be a non-interactive (t, n)-threshold signature scheme. For an algorithm A, define experiment $\mathbf{UF\text{-}CMA}^{\mathsf{A}}_{\Sigma,t}$ as follows:

- **Setup Phase.** Initialize sets $\mathcal{H} := \{1, \ldots, n\}$, $C := \emptyset$. Run A on input *par*.
- **Corruption Queries.** At any point of the experiment, A may corrupt a party P_i by submitting an index i. In this case, return the internal state of P_i and set $\mathcal{H} = \mathcal{H} \setminus \{i\}$, $C = C \cup \{i\}$. Henceforth, A has full control over P_i.
- **Distributed Key Generation.** Initiate an execution of DKG among parties $P_1, \ldots, P_n$. Denote by $(sk_1, \ldots, sk_n)$ and $(pk, pk_1, \ldots, pk_n)$ the secret and public key shares determined by DKG, respectively. In particular, the shares $\{sk_i\}_{i \in C}$ are also known to A.
- **Online Phase.** Here, A gets additional access to oracles that answer queries of the following types:
  - *Signing Queries.* When A submits a pair (i, m) for party $i \in \mathcal{H}$, return $\sigma \leftarrow \mathsf{SSign}(sk_i, m)$.
  - *Random Oracle Queries.* When A submits a query m, check if $H[m] = \perp$ and if so, set $H[m] \leftarrow \mathbb{G}$. Return $H[m]$.
- **Output Determination.** When A outputs a message m^* and a signature σ^*, let $\mathcal{S} \subset \{1, \ldots, n\}$ denote the subset of parties for which A made a signing query of the form (i, m^*). Output 1 if $|C \cup \mathcal{S}| \leq t$ and $\mathsf{Ver}(pk, m^*, \sigma^*) = 1$. Otherwise, output 0.

We say that Σ is $(\varepsilon, T, q_h, q_s)$-*unforgeable under chosen message attacks (UF-CMA)* if for all algorithms A running in time at most T, making at most q_h random oracle queries, and making at most q_s signing queries, $\Pr[\mathbf{UF\text{-}CMA}^{\mathsf{A}}_{\Sigma,t} = 1] \leq \varepsilon$. Conversely, we say that A $(\varepsilon, T, q_h, q_s)$-breaks unforgeability of Σ under chosen message attacks if it runs in time at most T, makes at most q_h to the random oracle, makes at most q_s queries to the signing oracle, and $\Pr[\mathbf{UF\text{-}CMA}^{\mathsf{A}}_{\Sigma,t} = 1] > \varepsilon$

3.4.3 Threshold BLS Signature Scheme Th-BLS$_{\mathsf{DKG}}$

In this section, we recall Boldyreva's BLS-based threshold signature scheme. We write Th-BLS$_{\mathsf{DKG}}$ to denote the scheme when setup is done using the distributed key generation algorithm DKG.

Definition 3.4.4 (Threshold BLS Signature Scheme [Bol03].)**.** Let DKG be a distributed key generation protocol. The algorithms of the (t, n)-threshold signature scheme Th-BLS$_{\mathsf{DKG}}$ = (DKG, SSign$_{\mathsf{BLS}}$, SVer$_{\mathsf{BLS}}$, Comb$_{\mathsf{BLS}}$, Ver$_{\mathsf{BLS}}$) are defined as follows:

- SSign$_{\mathsf{BLS}}$: On input a secret key share $sk_i \in \mathbb{Z}_p$ and a message $m \in \{0,1\}^*$ return the signature share $\sigma_i := \mathsf{H}(m)^{sk_i} \in \mathbb{G}$.
- SVer$_{\mathsf{BLS}}$: On input a public key share $pk_i \in \mathbb{G}$, a signature share σ_i, and a message m, return 1 if $e(g, \sigma_i) = e(pk_i, \mathsf{H}(m))$ and 0 otherwise.
- Comb$_{\mathsf{BLS}}$: On input a vector of public key shares $(pk_1, \ldots, pk_n)$, a set $\mathcal{S}$ of $t+1$ signature shares (and corresponding indices) (σ_i, i), and a message m, run SVer$_{\mathsf{BLS}}(\sigma_i, pk_i)$ for all $i \in \mathcal{S}_0 := \{i \in [n] \mid (\sigma_i, i) \in \mathcal{S}\}$. If any of these calls returns 0, return $\perp$. Otherwise, return $\sigma = \prod_{i \in \mathcal{S}_0} \sigma_i^{L_i}$, where $L_i = \prod_{j \in \mathcal{S}_0 \setminus \{i\}} j/(j-i)$ denotes the i-th Lagrange coefficient for the set $\mathcal{S}_0$.
- Ver$_{\mathsf{BLS}}$: On input a public key pk, a signature σ, and a message m, return 1 if $e(g, \sigma) = e(pk, \mathsf{H}(m))$ and 0 otherwise.

3.5 Security Analysis of Th-BLS$_{\mathsf{DKG}}$

In this section, we analyze the security of Th-BLS$_{\mathsf{DKG}}$ in the case where DKG has (t, k)-oracle-aided algebraic security. First, we show a tight reduction to the OMDL assumption of degree k. Second, we show that there is no algebraic black-box reduction from the OMDL assumption of degree 2 and from the q-DL assumption to the security of Th-BLS$_{\mathsf{DKG}}$ with corruption threshold t. On the other hand, we show that the trusted dealer key generation algorithm TD-DKG (see Section 3.5.3) has (t, k)-oracle-aided algebraic security with $k = t + 1$. In particular, there is a reduction from the $(t+1)$-OMDL assumption to the security of Th-BLS$_{\mathsf{TD\text{-}DKG}}$ (with corruption threshold t). Apart from that, we show oracle-aided algebraic security for JF-DKG and New-DKG. Finally, we show that any algebraic black-box reduction of a certain kind (which will be defined in Section 3.5.4) from the $(t+1)$-OMDL assumption to the security of Th-BLS$_{\mathsf{DKG}}$ loses a factor of q_s and can not possibly be improved, in the plain ROM.

3.5.1 Security proof of Th-BLS$_{\mathsf{DKG}}$

For DKG with (t, k)-oracle-aided algebraic security, our first theorem asserts the security of Th-BLS$_{\mathsf{DKG}}$ (in the AGM+ROM) under the assumption that the k-OMDL problem is hard. Our proof follows the tight security proof for the (standard) BLS scheme [FKL18; Los19]. The key idea is to embed an OMDL challenge ξ in either the secret key shares or inside the random oracle queries, a choice that remains hidden from the adversary. In the former case, we simulate by using the oracle-aided algebraic simulator coming from DKG. In the latter case, we solve ξ directly from the algebraic equation that comes from the forgery (with its representation).

Theorem 3.5.1. *If k-OMDL is (ε, T)-hard in the AGM and* DKG *has $(t, k, T', T_{\mathsf{Sim}})$-oracle-aided algebraic security, then* Th-BLS$_{\mathsf{DKG}}$ *is $(\varepsilon', T', q_h, q_s)$-secure in the AGM+ROM, where*

$$\varepsilon \geq \frac{\varepsilon'}{4} - \frac{q_h^2}{4p}, \quad T \leq T' + T_{\mathsf{Sim}} + 3q_h + q_s.$$

Proof. We prove the theorem via a sequence of games. Let A be an algebraic algorithm that $(\varepsilon', T', q_h, q_s)$-breaks unforgeability of Th-BLS$_{\mathsf{DKG}}$ under chosen message attacks. Without loss of generality we assume that A always queries the random oracle H before any signing query for the same message m. (Note that the challenger can always enforce this by making appropriate random oracle queries for itself.) Similarly, we may assume that queries to the random oracle are distinct and that A queries the random oracle H on m^* (the forgery message) before producing its forgery.

Game $\mathbf{G}_0$: This is the real game. The challenger runs DKG on behalf of the honest parties. Whenever A decides to corrupt a party P_i, the challenger faithfully returns the internal state of that party and sets $C = C \cup \{i\}$, $\mathcal{H} = \mathcal{H} \setminus \{i\}$. In addition, A gets full control over P_i. For all $i \in [n]$, let $y_i \in \mathbb{G}$ denote the public key share assigned to P_i by DKG and let x_i denote P_i's secret key share. Moreover, let $x \in \mathbb{Z}_p$ and $y = g^x$ denote the secret key and public key, respectively. Random oracle queries are answered by sampling $r_i \leftarrow \mathbb{Z}_p^*$ and returning $h_i = g^{r_i} \in \mathbb{G}$. Partial signing queries (j, m) are answered by returning $H[m]^{x_j}$. At the end of the game, A outputs a message-signature pair (m^*, σ^*).

Game $\mathbf{G}_1$: This game is identical to the game before, except that the game aborts and the adversary loses when there is a collision $H[m_1] = H[m_2]$ among distinct random oracle queries $m_1 \neq m_2$ from A. By a standard argument, $\Pr[\mathbf{G}_0^{\mathsf{A}} = 1] \leq \Pr[\mathbf{G}_1^{\mathsf{A}} = 1] + q_h^2/p$.

Let $\mathcal{V} = C \cup \mathcal{S}$, where $\mathcal{S} \subset \{1, \ldots, n\}$ is the subset of parties for which A made a signing query of the form (i, m^*). As A is an algebraic adversary, at the end of $\mathbf{G}_1$ it returns a forgery σ^* on a message m^* together with a representation

$$a = (\hat{a}, a', \check{a}_1, \ldots, \check{a}_n, \bar{a}_1, \ldots, \bar{a}_{q_h}, \tilde{a}_{1,1}, \ldots, \tilde{a}_{1,n}, \ldots, \tilde{a}_{q_s,1}, \ldots, \tilde{a}_{q_s,n})$$

of elements in $\mathbb{Z}_p$ such that

$$\sigma^* = H[m^*] = g^{\hat{a}} \cdot y^{a'} \cdot y_1^{\check{a}_1} \cdot \ldots \cdot y_n^{\check{a}_n} \cdot \prod_{i=1}^{q_h} h_i^{\bar{a}_i} \cdot \prod_{i=1}^{q_s} \sigma_{i,1}^{\tilde{a}_{i,1}} \cdot \ldots \cdot \sigma_{i,n}^{\tilde{a}_{i,n}}.$$

Here, the representation is split (from left to right) into powers of the generator g, the public key $y = g^x$, the public key shares $y_j = g^{x_j}$, $j \in [n]$, all of the answers to hash queries h_i, $i \in [q_h]$, and the partial signatures $\sigma_{i,j}$, $i \in [q_s]$ and $j \in [n]$, returned by the random oracle and the partial signing oracle, respectively. Note that we have tacitly combined the other elements that are publicly communicated during the key generation phase into the term $g^{\hat{a}}$. We will clarify later why this is possible. In the following, let m_i denote the ith query to H and let $i^* \in [q_h]$ denote the index corresponding to the forgery message m^*. Recall that we write r^* and r_i for $i \in [q_h]$ to denote the values such that $H[m^*] = g^{r^*}$ and $H[m_i] = g^{r_i}$. Having said that, the

above equation is equivalent to

$$\begin{aligned} r^*x &= \hat{a} + xa' + \sum_{i=1}^{n} \breve{a}_i x_i + \sum_{i=1}^{q_h} r_i \bar{a}_i + \sum_{i=1}^{q_s} r_i(\tilde{a}_{i,1}x_1 + \ldots + \tilde{a}_{i,n}x_n) \\ &= \hat{a} + xa' + \sum_{i=1}^{n} \breve{a}_i x_i + \sum_{i\in Q_h} r_i \bar{a}_i + \sum_{i\in Q_s} r_i(\tilde{a}_{i,1}x_1 + \ldots + \tilde{a}_{i,n}x_n) \\ &\quad + r^*\bar{a}^* + r^*(\tilde{a}^*_1 x_1 + \ldots + \tilde{a}^*_n x_n), \end{aligned} \tag{♠}$$

where in the last equation we split the answers to the (hash and partial signing) queries for m^* from those of the other messages, with appropriate sets Q_h and Q_s (to be precise, $Q_h = [q_h] \setminus \{i^*\}$ and $Q_s = [q_s] \setminus \{i^*\}$) and the notation $\tilde{a}^*_j = \tilde{a}^*_{i^*,j}$ for all $j \in [n]$. Since A wins $\mathbf{G}_1$, we remark that neither $\sum_{i\in Q_h} r_i\bar{a}_i$ nor $\sum_{i\in Q_s} r_i(\tilde{a}_{i,1}x_1 + \ldots + \tilde{a}_{i,n}x_n)$ may include the terms $r^*\bar{a}^*$ or $r^*(\tilde{a}^*_1x_1 + \ldots + \tilde{a}^*_n x_n)$, respectively.[2] We define event E as the event that $x \neq \bar{a}^* + \tilde{a}^*_1 x_1 + \ldots + \tilde{a}^*_n x_n$.

We have the following lemma.

Lemma 3.5.2. *Let $\mathbf{G}_1$ and E be as defined above. Then there exist (algebraic) algorithms A_1 and A_2 playing in game k-OMDL that run in time at most T such that:*

$$\begin{aligned} \Pr[k\text{-}\mathsf{OMDL}^{\mathsf{A}_1} = 1] &= \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge \neg E], \\ \Pr[k\text{-}\mathsf{OMDL}^{\mathsf{A}_2} = 1] &\geq \left(1 - \frac{1}{p}\right) \cdot \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge E]. \end{aligned}$$

Moreover, $T \leq T' + T_{\mathsf{Sim}} + 3q_h + q_s$.

Proof. Let $\xi = (\xi_1, \ldots, \xi_k) \in \mathbb{G}^k$ with $\xi_i = g^{z_i}$, $i \in [k]$, be the OMDL instance. A_1 and A_2 both have access to a discrete logarithm oracle $\mathsf{DL}_g()$ which they can query at most $k-1$ times. Both simulate $\mathbf{G}_1$, as we now describe.

Algorithm $\mathsf{A}_1(\xi, par)$: Algorithm A_1 works as follows. Since DKG has $(t, k, T', T_{\mathsf{Sim}})$-oracle-aided algebraic security, there exists an algebraic simulator Sim that runs in time at most T_{Sim} with $(k-1)$-time access to a discrete logarithm oracle. Sim takes as input the k-OMDL instance $\xi = (\xi_1, \ldots, \xi_k) \in \mathbb{G}^k$ and perfectly simulates an execution of DKG, where at most t parties can be corrupted. A_1 simulates the key generation phase by running Sim on input ξ. Whenever Sim queries its discrete logarithm oracle, A_1 forwards this query to its own oracle $\mathsf{DL}_g()$. Random oracle queries are answered by sampling $r_i \leftarrow \mathbb{Z}_p^*$ and returning $h_i = g^{r_i} \in \mathbb{G}$. A_1 aborts when there is a collision $H[m_1] = H[m_2]$ among different random oracle queries $m_1 \neq m_2$ from A. Signing queries (j, m_i) are answered by returning $h_i^{x_j}$ via the identity

$$H[m_i]^{x_j} = (g^{r_i})^{x_j} = (g^{x_j})^{r_i} = y_j^{r_i}.$$

Corruption queries are handled by Sim, which allows A_1 to return the internal state of up to t parties correctly. It is not hard to see that A_1's simulation of $\mathbf{G}_1$ is perfect and that A_1 can correctly answer Sim's (at most) $k-1$ oracle queries.

[2] Here, we consider r^* and r_i, $i \in [n]$, as formal variables over $\mathbb{Z}_p$ rather than the concrete value that they may take. Note that it is indeed possible that for some i, $r_i\bar{a}_i = r^*\bar{a}^*$ or $r^*(\tilde{a}^*_1x_1 + \ldots + \tilde{a}^*_n x_n) = r_i(\tilde{a}_{i,1}x_1 + \ldots + \tilde{a}_{i,n}x_n)$ when considering concrete values in $\mathbb{Z}_p$ for r^*, r_i.

Suppose that A wins $\mathbf{G}_1$ and that event $\neg E$ happens, i.e., $x = \bar{a}^* + \tilde{a}_1^* x_1 + \ldots + \tilde{a}_n^* x_n$. We note that the partial signing queries of the form (i, m^*) do not reveal any more information than if the challenger were simply handing over the corresponding private key share x_i. We thus treat these partial signing queries for m^* as corruption queries. We may also assume w.l.o.g. that $|C| = t$. Otherwise, A_1 simulates, for itself, $t - |C|$ corruption queries for random parties from the set of uncorrupted parties $\mathcal{H}$ after receiving the forgery, as if these were regular queries from A. (It does so by querying Sim.) Since A_1 knows all values $\{x_i\}_{i \in \mathcal{V}}$, it can compute the secret key x efficiently via the identity $x = \bar{a}^* + \tilde{a}_1^* x_1 + \ldots + \tilde{a}_n^* x_n$. Let $g^{a_1}, \ldots, g^{a_{k-1}}$ denote the discrete logarithm oracle queries made by Sim. As Sim is algebraic, it also outputs a representation $(\hat{a}_i, a_{i,1}, \ldots, a_{i,k})$ for each of these queries, $i \in [k-1]$. Similarly, let $y = g^x$ be the public key output by Sim together with its representation $(\hat{a}_0, a_{0,1}, \ldots, a_{0,k})$. Then A_1 obtains the following system of linear equations in the variables $z_1, \ldots, z_k$:

$$\begin{aligned} x &= \hat{a}_0 + a_{0,1} z_1 + \ldots + a_{0,k} z_k \\ a_1 &= \hat{a}_1 + a_{1,1} z_1 + \ldots + a_{1,k} z_k \\ &\vdots \\ a_{k-1} &= \hat{a}_{k-1} + a_{k-1,1} z_1 + \ldots + a_{k-1,k} z_k, \end{aligned}$$

which in matrix form is equivalent to

$$\begin{pmatrix} x - \hat{a}_0 \\ a_1 - \hat{a}_1 \\ \vdots \\ a_{k-1} - \hat{a}_{k-1} \end{pmatrix} = \begin{pmatrix} a_{0,1} & a_{0,2} & \cdots & a_{0,k} \\ a_{1,1} & a_{1,2} & \cdots & a_{1,k} \\ \vdots & \vdots & & \vdots \\ a_{k-1,1} & a_{k-1,2} & \cdots & a_{k-1,k} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_k \end{pmatrix}.$$

By definition, the simulatability matrix of Sim is invertible and hence A_1 can efficiently compute $(z_1, \ldots, z_k)$ and solve the OMDL instance. Overall, we obtain

$$\Pr[k\text{-}\mathsf{OMDL}^{\mathsf{A}_1} = 1] = \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge \neg E].$$

The bound on the running time of A_1 (number of group operations and exponentiations) comes from running the simulator Sim once, one exponentiation for each random oracle query and one exponentiation for each signing query.

Algorithm $\mathsf{A}_2(\xi, par)$: Algorithm A_2 works as follows. It runs DKG correctly on behalf of the honest parties. In particular, it knows all the secret key shares x_j. Whenever the adversary A decides to corrupt a party, A_2 faithfully reveals the internal state of that party. Random oracle queries are answered by sampling $b_i, d_i \leftarrow \mathbb{Z}_p^*$ and returning $h_i = g^{r_i} = \xi_1^{b_i} g^{d_i}$, which implicitly sets $r_i = z_1 b_i + d_i$. A_2 aborts in case it detects a collision among answers in the list H. Partial signing queries (j, m_i) are answered by returning $h_i^{x_j}$. Again, it is not hard to see that A_2's simulation of $\mathbf{G}_1$ is perfect.

In case A_2 does not abort, let $\tilde{A}_i := \tilde{a}_{i,1} x_1 + \ldots + \tilde{a}_{i,n} x_n$ for all i, where we write $\tilde{A}^*$ for $\tilde{A}_{i^*}$. Suppose that A wins $\mathbf{G}_1$ and that event E happens, i.e., $x \neq \bar{a}^* + \tilde{A}^*$. With our notation, equation

(♠) is equivalent to

$$z_1 b^* x + d^* x = \hat{a} + xa' + \sum_{i=1}^{n} \breve{a}_i x_i + \sum_{i \in Q_h} d_i \bar{a}_i + \sum_{i \in Q_s} d_i \tilde{A}_i + d^* \bar{a} + d^* \tilde{A}^* + z_1 \left(\sum_{i \in Q_h} b_i \bar{a}_i + \sum_{i \in Q_s} b_i \tilde{A}_i + b^* \bar{a}^* + b^* \tilde{A}^* \right).$$

With the further notations

$$B := b^* x - \left(\sum_{i \in Q_h} b_i \bar{a}_i + \sum_{i \in Q_s} b_i \tilde{A}_i + b^* \bar{a}^* + b^* \tilde{A}^* \right),$$

$$D := \hat{a} + xa' + \sum_{i=1}^{n} \breve{a}_i x_i + \sum_{i \in Q_h} d_i \bar{a}_i + \sum_{i \in Q_s} d_i \tilde{A}_i + d^* \bar{a} + d^* \tilde{A}^* - d^* x,$$

this reduces to $Bz_1 = D$. Recall that we have tacitly combined the other group elements that were publicly communicated during the key generation phase into the term $g^{\hat{a}}$. This is possible because A_2 faithfully runs DKG on behalf of the honest parties and therefore has knowledge of the exponents of those elements relative to the base g and can combine them into the value $\hat{a}$. Let us now consider the case where $B = 0$. With the definition of event E, we get

$$\begin{aligned} & 0 = b^* x - \left(\sum_{i \in Q_h} b_i \bar{a}_i + \sum_{i \in Q_s} b_i \tilde{A}_i + b^* \bar{a}^* + b^* \tilde{A}^* \right) \\ \iff \quad & b^* (x - \bar{a}^* - \tilde{A}^*) = \sum_{i \in Q_h} b_i \bar{a}_i + \sum_{i \in Q_s} b_i \tilde{A}_i \\ \iff \quad & b^* = \left(\sum_{i \in Q_h} b_i \bar{a}_i + \sum_{i \in Q_s} b_i \tilde{A}_i \right) \cdot \left(x - \bar{a}^* - \tilde{A}^* \right)^{-1}. \end{aligned}$$

As already noted, $\sum_{i \in Q_h} r_i \bar{a}_i$ and $\sum_{i \in Q_s} r_i \tilde{A}_i$ do not include the terms $r^* \bar{a}^*$ and $r^* \tilde{A}^*$, respectively. Hence, $\sum_{i \in Q_h} b_i \bar{a}_i$ and $\sum_{i \in Q_s} b_i \tilde{A}_i$ do not include $b^* \bar{a}^*$ and $b^* \tilde{A}^*$, respectively.[3] As defined by the identity $H[m^*] = g^{z_1 b^* + d^*}$, b^* remains information-theoretically hidden from A. This implies that the right-hand side of the equation is statistically independent of the uniform value b^*. Therefore, with probability $1 - 1/p$ we have $B \neq 0$, and thus A_2 can compute z_1 efficiently as $z_1 := B^{-1} D$. Overall, we obtain

$$\Pr[k\text{-}\mathsf{OMDL}^{\mathsf{A}_2} = 1] \geq \left(1 - \frac{1}{p}\right) \cdot \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge E].$$

The bound on the running time of A_2 comes from running the simulator Sim once, three operations for each random oracle query and one operation for each signing query. □

Consider algorithm B playing in k-OMDL as follows: B samples $i^* \leftarrow [2]$ and then internally emulates A_{i^*}. Clearly, B is an algebraic algorithm running in time at most T (the running time of

[3] Here, we consider b^* as a formal variable over $\mathbb{Z}_p$ rather than its concrete value.

A_1 and A_2). An application of the law of total probability yields for $p \geq 2$,

$$\begin{aligned}
\Pr[k\text{-OMDL}^{\mathsf{B}} = 1] &= \sum_{i=1}^{2} \Pr[k\text{-OMDL}^{\mathsf{B}} = 1 \mid i^* = 1] \cdot \Pr[i^* = 1] \\
&= \frac{1}{2} \sum_{i=1}^{2} \Pr[k\text{-OMDL}^{\mathsf{A}_i} = 1] \\
&\geq \frac{1}{2}\left(1 - \frac{1}{p}\right)\left(\Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge E] + \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge \neg E]\right) \\
&= \frac{1}{2}\left(1 - \frac{1}{p}\right)\Pr[\mathbf{G}_1^{\mathsf{A}} = 1] \geq \frac{1}{4}\Pr[\mathbf{G}_1^{\mathsf{A}} = 1] \\
&\geq \frac{1}{4} \cdot \Pr[\mathbf{G}_0^{\mathsf{A}} = 1] - q_h^2/p.
\end{aligned}$$

□

Remark 3.5.1. This result, however, does not rule out a tight reduction from the OMDL assumption of some lower degree $< k$ to the security of Th-BLS$_{\mathsf{DKG}}$ (for DKG with (t, k)-oracle-aided algebraic security). Indeed, using Th-BLS$_{\mathsf{TD\text{-}DKG}}$ as an example (see Section 3.5.3 for a definition), we find a tight reduction from the OMDL assumption of degree $t - r$ for any constant $r \geq 0$ which is independent of t (i.e., $r \in O(1)$) to the security of this scheme. Note that in Section 3.5.3, Theorem 3.5.7 we show that TD-DKG has (t, k)-oracle-aided algebraic security with $k = t + 1$. The reason for the existence of a tight reduction from OMDL of some lower degree $< k$ is based on the possibility that a potential simulator for TD-DKG might fail during the simulation. In our definition of (t, k)-oracle-aided algebraic security, we assume a perfect (oracle-aided) simulator that never fails. However, there does exist a simulator for the underlying DKG that fails during the simulation with some probability > 0, and such a "deficient" simulator may just be used by the reduction. This does not only apply to TD-DKG, but possibly also for any other DKG protocol that is employed in the threshold BLS scheme. Indeed, after our following explanation it is not hard to see that this also applies to JF-DKG and New-DKG.

In our proof of the (t, k)-oracle-aided algebraic security with $k = t + 1$ of TD-DKG (see proof for Theorem 3.5.7), on input $t + 1$ elements $\xi_0, \ldots, \xi_t \in \mathbb{G}$, the simulator Sim chooses the polynomial $f = \sum_{i=0}^{t} a_i X^i \in \mathbb{Z}_p[X]$ of degree t that determines the secret key shares by embedding ξ_i into the ith coefficient of f for all $i \in [0, t]$, that is $g^{a_i} = \xi_i$; corruption queries are answered with the oracle $\mathsf{DL}_g()$. This simulation is perfect and in line with our security notion of oracle-aided algebraic simulatability. We now describe a "deficient" simulator Sim that works for OMDL of some lower degree $< t + 1$ (that might fail with some probability > 0). Take the OMDL assumption of degree $t - r$ with $r \geq 0$ and let $\xi = (\xi_{r+1}, \ldots, \xi_t)$ with $\xi_i = g^{z_i}$ for all $i \in [r + 1, t]$ (where $z_i \in \mathbb{Z}_p$) be the challenge instance to be solved with $(t - r - 1)$-times access to an oracle $\mathsf{DL}_g()$. Firstly, Sim picks a random $(r + 1)$-element subset $\tilde{S} \subset [n]$ of $1, \ldots, n$ and $r + 1$ numbers $s_1, \ldots, s_{r+1} \in \mathbb{Z}_p$ uniformly at random. For convenience we may assume $\tilde{S} = \{1, \ldots, r + 1\}$. Secondly, Sim chooses the polynomial $f = \sum_{i=0}^{t} a_i X^i \in \mathbb{Z}_p[X]$ of degree t that determines the secret key shares such that (i) $g^{a_i} = \xi_i$ for all $i \in [r + 1, t]$ (which implicitly sets $a_i = z_i$), and (ii) $f(i) = s_i$ for all $i \in [r + 1]$. This choice determines f completely and its coefficients solely depend on the z_i. This is the case because the equations $f(i) = s_i$ for

$i \in [r+1]$ give the following system of linear equations in matrix form

$$\begin{pmatrix} s_1 - \sum_{i=r+1}^{t} a_i 1^i \\ s_2 - \sum_{i=r+1}^{t} a_i 2^i \\ \vdots \\ s_{r+1} - \sum_{i=r+1}^{t} a_i (r+1)^i \end{pmatrix} = \begin{pmatrix} 1 & 1^1 & \cdots & 1^r \\ 1 & 2^1 & \cdots & 2^r \\ \vdots & \vdots & & \vdots \\ 1 & (r+1)^1 & \cdots & (r+1)^r \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_r \end{pmatrix}$$

and the Vandermonde matrix on the right-hand side is invertible.[4] As a result, the polynomial f is completely described (even if implicitly). Now, Sim can work properly during a simulation (i.e., it does not fail) if and only if the adversary chooses a corruption set $S \subset [n]$ that contains $\tilde{S}$ as a subset. Otherwise, Sim would have to return at least $t - r$ distinct points on f which it does not know a priori, so that it would ultimately know $t - r + r + 1 = t + 1$ distinct points on f, which conversely means that Sim would have to solve the $(t-r)$-OMDL problem. On the other hand, the probability that $\tilde{S} \subset S$ is roughly $1/2^{r+1}$, and thus the simulator Sim might fail with probability $1/2^{r+1}$. In the other cases, Sim works perfectly and is able to return the internal states of all the corrupted parties properly. Finally, the security reduction simply uses such a "deficient" simulator as described above and fails with the additional probability $1/2^{r+1}$. The rest of the reduction, however, proceeds exactly as in the above proof for $(t+1)$-OMDL. Thus, for $(t-r)$-OMDL, the reduction has an additional security loss of $1/2^{r+1}$ and thus remains tight as long as $r \geq 0$ is independent of t, i.e., $r \in O(1)$. For instance, already the case $r = \log(t) - 1$ gives a (non-tight) multiplicative security loss of $1/t$.

3.5.2 No reduction from 2-OMDL and q-DLOG

Our next theorem asserts that the security of Th-BLS$_{\mathsf{DKG}}$ (for DKG with (t,k)-oracle-aided algebraic security) can not be derived from the OMDL assumption of degree 2, even in the AGM+ROM. Furthermore, we show that such a reduction can not be derived from the q-DLOG assumption for any q. In particular, the security of Th-BLS$_{\mathsf{DKG}}$ has to rely on some stronger mathematical assumption such as OMDL of some higher degree. Our proofs for these impossibility results follow Coron's metareduction technique [Cor02].

The idea behind the proof of the first theorem is the following. Upon receiving an OMDL challenge ξ of degree 2, the metareduction M runs the reduction R providing it with (par, ξ). The discrete logarithm oracle for R is simulated by M's own oracle $\mathsf{DL}_g()$. At the end of the key generation phase, R outputs the public key shares and the public key with their respective algebraic coefficients. The coefficients of the public key shares form an $(n \times 2)$-matrix Q over $\mathbb{Z}_p$ (disregarding the coefficients corresponding to g). By a mathematical result, we find that with overwhelming probability (to be precise, with probability at least $1 - 2^{-t+1}$) a set of randomly chosen t row vectors of the matrix Q spans the row space of Q. This eventually allows M to compute all the secret key shares (and hence the secret key x) after corrupting some t parties. With the knowledge of x, it can forge on any message and perfectly simulate an algebraic adversary F for the reduction. We point out that in this step, it is crucial for the adversary to be

[4] Note that Sim could likewise embed the instance elements $\xi_{r+1}, \ldots, \xi_t$ into some different (arbitrary) $t - r$ coefficients of the polynomial f, and not necessarily into the last $t - r$ ones. Eventually, the matrix on the right-hand side of the above equation would be a generalized Vandermonde matrix, which is also known to be invertible for pairwise distinct, positive numbers (which is the case here, since $\tilde{S} = \{1, \ldots, r+1\}$). Thus, this would also result in a complete determination of f.

allowed to corrupt parties adaptively, i.e., even after the termination of the DKG protocol. If the adversary were a static one, R would get the set of corrupted parties as an input and could take the algebraic coefficients of these parties in such a way that the corresponding vectors do not span the row space of Q. This results in M not being able to compute the secret key and the proof would fail.

We note that the same proof strategy (for the metareduction) is also applicable to the case where the underlying hardness assumption is OMDL of some higher degree $r \leq t$ instead of degree 2. The problem that arises there, however, is that the probability that a set of randomly chosen t row vectors of the matrix Q (which is now an $(n \times r)$-matrix over $\mathbb{Z}_p$) spans the whole row space of Q is extremely difficult to determine (especially, in order to tell something about the existence of a tight reduction). We will dive more into this later.

Theorem 3.5.3. *Let* DKG *have* $(t, k, T_{\mathsf{F}_{alg}}, T_{\mathsf{Sim}})$*-oracle-aided algebraic security. Let* R *be an algebraic reduction such that for every algebraic forger* F_{alg} *that* $(\varepsilon_{\mathsf{F}_{alg}}, T_{\mathsf{F}_{alg}}, q_h, q_s)$*-breaks* Th-BLS$_{\mathsf{DKG}}$*,* $\mathsf{R}^{\mathsf{F}_{alg}}$ *is an algorithm that* $(\varepsilon_{\mathsf{R}}, T_{\mathsf{R}})$*-breaks* 2-OMDL*. Then there exists an algorithm* M *such that* M^{R} $(\varepsilon_{\mathsf{M}}, T_{\mathsf{M}})$*-breaks* 2-OMDL *with*

$$\varepsilon_{\mathsf{M}} \geq \varepsilon_{\mathsf{R}} - 2^{-t+1}, \quad T_{\mathsf{M}} \leq T_{\mathsf{R}} + T_{\mathsf{F}_{alg}}.$$

Proof. Assume that R is an algebraic reduction as defined above. We will now build an efficient solver M against 2-OMDL. Let $\xi = (g^{z_1}, g^{z_2}) \in \mathbb{G}^2$ be the OMDL instance. M gets access to $\mathsf{DL}_g()$ at most one time and his goal is to return (z_1, z_2). Algorithm M works as follows.

1. M runs the reduction R providing it with (par, ξ). The discrete logarithm oracle for R is simulated by M's own oracle $\mathsf{DL}_g()$. As R is an algebraic reduction, at the end of the key generation phase, it returns a vector of public key shares $(g^{x_1}, \ldots, g^{x_n})$ together with a representation $(\hat{a}_i, a_{i,1}, a_{i,2})$ for all $i \in [n]$ such that

$$g^{x_i} = g^{\hat{a}_i} \cdot \prod_{j=1}^{2} (g^{z_j})^{a_{i,j}} = g^{\hat{a}_i} \cdot (g^{z_1})^{a_{i,1}} \cdot (g^{z_2})^{a_{i,2}}.$$

2. After termination of the key generation protocol, M chooses a random subset $S \subset \{1, \ldots, n\}$ of parties of order t. W.l.o.g. we may assume $S = \{1, \ldots, t\}$. Then M queries R, on behalf of a simulated algebraic forger F_{alg}^{sim}, for corruptions of the random t parties given by S, that is $P_1, \ldots, P_t$. Reduction R returns the internal states of these parties and M additionally gets full control over them. We stress that the secret key shares of parties $P_1, \ldots, P_t$ returned by R are all correct, which can be checked using the public key shares.

3. Afterwards, M forms the $(n \times 2)$-matrix over the field $\mathbb{Z}_p$

$$Q := \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ \vdots & \vdots \\ a_{n,1} & a_{n,2} \end{pmatrix}$$

and computes its rank efficiently via Gaussian elimination. We may assume that the rank of this matrix is 2. The other cases where the rank of Q is less than 2 work analogously.

Consider the $(t \times 2)$-matrix Q_S resulting from the t rows of Q corresponding to the set S, i.e.,

$$Q_S := \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ \vdots & \vdots \\ a_{t,1} & a_{t,2} \end{pmatrix}.$$

We consider the following three cases regarding Q:

(i) There are $t + 1$ row vectors in Q that span a 0-dimensional space.

(ii) For $n_0 \leq t$ the number of zero row vectors in Q, there are $t + 1 - n_0$ non-zero row vectors in Q that span a 1-dimensional space.

(iii) For $n_0 \leq t$ the number of zero row vectors in Q, any $t + 1 - n_0$ non-zero row vectors in Q span a 2-dimensional space.

In each of these cases we describe how M proceeds. For all $i \in [n]$, we let $\vec{a}_i := (a_{i,1}, a_{i,2})$. For each case, we define the two events E and F_i, $i \in [3]$, as follows. E is in each case the event that the reduction R is successful upon M corrupting these t parties. In case (i), F_1 is the event that there are at most t row vectors in Q that are zero (obviously, the probability that this happens is zero). In case (ii), F_2 is the event that all vectors $\vec{a}_1, \ldots, \vec{a}_t$ (corresponding to the corrupted parties $P_1, \ldots, P_t$) are zero. In case (iii), F_3 is the event that the matrix Q_S has rank ≤ 1. In case event F_i, $i \in [3]$, happens, M fails and just aborts. But in case F_i does not happen, M is able to determine the secret key x in each of the three cases, as we will see now.

Case (i): In that case, there are $t + 1$ row vectors, say $\vec{a}_{i_1}, \ldots, \vec{a}_{i_{t+1}}$, such that the corresponding secret key shares are defined as $x_j = \hat{a}_{i_j}$ for $j \in [t + 1]$ and M efficiently computes the secret key x by Lagrangian interpolation of these $t + 1$ points. M proceeds with step 4.

Case (ii): In that case, there are $t + 1$ row vectors in Q (and $n_0 \geq 0$ of them being zero), say $\vec{a}_{i_1}, \ldots, \vec{a}_{i_{t+1}}$ (w.l.o.g. with $\vec{a}_{i_1}, \ldots, \vec{a}_{n_0}$ being zero), that span a 1-dimensional space, so that for all $j \in [t]$ it is $\vec{a}_{i_j} = \mu_j \vec{a}_{i_{t+1}}$ for some efficiently computable $\mu_j \in \mathbb{Z}_p$. Note that we could replace the role of $\vec{a}_{i_{t+1}}$ with any other $\vec{a}_{i_j}$ for $j \in [n_0 + 1, t + 1]$. Since $x_i - \hat{a}_i = \vec{a}_i \cdot \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$ for all $i \in [n]$, we find that

$$\mu_j (x_{i_{t+1}} - \hat{a}_{i_{t+1}}) = \mu_j \vec{a}_{i_{t+1}} \cdot \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \vec{a}_{i_j} \cdot \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = x_{i_j} - \hat{a}_{i_j}$$

for all $j \in [t]$. This shows that the polynomial $f \in \mathbb{Z}_p[X]$ of degree t that gives the secret key shares is by Lagrangian interpolation determined by only one point, namely $x_{i_{t+1}}$ corresponding to the row vector $\vec{a}_{i_{t+1}} \neq 0$. Note that by the same argumentation, f is determined by any one point x_{i_j} for $j \in [n_0 + 1, t + 1]$) corresponding to a row vector $\vec{a}_{i_j}$ that is not zero. And in particular, any secret key share is computable via the Lagrangian interpolation formula by knowledge of $x_{i_{t+1}}$ (or alike secret key share corresponding to a non-zero row vector). We conclude that any secret key share $x_i = f(i)$ for $i \in [n]$ has an

expression as $l_i + l'_i \cdot x_{i_{t+1}}$ for some efficiently computable constants $l_i, l'_i \in \mathbb{Z}_p$, where $l'_i \neq 0$ if and only if the ith row vector of Q is non-zero. Hence, knowledge of any x_i corresponding to a non-zero row vector of Q determines f completely. So in case event F_2 does not happen, M determines f as described, computes the secret key x, and proceeds with step 4.

Case (iii): In that case, assuming event F_3 does not happen, Q_S has full rank 2 and therefore the vectors $\vec{a}_1, \ldots, \vec{a}_t$ give a generating set for the row space of Q. In particular, $\vec{a}_{t+1}$ is linearly dependent from the other first t vectors $\vec{a}_1, \ldots, \vec{a}_t$. This yields a linear system of equations $\lambda_1\vec{a}_1 + \ldots + \lambda_t\vec{a}_t = \vec{a}_{t+1}$, where $\lambda_i \in \mathbb{Z}_p$ for all $i \in [t]$. Via Gaussian elimination, M determines the linear coefficients $\lambda_1, \ldots, \lambda_t$ of this system of equations. Note that this approach would not work in the static corruption model, as previously explained. As a result, M obtains the following system of linear equations in z_1, z_2:

$$\begin{aligned} x_1 &= \hat{a}_1 + a_{1,1}z_1 + a_{1,2}z_2 \\ x_2 &= \hat{a}_2 + a_{2,1}z_1 + a_{2,2}z_2 \\ &\vdots \\ x_t &= \hat{a}_t + a_{t,1}z_1 + a_{t,2}z_2, \end{aligned}$$

which in matrix form is equivalent to

$$\begin{pmatrix} x_1 - \hat{a}_1 \\ x_2 - \hat{a}_2 \\ \vdots \\ x_t - \hat{a}_t \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ \vdots & \vdots \\ a_{t,1} & a_{t,2} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}.$$

Multiplying the i-th row of the matrix with λ_i for all $i \in [t]$ and adding up the equations yields

$$\sum_{i=1}^{2} \lambda_i (x_i - \hat{a}_i) = \vec{a}_{t+1} \cdot \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \sum_{i=1}^{2} a_{t+1,i} z_i.$$

Since $\sum_{i=1}^{2} a_{t+1,i} z_i = x_{t+1} - \hat{a}_{t+1}$, M efficiently computes x_{t+1} via the identity $x_{t+1} = \hat{a}_{t+1} + \sum_{i=1}^{2} \lambda_i (x_i - \hat{a}_i)$ by knowledge of the secret key shares $x_1, \ldots, x_t$ it got at the beginning of this step through the corruptions. Using Lagrange interpolation, M determines the secret key x and proceeds with step 4.

4. M picks a set $\mathcal{M} \subset \{0,1\}^*$ of q_h arbitrary messages (e.g., uniformly at random or the lexicographically first). Then it samples a message $m^* \leftarrow \mathcal{M}$ and a tuple of messages $(m_1, \ldots, m_{q_s}) \leftarrow (\mathcal{M} \setminus \{m^*\})^{q_s}$.

5. M queries the signing oracle, with implicit hash queries, on the messages $m_1, \ldots, m_{q_s}$. Thereafter, M makes a hash query on m^* and $q_h - q_s - 1$ additional messages from the set $\mathcal{M}$. In total, M has made exactly q_h hash queries, including the implicit hash queries from signing, and exactly q_s signing queries, so that it corresponds to what the reduction expects.

6. M then tosses a biased coin $\zeta \in \{0,1\}$ that takes the value 1 with probability $\varepsilon_{\mathsf{F}_{alg}}$ and the value 0 with probability $1-\varepsilon_{\mathsf{F}_{alg}}$. If $\zeta = 0$, then M sends $\perp$ to R. And if $\zeta = 1$, then M computes $\sigma^* = H[m^*]^x$ and submits (m^*, σ^*) as a forgery with algebraic representation $(x, 0, \ldots, 0)$, so that $\sigma^* = g_0^x \cdot \prod_{i\geq 1} g_i^0$ where $(g_0, g_1, \ldots, g_r)$ is the list of all group elements M has received during the execution of R and we assume w.l.o.g. $g_0 = H[m^*]$. This is done in time $T_{\mathsf{F}_{alg}}$ in order to correctly simulate an algebraic forger.

7. We see that this constitutes a valid forgery as follows. First, m^* was not queried to the signing oracle and σ^* is indeed a valid signature on m^*. Second, consider (as a thought-experiment) an unbounded algebraic forger $\mathsf{F}_{alg} = \mathsf{F}_{alg}^{unb}$ that brute-forces the secret key x from the public key g^x and outputs a valid forgery σ^* on m^* with probability $\varepsilon_{\mathsf{F}_{alg}}$ in case event F_i for an $i \in [3]$ from step 3 does not happen. By assertion of the theorem, R has to work even against such an unbounded forger. Clearly, the view of R when interacting with F_{alg}^{sim} is indistinguishable from its view when interacting with F_{alg}^{unb} whenever event F_i does not happen. Hence, σ^* is a valid signature on m^*. Additionally, this yields $\varepsilon_{\mathsf{M}} \geq \varepsilon_{\mathsf{R}} - \Pr[F_i]$ for $i \in [3]$. R will then return (z_1, z_2), which M submits as its solution against 2-OMDL.

Finally, we bound the probability that F_i happens for each $i \in [3]$. In case (i), we clearly have $\Pr[F_1] = 0$. In case (ii), since there are at most t row vectors in Q being zero, the probability that M's corruption set S contains exactly these parties (corresponding to zero row vectors in Q) is approximately 2^{-t}, i.e., $\Pr[F_2] \leq 2^{-t}$. In case (iii), we have the following fact.

Lemma 3.5.4. *The matrix Q_S has rank 2 with probability at least $1 - 2^{-t+1}$.*

Proof. We consider the projective 1-space $\mathbb{P}^1(\mathbb{F}_p)$ over the finite field $\mathbb{F}_p = \mathbb{Z}_p$ with p elements. The space $\mathbb{P}^1(\mathbb{F}_p)$ consists of all 1-dimensional subspaces of $\mathbb{F}_p^2$ (considered as a 2-dimensional vector space over $\mathbb{F}_p$). By assumption, there are at most $t - n_0$ non-zero row vectors in Q that span a 1-dimensional space. Thus, at most $t - n_0$ non-zero row vectors of Q correspond to one particular point in $\mathbb{P}^1(\mathbb{F}_p)$. In order to Q_S to have rank less than 2, the row vectors corresponding to the (randomly chosen) corruption set $S = \{1, \ldots, t\}$ would precisely have to be the n_0 zero row vectors and remaining $t - n_0$ (non-zero) vectors that correspond to one particular point in the projective space, that is these $t - n_0$ vectors would span a 1-dimensional space. Otherwise, there would be at least two row vectors that correspond to different points in the projective space and would therefore span a 2-dimensional space. The probability of this happening is at most $\frac{1}{2^{n_0}} \cdot \frac{2}{2^{t-n_0}} = \frac{1}{2^{t-1}}$. Therefore, Q_S has rank 2 with probability $\geq 1 - 2^{-t+1}$. □

As a result, we get $\Pr[F_3] \leq 2^{-t+1}$. Overall, the bound on M's success probability in breaking 2-OMDL is given by $\varepsilon_{\mathsf{M}} \geq \varepsilon_{\mathsf{R}} - \max_{i\in[3]}\{\Pr[F_i]\} \geq \varepsilon_{\mathsf{R}} - 2^{-t+1}$. The bound on M's time comes from running the reduction R once and simulating the forger. □

In fact, the exact same proof strategy of the metareduction M can also be applied to the case where the underlying hardness assumption is OMDL of some higher degree $r \leq t$ instead of degree 2. The main difference to the former case 2-OMDL is that the probability that the matrix Q_S (which is now a $(t \times r)$-matrix over $\mathbb{Z}_p$) has full rank r is no longer $1 - 2^{-t+1}$, but another expression that we believe to depend on r. Based on that very probability, it is possible to extend our result as follows. For this, we first formulate the underlying mathematical problem and our conjecture for it.

Conjecture 3.5.4.1. *As usual let $t < n/2$. Let Q be an $(n \times r)$-matrix over $\mathbb{Z}_p$ with $1 \leq r \leq t$ such that any set of $t+1$ row vectors of Q generates a space of dimension r. Further, let $S \subset [n]$ be a t-element subset of $\{1, \ldots, n\}$ chosen uniformly at random, and let Q_S be the $(t \times r)$-matrix over $\mathbb{Z}_p$ formed by the row vectors of Q given by the index set S. Then matrix Q_S has full rank r with probability at least $1 - 2^{-t+r-1}$.*

The case $r = 1$ is clear, and the case $r = 2$ was treated in the preceding proof. We try to give an intuition of why the conjecture should be true. First assume that r is small (in the order of magnitude of $O(1)$). In that case there is much linear dependence among any set of $t+1$ row vectors of Q. Now, for a t-element subset $S \subset [n]$ to generate a subspace of smaller dimension $r-1$, all the other $n-t$ row vectors of Q would have to be linearly independent from the t vectors of S. It is now intuitively clear that this can be true only with extremely small probability (in the order of magnitude of $O(2^{-t})$). On the other hand, assume that r is large (in the order of magnitude of $t - O(1)$). In that case there is much linear independence among any set of $t+1$ row vectors of Q. Therefore, it is much more likely that a t-element subset $S \subset [n]$ will generate a subspace of the same dimension r.

For our purposes, Q is the matrix given by the algebraic coefficients of the public key shares as defined in the proof of Theorem 3.5.3. Furthermore, we note that the condition that any set of $t+1$ row vectors of Q generates a space of dimension r is satisfied if the reduction R is generic, assuming the matrix Q is of full rank (which can be assumed from the beginning w.l.o.g., as the lower rank cases work analogously). We will prove this in Section 3.5.4. Assuming Conjecture 3.5.4.1 to be true, we extend our result as follows.

Theorem 3.5.5. *Assume Conjecture 3.5.4.1 to be true. Let* DKG *have $(t, k, T_{\mathsf{F}_{alg}}, T_{\mathsf{Sim}})$-oracle-aided algebraic security. Let* R *be an algebraic reduction such that for every algebraic forger F_{alg} that $(\varepsilon_{\mathsf{F}_{alg}}, T_{\mathsf{F}_{alg}}, q_h, q_s)$-breaks* Th-BLS$_{\mathsf{DKG}}$*, $\mathsf{R}^{\mathsf{F}_{alg}}$ is an algorithm that $(\varepsilon_{\mathsf{R}}, T_{\mathsf{R}})$-breaks r-*OMDL *(with $1 \leq r \leq t$ as in Conjecture 3.5.4.1). Then there exists an algorithm* M *such that* M^{R} *$(\varepsilon_{\mathsf{M}}, T_{\mathsf{M}})$-breaks r-*OMDL *with*

$$\varepsilon_{\mathsf{M}} \geq \varepsilon_{\mathsf{R}} - 2^{-t+r-1}, \quad T_{\mathsf{M}} \leq T_{\mathsf{R}} + T_{\mathsf{F}_{alg}}.$$

For our next impossibility result regarding the existence of a security reduction from q-DLOG, the metareduction uses a very similar strategy as in the previous proof. The only difference is in how the metareduction M determines the secret key in step 2 of its strategy. Essentially, after some t corruptions, M obtains a non-trivial polynomial equation over $\mathbb{Z}_p$ in the variable z (the sought-for solution to the given q-DLOG instance $(g^z, \ldots, g^{z^q}) \in \mathbb{G}^q$), which it can solve efficiently by standard techniques.

Theorem 3.5.6. *Let* DKG *have $(t, k, T_{\mathsf{F}_{alg}}, T_{\mathsf{Sim}})$-oracle-aided algebraic security. Let* R *be an algebraic reduction such that for every algebraic forger F_{alg} that $(\varepsilon_{\mathsf{F}_{alg}}, T_{\mathsf{F}_{alg}}, q_h, q_s)$-breaks* Th-BLS$_{\mathsf{DKG}}$*, $\mathsf{R}^{\mathsf{F}_{alg}}$ is an algorithm that $(\varepsilon_{\mathsf{R}}, T_{\mathsf{R}})$-breaks q-*DLOG*. Then there exists an algorithm* M *such that* M^{R} *$(\varepsilon_{\mathsf{M}}, T_{\mathsf{M}})$-breaks q-*DLOG *with*

$$\varepsilon_{\mathsf{M}} \geq \varepsilon_{\mathsf{R}} - 2^{-t}, \quad T_{\mathsf{M}} \leq T_{\mathsf{R}} + T_{\mathsf{F}_{alg}}.$$

Proof. Assume that R is an algebraic reduction as defined above. We will now build an efficient solver M against q-DLOG. Let $\xi = (g^z, \ldots, g^{z^q}) \in \mathbb{G}^q$ be the q-DLOG instance. M his goal is to return $z \in \mathbb{Z}_p$. Algorithm M works as follows.

1. M runs the reduction R providing it with (par, ξ). As R is an algebraic reduction, at the end of the key generation phase, it returns a vector of public key shares $(g^{x_1}, \ldots, g^{x_n})$ together with a representation $(\hat{a}_i, a_{i,1}, \ldots, a_{i,q})$ for all $i \in [n]$ such that

$$g^{x_i} = g^{\hat{a}_i} \cdot \prod_{j=1}^{q} \left(g^{z^j}\right)^{a_{i,j}} = g^{\hat{a}_i} \cdot (g^z)^{a_{i,1}} \cdot \ldots \cdot \left(g^{z^q}\right)^{a_{i,q}}.$$

2. After termination of the key generation protocol, M forms the $(n \times q)$-matrix over the field $\mathbb{Z}_p$,

$$Q := \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,q} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,q} \\ \vdots & \vdots & & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,q} \end{pmatrix}.$$

   For all $i \in [n]$, we let $\vec{a}_i := (a_{i,1}, \ldots, a_{i,q})$. With that, we consider the following three cases:

   (i) There are at least $t + 1$ row vectors in Q identical to the zero vector.

   (ii) There are at most $t - 1$ row vectors in Q identical to the zero vector.

   (iii) There are exactly t row vectors in Q identical to the zero vector.

   In each of these cases we describe how M proceeds.

   Case (i): In that case, there are $t + 1$ rows, say $\vec{a}_1, \ldots, \vec{a}_{t+1}$, such that the corresponding secret key shares are defined as $x_j = \hat{a}_j$ for $j \in [t + 1]$ and M efficiently computes the secret key x by Lagrangian interpolation of these $t + 1$ points. M proceeds with step 3.

   Case (ii): In that case, M queries R, on behalf of a simulated algebraic forger F_{alg}^{sim}, for corruptions of some random t parties, say $P_1, \ldots, P_t$. Reduction R returns the internal states of these parties and M additionally gets full control over them. We stress that the secret key shares of parties $P_1, \ldots, P_t$ returned by R are all correct, which can be checked using the public key shares. Since there are at most $t - 1$ rows in Q identical to the zero vector, one of the rows $\vec{a}_1, \ldots, \vec{a}_t$ (corresponding to the corrupted parties $P_1, \ldots, P_t$) is non-zero (w.l.o.g. $\vec{a}_1 \neq 0$). As a result, M obtains the following polynomial equation over $\mathbb{Z}_p$ in the variable z: $x_1 = \hat{a}_1 + a_{1,1}z + a_{1,2}z^2 + \ldots + a_{1,q}z^q \in \mathbb{Z}_p[z]$. By standard techniques, M efficiently computes z and is done. It aborts the reduction and outputs z as the solution of the q-DLOG instance.

   Case (iii): In that case, M queries R, on behalf of a simulated algebraic forger F_{alg}^{sim}, for corruptions of some random t parties, say $P_1, \ldots, P_t$. Reduction R returns the internal states of these parties and M additionally gets full control over them. We define the following two events: E is the event that the reduction R is successful upon M corrupting these t parties, and F is the event that not all vectors $\vec{a}_1, \ldots, \vec{a}_t$ (corresponding to the corrupted parties $P_1, \ldots, P_t$) are zero. In case event F happens, proceed as in case (ii). Now, the reduction is successful with probability ε_{R} if the algebraic forger is successful.

That means, $\varepsilon_{\mathsf{R}} = \Pr[E]$ (conditioned on the algebraic forger F_{alg} being successful). Conditioned on the algebraic forger being successful, we obtain the following identities:

$$\begin{aligned}\varepsilon_{\mathsf{R}} = \Pr[E] &= \Pr[E \wedge F] + \Pr[E \wedge \neg F] \\ &\leq \Pr[E \wedge F] + \Pr[E \mid \neg F] \Pr[\neg F] \\ &\leq \Pr[E \wedge F] + \Pr[\neg F] \leq \Pr[E \wedge F] + 2^{-t}.\end{aligned}$$

It follows that $\Pr[E \wedge F] \geq \varepsilon_{\mathsf{R}} - 2^{-t}$ and thus $\varepsilon_{\mathsf{M}} \geq \varepsilon_{\mathsf{R}} - 2^{-t}$.

3. M picks a set $\mathcal{M} \subset \{0,1\}^*$ of q_h arbitrary messages (e.g., at random or the lexicographically first). Then it samples a message $m^* \leftarrow \mathcal{M}$ and a tuple of messages $(m_1, \ldots, m_{q_s}) \leftarrow (\mathcal{M} \setminus \{m^*\})^{q_s}$.

4. M queries the signing oracle, with implicit hash queries, on the messages $m_1, \ldots, m_{q_s}$. Thereafter, M makes a hash query on m^* and $q_h - q_s - 1$ additional messages from the set $\mathcal{M}$. In total, M has made exactly q_h hash queries, including the implicit hash queries from signing, and exactly q_s signing queries, so that it corresponds to what the reduction expects.

5. M then tosses a biased coin $\zeta \in \{0,1\}$ that takes the value 1 with probability $\varepsilon_{\mathsf{F}_{alg}}$ and the value 0 with probability $1 - \varepsilon_{\mathsf{F}_{alg}}$. If $\zeta = 0$, then M sends $\perp$ to R. And if $\zeta = 1$, then M computes $\sigma^* = H[m^*]^x$ and submits (m^*, σ^*) as a forgery with algebraic representation $(x, 0, \ldots, 0)$, so that $\sigma^* = g_0^x \cdot \prod_{i \geq 1} g_i^0$ where $(g_0, g_1, \ldots, g_r)$ is the list of all group elements M has received during the execution of R and we assume w.l.o.g. $g_0 = H[m^*]$. This is done in time $T_{\mathsf{F}_{alg}}$ in order to correctly simulate an algebraic forger.

6. We see that this constitutes a valid forgery as follows. First, m^* was not queried to the signing oracle and σ^* is indeed a valid signature on m^*. Second, consider (as a thought-experiment) an unbounded algebraic forger $\mathsf{F}_{alg} = \mathsf{F}_{alg}^{unb}$ that brute-forces the secret key x from the public key g^x and outputs a valid forgery σ^* on m^* with probability $\varepsilon_{\mathsf{F}_{alg}}$. By assertion of the theorem, R has to work even against such an unbounded forger. Clearly, the view of R when interacting with F_{alg}^{sim} is indistinguishable from its view when interacting with F_{alg}^{unb}. Hence, σ^* is a valid signature on m^*. R will then return z with probability ε_{R}, which M submits as its solution against q-DLOG.

Since F_{alg}^{sim} perfectly simulates a real algebraic forger for R, the bound on A's success probability in breaking q-DLOG is clear. The bound on M's time comes from running the reduction R once and simulating the forger. □

Remark 3.5.2. An important implication of this theorem is that the security of Th-BLS$_{\mathsf{DKG}}$ can not be derived from a static mathematical assumption in the AGM (and in particular not in the plain ROM). This follows from the fact that q-DLOG implies every conceivable hardness assumption in the AGM (and even some non-static assumptions) [BFL20]. If there was a reduction from such a static assumption, this would mean that a successful forgery would lead to the breaking of this underlying assumption. And this would in turn lead to the breaking of q-DLOG for some q. Thus, there would ultimately exist a reduction from q-DLOG for some q to the security of Th-BLS$_{\mathsf{DKG}}$, which stands in direct contradiction to Theorem 3.5.6.

3.5.3 Security of several DKG protocols

Before we proceed with our impossibility result on the tightness of a security proof for Th-BLS$_{\mathsf{DKG}}$ under the $(t+1)$-OMDL assumption, we focus on some concrete DKG protocols. In more detail, we show the (t,k)-oracle-aided algebraic security of several well-known DKG protocols, so that these can be safely employed into Th-BLS$_{\mathsf{DKG}}$. We begin our excursion with TD-DKG, which serves as a DKG protocol that represents the traditional trusted dealer scheme. In this protocol, a trusted dealer TD chooses a random polynomial $f \in \mathbb{Z}_p[X]$ of degree t. Then, for all $i \in [n]$, it secretly sends the secret key share $sk_i = f(i)$, the vector of public key shares $(pk_1, \ldots, pk_n) = (g^{f(1)}, \ldots, g^{f(n)})$, and the public key $pk = g^{f(0)}$ to party P_i.

Our strategy for the proof is by building an (t,k)-oracle-aided algebraic simulator Sim with $k = t+1$ that simulates the role of the trusted dealer TD in an execution of TD-DKG. On input $t+1$ elements $\xi = (\xi_0, \ldots, \xi_t) \in \mathbb{G}^{t+1}$, Sim defines the polynomial $f \in \mathbb{Z}_p[X]$ of degree t by embedding ξ_i into the ith coefficient of f for all $i \in [0,t]$. Corruption queries are answered with the oracle $\mathsf{DL}_g()$.

Theorem 3.5.7. *Protocol* TD-DKG *has* $(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$*-oracle-aided algebraic security with* $k = t+1$ *and* $T_{\mathsf{Sim}} \leq T_{\mathsf{A}} + 2n(t+1)$.

Proof. Let A be an adversary that runs in time at most T_{A} and corrupts at most t parties during an execution of the protocol. Clearly, TD-DKG is t-consistent and t-correct. It remains to show $(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$-oracle-aided algebraic simulatability for $k = t+1$. Theorem 3.5.3 then implies the simulatability factor $t+1$. For this, we build an $(t, t+1, T)$-oracle-aided algebraic simulator Sim as follows. On input $t+1$ elements $\xi_0, \ldots, \xi_t$ where $\xi_i = g^{z_i}$ for $i \in [0,t]$ with t-time access to an oracle $\mathsf{DL}_g()$, Sim lets the polynomial $f = \sum_{i=0}^{t} a_i X^i \in \mathbb{Z}_p[X]$ of degree t be such that $g^{a_i} = \xi_i$ for all $i \in [0,t]$, which implicitly sets $a_i = z_i$. Then, for all $i \in [n]$, Sim computes $g^{f(i)}$ as

$$g^{f(i)} = g^{\sum_{j=0}^{t} a_j i^j} = \prod_{j=0}^{t} (g^{a_j})^{i^j} = \prod_{j=0}^{t} \xi_j^{i^j}$$

and sends the public key shares $(pk_1, \ldots, pk_n) = (g^{f(1)}, \ldots, g^{f(n)})$ along with the public key $pk = g^{f(0)} = \xi_0$ to party P_i. Whenever A decides to corrupt a party P_j, Sim queries $\mathsf{DL}_g(g^{f(j)})$ and returns $sk_j = f(j)$. Since A makes at most t corruption queries, Sim accesses the oracle $\mathsf{DL}_g()$ at most t times and hence is a well-defined simulator. Let $C \subset \{1, \ldots, n\}$ denote the subset of corrupted parties at the end of an execution of Sim. W.l.o.g. we may assume that $|C| = t$. By construction, the simulatability matrix of Sim is the square Vandermonde matrix $V(\ldots)$ for the $t+1$ distinct numbers in $C \cup \{0\}$, which is invertible. Finally, f is indistinguishable from a random polynomial over $\mathbb{Z}_p$ of degree t and Sim's simulation of TD-DKG is perfect. The claim on the running time is easy to verify and thus omitted. □

Now we turn to Pedersen's JF-DKG protocol. We note that the public key shares $(g^{x_1}, \ldots, g^{x_n})$ are not output explicitly by JF-DKG, but can be computed from publicly available information [Gen+07]. Therefore, we may simply assume that these values are publicly known. The proof of the following theorem is essentially just an adaption of the preceding proof to the setting where each party P_i acts as a dealer with its own polynomial $f_i \in \mathbb{Z}_p[X]$. Concretely, we build an (t,k)-oracle-aided algebraic simulator Sim with $k = n(t+1)$ that simulates the role of

the honest parties in an execution of JF-DKG. On input $n(t+1)$ elements $\xi = \xi_0, \ldots, \xi_{k-1} \in \mathbb{G}$, Sim defines the polynomials $f_i \in \mathbb{Z}_p[X]$ of degree t by embedding different $t+1$ $\xi_{i_0}, \ldots, \xi_{i_t}$ into the coefficients of f_i for all $i \in [n]$ such that $\bigcup_{i\in[n]}\{i_0, \ldots, i_t\} = [n(t+1)-1]$. This means, Sim just evenly distributes ξ among all the polynomials f_i. And corruption queries are answered through the oracle $\mathsf{DL}_g()$. By the end of the simulation, the simulatability matrix of Sim is just a block diagonal matrix with the ith block being the square Vandermonde matrix for some $t+1$ distinct numbers. This is due to the fact that the i-th block corresponds to the polynomial f_i and these f_i each have different $\xi_I = \xi_{i_0}, \ldots, \xi_{i_t}$ elements embedded. Since this block diagonal matrix is invertible, Sim is a well-defined oracle-aided algebraic simulator.

Theorem 3.5.8. *Protocol* JF-DKG *has* $(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$*-oracle-aided algebraic security with* $k \leq n(t+1)$ *and* $T_{\mathsf{Sim}} \leq T_{\mathsf{A}} + 2n^2(t+1) + n$.

Proof. Let A be an adversary that runs in time at most T_{A} and corrupts at most t parties during an execution of the protocol. JF-DKG is known to be t-consist and t-correct. It remains to show $(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$-oracle-aided algebraic simulatability for $k = n(t+1)$. For this, we build an appropriate algebraic simulator Sim as follows. Let $C, \mathcal{H}$ denote the dynamically evolving subsets of corrupted and honest parties, respectively. Whenever A decides to corrupt a party P_i, Sim returns the internal state of P_i before setting $C = C \cup \{i\}$ and $\mathcal{H} = \mathcal{H} \setminus \{i\}$. Initially, we set $C = \emptyset$. On input k elements $\xi_0 = g^{z_0}, \ldots, \xi_{k-1} = g^{z_{k-1}}$ with $(k-1)$-time access to an oracle $\mathsf{DL}_g()$, for all $i \in [n]$, Sim lets the polynomial $f_i = \sum_{k=0}^{t} a_{ik}X^k \in \mathbb{Z}_p[X]$ of degree t be such that $g^{a_{ij}} = \xi_{(i-1)(t+1)+j}$ for all $j \in [0,t]$. In particular, the coefficients of f_1 are $z_0, \ldots, z_t$, those of f_2 are $z_{t+1}, \ldots, z_{2t+1}$, etc. f_i is the polynomial of party P_i.

For all $i, j \in [n]$, Sim computes $g^{f_i(j)}$ as usual. We note that as long as Sim works properly, the only elements broadcast by it are those in Step 1. Here, for all $i \in \mathcal{H}$, Sim broadcasts $A_{ik} = g^{a_{ik}}$ for $k \in [0,t]$. Whenever A decides to corrupt P_i, Sim samples a subset $\mathcal{G}_i \leftarrow \{1, \ldots, n\} \setminus C$ of order $t+1-|C|$ and queries $\mathsf{DL}_g(g^{f_i(k)})$ for all $k \in \mathcal{G}_i$. In addition, Sim queries $\mathsf{DL}_g(g^{f_j(i)})$ for all $j \in \mathcal{H} \setminus \{i\}$. (The idea here is that for each new corruption query j, Sim already knows $|C|$ points on the polynomial f_j from previous corruption queries. Thus, in order to return f_j to the adversary, Sim has to query $\mathsf{DL}_g()$ only on some other (random) $t+1-|C|$ points. Additionally, in order to return P_j's internal state properly, Sim also has to query $\mathsf{DL}_g()$ on the polynomial shares $f_j(i)$ this party P_j is supposed to get from the other (honest) parties.) By drawing those values from previous corruption queries, Sim determines the polynomial f_i and returns the data f_i and $f_j(i)$ for $j \in \mathcal{H}$ before updating C and $\mathcal{H}$. We may w.l.o.g. assume that A corrupts exactly t parties. At the end of the simulation, Sim samples $i^* \leftarrow \mathcal{H}$ and queries $\mathsf{DL}_g(g^{a_{i0}})$ for all $i \in \mathcal{H} \setminus \{i^*\}$. (This is done in order to determine an algebraic representation for the public key.) Subsequently, it outputs the public key $pk = \prod_{i\in Q} g^{a_{i0}}$ with representation $(\sum_{i\in Q\setminus\{i^*\}} a_{i0}, 0, \ldots, 0, 1, 0, \ldots 0)$ such that it correctly is $y = g^{\sum_{i\in Q\setminus\{i^*\}} a_{i0}} \cdot (g^{a_{i^*0}})^1$.

First, we verify that Sim is well-defined by counting the total number of queries to $\mathsf{DL}_g()$ it has made. The j-th corruption query (on party P_i) yields $|\mathcal{G}_i| + |\mathcal{H} \setminus \{i\}| = t+1-|C|+|\mathcal{H}|-1 = (t+2-j)+(n-j)$ oracle queries. Summing up these values from $j=1$ to $j=t$ yields $\sum_{j=1}^{t}(t+2-j)+(n-j) = t(n+1)$. At the end of the simulation, Sim makes $|\mathcal{H}|-1 = n-t-1$ additional queries, which gives a total number of $t(n+1)+(n-t-1) = n(t+1)-1$ oracle queries. Thus, Sim is a well-defined oracle-aided algebraic simulator.

We consider the simulatability matrix L of Sim. By construction, Sim has eventually queried

$\mathsf{DL}_g()$ on $g^{f_i(\cdot)}$ for all $i \in [n] \setminus \{i^*\}$ on some $t+1$ distinct arguments (input values) and $g^{f_{i^*}(\cdot)}$ on some t distinct arguments $\neq 0$ with the public key corresponding to $a_{i^*0} = f_{i^*}(0)$. Thus, the queried elements along with the public key essentially correspond to $t+1$ distinct arguments of the polynomials $f_1, \ldots, f_n$. Since the invertibility of a square matrix is invariant under row switching transformations (corresponding to the chronological order of queries made to $\mathsf{DL}_g()$ during the execution of the DKG protocol), we may assume that we make the queries corresponding to each polynomial f_i at once. Individually, the matrix corresponding to the queries $g^{f_i(\cdot)}$ is the square Vandermonde matrix of the $t+1$ distinct input arguments, which is invertible. Overall, we find that L is a block diagonal matrix

$$\begin{pmatrix} V(f_1) & & & \\ & V(f_2) & & \\ & & \ddots & \\ & & & V(f_n) \end{pmatrix},$$

since every polynomial f_i has different coefficients: f_1 has coefficients $z_0, \ldots, z_t$ and thus $V(f_1)$ occupies the first $t+1$ rows/columns, f_2 has coefficients $z_{t+1}, \ldots, z_{2t+1}$, etc. Each block matrix is invertible and therefore L is also invertible. Finally, each f_i is indistinguishable from a random polynomial over $\mathbb{Z}_p$ of degree t and Sim's simulation of the protocol is perfect. The claim on the running time is easy to verify. □

The proof for Gennaro et al.'s New-DKG protocol is essentially the same as the preceding one for JF-DKG, since the „masking" polynomials g_i appearing in New-DKG do not contribute to the secret key shares. For these, Sim simply honestly samples $g_i \leftarrow \mathbb{Z}_p[X]$ at random and proceeds otherwise as in the proof for JF-DKG.

Theorem 3.5.9. *Protocol* New-DKG *has* $(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$*-oracle-aided algebraic security with* $k \leq n(t+1)$ *and* $T_{\mathsf{Sim}} \in T_{\mathsf{A}} + O(n^3)$.

Proof. This is done in the same fashion as the previous proof. Again, New-DKG is known to be t-consist and t-correct. The $(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$-oracle-aided algebraic simulator for $k = n(t+1)$ is described as follows. Sim works in the same way as the algebraic simulator in the previous proof, whereby honestly sampling the „masking" polynomials $g_i \leftarrow \mathbb{Z}_p[X]$ at random, so that these are known to Sim explicitly. The further analysis works analogously. The claim on the running time is easy to verify. □

3.5.4 Non-tightness of naive reduction from $(t+1)$-OMDL in ROM

We close this work with an impossibility result on the tightness of a security proof for $\mathsf{Th\text{-}BLS}_{\mathsf{DKG}}$ under the $(t+1)$-OMDL in the plain random oracle model, assuming the reduction is *naive*. We give a proper definition of what this means in our context.

Definition 3.5.1 (Naive Security Reduction)**.** Let R be an algebraic reduction from the hardness of $(t+1)$-OMDL to the security of $\mathsf{Th\text{-}BLS}_{\mathsf{DKG}}$. Then we say that R is *naive* if the $n \times (t+1)$-matrix Q formed by the algebraic coefficients of the public key shares satisfies the following condition: any set of $t+1$ row vectors of Q generates a space of dimension $t+1$.

As we now argue, naive reductions are an interesting class of reductions that appear difficult to bypass. As we show at the end of this section, any generic reduction that embeds the $(t+1)$-element OMDL instance "fully" into the polynomial $f \in \mathbb{Z}_p[X]$ of degree t that determines the secret key shares is indeed naive. By "fully", we simply mean that the matrix Q has full rank $t+1$. Furthermore, we emphasize that our tight reduction from Theorem 3.5.1 is an example of a naive reduction and that we do not know of a tighter reduction strategy. An example of a non-naive reduction would be one that embeds only t or fewer elements of the $(t+1)$-element OMDL instance into the polynomial f. However, it is unclear in what way this would be helpful.

As before, our proof follows the metareduction technique. In a typical scenario, the metareduction M rewinds the reduction R back to a previous state, with the consequence that M gains some new information from the second run of R which eventually allows M to simulate a forger to R successfully. In our case, however, the reduction has access to the $\mathsf{DL}_g()$ oracle which M also has to simulate to R. This comes with a subtle but severe problem: after rewinding R back to a previous state, M additionally has to answer the same number of $\mathsf{DL}_g()$ queries that R has made in its first run. This is a non-trivial or even impossible task for M, unless R has made none oracle queries in its first run. But a priori, M can not predict or control R's behaviour at all. We resolve this issue by finding a state $\mathcal{I}_{\mathsf{R}}$ of R in which R necessarily must have queried the $\mathsf{DL}_g()$ oracle t times. This state $\mathcal{I}_{\mathsf{R}}$ will then be the state to which we rewind R later. In fact, $\mathcal{I}_{\mathsf{R}}$ will be shortly after the termination of DKG. The remainder of the proof proceeds as in the spirit of [Cor02; KK18] which equally leads to a security loss linear in the number of signing queries q_s.

Theorem 3.5.10. *Let* DKG *have* $(t, k, T_{\mathsf{F}}, T_{\mathsf{Sim}})$*-oracle-aided algebraic security with* $k = t+1$*. Let* R *be a naive reduction as defined above such that for every forger* F *that* $(\varepsilon_{\mathsf{F}}, T_{\mathsf{F}}, q_h, q_s)$*-breaks* Th-$\mathsf{BLS}_{\mathsf{DKG}}$*,* R^{F} *is an algorithm that* $(\varepsilon_{\mathsf{R}}, T_{\mathsf{R}})$*-breaks* $(t+1)$-OMDL*. Then there exists an algorithm* M *such that* M^{R} $(\varepsilon_{\mathsf{M}}, T_{\mathsf{M}})$*-breaks* $(t+1)$-OMDL *with*

$$\varepsilon_{\mathsf{M}} \geq \varepsilon_{\mathsf{R}} - \varepsilon_{\mathsf{F}} \cdot \frac{2}{eq_s}, \quad T_{\mathsf{M}} \leq 2(T_{\mathsf{R}} + T_{\mathsf{F}}).$$

In particular, a naive reduction from Th-$\mathsf{BLS}_{\mathsf{DKG}}$ *to* $(t+1)$-OMDL *with a security loss less than* q_s *yields a solver* M *for* $(t+1)$-OMDL *with positive success probability* ε_{M}*.*

Proof. Assume that R is an algebraic reduction as defined above. We will now build an efficient solver M against $(t+1)$-OMDL. Let $\xi = (g^{z_1}, \ldots, g^{z_{t+1}}) \in \mathbb{G}^{t+1}$ be the OMDL instance. M gets access to $\mathsf{DL}_g()$ at most t times and his goal is to return $(z_1, \ldots, z_{t+1})$. Algorithm M works as follows.

1. M runs the reduction R providing it with (par, ξ) and access to $\mathsf{DL}_g()$. As R is an algebraic reduction, at the end of the key generation phase, it returns a vector of public key shares $(g^{x_1}, \ldots, g^{x_n})$ together with a representation $(\hat{a}_i, a_{i,1}, \ldots, a_{i,t+1})$ for all $i \in [n]$ such that

$$g^{x_i} = g^{\hat{a}_i} \cdot \prod_{j=1}^{t+1} (g^{z_j})^{a_{i,j}} = g^{\hat{a}_i} \cdot (g^{z_1})^{a_{i,1}} \cdot \ldots \cdot (g^{z_{t+1}})^{a_{i,t+1}}.$$

2. After termination of the key generation phase, M forms the $n \times (t+1)$-matrix Q over the

field $\mathbb{Z}_p$

$$Q := \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,t+1} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,t+1} \\ \vdots & \vdots & & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,t+1} \end{pmatrix}.$$

Since R is a naive reduction, any set of $t+1$ row vectors of Q generates a space of dimension $t+1$. It follows that any set of t row vectors of Q generates a space of dimension t, i.e., any t row vectors of Q are linearly independent. Now, M samples a random subset $C \subset \{1, \ldots, n\}$ of parties of order t and queries R, on behalf of a simulated forger F_{sim}, for corruptions of these parties. Reduction R returns the internal states of these parties and M additionally gets full control over them. We stress that the secret key shares of parties $P_i \in C$ returned by R are all correct, which can be checked using the public key shares. We denote R's state up to this point by $\mathcal{I}_{\mathsf{R}}$. Suppose that R has queried $\mathsf{DL}_g()$ less than t times up to this point. In that case, M aborts the reduction R and queries $\mathsf{DL}_g(g^{x_{j^*}})$ for an arbitrary $j^* \leftarrow \{1, \ldots, n\} \setminus C$. Since any $t+1$ row vectors of Q span the whole row space of Q, M efficiently computes $(z_1, \ldots, z_{t+1})$ as usual. Thus, unless $(t+1)$-OMDL is easy, we may assume that R has queried $\mathsf{DL}_g()$ at least t times up to this point.

3. Let $q = \max\{q_h, 2q_s\}$. M chooses a set $\mathcal{M} \subset \{0,1\}^*$ of q_h arbitrary messages. Then it samples $i \leftarrow [q_s]$, $m^* \leftarrow \mathcal{M}$ and $(m_1, \ldots, m_{q_s}) \leftarrow (\mathcal{M} \setminus \{m^*\})^{q_s}$, which defines the two sequences of messages

$$\mathcal{M}_1 = (m_1, \ldots, m_{i-1}, m^*), \quad \mathcal{M}_2 = (m_1, \ldots, m_{q_s}).$$

4. M queries the signing oracle, with implicit hash queries, on the messages in $\mathcal{M}_1$. If R does not abort, M receives the signature list $\mathcal{S}_1 = (\sigma_1, \ldots, \sigma_{i-1}, \sigma^*)$. If any of these signatures is invalid, which can be checked using the public key shares, then M aborts.

5. We rewind R back to its previous state $\mathcal{I}_{\mathsf{R}}$. Now, M queries the signing oracle, with implicit hash queries, on the messages in $\mathcal{M}_2$. If R does not abort, M receives the signature list $\mathcal{S}_2 = (\sigma_1, \ldots, \sigma_{q_s})$.[5] If any of these signatures is invalid, then M aborts.

6. M makes a hash query on m^* and $q_h - q_s - 1$ additional messages from the set $\mathcal{M}$. In total, M has made exactly q_h hash queries, including the implicit hash queries from signing, and exactly q_s signing queries, so that it correspond to what the reduction expects.

7. M then tosses a biased coin $\zeta \in \{0,1\}$ that takes the value 1 with probability ε_{F} and the value 0 with probability $1 - \varepsilon_{\mathsf{F}}$. If $\zeta = 0$, then M sends $\perp$ to R. And if $\zeta = 1$, then M submits (m^*, σ^*) as a forgery. This is done in time T_{F} in order to correctly simulate a forger.

8. Obviously, this constitutes a valid forgery. R will then return $(z_1, \ldots, z_t)$ with probability ε_{R}, which M submits as its solution against t-OMDL.

[5] Here we use that Th-$\mathsf{BLS}_{\mathsf{DKG}}$ is a signature scheme with unique signatures.

In the following, we analyze M's success probability in breaking $(t+1)$-OMDL as in the spirit of [KK18]. We denote by $\mathcal{V}$ the set of all sequences of indices such that the corresponding signature queries are correctly answered by R in time $\leq T_{\mathsf{R}}$. Obviously, if $(m_1, \ldots, m_j) \in \mathcal{V}$, then also $(m_1, \ldots, m_{j-1}) \in \mathcal{V}$. Consider (as a thought-experiment) an unbounded forger $\mathsf{F} = \mathsf{F}_{unb}$ that makes hash queries to messages as F_{sim}, signature queries to the messages from $\mathcal{M}_2$, and outputs a valid forgery σ^* on m^* with probability ε_{F}. By assertion of the theorem, reduction R has to work even against such an unbounded forger and outputs $(z_1, \ldots, z_{t+1})$ with probability at least ε_{R}. After the rewind, the view of R when interacting with F_{sim} is indistinguishable from its view when interacting with F_{unb} unless $\mathcal{M}_1 \not\subseteq \mathcal{V}$ and $\mathcal{M}_2 \subseteq \mathcal{V}$. The first condition means that R does not answer all the signing queries before the rewind correctly, while the second condition means that R does answer all the signing queries after the rewind correctly. In this case, F_{sim} would not output a valid signature, while F_{unb} would output a valid signature. This argument relies on the fact that our threshold signature scheme has unique signatures. Let $\vec{z} = (z_1, \ldots, z_{t+1})$ be the solution to the OMDL instance. This discussion yields

$$\begin{aligned} &|\Pr[R^{\mathsf{F}_{sim}}(par, \xi) = \vec{z}] - \Pr[R^{\mathsf{F}_{unb}}(par, \xi) = \vec{z}]| \\ &\leq \varepsilon_{\mathsf{F}} \cdot \Pr[\mathcal{M}_1 \not\subseteq \mathcal{V} \wedge \mathcal{M}_2 \subseteq \mathcal{V}]. \end{aligned}$$

We have the following lemma by Coron [Cor02], Appendix D.

Lemma 3.5.11. *Let $\mathcal{V}$ be a set of sequences of at most q_s integers from $\mathcal{M}$ with the property that for any $(m_1, \ldots, m_j) \in \mathcal{V}$, it is also $(m_1, \ldots, m_{j-1}) \in \mathcal{V}$. Then the following identity holds*

$$\Pr[(m_1, \ldots, m_{q_s}) \in \mathcal{V} \wedge (m_1, \ldots, m_{i-1}, m^*) \notin \mathcal{V}] \leq \frac{1}{eq_s},$$

where the probability is taken over the randomness of sampling index $i \leftarrow [q_s]$ and tuple of messages $(m_1, \ldots, m_{q_s}, m^*) \leftarrow \mathcal{M}^{q_s+1}$.

The probability that $m_i \neq m^*$ for all $i \in [q_s]$ is bounded by $(1 - q_s/|\mathcal{M}|)$. This allows us to use the above lemma and get

$$\Pr[\mathcal{M}_1 \not\subseteq \mathcal{V} \wedge \mathcal{M}_2 \subseteq \mathcal{V}] \leq \frac{1}{eq_s} \left(1 - \frac{q_s}{|\mathcal{M}|}\right)^{-1}.$$

Overall, we get

$$\begin{aligned} \varepsilon_{\mathsf{M}} &= \Pr[R^{\mathsf{F}_{sim}}(par, \xi) = \vec{z}] \\ &\geq \Pr[R^{\mathsf{F}_{unb}}(par, \xi) = \vec{z}] - \varepsilon_{\mathsf{F}} \cdot \frac{1}{eq_s} \left(1 - \frac{q_s}{|\mathcal{M}|}\right)^{-1} \\ &\geq \varepsilon_{\mathsf{R}} - \varepsilon_{\mathsf{F}} \cdot \frac{2}{eq_s}, \end{aligned}$$

where the last inequality comes from $|\mathcal{M}| \geq 2q_s$, and therefore $\left(1 - \frac{q_s}{|\mathcal{M}|}\right)^{-1} \leq 2$. Finally, the bound on M's time comes from the rewind, which is essentially the same as running both R and F_{sim} twice. □

Finally, we also show that any generic reduction that embeds the $(t+1)$-element OMDL instance "fully" into the polynomial $f \in \mathbb{Z}_p[X]$ of degree t that determines the secret key shares is indeed naive. By "fully", we simply mean that the matrix Q has full rank $t+1$. We have the following theorem.

Theorem 3.5.12. *Let $1 \leq r \leq t+1$, and let* DKG *have (t,k)-oracle-aided algebraic security. Further, let* R *be a generic reduction such that for every algebraic forger F_{alg} that breaks* Th-BLS$_{\mathsf{DKG}}$, $\mathsf{R}^{\mathsf{F}_{alg}}$ *is an algorithm that breaks r-*OMDL*. Let the $(n \times r)$-matrix Q, formed by the algebraic coefficients of the public key shares, have rank r. Then any set of $t+1$ row vectors of Q generates a space of dimension r.*

Proof. Let $\xi = (g^{z_1}, \ldots, g^{z_r}) \in \mathbb{G}^r$ be the given OMDL instance of degree r. And let $f(X) = a_0 + a_1 X + \ldots + a_t X^t \in \mathbb{Z}_p[X]$ be the degree t polynomial that determines the secret key shares. As R is an algebraic reduction, at the end of the key generation phase, it returns a vector of public key shares $(g^{x_1}, \ldots, g^{x_n})$ together with a representation $(\hat{a}_i, a_{i,1}, \ldots, a_{i,r})$ for all $i \in [n]$ such that

$$g^{x_i} = g^{\hat{a}_i} \cdot \prod_{j=1}^{r} (g^{z_j})^{a_{i,j}} = g^{\hat{a}_i} \cdot (g^{z_1})^{a_{i,1}} \cdot \ldots \cdot (g^{z_r})^{a_{i,r}} .$$

Let Q be the $(n \times r)$-matrix over the field $\mathbb{Z}_p$ formed by these algebraic coefficients

$$Q := \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,r} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,r} \\ \vdots & \vdots & & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,r} \end{pmatrix}.$$

By Lagrange interpolation in the exponent though the public key shares, the values g^{a_i} also come with a representation $(\hat{d}_i, d_{i,1}, \ldots, d_{i,r})$ for all $i \in [0,t]$ such that

$$g^{a_i} = g^{\hat{d}_i} \cdot \prod_{j=1}^{r} (g^{z_j})^{d_{i,j}} = g^{\hat{d}_i} \cdot (g^{z_1})^{d_{i,1}} \cdot \ldots \cdot (g^{z_r})^{d_{i,r}} .$$

Consider the $(t+1) \times r$-matrix over the field $\mathbb{Z}_p$

$$D := \begin{pmatrix} d_{0,1} & d_{0,2} & \cdots & d_{0,r} \\ d_{1,1} & d_{1,2} & \cdots & d_{1,r} \\ \vdots & \vdots & & \vdots \\ d_{t,1} & d_{t,2} & \cdots & d_{t,r} \end{pmatrix},$$

and let $V := V(1, \ldots, n)$ be the $n \times (t+1)$-Vandermonde matrix for the numbers $1, \ldots, n$. By construction of Q, V, and D, we have the identity $Qz = VDz$ and thus $(Q - VD) \cdot z = 0$. Since R is a generic reduction, there is at least one component of the vector $z = (z_1, \ldots, z_r)$ that looks uniformly random for R. As a consequence of the Schwartz–Zippel lemma, it follows $Q - VD = 0$ with overwhelming probability (to be precise, with probability at least $1 - 1/p$). Therefore, we may assume that $Q = VD$. Since $\mathsf{rank}(Q) = \mathsf{rank}(VD) \leq \min\{\mathsf{rank}(V), \mathsf{rank}(D)\}$ and V has full rank $t+1$, it follows $r = \mathsf{rank}(Q) \leq \mathsf{rank}(D) \leq r$ so that $\mathsf{rank}(D) = r$. Now let $S \subset [n]$ be

some $(t+1)$-element subset of $\{1,\ldots,n\}$. We let Q_S (and V_S likewise) be the $(t+1)\times r$-matrix formed by the row vectors of Q specified by S. The equation $Q = VD$ translates into $Q_S = V_S D$, where V_S is a $(t+1)\times(t+1)$-matrix of rank $t+1$. Finally, Sylvester's rank inequality yields

$$\begin{aligned}&(t+1)+r-(t+1)\\&=\mathsf{rank}(V_S)+\mathsf{rank}(D)-(t+1)\\&\leq \mathsf{rank}(Q_S)\leq \min\{t+1,r\}=r,\end{aligned}$$

that is $r \leq \mathsf{rank}(Q_S)$ and thus $r = \mathsf{rank}(Q_S)$. This means that any set S of $t+1$ row vectors of Q generates a space of dimension r. □

Appendix 3A: DKG Protocols

We describe the DKG protocols relevant for this chapter.

3.5.5 TD-DKG Protocol

This is the usual key generation protocol where a trusted dealer shares a secret among n parties $P_1,\ldots,P_n$ via some secret sharing scheme. We treat this protocol as an instance of a DKG for the purpose of our results being consistent.

Protocol TD-DKG:

1. The trusted dealer TD chooses a random polynomial $f \in \mathbb{Z}_p[X]$ of degree t:

   $$f(X) := a_0 + a_1 X + \ldots + a_t X^t \in \mathbb{Z}_p[X].$$

   For all $k \in [0,t]$, TD broadcasts the elements $A_i = g^{a_i}$.

2. For all $i \in [n]$, TD computes the public key share $pk_i = g^{f(i)}$. It then secretly sends $sk_i = f(i)$, the vector $(pk_1,\ldots,pk_n)$, and $y = g^{a_0}$ to party P_i.

3. Each party P_i verifies the share he received from the dealer by checking the identity

   $$g^{sk_i} = \prod_{k=0}^{t} A_i^{i^k}. \tag{1}$$

   If the check fails, P_i broadcasts a complaint against TD.

4. For every complaint from party P_i, the dealer reveals the share $sk_i = f(i)$ matching (1).

5. The dealer is disqualified if either

   - he received more than t complaints in Step 3, or
   - he answered a complaint in Step 4 with values that do not match (1).

   Upon TD being disqualified the protocol terminates and the DKG was unsuccessful.

6. If TD is non-disqualified, every party sets his share of the secret key as $x_i = sk_i$. The vector of public key shares is set as $(pk_1,\ldots,pk_n)$ and the public key is set as $pk = y$. The secret key x itself is not computed by any party or sent by TD, but it is equal to $x = f(0)$.

3.5.6 JF-DKG Protocol

This is Pedersen's traditional DKG in which it shares a secret among n parties $P_1 \ldots, P_n$ via n parallel executions of Feldman's VSS scheme. This protocol can be found in [Ped92] and [Gen+07].

Protocol JF-DKG:

1. Each party P_i chooses a random polynomial f_i of degree t with coefficients in $\mathbb{Z}_p$:

$$f_i = a_{i0} + a_{i1}X + \ldots + a_{it}X^t.$$

   For all $k \in [0, t]$, P_i broadcasts $A_{ik} = g^{a_{ik}}$. Each P_i computes the polynomial shares $s_{ij} = f_i(j)$ for $j \in [n]$ and sends s_{ij} secretly to party P_j.

2. Each party P_j verifies the shares he received from the other parties by checking

$$g^{s_{ij}} = \prod_{k=0}^{t} (A_{ik})^{j^k} \tag{1}$$

   for all $i \in [n]$. If the check fails for an index i, P_j broadcasts a complaint against P_i.

3. Each party P_i who received a complaint from party P_j reveals the share s_{ij} matching (1). If any of the revealed shares fails this equation, P_i is disqualified. We define $\mathcal{Q} \subset \{1, \ldots, n\}$ as the set of non-disqualified parties.

4. The public key y is computed as $y = \prod_{i \in \mathcal{Q}} A_{i0}$. The public verification values are computed as $A_k = \prod_{i \in \mathcal{Q}} A_{ik}$ for $k \in [t]$. Each party P_j sets his secret key share as $x_j = \sum_{i \in \mathcal{Q}} s_{ij}$. The secret key x itself is not computed by any party, but is equal to $x = \sum_{i \in \mathcal{Q}} a_{i0}$.

3.5.7 New-DKG Protocol

This is Gennaro et al.'s well-known DKG protocol in which the n parties $P_1, \ldots, P_n$ take part to share a secret. This protocol can be found in [Gen+07]. For the protocol, let $\mathsf{H} : \{0,1\}^* \to \mathbb{G} \backslash \{1\}$ be a hash function (modeled as a random oracle).

Protocol New-DKG:

0. This step is done only once. Set $h = H[1]$.

1. Each party P_i performs a Pedersen-VSS of a random value z_i as a dealer:

   (a) P_i chooses two random polynomials f_i, g_i of degree t with coefficients in $\mathbb{Z}_p$:

$$f_i(X) = a_{i0} + a_{i1}X + \ldots + a_{it}X^t,$$
$$g_i(X) = b_{i0} + b_{i1}X + \ldots + b_{it}X^t.$$

   For all $k \in [0, t]$, P_i broadcasts the elements $C_{ik} = g^{a_{ik}} h^{b_{ik}}$. Let $z_i = a_{i0}$. Each P_i computes the polynomial shares $s_{ij} = f_i(j)$, $u_{ij} = g_i(j)$ for $j \in [n]$ and sends (s_{ij}, u_{ij}) secretly to party P_j.

(b) Each party P_j verifies the shares he received from the other parties by checking the identity

$$g^{s_{ij}} h^{u_{ij}} = \prod_{k=0}^{t} (C_{ik})^{j^k} \tag{1}$$

for all $i \in [n]$. If the check fails for an index i, P_j broadcasts a complaint against P_i.

(c) Each party P_i who received a complaint from party P_j reveals the shares s_{ij}, u_{ij} matching (1).

(d) Each party marks as disqualified any party that either

- received more than t complaints in Step 1(b), or
- answered a complaint in Step 1(c) with values that do not meet (1).

2. Each party then builds the set of non-disqualified parties $Q \subset \{1, \ldots, n\}$.

3. Each party P_j sets his share of the secret key as $x_j = \sum_{i \in Q} s_{ij}$ and the value $x'_j = \sum_{i \in Q} u_{ij}$. The distributed secret key x itself is not computed by any party, but it is equal to $x = \sum_{i \in Q} z_i$.

4. Each party P_i, $i \in Q$, exposes $y_i = g^{z_i}$ via Feldman-VSS:

(a) Each party P_i, $i \in Q$, broadcasts the elements $A_{ik} = g^{a_{ik}}$ for all $k \in [0, t]$.

(b) Each party P_j verifies the values broadcast by the other parties in Q by checking the identity

$$g^{s_{ij}} = \prod_{k=0}^{t} (A_{ik})^{j^k} \tag{2}$$

for all $i \in Q$. If the check fails for an index i, P_j complains against P_i by broadcasting the values s_{ij}, u_{ij} that satisfy (1) but not (2).

(c) For parties P_i who receive at least one valid complaint, the other parties run the reconstruction phase of Pedersen-VSS to compute z_i, f_i and A_{ik} for $k \in [0, t]$ in the clear. Let $y_i = g^{z_i}$ for $i \in Q$. The public value y is computed as $y = \prod_{i \in Q} y_i$.

Remark 3.5.3. Note that the New-DKG protocol requires the additional element $h \in \mathbb{G}$ in order to run Pedersen's verifiable secret sharing (VSS). One possibility is to assume that $h = H[1]$ is made public as part of the global parameters. Another possibility is to have h generated jointly by the parties in a preliminary phase of the protocol, e.g., by using JF-DKG.

4

Aggregatable PVSS and Its Application to Distributed Randomness Beacons

Details on this Chapter

This chapter is based on the conference publication [BL23a] and its full version [BL23b]. I contributed to it as the main author. The original publication has been partially reordered, and editorial improvements have been made throughout the text.

Publicly Verifiable Secret Sharing (PVSS) is a fundamental primitive that allows to share a secret S among n parties via a publicly verifiable transcript T. Existing (efficient) PVSS are only proven secure against *static adversaries* who must choose who to corrupt ahead of a protocol execution. As a result, any protocol (e.g., a distributed randomness beacon) that builds on top of such a PVSS scheme inherits this limitation. To overcome this barrier, we revisit the security of PVSS under *adaptive corruptions* and show that, surprisingly, many protocols from the literature already achieve it in a meaningful way:

- We propose a new security definition for *aggregatable PVSS*, i.e., schemes that allow to homomorphically combine multiple transcripts into one compact aggregate transcript AT that shares the sum of their individual secrets. Our notion captures that if the secret shared by AT contains at least one contribution from an honestly generated transcript, it should not be predictable. We then prove that several existing schemes satisfy this notion against adaptive corruptions in the algebraic group model (AGM).

- To motivate our new notion, we show that it implies the adaptive security of two recent random beacon protocols, SPURT (IEEE S&P 2022) and OptRand (NDSS 2023), who build on top of aggregatable PVSS schemes satisfying our notion of unpredictability. For a security parameter λ, our result improves the communication complexity of the best known adaptively secure random beacon protocols to $O(\lambda n^2)$ for synchronous networks with $t < n/2$ corruptions and partially synchronous networks with $t < n/3$ corruptions.

4.1 Introduction

In *publicly verifiable secret sharing* (PVSS) [Sta96], a dealer D shares a secret S among n parties $P_1, \ldots, P_n$ by broadcasting a transcript T consisting of encrypted shares $\vec{E} = (E_1, \ldots, E_n)$ along with a proof π. Any subset of $t + 1$ parties can pool their (decrypted) shares to reconstruct S, whereas t or fewer shares give no information about S. Using π, anyone can efficiently determine whether the shares in $\vec{E}$ can be decrypted by the appropriate parties and indeed yield a sharing of S. This sets PVSS apart from the more common notion of *verifiable secret sharing* (VSS) [Cho+85a], which typically requires expensive communication among parties to ensure that the sharing is correct. As such, PVSS is an important building block in high-performance distributed protocols that aim to minimize communication. Recent examples of such protocols include distributed randomness beacons [Sch+20; Bha+21; Das+22a; Bha+23] and distributed key generation (DKG) [Gur+21b; Abr+21a]. In these types of protocols, one typically assumes a malicious adversary who can corrupt some $t < n$ of the parties and make them behave arbitrarily. Most of the literature considers a *static adversary* who must commit to its corruptions before the protocol execution begins. However, a recent trend in this area has been toward considering a stronger *adaptive adversary* who can corrupt parties dynamically over the course of the protocol execution [Bha+21; BL22a; Abr+22a; CKM23a].

Unfortunately, protocol designers currently face the following limitation: existing (efficient) PVSS schemes are only proven secure with respect to static corruptions. Hence, adaptively secure protocols must often resort to less efficient (but adaptively secure) alternatives such as VSS. To ameliorate this unsatisfactory state of affairs, we ask the following question: *Are there efficient and adaptively secure PVSS protocols?*

4.1.1 Our Contributions

In this work, we provide a nuanced answer to the above question. Our contributions are summarized in the following.

New Security Notions for Aggregatable PVSS. One particularly useful feature supported by some PVSS schemes is the ability to homomorphically *aggregate* sharings. In more detail, suppose that we are given $t + 1$ PVSS transcripts $T_1, \ldots T_{t+1}$ sharing respective secrets $S_1, \ldots, S_{t+1}$. Then aggregation allows to efficiently combine them into a compact transcript T sharing $S = \sum_i S_i$.

Aggregatable PVSS has served as an indispensible building block in many higher-order constructions, most notably leader-based randomness beacons [Bha+21; Das+22a; Bha+23]. In such constructions, a designated leader L aggregates PVSS transcripts of different parties and commits them to consensus. To ensure that a malicious leader cannot propose a self-chosen value, T should prove that at least one honest party has contributed to the combined secret S. This, intuitively, ensures that S remains unpredictable. We observe that while several constructions from the literature already have this property, it is usually proven as part of a security proof for a broader system (see, e.g., the recent work of Bhat et al. [Bha+23]). Given the importance of aggregated PVSS as a modular building block, we believe that it is useful to capture the above unpredictability property in a new standalone security notion that we call *aggregated unpredictability*. While aggregatable unpredictability does not ensure full secrecy in the sense of previous indistinguishability-based notions [HV09], we show that it is sufficient to prove the security of recent distributed randomness beacons (see below).

We prove that several existing aggregatable PVSS protocols achieve our notion of unpredictability against *adaptive corruptions* in the algebraic group model (AGM). Here, we rely on techniques from the recent work of Bacho and Loss [BL22a], who gave the first adaptive security analysis of the threshold BLS signature [BLS01; Bol03]. Our proof faces many additional challenges compared to theirs that we elaborate on in more detail in our technical overview. In particular, our proofs are complicated by the fact that the adversary obtains partial information about the secret S from the encrypted shares $\vec{E}$. Therefore, it must be argued that it cannot use this information to cancel out honest parties' contribution to the aggregated secret S and render it predictable.

Applications to Randomness Beacons. We conclude by showing that our newly introduced notion of unpredictability for PVSS suffices to prove the security of two recent distributed randomness beacon protocols, SPURT [Das+22a] and OptRand [Bha+23]. Recall that the objective of a distributed random beacon protocol is for n parties $P_1, \ldots, P_n$ to agree on an a sequence of (computationally) uniformly random values $\sigma_1, \sigma_2, \ldots$. The crucial property of a randomness beacon is that an adversary controlling some minority of $t < n$ parties can neither *predict* these values too early before they are output nor *bias* them. While both SPURT and OptRand achieve these properties (under different network conditions and corruption regimes), both of them are proven secure only with respect to *static adversaries*. We observe, however, that this limitation is directly inherited from the respective (statically secure) PVSS schemes that they are built on. Hence, it is plausible that their security can be improved to the same number of adaptive corruptions if the underlying PVSS provides such security guarantees. We confirm this intuition by introducing a weak unpredictability notion for randomness beacons and showing that both SPURT and OptRand achieve it against adaptive adversaries. In our new notion, a

beacon produces values that remain unpredictable, yet possibly not uniformly distributed from the perspective of the adversary, up to a certain point before being output. However, since most beacon protocols assume the random oracle model [BR93] (ROM) anyway, it is trivial to transform an unpredictable beacon into one fully-fledged one. To do so, each party simply hashes each value that it outputs from the weak (i.e., unpredictable) beacon to obtain its final output. In this manner, one immediately obtains the first adaptively secure randomness beacons achieving $O(\lambda n^2)$ communication complexity per computed value (λ denotes a security parameter) in the synchronous regime with $t < n/2$ corruptions and in the partially synchronous regime with $t < n/3$ corruptions. Previously, adaptively secure randomness beacons in these settings relied on (more expensive) VSS [Bha+21], thus incurring at least $O(\lambda n^3)$ communication per output.

Our proofs also give a modular template which allows to infer unpredictability of leader-based beacon protocols in a black-box fashion from the unpredictability of the underlying PVSS scheme. Thus, we believe that our new security notions will be of use to the design of randomness beacons in the future.

4.2 Technical Overview

We proceed with a brief overview of our techniques. We remark that the discussion below is informal and as such does not depend on particular components of the PVSS we consider in this work. For example, non-interactive zero-knowledge proofs (NIZKs) can be implemented using Fiat-Shamir type proofs of discrete logarithm equality, pairing-based proofs, or code-based proofs, but we omit these distinctions here as they are not relevant for this high-level overview.

4.2.1 A Short Recap of PVSS and Adaptive Corruptions

To begin, we describe the common high-level idea behind many efficient PVSS schemes in the literature. Let again g and h be known generators of some cyclic group $\mathbb{G}$ of prime order p. In the sharing phase, the dealer D picks a value $\alpha \in \mathbb{Z}_p$ and computes a (t, n)-sharing of a group element $S := h^\alpha$ by interpolating a random polynomial P over $\mathbb{Z}_p$ of degree t through points $\alpha_i =: P(i)$ and computing the shares h^{α_i} for all $i = 0, \ldots, n$. In addition, D also computes the commitments $g^{\alpha_0}, \ldots, g^{\alpha_n}$. It then computes ciphertexts $E_i := \mathsf{Enc}(pk_i, h^{\alpha_i})$ for all i and shares the vector $\vec{E}$ of encrypted shares together with the proof π consisting of the values $g^{\alpha_0}, \ldots, g^{\alpha_n}$ and NIZKs proving that $\vec{E}$ is an encryption of values h^{α_i}. This can be achieved by using the values $g^{\alpha_0}, \ldots, g^{\alpha_n}$. Using the NIZKs and these values, anyone is convinced that $\vec{E}$ provides a correct sharing of S. To reconstruct, party i decrypts its share via $h^{\alpha_i} = \mathsf{Dec}(sk_i, \vec{E}_i)$ and sends this value to all parties. Upon receiving $t + 1$ shares $h^{\alpha_{k_1}}, \ldots, h^{\alpha_{k_{t+1}}}$, the secret can be recovered via Lagrange interpolation in the exponent of h.

A Common Proof Strategy. The main difficulty in the context of adaptive corruptions that our simulator Sim has to overcome is to balance two seemingly mutually exclusive tasks. First, Sim has to simulate the security experiment without knowing the values sk_i and the secret shares of all of the honest parties. If, instead, it knew all these values, the adversary's final output would be useless to Sim with regards to breaking the hardness assumption underlying the security of the PVSS scheme. Also, it is clear that guessing the subset of eventually corrupted parties is out of the question, as it would lead to a security loss exponential in t and n. On the other hand, Sim must be able to provide the values sk_i along with the share of party i, upon i becoming adaptively

corrupted during simulation. We stress that this issue does not occur when corruptions are static, as Sim knows all the corrupted parties upfront. This allows Sim to interpolate a properly distributed polynomial through a secret S as well as the corrupted parties' shares, without actually knowing them. This simulation strategy is well-known from the literature on distributed key generation protocols [Gen+99; Can+99; JL00; Gen+07]. Unfortunately, it is not applicable to a setting with adaptive corruptions, which, instead, requires different arguments. As such, schemes commonly resort to heavy machinery such as non-committing encryption [JL00] to attain adaptive security.

Our techniques for addressing this problem are inspired by the recent work of Bacho and Loss [BL22a] who proved adaptive security of the (symmetric) threshold BLS signature scheme in the AGM under the OMDL assumption. Loosely speaking, the OMDL assumption of degree k asserts that it is difficult to return the discrete logarithms $z_1, \ldots, z_k$ of k discrete logarithm challenges $g^{z_1}, \ldots, g^{z_k}$ when given $(k-1)$-time access to a (perfect) discrete logarithm oracle $\mathsf{DL}_g()$. In more detail, on input a group element $\xi \in \mathbb{G}$, $\mathsf{DL}_g()$ returns its discrete logarithm $z \in \mathbb{Z}_p$ to base g (where p is the prime order of $\mathbb{G}$). The key insight of their work is the construction of a simulator Sim which reduces from this assumption and hence can leverage the oracle $\mathsf{DL}_g()$ to simulate adaptive corruptions. Follow-up works have leveraged similar techniques to obtain adaptively secure asynchronous DKG [Abr+22a] and threshold Schnorr signatures [CKM23a].

4.2.2 Challenges in the Context of PVSS

As explained above, aggregatable PVSS are typically composed of three main components: an encryption scheme, a commitment scheme, and a NIZK. This combination opens up many different vectors of attack that add unique challenges to our security proofs when compared to structurally simpler primitives such as signatures and VSS. Intuitively, there are three ways in which an attacker can learn the secret S, each corresponding to one of the three aforementioned components: (1) it can break security of the encryption scheme Enc to learn a $(t+1)st$ share h^{α_i}, (2) it can find the discrete logarithm a of h to base g and compute h^{α_0} from the commitment (g^{α_0}) via $(g^{\alpha_0})^a$, (3) it can pick up to t transcripts $T_1, \ldots, T_t$ dependent on a single honest transcript T^* in such a way that their aggregate becomes entirely independent of T^*. In particular, it could choose them in such a way that T^*'s contribution is cancelled out entirely, in which case the secret shared by the aggregate transcript AT is no longer unpredictable. Intuitively, this lane of attack should be prevented by the NIZK component of the scheme, as it forces the attacker to know the discrete logarithms of the secrets.

Following this high-level template, our proof broadly distinguishes multiple cases by providing appropriate simulations of the unpredictability experiment to the adversary in each of which the OMDL instance is embedded into different components of the scheme. The main difficulty of our proof is to balance these simulation strategies without the adversary being able to tell them apart.

Additional Issues with Asymmetric Groups. While we could prove all of our claims for symmetric variants of the pairing-based PVSS schemes we consider directly under the OMDL assumption, we insist on proving these schemes directly in their original and more performant versions over asymmetric pairing groups. Because of this, the OMDL assumption unfortunately turns out to be insufficient for our purposes. To see the issue, note that PVSS schemes over

asymmetric pairing groups typically share the secret in *both source groups* $\mathbb{G}_1$ *and* $\mathbb{G}_2$. As a consequence, our reduction would have to supply the discrete logarithm challenges in both groups as $g_1^{x_1}, \ldots, g_1^{x_k}, g_2^{x_1}, \ldots, g_2^{x_k}$, where g_1 and g_2 are the groups' respective generators. To remedy this issue, we introduce a natural extension of OMDL to asymmetric pairing groups, in which the adversary obtains all of these generators and can query the oracle DL_{g_1} for elements in $\mathbb{G}_1$. We refer to this assumption as *co-OMDL* and provide a rigorous proof of its hardness in the generic group model (GGM). Our proof follows along the lines of Bauer et al. [BFP21], but requires a new mathematical lemma due to the higher degree of polynomials in the exponents of target group elements.

We believe that, similar to established asymmetric hardness assumptions such as SXDH [ACM05] or co-CDH [BLS01], co-OMDL has many applications to schemes based on asymmetric pairing groups and, as such, is of independent interest. As an example, we refer again to the work of Bacho and Loss who prove adaptive security for the (symmetric) threshold BLS from OMDL. As we later explain in more detail in Appendix 4.6.2, their proof faces similar issues in the asymmetric setting that would also require co-OMDL.

4.2.3 More on Related Work

We give an overview of the literature on PVSS and how our work fits in. We also briefly discuss some limitations of our work.

Publicly Verifiable Secret Sharing. The idea of publicly verifiable secret sharing (PVSS) was first formally stated in the seminal work of Stadler [Sta96], although Stadler notes that it already was implicitly conceived in the work of Chor et al. [Cho+85c]. Over the years, many improved schemes have been proposed. The common idea behind many of these schemes is the following. The dealer samples a polynomial $f \in \mathbb{Z}_p[X]$ of degree t uniformly at random and commits to it via Feldman commitments [Fel87a] (i.e., it commits to the $t+1$ coefficients of f). The dealer also provides encryptions of shares to an (t, n)-Shamir secret sharing of f. The Feldman commitments are used by non-dealer parties to compute commitments to the shares $f(i)$ that are proven via zero-knowledge proofs to correspond to the encrypted shares.

Stadler realized these proofs via the Fiat-Shamir heuristics in the random oracle model. Security of the scheme is reduced from the Decisional Diffie-Hellman (DDH) assumption. Schoenmakers [Sch99] gives a more efficient variant of Stadler's construction, in which the security of the scheme is reduced from the computational Diffie-Hellman (CDH) assumption. Ruiz and Villar [RV05] and Jhanwar et al. [JVSN14] gave standard model constructions which replace random oracle model proofs through checks based on Paillier encryption [Pai99]. The security of both these schemes is reduced from the Decisional Composite Residuosity (DCR) assumption. Heidarvand and Villar [HV09] and Jhanwar [Jha11] proposed alternative pairing-based PVSS constructions in the plain model with security under the Decisional Bilinear Square (DBS) assumption and the multi-sequence of exponents Diffie-Hellman (MSE-DDH) assumption. A significant drawback of these schemes is that parties must each compute $O(nt)$ exponentiations to verify the transcript. Since one generally assumes $t \in O(n)$, this results in high computation cost of $O(n^2)$, limiting their practicality.

This barrier in quadratic computation cost was first overcome by SCRAPE, an elegant scheme proposed by Cascudo and David [CD17]. The idea of their scheme is the following. Instead of committing to the coefficients of f, the dealer directly commits to the polynomial

evaluations $f(i)$ by publishing $g^{f(i)}$. With the help of linear error correcting codes (and their dual codes), parties can verify with high probability that the commitments published by the dealer actually correspond to a polynomial of degree t. In this manner, the total computation cost reduces to $O(n)$ exponentiations. The authors provide two construction based on the underlying model. In the random oracle model, the proofs are realized via NIZKs and security of the scheme is reduced from the DDH assumption. In the standard model, the authors use pairings to realize these proofs and security of the scheme is reduced from the Decisional Bilinear Square (DBS) assumption. This construction has inspired several followups.

In the context of randomness beacons, Das et al. [Das+22a] propose SPURT, which gives a variant of SCRAPE that relies on the standard Decisional Bilinear Diffie-Hellman (DBDH) assumption and achieves similar performance to SCRAPE. The work of Gurkan et al. [Gur+21b] also gives a variant of the pairing-based SCRAPE and uses it as a building block to design a DKG protocol. To support efficient aggregation of PVSS transcripts, their construction relies on signatures of knowledge and is proven secure under the Symmetric External Diffie-Hellman (SXDH) assumption for Type 3 pairings.

Security of Publicly Verifiable Secret Sharing. The literature on PVSS has considered two main security notions, both of which capture the notion of indistinguishability of secrets. These notions were first formally defined by Heidarvand and Villar in [HV09]. In the weaker notion of *IND1-secrecy*, the adversary cannot distinguish between the sharings of two secrets S_1, S_2 chosen uniformly at random by the challenger. In the stronger notion of *IND2-secrecy*, the adversary has the additional power to choose the two secrets S_1, S_2 by itself. As already pointed out by Heidarvand and Villar, there is a generic transformation from an IND1-secure PVSS scheme to an IND2-secure PVSS scheme. Omitting some details, the transform uses an IND1-secret PVSS to share a uniform key K which in turn is used to encrypt a secret S.

In the following, we compare these notions to our notion of unpredictability. Intuitively, IND-secrecy says that an adversary cannot learn any information about the secret S shared in the distribution protocol. Therefore, proofs achieving this security notion have to provide simulator Sim that on input a uniformly random S simulates protocol execution in which the secret S is shared. As discussed above, it is unknown how to instantiate Sim efficiently for adaptive corruption without modifying the scheme. Our notion of (aggregated) unpredictability obviates the need for this type of simulation. This is because unpredictability allows the adversary to obtain partial information about the secret S, with the only condition that it cannot *fully recover* the secret.

4.2.4 Outline of this Chapter

The rest of this chapter is structured as follows. In Section 4.3, we define the preliminaries that are only relevant for this chapter. In Section 4.4, we give new definitions for standard PVSS schemes, and show security of several PVSS schemes and distributed randomness beacons from the literature. In Section 4.5, we give new definitions for aggregatable PVSS schemes, and show security of several aggregatable PVSS schemes from the literature. In Section 4.6, we show security of the state-of-the-art distributed randomness beacons OptRand and SPURT. In Appendix 4.6.2, we discuss related work on distributed randomness beacons. In Appendix 4.6.2, we give a proof of hardness in the generic group model for our new co-OMDL problem.

4.3 Preliminaries for this Chapter

In addition to the general preliminaries in Chapter 2, we introduce here preliminaries that are only relevant for this chapter.

General Notation. For this chapter, we define $\mathcal{LC}$ to be the Reed-Solomon code over $\mathbb{Z}_p$ of length n and dimension $t+1$ as

$$\mathcal{LC} := \{(f(1), \ldots, f(n)) \mid f \in \mathbb{Z}_p[X],\ \deg(f) \leq t\}.$$

Its dual code $\mathcal{LC}^\perp$ is defined as

$$\mathcal{LC}^\perp := \{(\vartheta_1 r(1), \ldots, \vartheta_n r(n)) \mid r(X) \in \mathbb{Z}_p[X],\ \deg(r) \leq n-t\},$$

with the coefficients $\vartheta_i := \prod_{j \in [n]\setminus\{i\}} 1/(i-j)$. Equivalently, $\mathcal{LC}^\perp$ is the vector space consisting of all $c^\perp \in \mathbb{Z}_p^n$ that are orthogonal to all of $\mathcal{LC}$ (i.e., $\langle c^\perp, c\rangle = 0$ for all $c \in \mathcal{LC}$ where $\langle\cdot,\cdot\rangle$ is the standard inner product on $\mathbb{Z}_p^n$).

Setup and Adversarial Model. In this chapter, we will make direct use of the established public key infrastructure (PKI) between parties. To recall, each party P_i has a public-secret key pair (pk_i, sk_i) for a public key encryption scheme, where pk_i is known to all parties. And we also assume that these pairs implicitly include key pairs for a digital signature scheme. Importantly, these keys are generated by parties locally, and malicious parties may choose their keys arbitrarily. The adversary is Byzantine, strongly adaptive, and can corrupt up to $t < n$ parties.

Idealized Models and Computational Assumptions. We assume the random oracle model and the algebraic group model. Both models are defined in Chapter 2. We will work with the one-more discrete logarithm assumption and a newly introduced assumption called *co-one-more discrete logarithm (co-OMDL)*, defined in Section 4.5. We also give a proof of its hardness in the generic group model, in Appendix 4.6.2.

Cryptographic Groups. Let λ be the security parameter. Throughout, we assume that global system parameters par are fixed and known to all parties. Depending on the setting, we either assume that $par = (\mathbb{G}, g, h, p)$ defines a cyclic group $\mathbb{G}$ of prime order p with generators g, h, or that $par = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \hat{g}, h, e)$ defines a triple of groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of prime order p such that $g, \hat{g} \in \mathbb{G}_1$, $h \in \mathbb{G}_2$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an asymmetric pairing of Type 3. That is, there is no efficiently computable isomomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$ and vice versa. For concrete choices, we will assume $\lambda = 128$ and that $\mathbb{G}_1, \mathbb{G}_2$ are instantiated with 256-bit elliptic curves.

4.4 Warm-Up: PVSS Schemes and Plain Unpredictability

In this warm-up section, we provide a formal definition for a standard PVSS scheme and propose a new security notion for it called *(plain) unpredictability*. Additionally, we prove the well-known Schoenmakers' PVSS scheme [Sch99] secure under this new security notion and show several distributed randomness beacons from the literature (that use this PVSS) adaptively secure.

Publicly Verifiable Secret Sharing (PVSS). In a VSS scheme, a dealer distributes shares of a secret among a group of parties such that it can be reconstructed only if a threshold of these parties collaborate. In a PVSS scheme, any third party can verify the correctness of the sharing,

thus avoiding the need for a complaint phase as in VSS schemes. In the following, we define a (non-interactive) PVSS scheme.

Definition 4.4.1 (PVSS Scheme)**.** Let $\hat{\mathbb{G}}$ be a cyclic group of prime order p specified by par. A (t, n)*-threshold PVSS scheme over* $\hat{\mathbb{G}}$ is a tuple of algorithms $\mathsf{PVSS} = (\mathsf{Keys}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Dist}, \mathsf{Ver}, \mathsf{Rec})$ with the following properties:

- Keys: The randomized *key generation algorithm* takes as input system parameters par and an identity index $i \in [n]$. It outputs a public-secret key pair (pk_i, sk_i).
- Enc: The randomized *encryption algorithm* takes as input a public key pk_i and a message m. It outputs a ciphertext c.
- Dec: The deterministic *decryption algorithm* takes as input a secret key sk_i and a ciphertext c. It outputs a message m (optionally with a proof of correct decryption). We require that for all messages m,

$$\Pr[\mathsf{Dec}_{sk_i}(\mathsf{Enc}_{pk_i}(m)) = m] = 1.$$

- Dist: The randomized *secret sharing algorithm* takes as input a list of n public keys $pk_1, \ldots, pk_n$. It outputs a vector of encrypted shares $\vec{E} = (\mathsf{Enc}_{pk_1}(S_1), \ldots, \mathsf{Enc}_{pk_n}(S_n))$ and a proof π, where $S_1, \ldots, S_n$ are shares of a secret $S \in \hat{\mathbb{G}}$. We refer to $T := (\vec{E}, \pi)$ as a *PVSS transcript*.
- Ver: The deterministic *verification algorithm* takes as input public keys $pk_1, \ldots, pk_n$, and a PVSS transcript $T = (\vec{E}, \pi)$. It outputs 1 (accept) or 0 (reject). In the first case we call the transcript T *valid* (relative to $pk_1, \ldots, pk_n$); otherwise we call it *invalid*.
- Rec: The deterministic *reconstruction algorithm* takes as input $t + 1$ shares $S_1, \ldots, S_{t+1}$. It outputs a reconstructed secret $S \in \hat{\mathbb{G}}$. *In case* Rec *gets more than* $t + 1$ *shares as input, it takes the first lexicographical* $t + 1$.

When the parameters t, n, and $\hat{\mathbb{G}}$ are clear from the context, we will sometimes refer to a (t, n)-threshold PVSS scheme over $\hat{\mathbb{G}}$ simply as a PVSS scheme. Next, we define correctness and verifiability notions for a PVSS scheme:

Definition 4.4.2 (Correctness and Verifiability)**.** Let $\mathsf{PVSS} = (\mathsf{Keys}, \mathsf{Dist}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ver}, \mathsf{Rec})$ be a (t, n)-threshold PVSS scheme over $\hat{\mathbb{G}}$. We define the following correctness and verifiability notions for PVSS.

- *Correctness (of PVSS).* We say that PVSS is *correct* if for all keys $(pk_1, sk_1), \ldots, (pk_n, sk_n) \in \mathsf{Keys}(par)$,

$$\Pr[\mathsf{Ver}(\{pk_i\}_i, \mathsf{Dist}(\{pk_i\}_i) = 1] = 1.$$

- *Public Verifiability (of Transcripts).* We say that PVSS is *publicly verifiable* if for all $(pk_1, sk_1), \ldots, (pk_n, sk_n) \in \mathsf{Keys}(par)$ and all $(\vec{E}, \pi)$ s.t. $\mathsf{Ver}(\{pk_i\}_i, (\vec{E}, \pi)) = 1$, there exists a unique $S \in \hat{\mathbb{G}}$ s.t.

$$\mathsf{Rec}(\{\mathsf{Dec}_{sk_i}(\vec{E}_i)\}_{i \in \mathcal{I}}) = S \quad \forall \mathcal{I} \subset [n],\ |\mathcal{I}| = t + 1.$$

We introduce a new security notion for PVSS schemes. The notion of *unpredictability* prohibits an adversary controlling t parties from learning the secret by observing a transcript. This models a passive adversary who can observe distributions of transcripts, but does not contribute itself to the final secret. The notion of *aggregated unpredictability* to be introduced later is a non-malleability kind of property specifically for aggregatable PVSS schemes. It prohibits an adversary controlling t parties from learning the secret of an aggregated transcript with at least one honest contribution, even if the adversary is allowed to contribute itself to the aggregate. This models an active adversary who can contribute to the final secret itself. We now define the notion of unpredictability formally via a security experiment.

Definition 4.4.3 (Unpredictability of PVSS Scheme)**.** Let $\mathsf{PVSS} = (\mathsf{Keys}, \mathsf{Dist}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ver}, \mathsf{Rec})$ be a (t, n)-PVSS scheme over $\hat{\mathbb{G}}$. For an algorithm A, define the *unpredictability* experiment $\mathbf{Pred}^{\mathsf{A}}_{\mathsf{PVSS},t}$ as follows:

- *Offline Phase.* For all $i \in [n]$, run Keys on input (par, i) to generate n keys $(pk_i, sk_i) \leftarrow \mathsf{Keys}(par, i)$. On input par and $\{pk_i\}_{i\in[n]}$, A returns an index set $C \subset [n]$ of initially corrupted parties along with updated public keys $\{\hat{pk}_j\}_{j\in C}$. Set $pk_j := \hat{pk}_j$ for all $j \in C$.
- *Corruption Queries.* At any point of the experiment, A may submit an index $i \in [n] \setminus C$. In this case, return the secret key sk_i and update $C = C \cup \{i\}$. *If* A *is static, it submits an index set* $C' \subset [n] \setminus C$ *at the beginning of the experiment. Return the secret keys* $\{sk_i\}_{i\in C'}$ *and update* $C := C' \cup C$.
- *Random Oracle Queries.* At any point of the experiment, A gets access to an oracle that answers queries of the following type: When A submits a query m, check if $H[m] = \perp$. If so, sample uniformly at random $H[m] \leftarrow \mathbb{Z}_p^*$ and return it. Otherwise, return $H[m]$.
- *Challenge Phase.* Run Dist on input $pk_1, \ldots, pk_n$ to obtain the challenge transcript $T = (\vec{E}, \pi)$. Then, run A on input T. *We assume* A *to be stateful.*
- *Output Determination.* Let $S = \mathsf{Rec}(\{\mathsf{Dec}_{sk_i}(\vec{E}_i)\}_{i\le t+1})$ be the secret. When A outputs $S^* \in \hat{\mathbb{G}}$, return 1 if $|C| \le t$ and $S^* = S$. Otherwise, return 0.

We say that PVSS is (ε, T, t, q_h)-*unpredictable* if for all algorithms A that run in time at most T, make at most q_h random oracle queries, $\Pr[\mathbf{Pred}^{\mathsf{A}}_{\mathsf{PVSS},t} = 1] \le \varepsilon$. Conversely, we say that A (ε, T, t, q_h)-*breaks unpredictability* of PVSS if it runs in time at most T, makes at most q_h random oracle queries, and $\Pr[\mathbf{Pred}^{\mathsf{A}}_{\mathsf{PVSS},t} = 1] > \varepsilon$.

4.4.1 Security Analysis of Schoenmakers' PVSS

In this section, we give a tight security reduction from the hardness of n-OMDL to the unpredictability of Schoenmakers' PVSS (cf. Figures 4.2 and 4.3). In this section, we specify $\mathbb{G}$ as a cyclic group of prime order p with independent generators g, h.

In the following, we provide an intuition for our proof of unpredictability. For this, we start with the observation that an adversary controlling t parties essentially has three options to successfully predict the secret $S \in \mathbb{G}$. Firstly, it learns an additional $(t + 1)$-th decryption key controlled by an honest party, in which case it can derive S from enough decryptions of secret shares. Secondly, it breaks the underlying encryption scheme directly and thus obtains an

Let $\mathbb{G}$ be a cyclic group of prime order p and independent generators $g, h \in \mathbb{G}$. The protocol takes as input a tuple (g, X, h, Y) of group elements in $\mathbb{G}$ and outputs a NIZK proof of knowledge of an $\alpha \in \mathbb{Z}_p^*$ such that $X = g^\alpha$ and $Y = h^\alpha$ holds.

1. Sample $s \leftarrow \mathbb{Z}_p^*$ and compute the challenge as $c := \mathsf{H}(X, g^s, Y, h^s) \in \mathbb{Z}_p^*$.
2. Compute the response $r := s - \alpha c \in \mathbb{Z}_p$, and output the proof $\pi := (c, r)$.
3. The proof $\pi = (c, r)$ is valid if and only if $c = \mathsf{H}(X, g^r X^c, Y, h^r Y^c)$.

Figure 4.1: NIZK proof Dleq() for equality of discrete logarithms.

additional secret share. Lastly, it finds the discrete logarithm e of the second generator $h \in \mathbb{G}$ to base g, in which case it can compute the secret $S = h^\alpha$ directly from the commitment g^α. The key idea of our reduction therefore is to embed the n-OMDL challenge ξ in the public keys $pk_1, \ldots, pk_n$ of parties, the polynomial $f \in \mathbb{Z}_p[X]$ chosen by the dealer, or the second generator $h \in \mathbb{G}$, a choice that remains hidden from the adversary. In the first case, we simulate by using the discrete logarithm oracle $\mathsf{DL}_g()$ to answer corruption queries. In the second case, we simulate by using an honest-verifier zero knowledge simulation in the random oracle to generate the NIZK proofs of correctness of the sharing. In the third case, we simulate by an honest execution of the protocol. In either case, we solve the n-OMDL challenge ξ directly from the algebraic equation that comes from the secret prediction/forgery (with its algebraic representation) by the adversary. Overall, our reduction is tight and loses only a factor of $1/4$. The running time of the reduction has a quadratic overhead.

Theorem 4.4.1. *If n-OMDL is (ε, T)-hard in the AGM, then Schoenmakers' PVSS is $(\varepsilon', T', t, q_h)$-unpredictable in the AGM+ROM, where*

$$\varepsilon \geq \frac{\varepsilon'}{4}, \quad T \leq T' + O(n^2).$$

Proof. Let A be an algebraic adversary that $(\varepsilon', T', t, q_h)$-breaks unpredictability of PVSS. In our proof, we assume that all parties are honest prior to the execution of PVSS. It is easy to adjust the proof to the case where the adversary has already corrupted some parties before the execution of the protocol. Let $C \subset \mathcal{P} = \{P_1, \ldots, P_n\}$ be the dynamically changing set of corrupt parties and $\mathcal{H} = \mathcal{P} \setminus C$ the set of honest parties. In particular, we assume that $C = \{\}$ prior to the execution of the protocol. We consider the following game between a challenger and the adversary.

Game **G**: This is the real game. The challenger runs PVSS on behalf of the honest parties and the designated dealer. In particular, it generates the system parameters $(\mathbb{G}, p, g, h)$, where $\mathbb{G}$ is a cyclic group of order p, and g and h are two independent generators of $\mathbb{G}$, and a uniformly at random chosen polynomial $P(X) = \alpha_0 + \alpha_1 X + \ldots + \alpha_t X^t \in \mathbb{Z}_p[X]$ of degree t such that $h^\alpha = h^{P(0)}$ is the secret to be shared among all parties. Furthermore, the challenger generates the public-secret key pairs $(pk_i, sk_i) = (h^{x_i}, x_i)$ of the honest parties. Whenever A decides to corrupt a party P_i, the challenger honestly returns the internal state of that party, which consists of P_i's secret key $x_i = sk_i$, and sets $C = C \cup \{P_i\}$, $\mathcal{H} = \mathcal{H} \setminus \{P_i\}$. In addition, A gets full control over party P_i. Random oracle queries m_i are answered by sampling $r_i \leftarrow \mathbb{Z}_p$ uniformly

Let $\mathbb{G}$ be a cyclic group of prime order p and independent generators $g, h \in \mathbb{G}$. Let (pk_i, sk_i) be the key pair of party P_i with $pk_i = h^{sk_i}$. The dealer P_L with key pair (pk_L, sk_L) wants to share secret h^α for an $\alpha \leftarrow \mathbb{Z}_p^*$. The Dist algorithm takes as input sk_L and public keys $pk_1, \ldots, pk_n$. It outputs the transcript $T_L := \{C_i, Y_i, \pi\}_{i\in[n]}$ defined as follows.

1. Sample $f(X) := \alpha + \alpha_1 X + \ldots + \alpha_t X^t \leftarrow \mathbb{Z}_p[X]$ of degree t.
2. Publish commitments $C_i = g^{\alpha_i} \in \mathbb{G}$ for $i \in [\![t]\!]$. Also publish encrypted shares $Y_i = pk_i^{f(i)} \in \mathbb{G}$ for $i \in [n]$. Let $X_i := \prod^{j\in[\![t]\!]} C_j^{i^j}$.
3. Compute NIZK proofs $\pi_i := \mathsf{Dleq}(g, X_i, pk_i, Y_i)$ for $i \in [n]$ using the simultaneous challenge $c := \mathsf{H}(\{X_i, g^{w_i}, Y_i, pk_i^{w_i}\}_{i\in[n]})$. Compute the responses $r_i \in \mathbb{Z}_p^*$ as $r_i := w_i - f(i)c$ for $i \in [n]$, and publish the proof $\pi := (c, r_1, \ldots, r_n)$.

The *transcript verification algorithm* Ver takes as input the public keys $pk_1, \ldots, pk_n$ (including pk_L) and transcript T_L. It outputs 1 (valid) or 0 (invalid).

4. Compute $X_i := \prod^{j\in[\![t]\!]} C_j^{i^j}$ for $i \in [n]$. Also compute $g^{w_i} = g^{r_i} X_i^c$ and $pk_i^{w_i} = y_i^{r_i} Y_i^c$ for $i \in [n]$ using $\{X_i, Y_i, pk_i, r_i, c\}_{i\in[n]}$.
5. Check that $c = \mathsf{H}(\{X_i, g^{w_i}, Y_i, pk_i^{w_i}\}_{i\in[n]})$ and the NIZK proof π verifies.
6. Output 1 if and only if all the above checks verify.

Figure 4.2: Distribution protocol Dist and transcript verification algorithm Ver of Schoenmakers' PVSS.

at random and returning $H[m_i] = r_i$. The challenger broadcasts the commitments $C_i = g^{\alpha_i}$ for all $i \in [\![t]\!]$ and the encrypted shares $Y_i = pk_i^{P(i)}$ for all $i \in [n]$ using the public keys pk_i of parties. Let $X_i = g^{P(i)}$ for all $i \in [n]$, which can be computed by Lagrange interpolation in the exponent from the commitments C_j. Additionally, the challenger broadcasts Chaum-Pedersen non-interactive zero-knowledge (NIZK) proofs π_i of knowledge of the polynomial shares $P(i)$ for all $i \in [n]$ such that $X_i = g^{P(i)}$ and $Y_i = pk_i^{P(i)}$. The common challenge c for the proof is computed as the hash $\mathsf{H} : \mathbb{G} \to \mathbb{Z}_p$ of all elements $X_i, Y_i, g^{w_i}, pk_i^{w_i}$, $i \in [n]$, where $w_i \leftarrow \mathbb{Z}_p$ is chosen uniformly at random. The NIZK π consists of c and the n responses $s_i := w_i - P(i)c$ for $i \in [n]$. At the end of the game, A outputs a secret $\sigma^* \in \mathbb{G}$.

As A is an algebraic adversary, at the end of game $\mathbf{G}$ it returns a secret σ^* together with a representation

$$(\hat{a}, \hat{b}, a_1, \ldots, a_n, b_0, \ldots, b_t, c_1, \ldots, c_n)$$

of elements in $\mathbb{Z}_p$ such that

$$\sigma^* = g^{\hat{a}} \cdot h^{\hat{b}} \cdot pk_1^{a_1} \cdot \ldots \cdot pk_n^{a_n} \cdot C_0^{b_0} \cdot \ldots \cdot C_t^{b_t} \cdot Y_1^{c_1} \cdot \ldots \cdot Y_n^{c_n}.$$

Here, the representation is split (from left to right) into powers of the generators g, h, the public keys $pk_1, \ldots, pk_n$, the polynomial commitments $C_0, \ldots, C_t$, and the encrypted shares $Y_1, \ldots, Y_n$.

On input the encrypted shares $Y_1, \ldots, Y_n$, the decryption Dec and reconstruction Rec algorithms work as follows.

1. Using sk_i, compute the secret share $S_i = h^{f(i)}$ from Y_i via extracting the root $S_i = Y_i^{1/sk_i}$. Also provide a proof of correct decryption $\theta_i := \mathsf{Dleq}(h, h^r, S_i, S_i^r)$ for an $r \leftarrow \mathbb{Z}_p^*$ sampled uniformly at random. Publish (S_i, θ_i).
2. Upon receiving a secret share tuple (S_ℓ, θ_ℓ) from party P_ℓ, check that the proof θ_ℓ verifies. Otherwise, the secret share is invalid.
3. Upon receiving $t + 1$ valid secret shares $S_j = h^{f(j)}$ from $t + 1$ different parties, compute the secret $S = h^{f(0)}$ via Lagrange interpolation in the exponent.

Figure 4.3: Decryption Dec and reconstruction Rec algorithms of Schoenmakers' PVSS.

In the following, let $e \in \mathbb{Z}_p$ denote the discrete logarithm of h to base g (i.e., $g^e = h$). Assuming the adversary wins the game $\mathbf{G}$ by outputting the secret $h^{P(0)} = g^{eP(0)}$ chosen by the challenger, the above equation is equivalent to

$$e\alpha = \hat{a} + e\hat{b} + e\sum_{i=1}^{n} x_i a_i + \sum_{i=0}^{t} \alpha_i b_i + e\sum_{i=1}^{n} x_i P(i) c_i. \qquad (\spadesuit)$$

We define the following three events:

- Event E_1 defined by the identity $\alpha = \hat{b} + \sum_{i=1}^{n} x_i a_i + \sum_{i=1}^{n} x_i P(i) c_i$.
- Event E_2 defined by the identity $1 = x_1 c_1 + \ldots + x_n c_n = \sum_{i=1}^{n} x_i c_i$.
- Event E_3 defined by the existence of an index $i \in \mathcal{H}$ such that $c_i \neq 0$.[1]

We have the following technical lemma.

Lemma 4.4.2. *Let* $\mathbf{G}$ *and* E_i *for* $i \in [3]$ *be defined as above. Then there exist (algebraic) algorithms* A_j *for* $j \in [4]$ *playing in game* n-OMDL *that run in time at most* T *such that:*

$$\begin{aligned}
\Pr[n\text{-OMDL}^{\mathsf{A}_1} = 1] &= \Pr[\mathbf{G}^{\mathsf{A}} = 1 \wedge \neg E_1], \\
\Pr[n\text{-OMDL}^{\mathsf{A}_2} = 1] &= \Pr[\mathbf{G}^{\mathsf{A}} = 1 \wedge E_1 \wedge \neg E_2], \\
\Pr[n\text{-OMDL}^{\mathsf{A}_3} = 1] &= \Pr[\mathbf{G}^{\mathsf{A}} = 1 \wedge E_1 \wedge E_2 \wedge \neg E_3], \\
\Pr[n\text{-OMDL}^{\mathsf{A}_4} = 1] &= \Pr[\mathbf{G}^{\mathsf{A}} = 1 \wedge E_1 \wedge E_2 \wedge E_3].
\end{aligned}$$

Moreover, the running time satisfies $T \leq T' + O(n^2)$.

Proof. Let $\xi = (\xi_1, \ldots, \xi_n) \in \mathbb{G}^n$ with $\xi = g^{z_i}$ for $i \in [n]$ be the OMDL instance of degree n. Algorithms A_i for $i \in [4]$ have access to a (perfect) discrete logarithm oracle $\mathsf{DL}_g()$ (to base

[1] At this stage, $\mathcal{H} \subset \mathcal{P}$ is the set of parties that remain honest at the end of the game.

g) which they can query at most $n-1$ times. The algorithms A_i, $i \in [4]$, simulate game $\mathbf{G}$ as described in the following.

Algorithm $\mathsf{A}_1(\xi, par)$: Algorithm A_1 works as follows. On input ξ, A_1 queries the discrete logarithm oracle $\mathsf{DL}_g()$ on $\xi_2, \ldots, \xi_n$ and gets $(z_2, \ldots, z_n)$. It publishes the second generator h by setting $h = \xi_1$. In particular, it is $e = \mathsf{DL}_g(h) = z_1$. Furthermore, A_1 generates the public-secret key pairs of parties and the polynomial $P(X) \in \mathbb{Z}_p[X]$ honestly (by sampling $sk_i, \alpha_j \leftarrow \mathbb{Z}_p$ uniformly at random). Commitments X_i, encrypted shares Y_i, and NIZK proofs π are computed honestly and published. Random oracle queries m_i are answered honestly by sampling $r_i \leftarrow \mathbb{Z}_p$ and returning $H[m_i] = r_i$. Corruption queries are answered by returning the secret key of the corresponding party. It is not hard to see that A_1's simulation of $\mathbf{G}$ is perfect.

Suppose A_1 wins $\mathbf{G}$ and that event $\neg E_1$ happens. Equation (♠) is then equivalent to

$$e = \left(\hat{a} + \sum_{i=0}^{t} \alpha_i b_i\right) \cdot \left(\alpha - \hat{b} - \sum_{i=1}^{n} x_i a_i - \sum_{i=1}^{n} x_i P(i) c_i\right)^{-1},$$

since the second factor is non-zero. As a result, A_1 can efficiently compute $e = z_1$ and solve the OMDL instance. Overall, we obtain

$$\Pr[n\text{-}\mathsf{OMDL}^{\mathsf{A}_1} = 1] = \Pr[\mathbf{G}^{\mathsf{A}} = 1 \wedge \neg E_1].$$

The bound on the running time of A_1 (number of group operations and exponentiations) is straightforward.

Algorithm $\mathsf{A}_2(\xi, par)$: Algorithm A_2 works on input $\xi_i = g^{z_i}$, $i \in [n]$, as follows. It samples $e \leftarrow \mathbb{Z}_p$ uniformly at random and publishes $h = g^e$. It generates the public-secret key pairs of parties honestly. It chooses the polynomial $P(X) = \alpha_0 + \alpha_1 X + \ldots + \alpha_t X^t$ such that $g^{\alpha_i} = \xi_{i+1}$ for all $i \in [\![t]\!]$. In particular, it is $\alpha_i = z_{i+1}$ for all $i \in [\![t]\!]$. The commitments $X_i = g^{\alpha_i} = \xi_i$ are published without the need of computation. The encrypted shares $Y_i = pk_i^{P(i)}$ are computed as $Y_i = (h^{P(i)})^{x_i}$ and published. Note that Lagrange interpolation in the exponent allows A_2 to compute $g^{P(i)}$ for any $i \in [n]$ from the commitments C_j, $j \in [\![t]\!]$. The NIZK proofs π are created via an honest-verifier zero-knowledge (HVZK) simulation using the random oracle. Corruption queries are answered by returning the secret key of the corresponding party. Random oracle queries m_i are answered honestly by sampling $r_i \leftarrow \mathbb{Z}_p$ and returning $H[m_i] = r_i$. It is not hard to see that A_2's simulation of $\mathbf{G}$ is perfect.

Suppose A_2 wins $\mathbf{G}$ and that event $E_1 \wedge \neg E_2$ happens. With the notation $P(X) := \alpha_0 + q(X)$ for polynomial $q(X) = \alpha_1 X + \ldots + \alpha_t X^t \in \mathbb{Z}_p[X]$, the equation defined by event E_1 then reduces to

$$\alpha = \hat{b} + \sum_{i=1}^{n} x_i a_i + \sum_{i=1}^{n} x_i q(i) c_i + \alpha \sum_{i=1}^{n} x_i c_i.$$

With the condition that $\neg E_2$ is satisfied, this equation is equivalent to

$$\alpha = \left(\hat{b} + \sum_{i=1}^{n} x_i a_i + \sum_{i=1}^{n} x_i q(i) c_i\right)\left(1 - \sum_{i=1}^{n} x_i c_i\right)^{-1}. \qquad (♦)$$

Algorithm A_2 proceeds as follows. It queries the discrete logarithm oracle $\mathsf{DL}_g()$ on $\xi_2, \ldots, \xi_n$ and obtains $(z_2, \ldots, z_n)$. In particular, it knows $\alpha_1 = z_2, \ldots, \alpha_t = z_{t+1}$ and thus $q(i)$ for all

$i \in \mathbb{Z}_p$ (i.e., it knows the coefficients of the polynomial $q(X)$). From identity (♦), A_2 can efficiently compute $z_1 = \alpha$ and solve OMDL. Overall, we get

$$\Pr[n\text{-OMDL}^{\mathsf{A}_2} = 1] = \Pr[\mathbf{G}^{\mathsf{A}} = 1 \wedge E_1 \wedge \neg E_2].$$

The bound on the running time is obvious, since A_2 has to compute the group elements $q^{P(i)}$ for all $i \in [n]$ by Lagrange interpolation in the exponent from the commitments, which results in additional $O(n^2)$ running time.

Algorithm $\mathsf{A}_3(\xi, par)$: Algorithm A_3 works on input $\xi_i = g^{z_i}$, $i \in [n]$, as follows. The simulation of the game $\mathbf{G}$ is identical to that of A_2; in particular, A_3 chooses $P(X) \in \mathbb{Z}_p[X]$ such that $g^{\alpha_i} = \xi_{i+1}$ for all $i \in [\![t]\!]$. The only difference lies in how A_3 extracts the solution to the OMDL instance after the game has finished. Again, A_3's simulation of $\mathbf{G}$ is perfect.

Suppose that A_3 wins $\mathbf{G}$ and that event $E_1 \wedge E_2 \wedge \neg E_3$ happens. With the same notation $P(X) = \alpha_0 + q(X)$ as before, the equations defined by events E_1 and E_2 then reduce to

$$0 = \hat{b} + \sum_{i=1}^{n} x_i a_i + \sum_{i=1}^{n} x_i q(i) c_i, \quad 1 = \sum_{i=1}^{n} x_i c_i, \tag{♣}$$

where $c_i = 0$ for all $i \in \mathcal{H}$ by condition $\neg E_3$. Therefore, the identity

$$1 = \sum_{i=1}^{n} x_i c_i = \sum_{i \in C} x_i c_i$$

implies that there is an index $\tilde{j} \in C$ such that $x_{\tilde{j}} c_{\tilde{j}} \neq 0$. The first equation in (♣) is then equivalent to

$$q(\tilde{j}) = -\frac{1}{x_{\tilde{j}} c_{\tilde{j}}} \cdot \left(\hat{b} + \sum_{i=1}^{n} x_i a_i + \sum_{i \in C \setminus \{\tilde{j}\}} x_i q(i) c_i \right).$$

Algorithm A_3 proceeds as follows. It queries the oracle $\mathsf{DL}_g()$ on input $\xi_1, \xi_{t+2}, \ldots, \xi_n$ and $g^{q(i)}$ for all $i \in C \setminus \{\tilde{j}\}$. Note that since $g^{P(X)} = g^{\alpha_0} \cdot g^{q(X)} = \xi_1 \cdot g^{q(X)}$, algorithm A_3 can compute (and hence query) $g^{q(i)}$ for any $i \in \mathbb{Z}_p$. W.l.o.g. we may assume that $|C| = t$ (otherwise, A_3 simply simulates, for itself, $t - |C|$ corruption queries for random parties from $\mathcal{H}$). As a result, A_3 can compute $q(\tilde{j})$ and has finally knowledge of $t + 1$ points in the range of $[n]$ on the polynomial $q(X)$ of degree t. In particular, A_3 knows the coefficients of q, i.e., $\alpha_1 = z_2, \ldots, \alpha_t = z_{t+1}$. From previous oracle queries it knows $z_1, z_{t+2}, \ldots, z_n$ and thus solves the OMDL instance with $1 + (n - t - 1) + (t - 1) = n - 1$ queries to $\mathsf{DL}_g()$. Overall, we obtain

$$\Pr[n\text{-OMDL}^{\mathsf{A}_3} = 1] = \Pr[\mathbf{G}^{\mathsf{A}} = 1 \wedge E_1 \wedge E_2 \wedge \neg E_3].$$

The bound on the running time of algorithm A_3 is obvious from the previous case.

Algorithm $\mathsf{A}_4(\xi, par)$: Algorithm A_4 works on input $\xi_i = g^{z_i}$, $i \in [n]$, as follows. It samples $e \leftarrow \mathbb{Z}_p$ uniformly at random and publishes $h = g^e$. It generates the polynomial $P(X) \in \mathbb{Z}_p[X]$ honestly by sampling $\alpha_i \leftarrow \mathbb{Z}_p$ for all $i \in [\![t]\!]$ uniformly at random. It chooses party P_j's public key as $pk_j = \xi_j$ for all $j \in [n]$. In particular, it is $x_j = sk_j = z_j$ for all $j \in [n]$. Commitments X_i, encrypted shares Y_i, and NIZK proofs π are computed honestly and published (which is

possible, since the polynomial $P(X)$ is completely known to A_4). Random oracle queries m_i are answered honestly by sampling $r_i \leftarrow \mathbb{Z}_p$ and returning $H[m_i] = r_i$. Corruption queries are answered with the help of the discrete logarithm oracle $\mathsf{DL}_g()$. A corruption query on party P_j is answered by computing $\mathsf{DL}_g(pk_j)/e$ and returning the secret key sk_j (note that $pk_j = h^{sk_j}$ and the oracle returns the discrete logarithm to base g so that we have to divide the result by e afterwards to get sk_j). It is not hard to see that A_4's simulation of $\mathbf{G}$ is perfect.

Suppose A_4 wins $\mathbf{G}$ and that event $E_1 \wedge E_2 \wedge E_3$ happens. The equation $1 = \sum_{i=1}^{n} x_i c_i$ defined by event E_2 together with E_3 implies there is an index $\tilde{i} \in \mathcal{H}$ such that

$$x_{\tilde{i}} = \frac{1}{c_{\tilde{i}}} \sum_{i \neq \tilde{i}} x_i c_i. \tag{♥}$$

Algorithm A_4 proceeds as follows. It queries the discrete logarithm oracle $\mathsf{DL}_g()$ on ξ_j for all $j \in \mathcal{H} \setminus \{\tilde{i}\}$ and obtains x_j for all $j \in \mathcal{H} \setminus \{\tilde{i}\}$. Therefore, A_4 has knowledge of x_j for all $j \in \mathcal{H} \setminus \{\tilde{i}\} \cup \mathcal{C} = \mathcal{P} \setminus \{\tilde{i}\}$ and computes the remaining value $x_{\tilde{i}}$ by the above identity (♥). As a result, it solves the OMDL instance with $n - 1$ oracle queries. Overall, we obtain

$$\Pr[n\text{-}\mathsf{OMDL}^{\mathsf{A}_4} = 1] = \Pr[\mathbf{G}^{\mathsf{A}} = 1 \wedge E_1 \wedge E_2 \wedge E_3].$$

The claim on the running time of A_4 is clear. □

To end the proof, consider algorithm B playing in n-OMDL as follows: B samples $i^* \leftarrow [4]$ and then internally emulates algorithm A_{i^*}. Clearly, B is an algebraic algorithm running in time at most T (the running time of A_i, $1 \leq i \leq 4$). An application of the law of total probability yields

$$\begin{aligned}
\Pr[n\text{-}\mathsf{OMDL}^{\mathsf{B}} = 1] &= \sum_{i=1}^{4} \Pr[n\text{-}\mathsf{OMDL}^{\mathsf{B}} = 1 \mid i^* = i] \cdot \Pr[i^* = i] \\
&= \frac{1}{4} \sum_{i=1}^{4} \Pr[n\text{-}\mathsf{OMDL}^{\mathsf{A}_i} = 1] \\
&\geq \frac{1}{4} \Pr[\mathbf{G}^{\mathsf{A}} = 1] = \frac{\varepsilon'}{4}.
\end{aligned}$$

□

Remark 4.4.1. By breaking down our above proof, we find that it can be adapted to obtain a security reduction (with the same parameters for tightness) from the plain discrete logarithm problem assuming a weaker static adversary. The main idea being to embed the discrete logarithm challenge $\xi \in \mathbb{G}$ into the public keys $\{pk_i\}_{i \in \mathcal{H}}$ of honest parties (which is fixed from the very beginning in the static corruption model) via $pk_i = \xi^{u_i} g^{v_i}$ for uniformly random $u_i, v_i \leftarrow \mathbb{Z}_p$, into the polynomial $f(X) = \alpha_0 + \alpha X + \ldots + \alpha_t X^t \in \mathbb{Z}_p[X]$ chosen by the dealer via $g^{\alpha_i} = \xi^{u_i} g^{v_i}$ for uniformly random $u_i, v_i \leftarrow \mathbb{Z}_p$, or into the second generator $h \in \mathbb{G}$ via $h = \xi$. Using this technique, the above proof can be adapted accordingly for the static case to reduce the security from plain DLOG.

4.4.2 Application to Distributed Randomness Beacons

We discuss the adaptive security of the previous state-of-the-art randomness beacon protocol from the literature, GRandPiper [Bha+21] in the synchronous network model. The protocol employs an unspecified PVSS scheme in its design. Thus, by instantiating their generic construction with Schoenmakers' PVSS, this results in the adaptive security of the randomness beacon. We briefly discuss the randomness beacon.

GRandPiper. The protocol employs an unspecified PVSS scheme (any secure PVSS scheme can be used) and a leader-based SMR protocol with a communication cost of $O(n\ell + \lambda n^2)$ bits per consensus decision on a block of size ℓ. In each epoch $e \geq 1$, the leader L_e puts a randomly sampled PVSS transcript on the ledger. If L_e does not put anything on the ledger or the transcript is invalid, parties blacklist L_e from future leader elections. Apart from that, parties adhere to a randomized election with blacklisting. Concretely, the leader for epoch e' is chosen based on the beacon output $O_{e'-1}$ and by removing the leaders $L_{e'-1}, \ldots, L_{e'-t}$ of the previous t epochs. Incorrectly behaving leaders are blacklisted from future leader elections. When the same party is elected as a leader $L_{e'}$ in epoch $e' > e + t$ the next time, parties take its previously published (valid) transcript and reconstruct the secret S_e. The beacon output $O_{e'}$ for epoch e' is computed as hash $O_{e'} = \mathsf{Hash}(O_{e'-1}, \ldots, O_{e'-t}, S_e)$. Finally, to ensure availability the first time a party is elected as the leader, the protocol relies on a setup where parties start with agreed-upon buffers $\mathsf{B}(P_i)$ for $i \in [n]$ that contain random PVSS transcripts each. Ignoring the pre-processing phase for the buffers, GRandPiper outputs a randomness beacon value with a communication cost of $O(\lambda n^2)$ bits and optimal resilience in the synchronous setting. However, we note that even if the underlying PVSS scheme was adaptively secure, the randomness beacon remains only $(t+1)$-unpredictable, since an adaptive adversary can predict the next t beacon values (by sequentially computing the next beacon value and corrupting the next leader). The reason for that is that single transcripts are proposed by leaders and not aggregated transcripts.

Theorem 4.4.3. *If Schoenmakers'* PVSS *is (ε, T, t, q_h)-unpredictable in the AGM+ROM, then GRandPiper is an $(\varepsilon', T', t, L, q_h', t+1)$-secure randomness beacon protocol in the AGM+ROM, where*

$$\varepsilon \geq \frac{\varepsilon'}{L} - \frac{q_h'}{p}, \quad T \leq T' + O(Ln^2).$$

Proof. The proof follows along the same lines as the proofs for OptRand and SPURT in Section 4.6.2, Theorem 4.6.1. Therefore, we omit them. □

4.5 Aggregatable PVSS Schemes

In this section, we provide a formal definition for *aggregatable PVSS (APVSS)*. Additionally, we propose our new security notion and prove several schemes from the literature secure with respect to it.

An APVSS scheme allows a dealer to share a secret S via ADist via a *transcript* $T = (\vec{E}, \pi)$. $\vec{E}$ contains a vector of encrypted shares of S, each to a different public key pk_i, such that any $t+1$ decryptions uniquely reconstruct S. T is publicly verifiable using algorithm Ver. The *aggregation* routine Agg allows to homomorphically combine the secrets corresponding to

transcripts $T_1, \ldots, T_k$ into an *aggregate transcript* AT. To be useful as a building block, we endow an aggregatable PVSS scheme with an additional verification routine AVer for aggregated transcripts. Intuitively, AVer can be used to detect whether an aggretated transcript AT has at least one contribution from an honest party. In order for this to be well-defined, we also define the notion of *ownership* of a transcript T. This is captured via the auxiliary algorithm OwnId that can efficiently find the creator of T. In concrete schemes, this is usually implemented by parties holding signing keys in addition to their encryption keys and digitally signing their transcripts. Rather than unnecessarily convoluting our syntax, we simply require that the distribution algorithm ADist take in a party's secret key as part of its input. (This would, for example, allow a party holding a signing key as part of its overall secret key to sign its transcript upon distributing it.) We remark that in our definitions, we syntactically distinguish between transcripts and aggregated transcripts.

Definition 4.5.1 (Aggregatable PVSS Scheme)**.** Let $\hat{\mathbb{G}}$ be a cyclic group of prime order p specified by par. A (t, n)*-threshold aggregatable PVSS (APVSS) scheme over* $\hat{\mathbb{G}}$ is a tuple of algorithms $\mathsf{APVSS} = (\mathsf{Keys}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ADist}, \mathsf{OwnId}, \mathsf{Ver}, \mathsf{AVer}, \mathsf{Rec}, \mathsf{Agg})$ with the following properties:

- Keys: The randomized *key generation algorithm* takes as input system parameters par and an index $i \in [n]$. It outputs a public key pk_i and a secret key sk_i.
- Enc: The randomized *encryption algorithm* takes as input a public key pk_i and a message m. It outputs a ciphertext c.
- Dec: The deterministic *decryption algorithm* takes as input a secret key sk_i and a ciphertext c. It outputs a message m (optionally with a proof of correct decryption). We require that for all messages m,

$$\Pr[\mathsf{Dec}_{sk_i}(\mathsf{Enc}_{pk_i}(m)) = m] = 1.$$

- ADist: The randomized *aggregatable secret sharing algorithm* takes as input a secret key sk_i and public keys $pk_1, \ldots, pk_n$. It outputs a vector of encrypted shares $\vec{E} = (\mathsf{Enc}_{pk_1}(S_1), \ldots, \mathsf{Enc}_{pk_n}(S_n))$ and a proof π, where $S_1, \ldots, S_n$ are shares of a secret $S \in \hat{\mathbb{G}}$. We refer to $T := (\vec{E}, \pi)$ as a *PVSS transcript*.
- Ver: The deterministic *verification algorithm* takes as input public keys $pk_1, \ldots, pk_n$, and a PVSS transcript $T = (\vec{E}, \pi)$. It outputs 1 (accept) or 0 (reject). In the first case we call the transcript T *valid* (relative to $pk_1, \ldots, pk_n$); otherwise we call it *invalid*.
- OwnId: The deterministic *owner identifier algorithm* takes as input a PVSS transcript $T = (\vec{E}, \pi)$ and a public key pk_i. It outputs 1 (accept) or 0 (reject). In the first case, we refer to P_i as the *owner of* T.[2]
- Agg: The deterministic *aggregation algorithm* takes as input $t + 1$ PVSS transcripts $(\vec{E}_1, \pi_1), \ldots, (\vec{E}_{t+1}, \pi_{t+1})$ with pairwise distinct owners. It outputs an *aggregated PVSS transcript* $AT := (\vec{E}, \pi)$.

[2]We remark that OwnId could return 1 on an invalid transcript.

- AVer: The deterministic *aggregation verification algorithm* takes as input public keys $pk_1, \ldots, pk_n$, and an aggregated PVSS transcript $AT = (\vec{E}, \pi)$. It outputs 1 (accept) or 0 (reject). In the first case we call the aggregated transcript *AT valid*; otherwise we call it *invalid*.

- Rec: The deterministic *reconstruction algorithm* takes as input $t + 1$ shares $S_1, \ldots, S_{t+1}$. It outputs a reconstructed secret $S \in \hat{\mathbb{G}}$. *In case* Rec *gets more than* $t + 1$ *shares as input, it takes the first lexicographical* $t + 1$.

For an aggregatable PVSS scheme APVSS = (Keys, Enc, Dec, ADist, OwnId, Ver, AVer, Rec, Agg) as defined above, we define public verifiability of transcripts and aggregated transcripts as well as correctness as follows:

- *Correctness (of Aggregatable PVSS).* We say that APVSS is *correct* if for all keys $(pk_1, sk_1), \ldots, (pk_n, sk_n) \in \mathsf{Keys}(par)$ and all $i \in [n]$,

$$\Pr[\mathsf{Ver}(\{pk_j\}_{j\in[n]}, T) = 1 \wedge \mathsf{OwnId}(pk_i, T) = 1] = 1,$$

  where the probability is taken over all $T \leftarrow \mathsf{ADist}(sk_i, \{pk_j\}_{j\in[n]})$.

- *Public Verifiability (of Transcripts).* We say that APVSS is *publicly verifiable* if for all $(pk_1, sk_1), \ldots, (pk_n, sk_n) \in \mathsf{Keys}(par)$ and all $(\vec{E}, \pi)$ s.t. $\mathsf{Ver}(\{pk_j\}_{j\in[n]}, (\vec{E}, \pi)) = 1$, there exists a unique $S \in \hat{\mathbb{G}}$ such that

$$\mathsf{Rec}(\{\mathsf{Dec}_{sk_i}(\vec{E}_i)\}_{i\in\mathcal{I}}) = S \quad \forall \mathcal{I} \subset [n],\ |\mathcal{I}| = t + 1.$$

- *Public Verifiability (of Aggregated Transcripts).* We say that APVSS is *publicly verifiable* if for all $(pk_1, sk_1), \ldots, (pk_n, sk_n) \in \mathsf{Keys}(par)$ and all aggregated transcripts $AT = (\vec{E}, \pi)$ s.t. $\mathsf{AVer}(\{pk_j\}_{j\in[n]}, (\vec{E}, \pi)) = 1$, there exists a unique $S \in \hat{\mathbb{G}}$ such that

$$\mathsf{Rec}(\{\mathsf{Dec}_{sk_i}(\vec{E}_i)\}_{i\in\mathcal{I}}) = S \quad \forall \mathcal{I} \subset [n],\ |\mathcal{I}| = t + 1.$$

We say that an APVSS scheme is publicly verifiable if both its transcripts and aggregated transcripts are publicly verifiable. We would also like to guarantee that the secret reconstructed from an aggregated transcript $AT = \mathsf{Agg}(T_1, \ldots, T_{t+1})$ corresponds to the sum of the secrets S_i that can be reconstructed from T_i. This is captured in the following definition.

Definition 4.5.2 (Correctness of Aggregation)**.** We say that an aggregatable and publicly verifiable (t, n)-threshold APVSS scheme APVSS = (Keys, Enc, Dec, ADist, OwnId, Ver, AVer, Rec, Agg) over $\hat{\mathbb{G}}$ is *correctly aggregatable* if for all keys $(pk_1, sk_1), \ldots, (pk_n, sk_n) \in \mathsf{Keys}(par)$ and all PVSS transcripts $T_1 = (\vec{E}_1, \pi_1), \ldots, T_{t+1} = (\vec{E}_{t+1}, \pi_{t+1})$ with pairwise distinct owners, the following holds. If for all $i \in [t+1]$, $\mathsf{Ver}(\{pk_j\}_{j\in[n]}, T_i) = 1$, then for all $\mathcal{I} \subset [n]$, $|\mathcal{I}| = t+1$, the aggregated transcript $AT = (\vec{E}', \pi') := \mathsf{Agg}(T_1, \ldots, T_{t+1})$ satisfies

$$\mathsf{Rec}(\{\mathsf{Dec}_{sk_i}(\vec{E}'_i)\}_{i\in\mathcal{I}}) = \prod_{j\in[t+1]} \mathsf{Rec}(\{\mathsf{Dec}_{sk_i}(\vec{E}_{j,i})\}_{i\in\mathcal{I}}),$$

where we write $\vec{E}_j = (\vec{E}_{j,1}, \ldots, \vec{E}_{j,n})$.

4.5.1 New Security Notions for APVSS

We introduce a new security notion for APVSS schemes called *aggregated unpredictability*. This is a kind of non-malleability kind property specifically for aggregatable PVSS schemes. It prohibits an adversary controlling t parties from learning the secret of an aggregated transcript with at least one honest contribution, even if the adversary is allowed to contribute itself to the aggregate. This models an active adversary who can contribute to the final secret itself. We now define this notion formally.

Definition 4.5.3 (Aggregated Unpredictability of Aggregatable PVSS Scheme)**.** Let $\mathsf{APVSS} = (\mathsf{Keys}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ADist}, \mathsf{OwnId}, \mathsf{Ver}, \mathsf{AVer}, \mathsf{Rec}, \mathsf{Agg})$ be a publicly verifiable aggregatable (t, n)-PVSS scheme over $\hat{\mathbb{G}}$. For an algorithm A, we define the *aggregated unpredictability* experiment $\mathbf{AggPred}^{\mathsf{A}}_{\mathsf{APVSS},t}$ as follows:

- *Offline Phase.* For all $i \in [n]$, run Keys on input (par, i) to generate keys $(pk_i, sk_i) \leftarrow \mathsf{Keys}(par, i)$. On input par and $\{pk_i\}_{i\in[n]}$, A returns an index set $C \subset [n]$ of initially corrupted parties along with updated public keys $\{\hat{pk}_j\}_{j\in C}$. Set $pk_j := \hat{pk}_j$ for all $j \in C$.

- *Corruption Queries.* At any point of the experiment, A may submit an index $i \in [n] \setminus C$. In this case, return the secret key sk_i and update $C := C \cup \{i\}$. *If* A *is static, it submits an index set* $C' \subset [n] \setminus C$ *at the beginning of the experiment. Return the secret keys* $\{sk_i\}_{i\in C'}$ *and update* $C := C' \cup C$.

- *Random Oracle Queries.* At any point of the experiment, A gets access to an oracle that answers queries of the following type: When A submits a query m, check if $H[m] = \perp$ (i.e., undefined value yet). If so, sample $H[m] \leftarrow \mathbb{Z}_p^*$ and return it. Otherwise, return $H[m]$.

- *Transcript Queries.* At any point of the experiment, A gets access to an oracle that answers queries of the following type: When A submits a request $(\mathsf{givePVSS}, i)$ for an $i \in [n] \setminus C$, do the following. On behalf of dealer P_i, run ADist on input sk_i and $pk_1, \ldots, pk_n$. Return the output transcript $T = (\vec{E}, \pi)$.

- *Output Determination.* When A outputs an aggregated transcript $AT' = (\vec{E}', \pi')$ and an element $S^* \in \hat{\mathbb{G}}$, do:

  - Return 1 if the following holds: $|C| \leq t$, $\mathsf{AVer}(\{pk_i\}_{i\in[n]}, (\vec{E}', \pi')) = 1$, and $S^* = \mathsf{Rec}(\{\mathsf{Dec}_{sk_i}(\vec{E}'_i)\}_{i\in[t+1]})$.
  - Return 0 otherwise.

We say that APVSS is $(\varepsilon, T, t, q_k, q_h)$*-aggregated unpredictable* if for all algorithms A that run in time at most T, make at most q_k transcript queries, and make at most q_h random oracle queries, $\Pr[\mathbf{AggPred}^{\mathsf{A}}_{\mathsf{APVSS},t} = 1] \leq \varepsilon$. Conversely, we say that A $(\varepsilon, T, t, q_k, q_h)$*-breaks aggregated unpredictability* of APVSS if it runs in time at most T, makes at most q_k transcript queries, makes at most q_h random oracle queries, and $\Pr[\mathbf{AggPred}^{\mathsf{A}}_{\mathsf{APVSS},t} = 1] > \varepsilon$.

Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be an asymmetric pairing and independent generators $g, \hat{g} \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$. Let (pk_i, sk_i) be the key pair of party P_i with $pk_i = h^{sk_i}$. The dealer P_L with key pair (pk_L, sk_L) wants to share secret $e(\hat{g}, h^\alpha)$ for an $\alpha \leftarrow \mathbb{Z}_p^*$. The ADist algorithm takes as input sk_L and public keys $pk_1, \ldots, pk_n$. It outputs the transcript $T_L := \{C_i, Y_i, \pi\}_{i\in[n]}$ defined as follows. In the following, $\langle m \rangle_i := (m, \sigma)$ denotes the pair consisting of message m and a signature σ on m from party P_i.

1. Sample $f(X) := \alpha + \alpha_1 X + \ldots + \alpha_t X^t \leftarrow \mathbb{Z}_p[X]$ of degree t.
2. Publish commitments $C_i = g^{f(i)} \in \mathbb{G}_1$ for $i \in [n]$. Also publish encrypted shares $Y_i = pk_i^{f(i)} \in \mathbb{G}_2$ for $i \in [n]$.
3. Compute $\zeta = g^\alpha$ and a NIZK proof $\theta = (c, r)$ of knowledge of α where the challenge is $c = \mathsf{H}(g^r \zeta^{-c}, \zeta)$. Publish $\pi := \langle \zeta, \theta \rangle_L$.

The *transcript verification algorithm* Ver takes as input the public keys $pk_1, \ldots, pk_n$ (including pk_L) and transcript T_L. It outputs 1 (accept) or 0 (reject). Let $\mathcal{LC}$ be the linear code as defined in General Notation 4.3 and let $\mathcal{LC}^\perp$ be its dual code.

4. Check that $e(g, Y_i) = e(C_i, pk_i)$ for all $i \in [n]$. Sample a random codeword $(\nu_1, \ldots, \nu_n) \in \mathcal{LC}^\perp$ and check that $C_1^{\nu_1} \cdot \ldots \cdot C_n^{\nu_n} = 1$.
5. Check that $\zeta = g^{f(0)}$ via Lagrange interpolation in the exponent from the C_i.
6. Check that the NIZK proof $\theta = (c, r)$ verifies using ζ and H. Check that the signature on $\langle \zeta, \theta \rangle_L$ verifies using pk_L.
7. Output 1 if and only if all the above checks verify.

Figure 4.4: Aggregatable distribution protocol ADist and transcript verification algorithm Ver of OptRand's APVSS.

4.5.2 Security Analysis of several APVSS schemes

We analyze the (adaptive) security of two recent APVSS schemes from the literature that are designed upon Type 3 asymmetric pairings, OptRand's and SPURT's APVSS. As already explained in the introduction, the standard OMDL assumption is not sufficient anymore for this setting. The reason is that the secret is shared in both source groups, which makes it impossible for the simulation to work when relying on OMDL. We elaborate on this in more detail in Appendix 4.6.2. We observe that this issue can be resolved by relying on an extended version of OMDL, which we call the *co one-more discrete logarithm (co-OMDL) assumption*. In the following, let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of prime order p with respective generators $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$. As usual, we denote by $\mathsf{DL}_g()$ an oracle that on input $\xi := g^z \in \mathbb{G}_1$ returns the discrete logarithm z of ξ (to the base g).

Definition 4.5.4 (co-OMDL Problem)**.** Let $par := (\mathbb{G}_1, \mathbb{G}_2, p, g, h)$ be as defined above. For an algorithm A and $n \in \mathbb{N}$, we define the experiment n-**co-OMDL**$^{\mathsf{A}}$ as follows:

On input the encrypted shares $Y_1, \ldots, Y_n$, the decryption Dec and reconstruction Rec algorithms work as follows.

1. Using sk_i, compute the secret share $S_i = h^{f(i)}$ from Y_i via extracting the root $S_i = Y_i^{1/sk_i}$. Publish the decryption S_i.
2. Upon receiving a secret share S_ℓ from party P_ℓ, check that $e(C_\ell, h) = e(g, S_\ell)$. Otherwise, the secret share is invalid.
3. Upon receiving $t + 1$ valid secret shares $S_j = h^{f(j)}$ from different parties, compute $S = h^{f(0)}$ via Lagrange interpolation in the exponent. Finally, the secret is computed as $e(\hat{g}, S) \in \mathbb{G}_T$ and output.

Figure 4.5: Decryption Dec and reconstruction Rec algorithms of OptRand's APVSS.

- *Offline Phase.* Sample $(z_1, \ldots, z_n) \leftarrow \mathbb{Z}_p^n$ uniformly at random and define elements $\xi_i := (g^{z_i}, h^{z_i}) \in \mathbb{G}_1 \times \mathbb{G}_2$ for all $i \in [n]$.
- *Online Phase.* Run A on input *par* and $(\xi_1, \ldots, \xi_n)$. In this phase, A gets access to the oracle $\mathsf{DL}_g()$.
- *Output Determination.* When A outputs $(z'_1, \ldots, z'_n)$, return 1 if (i) $z'_i = z_i$ for $i \in [n]$, and (ii) $\mathsf{DL}_g()$ was queried at most $n - 1$ times during the online phase. Otherwise, return 0.

We say that the co one-more discrete logarithm problem of degree n is (ε, T)-hard if for all algorithms A that run in time at most T, $\Pr[n\textbf{-co-OMDL}^{\mathsf{A}} = 1] \leq \varepsilon$. Conversely, we say that an algorithm A (ε, T)-solves the co one-more discrete logarithm problem of degree n if it runs in time at most T, and $\Pr[n\textbf{-co-OMDL}^{\mathsf{A}} = 1] > \varepsilon$.

In Appendix 4.6.2, we provide a proof of hardness of co-OMDL in the generic group model (GGM) when the groups are equipped with a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. This structure gives the adversary additional power and makes our proof even more valuable. Our proof follows along the lines of Bauer et al.'s [BFP21] proof of hardness of OMDL in the GGM. At the heart of their proof is a technical lemma on vector spaces generated by the vanishing set of linear (multivariate) polynomials. Their proof relies on techniques from linear algebra, especially the theory of linear vector spaces. In our case, however, this lemma does not suffice anymore, since we obtain polynomials of degree 2 from the pairing operation. Nevertheless, using techniques from algebraic geometry and the theory (of rational points) on projective varieties, we are able to extend their lemma to our setting and thus get a proof of hardness of co-OMDL in the generic group model. Further, our proof extends to the setting in which the source groups $\mathbb{G}_1$ and $\mathbb{G}_2$ might have different order (and the discrete logarithm oracle is given in the larger group).

Some notes on OptRand's APVSS. In their scheme, the authors use an unspecified digital signature scheme to sign the commitment $\zeta = g^\alpha$ along with the NIZK proof of knowledge θ. In our description of their scheme, we assume for convenience that the generated pairs (pk_i, sk_i) also (implicitly) include the verification-signing key pair (vk_i, dk_i) of the underlying signature scheme for party P_i so that we do not have to keep track of these pairs in our description of the

We demonstrate aggregation for the first $t+1$ parties $P_1, \ldots, P_{t+1}$. The algorithm Agg takes as input the individual parties' transcripts $\{C_{i,j}, Y_{i,j}, \pi_j\}_{i\in[n]}$ for all party indices $j \in [t+1]$ and outputs an aggregated transcript $AT := \{C_i, Y_i, \pi\}_{i\in[n]}$. In the following, let $\mu_1, \ldots, \mu_{t+1}$ denote the Lagrange coefficients for the set $[t+1]$ at the point $x = 0$, i.e., $\mu_i := \prod_{j\in[t+1]\setminus\{i\}} j/(j-i)$ for $i \in [t+1]$.

1. For $i \in [n]$, compute $C_i := C_{i,1} \cdot \ldots \cdot C_{i,t+1}$ and $Y_i := Y_{i,1} \cdot \ldots \cdot Y_{i,t+1}$. Let $\pi := (\pi_1, \ldots, \pi_{t+1})$ where as above $\pi_j = \langle \zeta_j, \theta_j \rangle_j$ for all $j \in [t+1]$. Publish the aggregated transcript $AT := \{C_i, Y_i, \pi\}_{i\in[n]}$.

The *aggregation transcript verification algorithm* AVer takes as input public keys $pk_1, \ldots, pk_n$ and an aggregated transcript $AT := \{C_i, Y_i, \pi\}_{i\in[n]}$ as above. It outputs 1 (valid aggregated transcript) or 0 (invalid aggregated transcript).

2. Check as usual that $\{C_i, Y_i\}_{i\in[n]}$ and that $\langle \zeta_i, \theta_i \rangle_i$ for $i \in [t+1]$ verify. Also check that $\zeta_1 \cdot \ldots \cdot \zeta_{t+1} = C_1^{\mu_1} \cdot \ldots \cdot C_{t+1}^{\mu_{t+1}}$. Output 1 if and only if all the above checks verify.

Figure 4.6: Aggregation algorithm Agg and aggregation transcript verification algorithm AVer of OptRand's APVSS.

scheme. We will use a signature scheme $\mathsf{DS} = (\mathsf{SKey}, \mathsf{Sign}, \mathsf{Ver})$ as defined in Definition 2.4.1 to implement their (and SPURT's) underlying APVSS scheme. For this, we will write $\mathsf{APVSS}_{\mathsf{DS}}$ to denote that APVSS is implemented with DS. In particular, $(vk_i, dk_i) \leftarrow \mathsf{SKey}(par, i)$ is used for the owner identifier algorithm. In Definition 2.4.2, we define the security of a signature scheme by means of the unforgeability under chosen message game.

Subsequently, we give a tight security reduction from the hardness of n-co-OMDL to the aggregated unpredictability of OptRand's APVSS scheme. In the following, we provide an intuition for our proof. Our analysis starts with the observation that the adversary controlling t parties essentially has four options to successfully predict the secret S of the aggregate. Firstly, it learns an additional $(t+1)$-th decryption key controlled by an honest party, in which case it can derive S from enough decryptions of secret shares. Secondly, it breaks the underlying encryption scheme directly and thus obtains an additional secret share. Thirdly, it finds the discrete logarithm ℓ of the second generator $\hat{g} \in \mathbb{G}_1$ to base g, in which case it can compute the secret $S = e(\hat{g}, h^\alpha)$ from the element g^α (which is derived via Lagrange interpolation in the exponent from the public commitments) by the identity $e(\hat{g}, h^\alpha) = e(g^\alpha, h)^\ell$. Lastly, it forms its contributions to the aggregate such that honest parties' contributions erase (malleability attack). The key idea of our reduction therefore is to embed the n-co-OMDL challenge ξ in the public keys $pk_1, \ldots, pk_n$ of parties, the polynomial $f \in \mathbb{Z}_p[X]$ chosen by the challenger to answer a transcript query, or the second generator $\hat{g} \in \mathbb{G}_1$, a choice that remains hidden from the adversary. In the first case, we simulate by using the discrete logarithm oracle $\mathsf{DL}_g()$ to answer corruption queries. In the second case, we simulate by using an honest-verifier zero knowledge simulation in the random oracle to generate the NIZK proofs for the transcripts of honest parties. In the third case, we execute the protocol honestly. Additionally, our reduction is able to leverage the algebraic equations that result from the random oracle queries by the adversary to generate

its NIZK proofs of knowledge to handle the malleability attack. Overall, our reduction is tight and loses only a factor of $1/6$. The running time of the reduction has a quadratic overhead.

Theorem 4.5.1. *If n-**co-OMDL** is (ε, T)-hard in the AGM and* DS *is $(\varepsilon_s, T_s, q_s)$-secure, then the OptRand* $\mathsf{APVSS}_{\mathsf{DS}}$ *is $(\varepsilon', T', t, q_k, q_h)$-aggregated unpredictable in the AGM+ROM, where*

$$\varepsilon \geq \frac{\varepsilon' - \varepsilon_s}{6} - \frac{q_h}{6p}, \quad T \leq T' + T_s + O(n^2).$$

Proof. Let A be an algebraic adversary that $(\varepsilon', T', t, q_k, q_h)$-breaks aggregated unpredictability of APVSS. In our proof, we assume that all parties are honest prior to the execution of APVSS. It is easy to adjust the proof to the case where the adversary has already corrupted some parties before the execution of the protocol. Additionally, we assume that the aggregated transcript output by the adversary at the end of the game has contribution from exactly one corrupt party. At the end of the proof we explain how to adjust the proof (at one place) to obtain the general case. In the following, let $C \subset \mathcal{P} = \{P_1, \ldots, P_n\}$ be the dynamically changing set of corrupt parties and $\mathcal{H} = \mathcal{P} \setminus C$ the set of honest parties. In particular, we assume that $C = \{\}$ prior to the execution of the protocol. We consider the following game against the adversary.

Game $\mathbf{G}_0$: This is the real game. The challenger generates the system parameters $(\mathbb{G}_1, \mathbb{G}_2, p, g, \hat{g}, h)$, where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an asymmetric pairing of cyclic groups of prime order p with independent generators $g, \hat{g} \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$. Furthermore, the challenger generates the public-secret key pairs $(pk_i, sk_i) = (h^{x_i}, x_i)$ of the honest parties. Whenever A decides to corrupt a party P_i, the challenger returns the internal state of that party, which consists of P_i's secret key $x_i = sk_i$, and sets $C = C \cup \{P_i\}$, $\mathcal{H} = \mathcal{H} \setminus \{P_i\}$. In addition, A gets full control over party P_i. Random oracle queries m_i are answered by sampling $r_i \leftarrow \mathbb{Z}_p^*$ uniformly at random and returning $H[m_i] = r_i$. Transcript oracle queries are answered by sampling a polynomial $f_k \leftarrow \mathbb{Z}_p[X]$ of degree t uniformly at random, running the ADist algorithm and returning the transcript T_k with reconstructed secret $e(\hat{g}, h^{\alpha_{0,k}})$ where $\alpha_{0,k} = f_k(0)$. The transcript also includes $\zeta_k = g^{\alpha_{0,k}}$ and a Chaum-Pedersen non-interactive zero-knowledge (NIZK) proof of knowledge $\pi_k = (c_k, r_k)$ of $\alpha_{0,k}$. The challenge c_k for the proof is computed as the hash $\mathsf{H}(g^{s_k} \| \zeta_k)$, where $s_k = r_k - c_k \alpha_{0,k}$ and $\|$ denotes the concatenation of elements in $\mathbb{G}_1$. From now on, we write $P := f_1 \in \mathbb{Z}_p[X]$ for f_1. At the end of the game, A outputs an aggregated transcript AT with contribution from $t + 1$ different parties along with a (predicted) secret $\sigma^* \in \mathbb{G}_T$.

Game $\mathbf{G}_1$: This game is identical to the game before, except that the game aborts and the adversary loses when it forges a signature of an honest party. Clearly, the statistical distance between game $\mathbf{G}_0$ and $\mathbf{G}_1$ is bounded by the advantage ε_s of A in the UF-CMA game of the underlying signature scheme DS. This observation is necessary, otherwise A could forge signatures on the NIZK and aggregate $t + 1$ transcripts it sampled itself. Note that in the APVSS scheme the signature is used as proof of ownership of a transcript.

The strategy of our reduction will be to embed the co-OMDL instance into the generator $\hat{g}$, the public keys $pk_1, \ldots, pk_n$ of parties, or the polynomials $f_k \in \mathbb{Z}_p[X]$ of transcripts T_k. In the following, we make the simplification by embedding the instance in only one particular polynomial, w.l.o.g. the first one f_1, and that the adversary picks the corresponding transcript T_1 for his aggregated transcript. At the end of the proof, we will eliminate these simplifications. Having said that, our reduction now samples all but the first queried transcript honestly. As A is

an algebraic adversary, it returns the secret σ^* together with a representation

$$\left(a, b, \{c_i\}_{i=1}^n, \{d_i\}_{i=1}^n, \{e_i\}_{i=1}^n, \{f_i\}_{i=1}^n, \{u_i\}_{i=1}^n, \{v_{i,j}\}_{i,j=1}^n, \{w_{i,j}\}_{i,j=1}^n\right)$$

of elements in $\mathbb{Z}_p$ such that

$$\begin{aligned}\sigma^* = e(g,h)^a \cdot e(\hat{g},h)^b \cdot \prod_{i=1}^n e(C_i,h)^{c_i} \cdot \prod_{i=1}^n e(g,pk_i)^{d_i} \cdot \prod_{i=1}^n e(g,Y_i)^{e_i} \\ \cdot \prod_{i=1}^n e(\hat{g},pk_i)^{f_i} \cdot \prod_{i=1}^n e(\hat{g},Y_i)^{u_i} \cdot \prod_{i,j=1}^n e(C_i,pk_j)^{v_{i,j}} \cdot \prod_{i,j=1}^n e(C_i,Y_j)^{w_{i,j}}. \end{aligned} \tag{1}$$

Here, the representation is split (from left to right) into powers of pairing evaluations on combinations of the generators $g, \hat{g} \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$, the public keys $pk_1, \ldots, pk_n \in \mathbb{G}_2$, the polynomial commitments $C_1, \ldots, C_n \in \mathbb{G}_1$ of f_1, and the encrypted shares $Y_1, \ldots, Y_n \in \mathbb{G}_2$. As already clarified, we do not explicitly present the elements from the outputs $\{T_k\}_{k\geq 2}$ in the equation because these can directly be put into the other terms in on the right-hand side of the equation (since the T_k for $k > 1$ are honestly generated). We also do not include ζ into the equation because it can be computed via Lagrange interpolation in the exponent from the commitments $C_1, \ldots, C_n$.

In the following, let R_i for $i \in [q_h]$ denote the random oracle queries made by the adversary. Let $R_\dagger$ and ζ' be the elements corresponding to the contribution of the corrupt party. Since A is an algebraic adversary, it returns the elements $R_\dagger \stackrel{!}{=} g^{r'}\zeta'^{-c'} \in \mathbb{G}_1$ and $\zeta' \in \mathbb{G}_1$ together with an algebraic representation. Note that we assume w.l.o.g. that the adversary queries the random oracle on $R_\dagger \| \zeta$ to obtain a challenge for the NIZK π' corresponding to its contribution. For $R_\dagger$, let $(a', b', c'_1, \ldots, c'_n)$ be elements in $\mathbb{Z}_p$ such that

$$R_\dagger = g^{a'} \cdot \hat{g}^{b'} \cdot C_1^{c'_1} \cdot \ldots \cdot C_n^{c'_n}. \tag{1'}$$

And for ζ', let $(a^\dagger, b^\dagger, c_1^\dagger, \ldots, c_n^\dagger)$ be elements in $\mathbb{Z}_p$ such that

$$\zeta = g^{a^\dagger} \cdot \hat{g}^{b^\dagger} \cdot C_1^{c_1^\dagger} \cdot \ldots \cdot C_n^{c_n^\dagger}. \tag{1"}$$

In the following, let $\ell \in \mathbb{Z}_p$ denote the discrete logarithm of $\hat{g}$ to base g (i.e., $g^\ell = \hat{g}$). And let $\alpha_{0,i} = f_i(0)$ for $i \in [2, q_k]$ denote the secret field elements chosen by the reduction to answer the i-th transcript oracle query T_i. Assuming the adversary wins the game $\mathbf{G}_1$ by outputting the secret of the aggregated transcript AT (w.l.o.g. it has contributions $(\alpha', \alpha, \alpha_{0,2}, \ldots, \alpha_{0,t})$, where α' comes from the adversary), the above equation (1) to base $e(g,h)$ yields

$$\begin{aligned}\ell(\alpha + \alpha' + \sum_{i=2}^t \alpha_{0,i}) = a + \ell b + \sum_{i=1}^n P(i)c_i + \sum_{i=1}^n x_i d_i + \sum_{i=1}^n x_i P(i) e_i \\ + \ell \sum_{i=1}^n x_i f_i + \ell \sum_{i=1}^n x_i P(i) u_i + \sum_{i,j=1}^n P(i) x_j v_{i,j} + \sum_{i,j=1}^n P(i)P(j)x_j w_{i,j}.\end{aligned}$$

Since the $\alpha_{0,i}$ for $i > 1$ are known and the sum with coefficients e_i also appears in the sum with coefficients $v_{i,i}$, this equation reduces to (where we use the same symbols for the coefficients)

$$\ell(\alpha + \alpha') = \ell\Big(b + \sum_{i=1}^{n} x_i f_i + \sum_{i=1}^{n} x_i P(i) u_i\Big) + a + \sum_{i=1}^{n} P(i) c_i + \sum_{i=1}^{n} x_i d_i + \sum_{i,j=1}^{n} P(i) x_j v_{i,j} + \sum_{i,j=1}^{n} P(i) P(j) x_j w_{i,j}, \tag{2}$$

which we write as $\ell(\alpha + \alpha') = \alpha A + B$ for appropriate variables A and B. On the other hand, equation (1') together with the condition that $R_\dagger \stackrel{!}{=} g^{r'} \zeta^{-c'}$ yields

$$\alpha' c' = r' - a' - \ell b' - \sum_{i=1}^{n} P(i) c'_i. \tag{2'}$$

We make the crucial observation that the adversary necessarily queries the random oracle on input $R_\dagger \| \zeta$ before obtaining the challenge c', thus fixing the value α' before c' was chosen by the reduction. Therefore, all appearing variables including α' are independent from c' and the above equation (2') is equivalent to $\alpha' = \tilde{a} + \ell \tilde{b} + \sum_{i \le n} P(i) \tilde{c}_i / c'$ (2"), where the elements $\tilde{a}, \tilde{b}, \tilde{c}_i \in \mathbb{Z}_p$ are appropriately defined. Plugging this equation into the above one (2) with the same notation for A and B yields

$$\ell^2 \tilde{b} + \ell(\alpha + \tilde{a} + \sum_{i=1}^{n} P(i) \tilde{c}_i / c' - A) - B = 0. \tag{♠}$$

Additionally, equation (1") yields

$$\alpha' = a^\dagger + \ell b^\dagger + \sum_{i=1}^{n} P(i) c_i^\dagger, \tag{†}$$

where the coefficients on the right-hand side are again independent from c'. In the following, we denote by V the Vandermonde matrix corresponding to the polynomial $\omega(X) = 1 + X + \ldots + X^t \in \mathbb{Z}_p[X]$ of degree t at the points $\{1, 2, \ldots, n\}$. Since V is Vandermonde, its rank is $t + 1$ and thus its kernel $\ker(V)$ is of dimension $n - (t + 1) = t$.

We define the following four events:

- Event E_1 defined by $\tilde{b} = 0 \ \wedge \ \alpha + \tilde{a} + \sum_{i=1}^{n} P(i) \tilde{c}_i / c' - A = 0$.
- Event E_2 defined by: $(\tilde{c}_1, \ldots, \tilde{c}_n) \in \mathbb{Z}_p^n$ is in the kernel of V.
- Event E_3 defined by $1 = x_1 u_1 + \ldots + x_n u_n$.
- Event E_4 defined by: There is no index $i \in \mathcal{H}$ s.t. $u_i \neq 0$.[3]

We have the following technical lemma.

[3] At this stage, $\mathcal{H} \subset \mathcal{P}$ is the set of parties that remain honest at the end of the game.

Lemma 4.5.2. *Let* $\mathbf{G}_1$ *and* E_i *for* $i \in [4]$ *be defined as above. Then there exist (algebraic) algorithms* A_j *for* $j \in [5]$ *playing in game* n**-co-OMDL** *that run in time at most* T *such that:*

$$\begin{aligned}
\Pr[n\text{-co-OMDL}^{\mathsf{A}_1} = 1] &= \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge \neg E_1],\\
\Pr[n\text{-co-OMDL}^{\mathsf{A}_2} = 1] &= \left(1 - \frac{1}{p}\right) \cdot \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge E_1 \wedge \neg E_2],\\
\Pr[n\text{-co-OMDL}^{\mathsf{A}_3} = 1] &= \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge E_1 \wedge E_2 \wedge \neg E_3],\\
\Pr[n\text{-co-OMDL}^{\mathsf{A}_4} = 1] &= \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge E_1 \wedge E_2 \wedge E_3 \wedge \neg E_4],\\
\Pr[n\text{-co-OMDL}^{\mathsf{A}_5} = 1] &= \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge E_1 \wedge E_2 \wedge E_3 \wedge E_4].
\end{aligned}$$

Moreover, the running time satisfies $T \leq T' + O(n^2)$.

Proof. Let $\xi = (\xi_1, \ldots, \xi_n) \in (\mathbb{G}_1 \times \mathbb{G}_2)^n$ with $\xi_i = (g^{z_i}, h^{z_i})$ for $i \in [n]$ be the co-OMDL instance of degree n. Algorithms A_i for $i \in [5]$ have access to a (perfect) discrete logarithm oracle $\mathsf{DL}_g()$ in $\mathbb{G}_1$ (to base g) which they can query at most $n-1$ times. When we say algorithm A_i queries the discrete logarithm oracle on ξ_j, we mean that it queries $\mathsf{DL}_g()$ on the first component of ξ_j which is a group element in $\mathbb{G}_1$. The algorithms A_i, $i \in [5]$, simulate game $\mathbf{G}_1$ as described in the following.

Algorithm $\mathsf{A}_1(\xi, par)$: Algorithm A_1 works as follows. On input ξ, A_1 queries the discrete logarithm oracle $\mathsf{DL}_g()$ on $\xi_2, \ldots, \xi_n$ and gets $(z_2, \ldots, z_n)$. It publishes the generator $\hat{g}$ by setting $\hat{g} = \xi_{1,1}$. In particular, it is $\ell = \mathsf{DL}_g(\hat{g}) = z_1$. Furthermore, A_1 generates the public-secret key pairs of parties and the polynomial $P(X) \in \mathbb{Z}_p[X]$ honestly (by sampling $sk_i, \alpha_j \leftarrow \mathbb{Z}_p$ uniformly at random). Random oracle queries m_i are answered honestly by sampling $r_i \leftarrow \mathbb{Z}_p$ and returning $H[m_i] = r_i$. Transcript oracle queries are answered honestly by sampling a polynomial $f_k \leftarrow \mathbb{Z}_p[X]$ of degree t uniformly at random and running the distribution phase on it. Corruption queries are answered by returning the secret key of the corresponding party. It is not hard to see that A_1's simulation of $\mathbf{G}_1$ is perfect.

Suppose that A_1 wins $\mathbf{G}_1$ and that event $\neg E_1$ happens. Equation (♠) is then a non-trivial equation of degree one or two in ℓ over the field $\mathbb{Z}_p$ (either the coefficient of ℓ^2 or the one from ℓ is non-zero). By standard techniques, A_1 can efficiently compute $\ell = z_1$ and thus solve the co-OMDL instance. Overall, we obtain

$$\Pr[n\text{-co-OMDL}^{\mathsf{A}_2} = 1] = \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge \neg E_1].$$

The bound on the running time of A_1 is obvious.

Algorithm $\mathsf{A}_2(\xi, par)$: Algorithm A_2 works on input elements $\xi_i = (g^{z_i}, h^{z_i})$, $i \in [n]$, as follows. It samples $\ell \leftarrow \mathbb{Z}_p$ uniformly at random and publishes $\hat{g} = g^{\ell}$. It generates the public-secret key pairs of parties honestly. It chooses the polynomial $P(X) := \alpha_0 + \alpha_1 X + \ldots + \alpha_t X^t$ such that $g^{\alpha_i} = \xi_{i+1,1}$ for all $i \in [\![t]\!]$. Hence, it is $\alpha_i = z_{i+1}$ for all $i \in [\![t]\!]$. Commitments $C_i = g^{P(i)}$ and encryptions $Y_i = pk_i^{P(i)} = (h^{P(i)})^{x_i}$ are computed via Lagrange interpolation in the exponent from the elements $\xi_1, \ldots, \xi_{t+1}$ and returned to the adversary. The NIZK proof π is generated via an HVZK simulation and returned. Random oracle queries m_i are answered by sampling $r_i \leftarrow \mathbb{Z}_p$ and returning $H[m_i] = r_i$. Transcript oracle queries for $k > 1$ are answered by sampling a polynomial $f_k \leftarrow \mathbb{Z}_p[X]$ of degree t uniformly at random and running the distribution phase on it. Corruption queries are answered by returning the secret key of the corresponding party. It is not hard to see that A_2's simulation of $\mathbf{G}_1$ is perfect.

Suppose that A_2 wins $\mathbf{G}_1$ and that event $E_1 \wedge \neg E_2$ happens. In particular, the vector $(\tilde{c}_1, \ldots, \tilde{c}_n) \in \mathbb{Z}_p^n$ is not in the kernel of V. Comparison of the equations (2") and (†) coming from the random oracle query on $R_\dagger \| \zeta'$ gives

$$\begin{aligned} \tilde{a} + \ell\tilde{b} + \sum_{i=1}^{n} P(i)\tilde{c}_i/c' &= a^\dagger + \ell b^\dagger + \sum_{i=1}^{n} P(i)c_i^\dagger \\ \iff \quad \sum_{i=1}^{n} P(i)\delta_i &= a^\dagger - \tilde{a} + \ell(b^\dagger - \tilde{b}), \end{aligned}$$

where $\delta_i := (\tilde{c}_i/c' - c_i^\dagger)$ for all $i \in [n]$. With the fixed notation $P(X) := \alpha_0 + \ldots + \alpha_t X^t$, the last equation is equivalent

$$\begin{aligned} \sum_{i=0}^{t} \alpha_i(\delta_1 + 2^i\delta_2 + 3^i\delta_3 + \ldots + n^i\delta_n) &= a^\dagger - \tilde{a} + \ell(b^\dagger - \tilde{b}) \\ \iff \quad \sum_{i=0}^{t} \alpha_i F(i) &= a^\dagger - \tilde{a} + \ell(b^\dagger - \tilde{b}), \end{aligned} \tag{♥}$$

where $F(X) := \delta_1 + 2^X\delta_2 + 3^X\delta_3 + \ldots + n^X\delta_n$. Assuming $F(i) = 0$ for all $i \in [\![t]\!]$, we get the following system of linear equations in the variables $\delta_1, \ldots, \delta_n$:

$$\begin{aligned} 0 &= \delta_1 + \delta_2 + \delta_3 + \ldots + \delta_n \\ 0 &= \delta_1 + 2\delta_2 + 3\delta_3 + \ldots + n\delta_n \\ &\vdots \\ 0 &= \delta_1 + 2^t\delta_2 + 3^t\delta_3 + \ldots + n^t\delta_n, \end{aligned}$$

which in matrix form is equivalent to

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 2 & \cdots & n \\ \vdots & \vdots & & \vdots \\ 1 & 2^t & \cdots & n^t \end{pmatrix} \cdot \begin{pmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \iff \quad V \cdot \begin{pmatrix} \tilde{c}_1 \\ \tilde{c}_2 \\ \vdots \\ \tilde{c}_n \end{pmatrix} / c' = V \cdot \begin{pmatrix} c_1^\dagger \\ c_2^\dagger \\ \vdots \\ c_n^\dagger \end{pmatrix}.$$

By assumption that event $\neg E_2$ happens, the left-hand side is an n-dimensional non-zero vector with scaling factor $1/c'$, whereas the right-hand side is an n-dimensional vector independent from c' (since the coefficients $c_i^\dagger$ and $\tilde{c}_i$ were fixed by the adversary before seeing c'). As a result, both sides are equal with probability at most $1/p$. Therefore, there is an $\tilde{i} \in [\![t]\!]$ such that $F(\tilde{i}) \neq 0$ with probability $1 - 1/p$ and algorithm A_2 proceeds as follows. It queries the discrete logarithm oracle $\mathsf{DL}_g(\xi_{i+1})$ for all $i \in [\![n-1]\!] \setminus \{\tilde{i}\}$ and obtains z_i for all $i \in [n] \setminus \{\tilde{i}+1\}$. In particular, A_3 has knowledge of the polynomial coefficients α_i for all $i \neq \tilde{i}$ and computes the remaining value $\alpha_{\tilde{i}}$ from the above equation (♥) using $F(\tilde{i}) \neq 0$. As a result, it solves the co-OMDL instance with $n-1$ oracle queries. Overall, we obtain

$$\Pr[n\textbf{-co-OMDL}^{\mathsf{A}_2} = 1] = \left(1 - \frac{1}{p}\right) \cdot \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge E_1 \wedge \neg E_2].$$

The bound on the running time of A_2 is clear.

Algorithm $\mathsf{A}_3(\xi, par)$: Algorithm A_3 works on input $\xi_i = (g^{z_i}, h^{z_i})$, $i \in [n]$, as follows. It queries the discrete logarithm oracle $\mathsf{DL}_g()$ on $\xi_2, \ldots, \xi_n$ and gets $(z_2, \ldots, z_n)$. It samples $\ell \leftarrow \mathbb{Z}_p$ uniformly at random and publishes $\hat{g} = g^\ell$. It generates the public-secret key pairs of parties honestly. It chooses the polynomial $q(X) = \alpha_1 X + \ldots + \alpha_t X^t \in \mathbb{Z}_p[X]$ uniformly at random and lets $P(X) = \alpha + q(X)$ such that $g^\alpha = \xi_{1,1}$. In particular, it is $\alpha = z_1$ and A_3 knows the coefficients α_i for $i \in [t]$ (since it chose them uniformly at random). Commitments $C_i = g^{P(i)}$ are computed as $C_i = \xi_{1,1} g^{q(i)}$ and returned. Encrypted shares $Y_i = pk_i^{P(i)}$ are computed as $Y_i = (h^{P(i)})^{x_i}$ where $h^{P(i)} = \xi_{1,2} h^{q(i)}$ and returned. The NIZK proof π is generated via an HVZK simulation and returned. Random oracle queries m_i are answered honestly by sampling $r_i \leftarrow \mathbb{Z}_p$ and returning $H[m_i] = r_i$. Transcript oracle queries for $k > 1$ are answered honestly by sampling a polynomial $f_k \leftarrow \mathbb{Z}_p[X]$ of degree t uniformly at random and running the distribution phase on it. Corruption queries are answered by returning the secret key of the corresponding party. It is not hard to see that A_3's simulation of $\mathbf{G}_1$ is perfect.

Suppose that A_3 wins $\mathbf{G}_1$ and that event $E_1 \wedge E_2 \wedge \neg E_3$ happens. The equation defining event E_1 is given by

$$\alpha + \tilde{a} + \sum_{i=1}^{n} P(i)\tilde{c}_i/c' = b + \sum_{i=1}^{n} x_i f_i + \sum_{i=1}^{n} x_i P(i) u_i.$$

The knowledge that $(\tilde{c}_1, \ldots, \tilde{c}_n) \in \ker(V)$ given by event E_2 reduces this equation to

$$\alpha + \tilde{a} = b + \sum_{i=1}^{n} x_i f_i + \sum_{i=1}^{n} x_i P(i) u_i.$$

With the same notation $P(X) = \alpha + q(X)$ as above, this yields

$$\alpha + \tilde{a} = b + \sum_{i=1}^{n} x_i f_i + \sum_{i=1}^{n} x_i q(i) u_i + \alpha \sum_{i=1}^{n} x_i u_i. \qquad (\blacklozenge)$$

With the condition that event $\neg E_3$ happens, equation $(\blacklozenge)$ is a non-trivial linear equation in α and thus yields

$$\alpha = \left(b - \tilde{a} + \sum_{i=1}^{n} x_i f_i + \sum_{i=1}^{n} x_i q(i) u_i\right)\left(1 - \sum_{i=1}^{n} x_i u_i\right)^{-1},$$

since the second factor is non-zero. As a result, A_3 can efficiently compute $\alpha = z_1$ and thus solve the co-OMDL instance with $n - 1$ oracle queries. Overall, we obtain

$$\Pr[n\textbf{-co-OMDL}^{\mathsf{A}_3} = 1] = \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge E_1 \wedge E_2 \wedge \neg E_3].$$

The bound on the running time of algorithm A_3 is obvious.

Algorithm $\mathsf{A}_4(\xi, par)$: Algorithm A_4 works on input $\xi_i = (g^{z_i}, h^{z_i})$, $i \in [n]$, as follows. It samples $\ell \leftarrow \mathbb{Z}_p$ uniformly at random and publishes $\hat{g} = g^\ell$. It generates the polynomial $P(X) \in \mathbb{Z}_p[X]$ honestly by sampling $\alpha_i \leftarrow \mathbb{Z}_p$ for all $i \in [\![t]\!]$ uniformly at random. It chooses party P_j's public key pk_j as $pk_j = \xi_{j,2}$ for all $j \in [n]$. In particular, it is $x_j = sk_j = z_j$ for all $j \in [n]$. Commitments C_i, encrypted shares Y_i, and NIZK proof π are computed

honestly and returned (which is possible, since the polynomial $P(X)$ is completely known to A_4). Random oracle queries m_i are answered honestly by sampling $r_i \leftarrow \mathbb{Z}_p$ and returning $H[m_i] = r_i$. Transcript oracle queries for $k > 1$ are answered honestly by sampling a polynomial $f_k \leftarrow \mathbb{Z}_p[X]$ of degree t uniformly at random and running the distribution phase on it. Corruption queries are answered with the help of the discrete logarithm oracle $\mathsf{DL}_g()$. A corruption query on party P_j is answered by computing $\mathsf{DL}_g(pk_j)$ and returning the secret key sk_j. It is not hard to see that A_4's simulation of $\mathbf{G}_1$ is perfect.

Suppose that A_4 wins $\mathbf{G}_1$ and that event $E_1 \wedge E_2 \wedge E_3 \wedge \neg E_4$ happens. In particular, event $\neg E_4$ implies that there is an index $\tilde{i} \in \mathcal{H}$ such that $u_{\tilde{i}} \neq 0$. Given the equation $1 = x_1 u_1 + \ldots + x_n u_n$ (♣) defined by E_3, algorithm A_4 proceeds as follows. It queries the discrete logarithm oracle $\mathsf{DL}_g()$ on ξ_i for all $i \in \mathcal{H} \setminus \{\tilde{i}\}$ and obtains x_i for all $i \in \mathcal{H} \setminus \{\tilde{i}\}$. Therefore, A_4 has knowledge of x_i for all $i \in \mathcal{H} \setminus \{\tilde{i}\} \cup C = \mathcal{P} \setminus \{\tilde{i}\}$ and computes the remaining value $x_{\tilde{i}}$ by the above equation (♣). As a result, it solves the co-OMDL instance with $n-1$ oracle queries. Overall, we obtain

$$\Pr[n\text{-co-OMDL}^{\mathsf{A}_4} = 1] = \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge E_1 \wedge E_2 \wedge E_3 \wedge \neg E_4].$$

The bound on the running time of algorithm A_4 is clear.

Algorithm $\mathsf{A}_5(\xi, par)$: Algorithm A_5 works on input $\xi_i = (g^{z_i}, h^{z_i})$, $i \in [n]$, as follows. The simulation of the game $\mathbf{G}_1$ is identical to that of A_2; in particular A_5 chooses the polynomial $P(X) = \alpha_0 + \alpha_1 X + \ldots + \alpha_t X^t$ such that $g^{\alpha_i} = \xi_{i+1,1}$ for all $i \in [\![t]\!]$, and thus it is $\alpha_i = z_{i+1}$ for all $i \in [\![t]\!]$. Again, A_5's simulation of $\mathbf{G}_1$ is perfect.

Suppose that A_5 wins $\mathbf{G}_1$ and that event $E_1 \wedge E_2 \wedge E_3 \wedge E_4$ happens. With the same notation $P(X) = \alpha + q(X)$ as before, events E_1 to E_4 yield the identities (note that $u_i = 0$ for all $i \in \mathcal{H}$ by definition of E_4)

$$\tilde{a} = b + \sum_{i=1}^{n} x_i f_i + \sum_{i \in C} x_i q(i) u_i, \quad 1 = \sum_{i \in C} x_i u_i. \tag{$\star$}$$

The latter implies that there is an index $\tilde{j} \in C$ such that $x_{\tilde{j}} u_{\tilde{j}} \neq 0$. The first equation in the above identity ($\star$) is then equivalent to

$$q(\tilde{j}) = -\frac{1}{x_{\tilde{j}} u_{\tilde{j}}} \cdot \left(b - \tilde{a} + \sum_{i=1}^{n} x_i f_i + \sum_{i \in C \setminus \{\tilde{j}\}} x_i q(i) u_i \right).$$

Algorithm A_5 proceeds as follows. It queries the oracle $\mathsf{DL}_g()$ on input $\xi_1, \xi_{t+2}, \ldots, \xi_n$ and $g^{q(i)}$ for all $i \in C \setminus \{\tilde{j}\}$. Note that since $g^{P(X)} = g^{\alpha_0} \cdot g^{q(X)} = \xi_1 \cdot g^{q(X)}$, algorithm A_5 can compute (and hence query) $g^{q(i)}$ for any $i \in \mathbb{Z}_p$. W.l.o.g. we may assume that $|C| = t$ (otherwise, A_5 simply simulates, for itself, $t - |C|$ corruption queries for random parties from $\mathcal{H}$). As a result, A_5 can compute $q(\tilde{j})$ from the above identity and has finally knowledge of $t+1$ points in the range of $[n]$ on the polynomial q of degree t. In particular, A_5 knows the coefficients of q, i.e., $\alpha_1 = z_2, \ldots, \alpha_t = z_{t+1}$. From previous oracle queries it knows $z_1, z_{t+2}, \ldots, z_n$ and thus solves the co-OMDL instance with $1 + (n-t-1) + (t-1) = n-1$ queries to $\mathsf{DL}_g()$. Overall, we obtain

$$\Pr[n\text{-co-OMDL}^{\mathsf{A}_5} = 1] = \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge E_1 \wedge E_2 \wedge E_3 \wedge E_4].$$

The bound on the running time of algorithm A_5 is obvious. □

To end the proof, consider algorithm B playing in n-**co-OMDL** as follows: B samples $i^* \leftarrow [5]$ and then internally emulates A_{i^*}. Clearly, B is an algebraic algorithm running in time at most T (the running time of A_i, $1 \le i \le 5$). An application of the law of total probability yields

$$\begin{aligned}\Pr[n\text{-co-OMDL}^{\mathsf{B}} = 1] &= \frac{1}{5}\sum_{i=1}^{5}\Pr[n\text{-co-OMDL}^{\mathsf{A}_i} = 1] \\ &\ge \frac{1}{5}\left(1 - \frac{1}{p}\right)\cdot \Pr[\mathbf{G}_1^{\mathsf{A}} = 1] \\ &\ge \frac{1}{6}\cdot \Pr[\mathbf{G}_1^{\mathsf{A}} = 1] \ge \frac{1}{6}\left(\varepsilon' - \varepsilon_s - \frac{q_h}{p}\right),\end{aligned}$$

where the last equality comes from the soundness error of the NIZK proof of knowledge of discrete logarithm output by the adversary, one try for each random oracle query.

Finally, we elaborate on the simplifications made at the beginning of the proof, which then completes our analysis. The first point was to assume that there is contribution in the aggregated transcript from precisely one corrupt party. If we more generally assume contribution from $f \le t$ corrupt parties, then the corresponding NIZKs yield f pairs of equations for $\alpha'_1, \ldots, \alpha'_f$ (instead of only α') as given in the identities (2') and (†):

$$\alpha'_j c'_j = r'_j - a'_j - \ell b'_j - \sum_{i=1}^{n} P(i) c'_{i,j}, \quad \alpha'_j = a^{\dagger}_j + \ell b^{\dagger}_j + \sum_{i=1}^{n} P(i) c^{\dagger}_{i,j}$$

for all $j \in [f]$. By summation of these equations over $j \in [f]$ and observing that the last queried challenge (w.l.o.g. the adversary queries and gets $c'_1, \ldots, c'_f$ in ascending order) is truly independent from all the algebraic coefficients output by A for all previously queried challenges, we can reduce to the single-challenge case that we already considered because $c'_1 + \ldots + c'_f$ is now completely independent from the algebraic coefficients on the right-hand side of the resulting equation. The second point was to embed the challenge in only one answer to the transcript queries and that the adversary picks this transcript for his aggregate (which actually only happens with probability at most $1/q_k$). We resolve this with the trick of embedding this instance re-randomized into all transcript. That is, whenever in a simulation that embeds $\xi_1, \ldots, \xi_{t+1}$ into the polynomial f_1, we instead embed $\xi_1^{(i)}, \ldots, \xi_{t+1}^{(i)}$ into polynomial f_i for $i \in [q_k]$, where $\xi_j^{(i)} := \xi_j^{u_{i,j}} \cdot g^{v_{i,j}}$ for uniformly at random chosen $u_{i,j}, v_{i,j} \leftarrow \mathbb{Z}_p^*$ for $j \in [t+1]$ and $i \in [q_k]$. The initial algebraic equation coming from the forgery of the adversary is then identical with polynomial $P = f_1$ replaced by $f := \sum_{i=1}^{q_k} f_i$. The corresponding coefficients are then not $\mathsf{DL}_g(\xi_i) = z_i$ (for $i \in [t+1]$), but they are $z'_i := \sum_{j=1}^{q_k} u_{i,j} z_i + v_{i,j}$. Still, the reduction can solve for the z'_i in each scenario and thus compute the z_i (since it chose the values $u_{i,j}, v_{i,j}$ by itself). This justifies our simplifications and completes our proof. □

We conclude with a theorem on the aggregated unpredictability of SPURT's APVSS scheme. The proof mostly follows the lines of the above one with some modifications. For this, we observe that there are only two differences between SPURT's and OptRand's APVSS. (1) The former assumes an additional generator $\hat{h} \in \mathbb{G}_2$ (resulting in a total of four generators $g, \hat{g} \in \mathbb{G}_1$, $h, \hat{h} \in \mathbb{G}_2$) whose purpose being solely to help their security analysis, which is a reduction from the decisional bilinear Diffie-Hellman (DBDH) problem. (2) The NIZK proof $\pi = (c, r)$ of

knowledge of $\alpha = f(0)$ is replaced by *n independently generated* knowledge-sound NIZK proofs $\pi_i = \{(c_i, r_i)\}$ for $i \in [n]$ of discrete logarithm equality of commitment C_i and encrypted share Y_i (thus obviating the need to compute n pairings for this task). Here, a challenge $c_i, i \in [n]$, is computed as the cryptographic hash $\mathsf{H}(C_i, g^{r_i} C_i^{c_i}, Y_i, h^{r_i} Y_i^{c_i})$ defined by the non-interactive Chaum-Pedersen Σ-protocol. We give a formal description of their scheme next and continue with a security proof of it.

Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be an asymmetric pairing and independent generators $g, \hat{g} \in \mathbb{G}_1$ and $h, \hat{h} \in \mathbb{G}_2$. Let (pk_i, sk_i) be the key pair of party P_i with $pk_i = h^{sk_i}$. The dealer P_L with key pair (pk_L, sk_L) wants to share secret $e(\hat{g}, h^{\alpha})$ for an $\alpha \leftarrow \mathbb{Z}_p^*$. The ADist algorithm takes as input sk_L and public keys $pk_1, \ldots, pk_n$. It outputs the transcript $T_L := \{C_i, Y_i, \pi_i\}_{i \in [n]}$ defined as follows.

1. Sample $f(X) := \alpha + \alpha_1 X + \ldots + \alpha_t X^t \leftarrow \mathbb{Z}_p[X]$ of degree t.
2. Publish commitments $C_i = g^{f(i)} \in \mathbb{G}_1$ for $i \in [n]$. Also publish encrypted shares $Y_i = pk_i^{f(i)} \in \mathbb{G}_2$ for $i \in [n]$.
3. Compute NIZK proofs $\pi_i = \mathsf{Dleq}(g, C_i, pk_i, Y_i)$ of discrete logarithm equality for $i \in [n]$. Publish π_i for $i \in [n]$.

The *transcript verification algorithm* Ver takes as input the public keys $pk_1, \ldots, pk_n$ (including pk_L) and transcript T_L. It outputs 1 (accept) or 0 (reject). Let $\mathcal{LC}$ be the linear code as defined in General Notation 4.3 and let $\mathcal{LC}^{\perp}$ be its dual code.

4. Check that $e(g, Y_i) = e(C_i, pk_i)$ for all $i \in [n]$. Sample a random codeword $(\nu_1, \ldots, \nu_n) \in \mathcal{LC}^{\perp}$ and check that $C_1^{\nu_1} \cdot \ldots \cdot C_n^{\nu_n} = 1$.
5. Check that the NIZK proofs $\pi_1, \ldots, \pi_n$ verify using keys $pk_1, \ldots, pk_n$ and H.
6. Output 1 if and only if all the above checks verify.

Figure 4.7: Aggregatable distribution protocol ADist and transcript verification algorithm Ver of SPURT's APVSS.

Theorem 4.5.3. *If n-co-OMDL is (ε, T)-hard in the AGM and DS is $(\varepsilon_s, T_s, q_s)$-secure, then the SPURT $\mathsf{APVSS}_{\mathsf{DS}}$ is $(\varepsilon', T', t, q_k, q_h)$-aggregated unpredictable in the AGM+ROM, where*

$$\varepsilon \geq \frac{\varepsilon' - \varepsilon_s}{6} - \frac{q_h}{6p}, \quad T \leq T' + T_s + O(n^2).$$

In the following, we explain how the proof differs from the proof of OptRand's APVSS. For this, we observe that there are only two differences between SPURT's and OptRand's APVSS. (1) The former assumes an additional generator $\hat{h} \in \mathbb{G}_2$ (resulting in a total of four generators $g, \hat{g} \in \mathbb{G}_1$, $h, \hat{h} \in \mathbb{G}_2$) whose purpose being solely to help their security analysis, which is a reduction from the decisional bilinear Diffie-Hellman (DBDH) problem. In particular, their PVSS scheme itself makes no use of this additional generator besides in the system parameter generation. (2) The NIZK proof $\pi = (c, r)$ of knowledge of $\alpha = f(0)$ is replaced

On input the encrypted shares $Y_1, \ldots, Y_n$, the decryption Dec and reconstruction Rec algorithms work as follows.

1. Using sk_i, compute the secret share $S_i = h^{f(i)}$ from Y_i via extracting the root $S_i = Y_i^{1/sk_i}$. Publish the decryption S_i.
2. Upon receiving a secret share S_ℓ from party P_ℓ, check that $e(C_\ell, h) = e(g, S_\ell)$. Otherwise, the secret share is invalid.
3. Upon receiving $t + 1$ valid secret shares $S_j = h^{f(j)}$ from different parties, compute $S = h^{f(0)}$ via Lagrange interpolation in the exponent. Finally, the secret is computed as $e(\hat{g}, S) \in \mathbb{G}_T$ and output.

Figure 4.8: Decryption Dec and reconstruction Rec algorithms of SPURT's APVSS.

by *n independently generated* knowledge-sound NIZK proofs $\pi_i = \{(c_i, r_i)\}_{i\in[n]}$ of discrete logarithm equality of commitment C_i and encrypted share Y_i for $i \in [n]$ (thus obviating the need to compute n pairings for this task). Here, a challenge $c_i, i \in [n]$, is computed as the cryptographic hash $\mathsf{H}(C_i, g^{r_i} C_i^{c_i}, Y_i, h^{r_i} Y_i^{c_i})$ defined by the non-interactive Chaum-Pedersen Σ-protocol. In particular, challenge c_i depends only on the pair (C_i, Y_i) and does not establish a connection to the whole transcript $\{C_i, Y_i, \ldots\}_{i\in[n]}$. In contrast to that, in Schoenmakers' PVSS scheme there is only one challenge c for all n NIZKs that is computed as hash of the whole transcript. The reason for SPURT's choice of using n separate challenges is that the design of their randomness beacon would not work otherwise, as we will see in the next section.

This choice, however, comes with a subtle nuance in the security analysis of SPURT's APVSS compared to the one from OptRand that we explain now. An argument in the analysis of OptRand's APVSS involves the observation that the challenge $c = \mathsf{H}(g^r \zeta^{-c}, g^\alpha)$ for the proof of knowledge of α is completely independent from the algebraic coefficients for g^α chosen by the adversary. In the case of SPURT's APVSS, the algebraic adversary could choose the algebraic coefficients for a tuple (C_i, Y_i) (that is input into the random oracle H to obtain the corresponding challenge c_i) dependent from previously obtained challenges c_j for different (C_j, Y_j), $j \neq i$, so that our above argument does not apply directly anymore. We solve this issue by regarding the on H last queried tuple (C_ℓ, Y_ℓ) which gives a challenge c_ℓ that is truly independent from the algebraic coefficients for all previously queried tuples (C_i, Y_i) chosen by the adversary. This allows us by summation over the algebraic equations coming from the n NIZKs to reduce to the single-challenge NIZK case as is given in OptRand's APVSS security analysis. Together with the fact that the additional generator $\hat{h} \in \mathbb{G}_2$ in SPURT's APVSS is an auxiliary element used only in their security analysis, allows us in the algebraic group model to transform the resulting algebraic equations to the algebraic equations that arose in the analysis of OptRand's APVSS, thus getting SPURT's APVSS aggregated unpredictability.

Proof. In the following, we explain how to adapt the proof for OptRand's PVSS into one for SPURT's PVSS. To this end, we first recapitulate the differences between these two designs. (1) SPURT's PVSS assumes an additional generator $\hat{h} \in \mathbb{G}_2$ (resulting in a total of four generators $g, \hat{g} \in \mathbb{G}_1$, $h, \hat{h} \in \mathbb{G}_2$) which the scheme itself makes no use of besides in the system parameter

We demonstrate aggregation for the first $t+1$ parties $P_1, \ldots, P_{t+1}$. The algorithm Agg takes as input the individual parties' transcripts $\{C_{i,j}, Y_{i,j}, \pi_{i,j}\}_{i\in[n]}$ for party indices $j \in [t+1]$ and outputs an aggregated transcript $AT := \{C_i, Y_i\}_{i\in[n]}$. In the following, let $\mu_1, \ldots, \mu_{t+1}$ denote the Lagrange coefficients for the set $[t+1]$ at the point $x = 0$, i.e., $\mu_i := \prod_{j\in[t+1]\setminus\{i\}} j/(j-i)$ for $i \in [t+1]$.

1. For $i \in [n]$, compute $C_i := C_{i,1} \cdot \ldots \cdot C_{i,t+1}$ and $Y_i := Y_{i,1} \cdot \ldots \cdot Y_{i,t+1}$. Publish the aggregated transcript $AT := \{C_i, Y_i\}_{i\in[n]}$.

The *aggregation transcript verification algorithm* AVer takes as input public keys $pk_1, \ldots, pk_n$ and an aggregated transcript $AT := \{C_i, Y_i\}_{i\in[n]}$ as above. It outputs 1 (valid aggregated transcript) or 0 (invalid aggregated transcript).

2. Check as usual that $\{C_i, Y_i\}_{i\in[n]}$ verifies using the pairing. Output 1 if and only if all these checks verify.

Note on aggregation verification. The aggregated transcript AT does not include any NIZK proofs in contrast to OptRand's scheme. However, to ensure security SPURT introduces a novel *collective verification* mechanism in which the set of elements of the individual transcripts $T_1, \ldots, T_{t+1}$ is distributed among all parties. This allows the parties to collectively verify that AT is indeed an aggregation of single transcripts $T_1, \ldots, T_{t+1}$.

Figure 4.9: Aggregation algorithm Agg and aggregation transcript verification algorithm AVer of SPURT's APVSS.

generation (its purpose is solely to help their security analysis, which is a reduction from the decisional bilinear Diffie-Hellman (DBDH) problem). (2) The NIZK proof $\pi = (c, r)$ of knowledge of $\alpha = f(0)$ in OptRand's PVSS is replaced in SPURT's PVSS by n independently generated knowledge-sound NIZK proofs $\pi_i = \{(c_i, r_i)\}_{i\in[n]}$ of discrete logarithm equality of commitment C_i and encrypted share Y_i for $i \in [n]$. Here, a challenge c_i for $i \in [n]$ is computed as the cryptographic hash $\mathsf{H}(C_i, g^{r_i} C_i^{c_i}, Y_i, h^{r_i} Y_i^{c_i})$ defined by the non-interactive Chaum-Pedersen Σ-protocol. In particular, challenge c_i depends only on the pair (C_i, Y_i) and does not establish a connection or dependence to other elements of the transcript $\{C_i, Y_i, \pi\}_{i\in[n]}$. In contrast to that, in Schoenmakers' PVSS scheme there is only one challenge c for all n NIZKs simultaneously that is computed as the hash of the whole transcript.

Let A be an algebraic adversary that $(\varepsilon', T', t, q_k, q_h)$-breaks aggregated unpredictability of PVSS. As in the previous proof, we make some simplifications. First, we assume that all parties are honest prior to the execution of PVSS. Second, we assume that there is contribution from exactly one corrupt party in the aggregated transcript. Third, we embed the co-OMDL instance in only the first answer to the transcript queries. The justification of these three points is analog to the one in the proof for OptRand's PVSS. The adversary wins the aggregated unpredictability game if it can predict the secret of the aggregation that it outputs at the end.

In the following, let $\mathcal{C} \subset \mathcal{P} = \{P_1, \ldots, P_n\}$ be the dynamic set of corrupt parties and $\mathcal{H} = \mathcal{P} \setminus \mathcal{C}$ the set of honest parties. The game between a challenger and the adversary is the same as in the previous proof with the following modifications as described above: (1) the system

parameters are generated as $(\mathbb{G}_1, \mathbb{G}_2, p, g, \hat{g}, h, \hat{h})$, where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an asymmetric pairing of cyclic groups of prime order p with independent generators $g, \hat{g} \in \mathbb{G}_1$ and $h, \hat{h} \in \mathbb{G}_2$, and (2) a transcript $\{C_i, Y_i\}_{i\in[n]}$ is generated with n NIZKs proofs $\{\pi_i\}_{i\in[n]}$. At the end of the game, A outputs an aggregated transcript AT along with a secret $\sigma^* \in \mathbb{G}_T$. As A is an algebraic adversary, it returns the secret σ^* together with a representation

$$\left(a, b, \{c_i\}_{i=1}^n, \{d_i\}_{i=1}^n, \{e_i\}_{i=1}^n, \{f_i, u_i\}_{i=1}^n, \{v_{i,j}\}_{i,j=1}^n, \{w_{i,j}\}_{i,j=1}^n, \{v_i\}_{i=1}^{n+2}\right)$$

of elements in $\mathbb{Z}_p$ such that

$$\begin{aligned}\sigma^* = {} & e(g,h)^a \cdot e(\hat{g},h)^b \cdot \prod_{i=1}^n e(C_i,h)^{c_i} \cdot \prod_{i=1}^n e(g,pk_i)^{d_i} \\ & \cdot \prod_{i=1}^n e(g,Y_i)^{e_i} \cdot \prod_{i=1}^n e(\hat{g},pk_i)^{f_i} \cdot \prod_{i=1}^n e(\hat{g},Y_i)^{u_i} \\ & \cdot \prod_{i,j=1}^n e(C_i,pk_j)^{v_{i,j}} \cdot \prod_{i,j=1}^n e(C_i,Y_j)^{w_{i,j}} \\ & \cdot e(g,\hat{h})^{v_1} \cdot e(\hat{g},\hat{h})^{v_2} \cdot \prod_{i=1}^n e(C_i,\hat{h})^{v_{i+2}}. \end{aligned} \tag{♠}$$

All our simulations will generate $\hat{h} \in \mathbb{G}_2$ honestly by sampling $\ell' \leftarrow \mathbb{Z}_p$ uniformly random and setting $\hat{h} = h^{\ell'}$. By replacing the elements $e(X, \hat{h})$ with $e(X, h)^{\ell'}$ for $X \in \{g, \hat{g}, C_1, \ldots, C_n\}$, the above equation reduces to the initial equation coming from the adversary's forgery obtained in OptRand's PVSS proof. The only difference is that the coefficients for $e(X, h)$ are shifted by ℓ' which sets no problem, since the reduction knows the value ℓ'. The other modification to OptRand's PVSS proof comes from the n NIZKs output by the adversary corresponding to the transcript of the contribution of the corrupt party (for the reduction, it simply simulates the n NIZKs for a transcript via n-time honest-verifier zero-knowledge (HVZK) simulation instead of one time). Let the corresponding equations be the following.

$$f'(j) = \tilde{a}_j + \ell\tilde{b}_j + \sum_{i=1}^n P(i)\tilde{c}_{j,i}/c'_j \quad \forall j \in [n], \tag{♥}$$

which come directly from the NIZKs $\pi'_j = (c'_j, r'_j)$ output by the adversary with degree-t polynomial $f' \in \mathbb{Z}_p[X]$. From the input to the random oracle, there are also the following equations

$$f'(j) = a_j^\dagger + \ell b_j^\dagger + \sum_{i=1}^n P(i)c_{j,i}^\dagger \quad \forall j \in [n]. \tag{†}$$

Without loss of generality we assume that the adversary queries the random oracle on the corresponding group elements (C_j, Y_j) in ascending order starting from $j = 1$ up to $j = n$. Since we assume that the adversary wins the game, its output transcript is valid and thus f' is a polynomial of degree t. Therefore, once the adversary queried the random oracle on the group elements (C_j, Y_j) for $j \in [t + 1]$, the other elements (C_i, Y_i) for $i > t + 1$ can be

computed (by the adversary) via Lagrange interpolation in the exponent and thus their algebraic representations won't give more information than needed. So we only consider the first $t+1$ equations. An argument in the analysis of OptRand's PVSS involved the observation that the challenge $c = \mathsf{H}(g^r \zeta^{-c}, g^\alpha)$ for the proof of knowledge of α is completely independent from the algebraic coefficients for g^α chosen by the adversary. In the above case, however, the algebraic adversary could choose the algebraic coefficients for a tuple (C_i, Y_i) dependent from previously obtained challenges c_j for different (C_j, Y_j), $j < i$, so that the previous argument does not apply directly anymore. We solve this issue by regarding the last queried tuple (C_{t+1}, Y_{t+1}) which gives a challenge c_{t+1} that is truly independent from the algebraic coefficients for all previously queried tuples (C_i, Y_i), $i < t+1$ chosen by the adversary. Let $\mu_1, \ldots, \mu_{t+1}$ be the Lagrange coefficients for the set $[t+1]$ at the point $x = 0$. By Lagrange interpolation (and summation), the equations (♥) and (†) with the notation $\alpha' = f'(0)$ reduce to

$$\alpha' = \tilde{a} + \ell\tilde{b} + \sum_{i=1}^{n} P(i)(\tilde{c}_{i,1} + \tilde{c}_{i,2}/c'_{t+1}), \qquad \alpha' = a^\dagger + \ell b^\dagger + \sum_{i=1}^{n} P(i) c_i^\dagger,$$

with appropriately defined coefficients. In OptRand's PVSS proof, event E_2 was defined by $(\tilde{c}_1, \ldots, \tilde{c}_n)/c' \in \mathbb{Z}_p^n$ being in the kernel of the linear map V. We replace this condition by E_2' defined by

$$(\tilde{c}_{1,1} + \tilde{c}_{1,2}/c'_{t+1}, \ldots, \tilde{c}_{n,1} + \tilde{c}_{n,2}/c'_{t+1}) \in \mathbb{Z}_p^n$$

being in the kernel of V. Since the coefficients $(\tilde{c}_{i,1}, \tilde{c}_{i,2})$ for $i \in [n]$ are independent from c'_{t+1}, the same argumentation as in the previous proof applies and there is no other difference. With the same simulations and same extraction of solutions to the OMDL challenge, the proof goes through. This completes our proof. □

Remark 4.5.1. By breaking down our above proof, we find that it can be adapted to obtain a security reduction (with the same parameters for tightness) from the plain discrete logarithm problem assuming a weaker static adversary. The main idea being to embed the discrete logarithm challenge $\xi \in \mathbb{G}$ into the public keys $\{pk_i\}_{i \in \mathcal{H}}$ of honest parties (which is fixed from the very beginning in the static corruption model) via $pk_i = \xi^{u_i} g^{v_i}$ for uniformly random $u_i, v_i \leftarrow \mathbb{Z}_p$, into the polynomial $f_1(X) = \alpha_0 + \alpha X + \ldots + \alpha_t X^t \in \mathbb{Z}_p[X]$ chosen by the simulator via $g^{\alpha_i} = \xi^{u_i} g^{v_i}$ for uniformly random $u_i, v_i \leftarrow \mathbb{Z}_p$, or into the second generator $\hat{g} \in \mathbb{G}$ via $\hat{g} = \xi$. Using this technique, the above proof can be adapted accordingly for the static case to reduce the security from the plain co-discrete logarithm problem (i.e., our asymmetric version of the discrete logarithm problem). The same is true for the security proof of OptRand's PVSS.

4.6 Application to State-of-the-Art Distributed Randomness Beacons

In this section, we discuss the adaptive security of the state-of-the-art randomness beacon protocols in their respective network models: OptRand [Bha+23] in the synchronous network model, and SPURT [Das+22a] in the partially synchronous network model. In Appendix 4.6.2, we provide a detailed discussion (including a comparison table) on existing work of randomness beacons. We begin by formally defining a randomness beacon protocol.

Distributed Randomness Beacon. A (distributed) randomness beacon is a distributed protocol that allows a system of n parties to generate a sequence of unpredictable and unbiased random values, one for each epoch. Each party P_i has a local log that is defined as a write-once array $\Sigma_i = (\Sigma_i[1], \Sigma_i[2], \dots)$ with $\Sigma_i[\ell]$ being its beacon output at epoch $\ell \geq 1$. Initially, each value is set to $\perp$. We say that party P_i outputs a beacon value in epoch ℓ if it writes a value on $\Sigma_i[\ell]$. A secure randomness beacon has to satisfy the following properties: consistency, availability, bias-resistance, and d-unpredictability.

Definition 4.6.1 (d-Secure Randomness Beacon)**.** Let $\mathcal{RB}$ be an epoch-based protocol executed by n parties $P_1, \dots, P_n$. We define the following security properties for $\mathcal{RB}$:

- *Consistency.* $\mathcal{RB}$ is (t, L)-consistent if the following holds whenever at most t parties are corrupted: if an honest party outputs a value $\sigma_\ell \in \{0,1\}^\lambda$ in epoch $\ell \in [L]$, then all honest parties output σ_ℓ in epoch ℓ.

- *Availability.* $\mathcal{RB}$ is (t, L)-available if the following holds whenever at most t parties are corrupted: for each $\ell \in [L]$, every honest party outputs a value $\sigma_\ell \in \{0,1\}^\lambda$ in epoch ℓ.

- *Bias-Resistance.* $\mathcal{RB}$ is (ε, T, t, L)-bias-resistant if it is (t, L)-available, (t, L)-consistent, and the following holds for all algorithms A, D s.t. A corrupts at most t parties and both A and D run in time at most T. Denote by $\Sigma_{\mathsf{A},L}$ the probability distribution induced by the outputs of an honest party in an execution of $\mathcal{RB}$ until epoch L with A as adversary. Then

$$\Big| \Pr_{\sigma \leftarrow \Sigma_{\mathsf{A},L}} [\mathsf{D}(\sigma) = 1] \; - \Pr_{u \leftarrow U_L} [\mathsf{D}(u) = 1] \Big| \leq \varepsilon,$$

  where U_L denotes the uniform distribution over the L-fold Cartesian product of $\{0,1\}^\lambda$ with itself.

- *d-Unpredictability.* $\mathcal{RB}$ is $(\varepsilon, T, t, L, q_h, d)$-unpredictable if it is (t, L)-available, (t, L)-consistent, and for all $\ell \in [L]$ and algorithms A that run in time at most T and make at most q_h random oracle queries, the following experiment outputs 1 with probabillity at most ε :

  - *Offline Phase.* For all $i \in [n]$, run Keys on input (par, i) to generate keys $(pk_i, sk_i) \leftarrow \mathsf{Keys}(par, i)$. On input par and $\{pk_i\}_{i \in [n]}$, A returns an index set $C \subset [n]$ of initially corrupted parties along with updated public keys $\{\hat{pk}_j\}_{j \in C}$. Set $pk_j := \hat{pk}_j$ for all $j \in C$. Initiate an execution of $\mathcal{RB}$ with A controlling parties in C.
  - *Random Oracle Queries.* At any point of the experiment, A gets access to an oracle that answers queries of the following type: When A submits a query m, check if $H[m] = \perp$. If so, set $H[m] \leftarrow \{0,1\}^\lambda$. Return $H[m]$.
  - *Online Phase.* Run $\mathcal{RB}$ with A. When A outputs a tuple (σ'_e, e) for an $e > \ell$, the experiment ends with output 0 if there is an honest party that has output a value $\sigma_{\ell+1}$ for epoch $\ell + 1$. Continue the execution of $\mathcal{RB}$ for another $e - \ell$ epochs.
  - *Corruption Queries.* During the online phase, A may corrupt a party P_i by submitting an index $i \in [n] \setminus C$. In this case, return the internal state of P_i and set $C := C \cup \{i\}$. Henceforth, A controls P_i.

– *Output Determination.* Return 1 if $|C| \leq t$, $e \geq \ell + d$, $L \geq e$, and $\sigma'_e = \sigma_e$. Otherwise, return 0.

We say that $\mathcal{RB}$ is a $(\varepsilon, T, t, L, q_h, d)$-secure randomness beacon protocol if it is (ε, T, t, L)-bias-resistant, $(\varepsilon, T, t, L, q_h, d)$-unpredictable, (t, L)-available, and (t, L)-consistent.

Discussion. We briefly elaborate on the security notions defined above. Consistency and availability guarantee that each honest party outputs the same value $\sigma_e \in \{0,1\}^\lambda$ in each epoch $e \geq 1$. Bias-resistance guarantees that the beacon outputs are indistinguishable from uniformly random numbers. This property ensures that the adversary has no power in biasing the beacon output, even when controlling up to t parties in the system. On the other hand, this notion does not prohibit the adversary from learning the beacon output some epochs ahead of the honest parties. That is ensured by the notion of d-unpredictability, which states that the adversary does not learn the beacon output d epochs before the honest parties. Conversely, an adversary could predict the beacon output some epochs ahead of the honest parties, e.g., by corrupting the next t leaders whose previously committed values determine the next t beacon outputs as in GRandPiper [Bha+21] or HydRand [Sch+20], without having the power to bias it. In our notions, we introduce an epoch bound L upon which the randomness beacon protocol can run.

In the following, we introduce the notion of a *weakly secure* randomness beacon to capture the properties of SPURT. Since the construction of SPURT allows the parties to output a bot symbol $\perp_{\mathcal{RB}}$ whenever the leader of an epoch does not behave correctly, the protocol does not achieve full availability. Instead, every n epochs the randomness beacon outputs at least $n - t$ truly random outputs $\sigma_i \in \{0,1\}^\lambda$ (and thus non-$\perp_{\mathcal{RB}}$ values) and thus has a form of weak availability. We adapt the other security notions of a randomness beacon accordingly.

Definition 4.6.2 (d-Weakly Secure Randomness Beacon)**.** Let $\mathcal{RB}$ be an epoch-based distributed protocol executed by parties $P_1, \ldots, P_n$. We define the following security properties for $\mathcal{RB}$:

- *Weak Consistency.* $\mathcal{RB}$ is (t, L)-weakly consistent if the following holds whenever at most t parties are corrupted: if an honest party outputs a value $\sigma \in \{0,1\}^\lambda \cup \{\perp_{\mathcal{RB}}\}$ in epoch $\ell \in [L]$, then all honest parties output σ in epoch ℓ.

- *Weak Availability.* $\mathcal{RB}$ is (t, L)-weakly available if the following holds whenever at most t parties are corrupted: for each $\ell \in [L]$, every honest party outputs a value $\sigma_\ell \in \{0,1\}^\lambda \cup \{\perp_{\mathcal{RB}}\}$ in epoch ℓ. Furthermore, the sequence $\sigma_1, \sigma_2, \ldots$ of beacon outputs has the following property: for all $\ell \in [\![L - n]\!]$, any n consecutive values $\sigma_{\ell+1}, \ldots, \sigma_{\ell+n}$ contain at most t values $\perp_{\mathcal{RB}}$.

- *Weak Bias-Resistance.* $\mathcal{RB}$ is (ε, T, t, L)-weakly bias-resistant if it is (t, L)-weakly available, (t, L)-weakly consistent, and the following holds for all algorithms A, D such that A corrupts at most t parties and both A and D run in time at most T. Denote by $\Sigma_{\mathsf{A},L}$ the probability distribution induced by the outputs of an honest party in an execution of $\mathcal{RB}$ until epoch L with A as adversary. And denote by U_L the uniform distribution over the L-times Cartesian product of $\{0,1\}^\lambda \cup \{\perp_{\mathcal{RB}}\}$ with itself. Then

$$\left|\Pr[\mathsf{D}(\sigma) = 1] - \Pr[\mathsf{D}(u) = 1]\right| \leq \varepsilon,$$

where the probabilities are taken over all $(\sigma, u) \in \Sigma_{\mathsf{A},L} \times U_L$ such that $u_i = \perp_{\mathcal{RB}}$ whenever $\sigma_i = \perp_{\mathcal{RB}}$ for $i \in [L]$. *This captures the condition that u is sampled from a distribution with $\perp_{\mathcal{RB}}$ at all points where $\sigma \leftarrow \Sigma_{\mathsf{A},L}$ also has $\perp_{\mathcal{RB}}$.*

- *d-Weak Unpredictability.* $\mathcal{RB}$ is $(\varepsilon, T, t, L, d)$-weakly unpredictable if it is (t, L)-weakly available, (t, L)-weakly consistent, and the following holds for all algorithms A that corrupt at most t parties and run in time at most T: A's advantage in the d-weak unpredictability experiment defined hereafter is at most ε.

We say that $\mathcal{RB}$ is a $(\varepsilon, T, t, L, d)$-weakly secure randomness beacon protocol if it is (ε, T, t, L)-weakly bias-resistant, $(\varepsilon, T, t, L, d)$-weakly unpredictable, (t, L)-weakly available, and (t, L)-weakly consistent.

Definition 4.6.3 (d-Weak Unpredictability for $\mathcal{RB}$)**.** Let $\mathcal{RB}$ be an epoch-based protocol as defined above. For an algorithm A and an $\ell \in [L]$, define the d-weak unpredictability experiment $d\text{-}\mathbf{wUnpred}^{\mathsf{A},\ell}_{\mathcal{RB},t}$ as follows:

- *Offline Phase.* Initialize sets $C = \{\}$ and $\mathcal{H} = \mathcal{P} \setminus C$. Run the parameter generation algorithm on input λ to obtain *par*. Run A on input *par*.
- *Corruption Queries.* At any point of the experiment, A may corrupt a party P_i by submitting an index $i \in \mathcal{P}$. In this case, return the internal state of P_i and set $\mathcal{H} = \mathcal{H} \setminus \{i\}$, $C = C \cup \{i\}$. Henceforth, A controls P_i.
- *Online Phase I.* Run $\mathcal{RB}$ for ℓ epochs until the first honest party outputs value σ_ℓ. *Note that A may also participate in the protocol through the corrupt parties.* Let $\sigma = (\sigma_1, \ldots, \sigma_\ell)$ denote its output values and T_ℓ its protocol transcript up to that point. Run A on input (T_ℓ, σ).
- *Online Phase II.* When A outputs a value (σ'_e, e) for an $e > \ell$, run $\mathcal{RB}$ for $e - \ell$ further epochs to obtain the output values $(\sigma_{\ell+1}, \ldots, \sigma_e)$. Again, A may participate in the execution of $\mathcal{RB}$ through the corrupt parties.
- *Output Determination.* Return 1 if $|C| \leq t$, $e \geq \ell + d$, $L \geq e$, and $\sigma'_e = \sigma_e \neq \perp_{\mathcal{RB}}$. Otherwise, return 0.

Define the advantage of A in the above experiment as

$$\mathbf{Adv}^{\mathsf{A}}_{\mathcal{RB}} = \Pr[d\text{-}\mathbf{wUnpred}^{\mathsf{A},\ell}_{\mathcal{RB},t} = 1],$$

where the probability is taken over all possible executions of $\mathcal{RB}$ and all $\ell \leq L$.

4.6.1 OptRand's and SPURT's Beacon Design

We give a high-level description of the randomness beacons of interest, OptRand and SPURT. We further provide formal descriptions in Figure 4.10 (OptRand) and Figure 4.11 (SPURT) in an abstracted manner. We do this for two reasons: (i) the abstractions that we provide, capture the most important building blocks of the beacons that are sufficient to understand them, and (ii) the actual protocols contain a lot of involved consensus parts. For more details, we refer the reader

directly to the respective papers [Bha+23; Das+22a]. Both protocols are built upon (respective) leader-based *state machine replication (SMR)* protocols. In leader-based SMR, parties run a protocol to agree on a public ledger. The protocol proceeds through *epochs*, where each epoch e has a designated *leader* L_e responsible for choosing the value to agree on for that epoch. In our setting, L_e will be instructed to gather and aggregate PVSS transcripts that other parties send to it at the beginning of epoch e.

The protocol rotates through leaders in round-robin fashion (or using a randomized scheduling) so that even a malicious leader cannot stall progress for more than one round. Whenever L_e is honest, parties are guaranteed to agree on a correct value for epoch e (where correctness here refers to that of the aggregate transcript L_e proposes). We stress that the details of these consensus protocols are immaterial for the ensuing discussion. However, it is important to note that while both OptRand and SPURT achieve only static security for their beacon constructions, their underlying SMR protocols are *adaptively* secure. We elaborate more on this below.

Network Models. In a *synchronous network* as in OptRand, honest parties have local clocks that move at the same speed. Protocols proceed in rounds of fixed and a-priori known length Δ and parties start executing the protocol within Δ time from each other. Here, Δ corresponds to an upper known bound on the *network delay*: hen an honest party P sends a message at time τ, the message is guaranteed to be delivered by time $\tau + \Delta$. In particular, messages sent by honest parties cannot be dropped from the network and are always delivered. Thus, messages sent at the beginning of a round r are guaranteed to be delivered by the end of round r. However, the adversary has full control of the network subject to the above constraints and may deliver some messages much faster, i.e., within time $\delta \ll \Delta$. In addition, the adversary is *rushing* and may pick its messages *after* seeing the honest parties' messages within a round. In SPURT, the network is *partially synchronous*. This means that the network can have unbounded message delays which are under full control of the adversary (however, messages of honest parties may not be dropped). However, there is an unknown *global stabilization time (GST)*, which occurs *eventually*. After GST, the network behaves synchronously with parameter Δ as above.

OptRand. The protocol employs their APVSS scheme and an optimistically responsive extension of RandPiper's adaptively secure leader-based SMR [Bha+21]. This gives a communication complexity of $O(n\ell + \lambda n^2)$ bits per consensus decision on a block of size ℓ bits. In each epoch $e \geq 1$, the leader L_e first collects $t + 1$ valid PVSS transcripts from other parties, aggregates them, and then puts the aggregate on the ledger. If L_e does not put anything on the ledger or the aggregate is invalid, parties blacklist L_e from future leader elections. Apart from this policy, parties adhere to a round-robin leader election. When the same party is elected as a leader $L_{e'}$ in epoch $e' > e + t$ the next time, parties take its previously published (valid) aggregate and reconstruct the secret S_e. The beacon output $O_{e'}$ for epoch e' is computed as hash $O_{e'} = \mathsf{H}(S_e)$. Finally, to ensure availability the first time a party is elected as the leader, the protocol relies on a setup where parties start with agreed-upon buffers $\mathsf{B}(P_i)$ for $i \in [n]$ that contain random PVSS transcripts each. Ignoring the pre-processing phase for buffers, OptRand outputs a randomness beacon value with a communication cost of $O(\lambda n^2)$ bits and optimal resilience $t < n/2$ in the synchronous setting.

SPURT. The protocol employs their APVSS scheme and a leader-based SMR protocol based on HotStuff [Yin+19]. Adaptive security of this SMR protocol follows directly from the adaptive security of HotStuff [Yin+19; Nay23b; Nay23a]. Furthermore, their SMR has a communication

complexity of $O(\ell n^2)$ bits per consensus decision on a block of size ℓ bits. In each epoch $e \geq 1$, the leader L_e collects $t + 1$ valid PVSS transcripts from other parties, aggregates them and multicasts the aggregate. Additionally, L_e distributes other parts of the collection of $t + 1$ transcripts among the parties via the private channels such that each non-leader party checks a disjoint part of the aggregation such that any subset of $t + 1$ honest parties collectively checks the entire aggregation. For efficiency reasons, the SMR is only used for the hash of the aggregate. Again, parties adhere to a round-robin leader election. However, SPURT does not use a blacklisting strategy and a pre-processing phase for buffers and thus does not guarantee availability. When the same party is elected as a leader $L_{e'}$ in epoch $e' = e + n$ the next time, parties take its previously agreed-upon aggregate and reconstruct the secret S_e (in case the leader did not behave correctly, parties do not output a beacon value for that epoch). The randomness beacon value $O_{e'}$ is computed as $e(S_e, h)$. Spurt outputs a beacon with a communication cost of $O(\lambda n^2)$ bits and optimal resilience $t < n/3$ in the partially synchronous setting.

4.6.2 Security Analysis of OptRand and SPURT

We prove that the state-of-the-art randomness beacons OptRand and SPURT are indeed adaptively secure. In their respective papers, the authors only provide a security analysis against a much weaker static adversary. For this, we employ our results from the previous section. For our analysis, we consider the derived protocol SPURT+ which results from SPURT by hashing its final output (OptRand hashes the reconstructed secret at the end anyway). This is necessary, since our aggregated unpredictability notion allows the adversary to obtain partial information about the secret. Thus, by hashing the result, we obtain a truly random beacon output (in the random oracle model, which both protocols assume anyway).

Theorem 4.6.1. *If the underlying* $\mathsf{APVSS}_{\mathsf{DS}}$ *is* $(\varepsilon, T, t, q_k, q_h)$*-aggregated unpredictable in the AGM+ROM, then OptRand (SPURT+) is an* $(\varepsilon', T', t, L, q'_h, 1)$*-(weakly) secure randomness beacon protocol in the AGM+ROM, where*

$$\varepsilon \geq \frac{\varepsilon'}{L} - \frac{q'_h}{p}, \quad T \leq T' + O(Ln^2).$$

Proof. In the following, we provide a security analysis of the randomness beacons of interest. There are four security notions to consider: consistency, availability, unpredictability, and bias-resistance. Regarding the first two notions, their validity follows directly from the analysis of the randomness beacon protocols in their respective papers [Bha+23; Das+22a] by additionally noting that the underlying SMR protocols are already proven adaptively secure. The defining reason is that consistency and availability (also called *safety* and *liveness* in the context of SMR protocols) are exclusively part of the consensus layer and not affected by the security of the underlying APVSS scheme. For the last two notions, we provide a reduction to the aggregated unpredictability of the underlying APVSS schemes (which themselves reduce to the hardness of n-co-OMDL as seen before).

To provide an intuition for the latter two notions, we have the following. Regarding 1-unpredictability of the randomness beacons, we observe: until reconstruction of a secret S_e in epoch e, the hash $\mathsf{H}(S_e)$ is uniformly random for the adversary A that corrupts up to t parties and does not break aggregated unpredictability of the underlying APVSS. Therefore, the inequality for the parameter ε in the theorem directly follows from the results on the underlying APVSS

Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the distributed system of n parties. All parties run the underlying (responsive) state machine replication protocol SMR. The beacon protocol starts with epoch $e = 1$.

- **Setup Phase.** Set $e = 1$ and let $\mathcal{G} := \{P_1, \ldots, P_n\}$ denote the dynamic set of potentially good leaders. Parties agree on a buffer $\mathsf{B}(P_i) := \{AT_i\}$ of a valid random PVSS transcript for each $i \in [n]$.
- **Leader Election.** Use a round-robin leader election with respect to $\mathcal{G}$. If an epoch leader L_e fails to publish a valid aggregated PVSS transcript on SMR, blacklist it from future leader elections by setting $\mathcal{G} := \mathcal{G} \setminus \{L_e\}$.
- **Leader Proposal.** Upon receiving $t + 1$ valid PVSS transcripts from other parties at the beginning of epoch e, the leader L_e creates an aggregated PVSS transcript AT_e and publishes it on SMR.
- **Buffer Update.** Upon observing a valid aggregated transcript AT_e published by the leader L_e of some epoch e, update the buffer $\mathsf{B}(L_e) := \mathsf{B}(L_e) \cup \{AT_e\}$. At the end of epoch e, if no valid aggregated transcript was proposed for epoch $e - t$ by leader L_{e-t}, remove L_{e-t} from future leader elections by setting $\mathcal{G} := \mathcal{G} \setminus \{L_{e-t}\}$.
- **Reconstruction Phase.** Upon entering epoch e with leader L_e, pick the aggregated PVSS transcript AT_e from $\mathsf{B}(L_e)$ and run the PVSS reconstruction protocol on AT_e among all parties. Also update $\mathsf{B}(L_e) := \mathsf{B}(L_e) \setminus \{AT_e\}$.
- **Output Generation.** Upon reconstruction of the secret S_e in epoch e, output the beacon value $\rho_e = \mathsf{Hash}(S_e) \in \{0,1\}^\lambda$.

On the Setup Phase. In order to agree on n valid random PVSS transcripts in their buffers, parties can run the SMR protocol for an initial $2n$ epochs, after which the first epoch $e = 1$ of the actual randomness beacon starts. This ensures that by the time the beacon protocol starts at least n valid aggregated transcripts are available to every party.

Figure 4.10: Distributed randomness beacon protocol OptRand described from the view of party P_i.

schemes from Section 4.5; the loss in $1/L$ comes from the reduction now guessing which of the at most L beacon outputs the adversary can predict. This implies 1-unpredictability. To argue bias-resistance, we observe that according to the above $\mathsf{H}(S_e)$ is uniformly random to A up to reconstruction of S_e, at which point it is fixed in the view of all honest parties and can no longer be altered. The algorithm D gets strictly less information than A about all output beacon values (since D only gets these values, but no information about the protocol execution). Therefore, the uniformity of $\mathsf{H}(S_e)$ for A prior to reconstruction of S_e implies uniformity for D of $\mathsf{H}(S_e)$ under the same assumptions. In the following, we denote by $\mathcal{RB}$ the DRB of consideration, i.e., $\mathcal{RB} \in \{\text{OptRand}, \text{SPURT}\}$.

- (t, L)-consistency and (t, L)-availability. From previous work [Bha+21; Yin+19; Nay23b],

Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the distributed system of n parties. All parties run the underlying (responsive) state machine replication protocol SMR. The beacon protocol starts with epoch $e = 1$.

- **Setup Phase.** There is no setup phase of the protocol.
- **Leader Election.** Use a deterministic round-robin leader election for $\mathcal{P}$.
- **Leader Proposal.** Upon receiving $t + 1$ valid PVSS transcripts $T_1, \ldots, T_{t+1}$ from other parties at the beginning of epoch e, the leader L_e creates an aggregated PVSS transcript AT_e that consists of the commitments and encrypted shares only. It publishes $\mathsf{Hash}(AT_e)$ on SMR, multicasts AT_e, and additionally sends a part $(AT_e)_i$ to party P_i (see bottom explanation).
- **Buffer Update.** Upon observing a valid aggregated transcript AT_e published by the leader L_e of some epoch e, update the buffer $\mathsf{B}(L_e) := \mathsf{B}(L_e) \cup \{AT_e\}$. At the end of epoch e, if no valid aggregated transcript was proposed for epoch $e - t$ by leader L_{e-t}, output $\perp_{\mathcal{RB}}$ as beacon output for that epoch.
- **Output Generation.** Upon entering epoch e with leader L_e, pick the aggregated PVSS transcript AT_e from $\mathsf{B}(L_e)$ and run the PVSS reconstruction protocol on AT_e among all parties to obtain the secret S_e. Output the beacon value $\rho_e = S_e \in \{0,1\}^\lambda$. Additionally, update $\mathsf{B}(L_e) := \mathsf{B}(L_e) \setminus \{AT_e\}$.

On the Leader Proposal and Aggregation. The aggregated transcript AT_e consists only of the commitments and encrypted shares, the NIZK proofs are ignored for AT_e. However, the leader also sends $(AT_e)_i$ to party P_i which is defined as the i-th part of the collection $\{T_1, \ldots, T_{t+1}\}$ of individual transcripts including the NIZKs. Via quorum certificates the parties can collectively verify that the aggregate AT_e is indeed a valid aggregation of the single transcripts $T_1, \ldots, T_{t+1}$.

Figure 4.11: Distributed randomness beacon protocol SPURT described from the view of party P_i.

we know that the randomness beacon $\mathcal{RB}$ builds upon an already adaptively secure SMR (state machine replication) protocol. As a consequence, the analysis on (the weak variants of) consistency and availability remains the same as in their respective papers [Bha+23; Das+22a], since the security of the consensus layer is not affected by the security of the underlying APVSS scheme.

- $(\varepsilon', T', t, L, q_h, 1)$-unpredictability. Let A be an algebraic adversary that breaks $(\varepsilon', T', t, L, q_h, 1)$-unpredictability of the randomness beacon protocol $\mathcal{RB}$. In particular, there is an epoch $\ell \in [L]$ in which A outputs a prediction (σ'_e, e) for a future epoch $e \in [\ell + 1, L]$ such that $\sigma'_e = \sigma_e$ where σ_e is the randomness beacon output for epoch e. We then use algorithm A and its prediction to build an algebraic reduction R that breaks aggregated unpredictability of the underlying APVSS. For this, we simulate an execution of $\mathcal{RB}$ for the adversary A with the help of the oracles provided by the aggregated unpredictability

game for APVSS. In the following, let $C \subset \mathcal{P} := \{P_1, \ldots, P_n\}$ be the dynamically changing set of corrupt parties and $\mathcal{H} := \mathcal{P} \setminus C$ the set of honest parties. We consider the following game of aggregated unpredictability of APVSS between R (with black-box access to the randomness beacon predictor A) and a challenger C.

Building a Reduction. In the following, we describe the workings of our reduction R. The challenger C provides R with public keys $\{pk_i\}_{i\in[n]}$ along with system parameters par. Before starting the simulation of the randomness beacon $\mathcal{RB}$ with A (having full control over parties in C), algorithm R makes a guess which of the at most L beacon outputs the adversary will predict. For this, the reduction samples a number $e^* \leftarrow [L]$ uniformly at random from the set $[L]$. Then, R begins the offline phase of an execution of $\mathcal{RB}$ and runs A on input par and $\{pk_i\}_{i\in[n]}$. A returns an index set $C \subset [n]$ of initially corrupted parties along with updated public keys $\{\hat{pk}_j\}_{j\in C}$, which R forwards to C. The challenger sets $pk_j := \hat{pk}_j$ for all $j \in C$. Subsequently, R begins an execution of $\mathcal{RB}$ with A controlling parties in C. At any point of the execution, A may corrupt a party P_i by submitting an index $i \in [n] \setminus C$. In this case, R forwards this corruption query to C who returns the secret key sk_i of P_i to R. Upon receiving this secret key, R forwards it to A and gives A also full control over P_i. Additionally, at any point of the execution of $\mathcal{RB}$, A may query the random oracle H on input m_i. In this case, R checks if $H[m_i] = \perp$. If so, it samples $r_i \leftarrow \{0,1\}^\lambda$ uniformly at random and returns $H[m_i] := r_i$. Now, for all epochs $\ell \in [L] \setminus \{e^*\}$, algorithm R simulates an honest execution of $\mathcal{RB}$ on behalf of all honest parties. For this, R samples on behalf of each honest party $P_i \in \mathcal{H}$ a polynomial $f_{i,\ell} \leftarrow \mathbb{Z}_p[X]$ of degree t uniformly at random, runs the aggregated distribution protocol ADist on it, and sends the corresponding PVSS transcript $T_{i,\ell}$ to the leader L_ℓ of the epoch. All other instructions of the randomness beacon protocol are also executed honestly. For epoch $e^* \in [L]$, however, R does the following. On behalf of all honest parties $P_i \in \mathcal{H}$, it queries the transcript oracle provided by C. More precisely, it submits $(\mathsf{givePVSS}, i)$ for all $i \in \mathcal{H}$, upon which C returns PVSS transcripts T_{i,e^*}. Algorithm R uses these transcripts to simulate the behavior of honest parties in the execution of $\mathcal{RB}$ with A. Denote the aggregated transcript of that epoch e^* by AT_{e^*}. We define the event E by $e = e^*$. Obviously, we have $\Pr[E] = 1/L$, since $e^* \in [L]$ is sampled uniformly at random and unknown to A. In case event E does not happen, R aborts the game with C and the execution of $\mathcal{RB}$. Note that this happens with probability $1 - \Pr[E] = 1 - 1/L$. Otherwise, A outputs with probability ε' a successful prediction (σ'_e, e) such that $\sigma'_e = \sigma_{e*}$ before the reconstruction of AT_{e^*}. Since $\sigma_{e*} = \mathsf{H}(S_{e^*})$, where S_{e^*} is the reconstructed secret of the aggregated transcript AT_{e^*}, algorithm A must have queried the random oracle H on S_{e^*} before outputting its prediction. As a consequence, it must have submitted S_{e^*} to R before outputting σ_{e^*}. Following the prediction output by A, R aborts the execution of $\mathcal{RB}$ with A and submits (AT_{e^*}, S_{e^*}) as a solution to C for the aggregated unpredictability game. R's success probability in breaking aggregated unpredictability of APVSS is then given by

$$\begin{aligned}\varepsilon = \Pr[\mathbf{AggPred}^{\mathsf{R}}_{\mathsf{APVSS}} = 1] &\geq \Pr[E] \cdot \Pr[\mathbf{Unpred}^{\mathsf{A}}_{\mathcal{RB}} = 1] - \frac{q'_h}{2^\lambda} \\ &\geq \frac{1}{L} \cdot \varepsilon' - \frac{q'_h}{2^\lambda} = \frac{\varepsilon'}{L} - \frac{q'_h}{p},\end{aligned}$$

where the negative summand $q'_h/2^\lambda$ (which is equal to q'_h/p, since p has λ-bit size) comes from the fact that the hash $\mathsf{H}(S_{e^*}) \in \{0,1\}^\lambda$ could occur more than once in the list of q'_h queried hashes by A, in which case R cannot recover S_{e^*}. Regarding the running time bound, R has to simulate each of the at most L epochs which comes with a computational overhead of $L \cdot O(n^2)$.

- (ε', T', t, L)-bias-resistance. We begin with the observation that D gets strictly less information than A about the beacon outputs (since D only gets these output values and no other information about the $\mathcal{RB}$ protocol execution). Now, our previous analysis on the 1-unpredictability of the randomness beacon output for A, however, implies that even A cannot distinguish the beacon output before reconstruction from uniform with probability better than ε'. As a result, D's success probability in distinguishing the beacon outputs from uniform is bounded by A's success probability ε' in predicting the beacon output. Thus yielding the desired (ε', T', t, L)-bias-resistance property.

□

Appendix 4A: Related Work on Distributed Randomness Beacons

In this appendix, we discuss existing works on distributed randomness beacons.

Distributed Randomness Beacons. Over the years, numerous (distributed) randomness beacon protocols have been proposed, each having its own set of strengths and weaknesses, depending on factors such as the network model, setup assumptions, desired efficiency and security properties, and reliance on cryptographic tools or specialized hardware. We briefly categorize several notable randomness beacon protocols found in the literature:

Table 4.1: Comparison table of existing distributed randomness beacon protocols.

Protocol	Network	Resil	Adapt	Unpred	Unbias	Commun	Crypto Primitive	Assump	Setup
Cachin et al. [CKS05]	*async*	1/3	✗	✓	✓	$O(\lambda n^2)$	Unique Th-Sig	CDH	*DKG*
RandHerd [Syt+17]	*async*	1/3	✗	✓	✓	$O(\lambda c^2 \log(n))$	PVSS, Th-Schnorr	DL	*DKG*
Dfinity [Abr+18; Cam+21]	*part sync*	1/3	✗	✓	✓	$O(\lambda n^2)$	Th-BLS	CDH	*DKG*
Herb [CSS19]	*sync*	1/3	✗	✓	✓	$O(\lambda n^3)$	Th-ElGamal	DDH	*DKG*
Drand [Dra20]	*sync*	1/2	✗	✓	✓	$O(\lambda n^2)$	Th-BLS	CDH	*DKG*
Algorand [Gil+17b]	*part sync*	1/3	✗	$\Omega(t)$	✗	$O(\lambda cn)$	VRF	VRF	*CRS*
SPURT [Das+22a]	*part sync*	1/3	✗	✓	✓	$O(\lambda n^2)$	PVSS, Pairings	DBS	*CRS*
HydRand [Sch+20]	*sync*	1/3	✗	$t+1$	✓	$O(\lambda n^2)$	PVSS	DDH	*CRS*
Proof-of-Work [Nak08]	*sync*	1/2	✗	$\Omega(t)$	✗	$O(\lambda n)$	Hash function	PoW	*CRS*
GRandPiper [Bha+21]	*sync*	1/2	✗	$t+1$	✓	$O(\lambda n^2)$	PVSS	SXDH	*PoT*
OptRand [Bha+23]	*sync*	1/2	✗	✓	✓	$O(\lambda n^2)$	PVSS, Pairings	SXDH	*PoT*
RandShare [Syt+17]	*async*	1/3	✓	✓	✓	$O(\lambda n^4)$	Async VSS	DL	*CRS*
RandChain [HYL20]	*sync*	1/2	✓	$O(\lambda)$	✓	$O(\lambda n^2)$	Naka Consensus	VDF, PoW	*CRS*
RandRunner [Sch+21]	*sync*	1/2	✓	$t+1$	✓	$O(\lambda n^2)$	Trapdoor VDF	tVDF, DL	*CRS*
BRandPiper [Bha+21]	*sync*	1/2	✓	✓	✓	$O(\lambda f n^2)$	Multi-VSS	SXDH	*PoT*

We explain the table. **Resil** denotes the Byzantine resilience threshold. **Adapt** denotes adaptive adversary. **Unpred** denotes unpredictability. **Unbias** denotes bias-resistance. **Commun** denotes communication complexity in bits. In RandHerd and Algorand, c denotes the average size of a randomly chosen committee. In BRandpiper, $f \leq t$ denotes the actual number of faults in the system. **Crypto Primitive** denotes the cryptographic primitives in usage. The prefix "Th-" stands for *threshold*, and "Naka Consensus" stands for *Nakamoto Consensus*. **Assump** denotes the underlying hardness assumption for the security proof. **Setup** denotes the setup assumption, where "PoT" stands for *powers-of-tau*. We note that all protocols achieve availability.

- *DKG-based protocols.* These distributed randomness beacon protocols employ threshold cryptography to generate random values. More specifically, the randomness beacon output is a unique threshold signature on the hash of the epoch number. Typically, the threshold BLS signature is used in conjunction with an initial distributed key generation (DKG) phase. Most protocols in this category achieve a communication cost of $O(\lambda n^2)$ bits per beacon output when ignoring the setup phase. A significant drawback of these protocols is their reliance on a DKG, which implies an expensive setup phase and limited reconfiguration guarantees.

- *VDF- and PoW-based protocols* These randomness beacon protocols employ verifiable delay functions (VDFs) or Proof-of-Work (PoW) to generate random values. VDFs are functions that require a certain amount of time to compute but can be verified quickly. While VDF-based protocols can offer high efficiency, they require specialized hardware

to compute the VDF, which might not be accessible to all parties. The same applies to PoW-based protocols, which have a very high computation cost. These protocols utilize heavy and expensive cryptographic or hardware tools.

- *Other protocols.* These randomness beacon protocols do not rely on a DKG and do not utilize heavy cryptographic tools such as VDFs. Most of these protocols employ a PVSS or a VSS to generate random values. As such, they are reconfiguration-friendly, efficient, and simple. With some exceptions, these protocols have cubic or even quartic communication cost.

Protocols in the first category include [Dra20; CKS05; Syt+17; Abr+18; Cam+21; CSS19]. Most of these works rely on the clever idea from Cachin, Kursawe, and Shoup, in which a randomness beacon value in epoch $e \geq 1$ is computed as a unique threshold signature on $\mathsf{Hash}(e)$. In more detail, each party P_i generates a partial signature $\sigma_e^{(i)}$ on the message $\mathsf{Hash}(e)$ and sends this share to every other party. Upon collecting $t + 1$ of these partial signatures, any party can locally compute the beacon output as a full signature σ_e. Cachin et al. works in asynchrony and neither specifies the threshold signature nor the DKG. Dfinity works in partial synchrony and uses threshold BLS together with a non-interactive DKG [Gro21]. Drand [Dra20] works in synchrony and uses threshold BLS together with Gennaro et al.'s DKG [Gen+07]. Other threshold signatures are also employed: Herb works in synchrony and uses the threshold ElGamal signature [DF90], and RandHerd works in asynchrony and uses threshold Schnorr signatures with collective signing [SS01; Syt+16]. For setup, all these protocols incur at least a communication cost of $O(\lambda n^3)$ bits. Once the setup phase is finished, the protocols achieve an improved communication cost of $O(\lambda n^2)$ bits per beacon output. The protocols in this category are unpredictable and bias-resistance, but security is only guaranteed against a static adversary. The most significant drawback of the protocols in this category is that they do not support efficient reconfiguration. In case a party is removed from the network and another one joins, the DKG must be run again among all parties. In particular, these protocols are not well-suited for public permissionless blockchain environments.

Protocols in the second category include [HYL20; CD20; Sch+21; Cho+23; Bau+20; BGB17; Dra18]. These protocols rely on computationally expensive tools such as Proof-of-Work (PoW) [Nak08] and Verifiable Delay Functions (VDF) [Bon+18; Dra18] that require specialized hardware and rely on strong and new assumptions about verifiable time-lock puzzles. Protocols in this category are highly energy-consuming. These works essentially rely on the same idea, in which it takes a certain amount of energy (in the case of PoW) or time (in the case of VDF) to compute the next block or beacon output. VDF-based protocols rely on the assumption that the adversary has no advantage over the honest parties in computing the VDF faster. PoW-based protocols rely on the assumption that the adversary has less computational hash power than the honest parties. RandRunner [Sch+21] works in synchrony and uses a trapdoor VDF. Such a trapdoor VDF can generate unique function values efficiently with the knowledge of the trapdoor, but takes some high specified time T otherwise. RandRunner is bias-resistant against an adaptive adversary and has a communication cost of $O(\lambda n^2)$ bits per beacon output. However, it only achieves $(t + 1)$-unpredictability, since an adaptive adversary can simply corrupt the next t leaders and thus learn the beacon values for the next t epochs. Other protocols in this category such as RandChain [HYL20] that uses a combination of sequential PoW, Nakamoto Consensus and VDF even achieve a communication cost of $O(\lambda n)$ bits per beacon output while providing

security against an adaptive adversary. Besides the high energy costs another drawback is that the beacon output is only guaranteed to be $1/5$-fair (1-fairness means that each party in the system controls comparable power on deciding random outputs). Apart from that, it also suffers from problems concerning blockchain-oriented attacks.

Protocols in the third category include [Syt+17; HYL20; Sch+21; Bha+21; Bha+23; Sch+20; Das+22a; CD17; Dav+18; Gil+17b; Kia+17; AMM18]. These protocols are reconfiguration-friendly and further do not rely on heavy tools such as PoW or VDF. Most notably are HydRand [Sch+20], SPURT [Das+22a], GRandPiper [Bha+21], BRandPiper [Bha+21], and OptRand [Bha+23]. We briefly elaborate on some of them. SPURT works in partial synchrony, achieves bias-resistance and unpredictability against a static adversary, and has a communication cost of $O(\lambda n^2)$ bits per beacon output. It relies on a pairing-based PVSS scheme and a modified version of the HotStuff SMR protocol [Yin+19]. Since it relies on a PVSS scheme, it does not provide security against an adaptive adversary. GRandPiper is a protocol from the RandPiper family of randomness beacons. It works in synchrony, achieves bias-resistance against a static adversary, and has a communication cost of $O(\lambda n^2)$ bits per beacon output. It relies on a PVSS scheme and the RandPiper SMR protocol. However, against an adaptive adversary it only provides $(t+1)$-unpredictability (apart from the problem of adaptive security of the PVSS scheme) and even against a static adversary it achieves only $\min(\lambda, t)$-unpredictability. HydRand works in synchrony with sub-optimal Byzantine resilience threshold $t < n/3$ and achieves a communication cost of $O(\lambda n^2)$ bits. Similar to GRandPiper, it only provides bias-resistance and $(t+1)$-unpredictability against a static adversary. It relies on the Scrape PVSS scheme and the repeated execution of a Byzantine agreement protocol. Since it uses a PVSS scheme, it also suffers from the existence of a security proof against adaptive adversaries. BRandPiper is a self-claimed better version of GRandPiper from the family of RandPiper protocols. It works in synchrony with optimal-resilience threshold and provides complete security (i.e., unpredictability and bias-resistance) against an adaptive adversary. It achieves a communication cost of $O(\lambda f n^2)$ bits per beacon output, where $f \leq t$ is the actual number of faults in the system. It relies on an efficient VSS scheme and the RandPiper SMR protocol. In BRandPiper, the leader of an epoch shares n secrets at once and for the beacon output parties reconstruct a random value accumulating secrets from $t+1$ different (previous) leader parties. The other only protocol in this category that provides complete security against an adaptive adversary is RandShare [Syt+17]. It works in asynchrony with optimal-resilience threshold, and has a worst-case communication cost of $O(\lambda n^4)$ bits per beacon output with best-case communication of $O(\lambda n^3)$ bits per beacon output. It relies on a VSS scheme and a Byzantine agreement (BA) protocol. In RandShare, parties run n independent Byzantine agreement instances in parallel, from which the beacon value is then computed.

Appendix 4B: On the Hardness of co-OMDL in the Generic Group Model

In this appendix, we provide a proof for the hardness of the co-OMDL assumption in the generic group model (GGM). We assume that the groups are equipped with a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. In that case the adversary has access to an additional pairing evaluation oracle that on input $(u, v) \in \mathbb{G}_1 \times \mathbb{G}_2$ returns $e(u, v) \in \mathbb{G}_T$. Our proof is inspired from the original proof for the hardness of the standard OMDL assumption in the GGM of Bauer et al. [BFP21].

4.6.3 On the Necessity of the co-OMDL Assumption

We give two examples from the literature (including this work) and explain why the ordinary OMDL assumption does not suffice anymore to provide adaptive security in the context of asymmetric pairings. We believe this observation to be of high significance, since actual implementations of pairing-based schemes do use asymmetric pairings for efficiency reasons.

Asymmetric PVSS Schemes. In our warm-up section, we gave a reduction from plain unpredictability of Schoenmakers' PVSS (which uses a group $\mathbb{G}$ with generators g, h and no pairing is involved) to the hardness of n-OMDL. In the setting of Type 3 asymmetric pairing, our reduction cannot rely on the n-OMDL assumption anymore. We explain the reason for that. Summarizing our proof for Schoenmakers' PVSS in Section 4.4.1, the reduction embeds the OMDL challenge ξ in the second generator h, the public keys $pk_1, \ldots, pk_n$ of parties, or the degree-t polynomial $f \in \mathbb{Z}_p[X]$ chosen by the dealer/challenger (in the plain unpredictability notion, there is no aggregation involved). For the simulation to work, it needs to generate the commitments $g^{f(i)}$ and encrypted shares $pk_i^{f(i)}$ for $i \in [n]$. In the case where the reduction embeds ξ into the polynomial f, it relies on the knowledge of the discrete logarithm of h to base g to compute the elements $h^{f(i)}$ and from that generate the encrypted shares. This strategy also works for symmetric (Type 1) pairings and Type 2 pairings where there is an efficient way to map between g and h, even if these elements live in different source groups. However, for Type 3 pairings $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, there is no such connection between generators $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$ from different groups. Thus, the reduction is not able to compute $h^{f(i)}$ from $g^{f(i)}$ (and vice versa) and thus fails. This issue can be resolved by relying instead on the co-OMDL assumption that simultaneously provides challenges with the same exponents in both source groups.

Asymmetric Threshold BLS. We recall the threshold BLS signature scheme. Given a symmetric pairing $e' : \mathbb{G} \times \mathbb{G} \to \mathbb{G}'$ and a random oracle $\mathsf{H} : \{0,1\}^* \to \mathbb{G}$, a threshold BLS signature share σ_i from party P_i on a message m is computed as $\sigma_i = \mathsf{H}(m)^{sk_i}$. Verification of σ_i is done by checking the equality $e'(g, \sigma_i) = e'(pk_i, \mathsf{H}(m))$. Now, the security reduction of Bacho and Loss [BL22a] embeds the OMDL instance either in the public keys $pk_1, \ldots, pk_n$ of parties or in the answers $\mathsf{H}(m)$ to random oracle queries. In the first case, signature shares σ_i on a message m are simulated by sampling $r \leftarrow \mathbb{Z}_p^*$ uniformly at random and returning $\mathsf{H}(m)^{sk_i} = (g^{sk_i})^r = pk_i^r$. In the asymmetric version of the threshold BLS scheme, the underlying pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is of Type 3 with public keys $pk_i \in \mathbb{G}_1$ and hash values $\mathsf{H}(m) \in \mathbb{G}_2$. Since there is no connection between the two source groups, signature shares cannot be simulated as before if the secret keys are unknown to the reduction. More concretely, a signature share on m is $\sigma_i = \mathsf{H}(m)^{sk_i} = h^{sk_i r}$ for an $r \in \mathbb{Z}_p^*$ and the generator $h \in \mathbb{G}_2$. Obviously, there is no way to compute this value unless h^{sk_i} is known. However, the public keys g^{sk_i} are elements in $\mathbb{G}_1$ and do not help in the Type 3 setting. Consequently, the simulation fails. We believe this issue can be resolved by relying instead on the co-OMDL assumption that simultaneously provides challenges with the same exponents in both source groups.

4.6.4 Proof of Hardness of co-OMDL in GGM

Theorem 4.6.2. *Let* A *be an adversary that* (ε, T)*-solves the co-OMDL problem of degree* n *in the generic group model, making at most* m *group operation queries and* q_e *pairing evaluation*

queries. Then

$$\varepsilon \leq \frac{2m^2}{p-m^2} + \frac{4(m+q_e)^2}{p-4(m+q_e)^2} + \frac{1}{p}.$$

Proof. Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a pairing of prime order p cyclic groups with generators $g \in \mathbb{G}_1$, $h \in \mathbb{G}_2$, and $e(g,h) \in \mathbb{G}_T$. And let A be an adversary that wants to break n-co-OMDL defined for $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We consider the generic group model (GGM). In the GGM, the adversary does not see actual group elements of the form $g^x \in \mathbb{G}_1$ with $x \in \mathbb{Z}_p$, but encodings $\Xi_1(x)$ of them, where $\Xi_1 : \mathbb{Z}_p \to \{0,1\}^{\log(p)}$ is a random injective function corresponding to the group $\mathbb{G}_1$. The same holds true for the groups $\mathbb{G}_2$ and $\mathbb{G}_T$ with encoding functions Ξ_2 and Ξ_T, respectively. Furthermore, the adversary cannot compute the encoding of $g^x \cdot g^y$ from encodings of g^x and g^y. Instead, it is provided with an group operation oracle that on input the pair $(\Xi_1(x), \Xi_1(y))$ returns $\Xi_1(x+y)$. The same holds true for the group $\mathbb{G}_2$ and $\mathbb{G}_T$ with its encoding functions. Additionally, the adversary can query the pairing on evaluations. That is, on input a pair $(\Xi_1(a), \Xi_2(b))$ which corresponds to (g^a, h^b), the oracle returns $\Xi_T(ab)$ which corresponds to $e(g,h)^{ab} \in \mathbb{G}_T$.

In their proof, the authors of [BFP21] observed the following. In general GGM proofs, the challenger replaces the secrets (that the adversary is supposed to find) by indeterminates. For instance in the case of the discrete logarithm problem, the challenger provides the adversary with an encoding $\Xi(X)$ of a group element g^X where $X \in \mathbb{Z}_p[X]$ is an undefined variable instead of an actual value $x \in \mathbb{Z}_p$. As long as the simulation succeeds, the adversary cannot distinguish between g^X and an actual group element g^x. The idea is then, after the adversary outputs a solution $x' \in \mathbb{Z}_p$ to the discrete logarithm problem g^X, the challenger chooses an actual value $x \leftarrow \mathbb{Z}_p$ uniformly at random to replace X with x. However, an issue arises. During the game the adversary queried the group operation oracle GC to compute several group elements, e.g., $g^X \cdot g^X, g \cdot g^X, (g^X)^7 \cdot g^5, \ldots$ (note that the adversary can only query the oracle GC on input two already computed group elements A, B along with a bit $b \in \{-1,1\}$ to compute $A \cdot B^b$, but we simply assume the adversary already computed the elements $(g^X)^7, g^5$ etc.). More general, we assume that the adversary computed the group elements $g^{P_i(X)}$ for $i \in [m]$ (where m denotes the total number of group operation oracle queries) for polynomials $P_i \in \mathbb{Z}_p[X]$ of degree 1. Obviously, different polynomials $P_i \neq P_j$ correspond to different group elements and thus different encodings $\xi_i \neq \xi_j$ that the challenger provided the adversary with. However, if the variable X is set to be defined $X := x$ for the actual (uniformly random) value $x \in \mathbb{Z}_p$, it could happen that there are polynomials $P_i \neq P_j$ such that $P_i(x) = P_j(x)$. In that case, the adversary detects incorrect behavior and thus the simulation fails. However, by the well-known Schwartz-Zippel lemma, for a fixed pair (P_i, P_j) of polynomials $P_i \neq P_j$, the probability that $P_i(x) = P_j(x)$ is the probability that $(P_i - P_j)(x) = 0$, which is at most $1/p$ by Schwartz-Zippel, since $x \leftarrow \mathbb{Z}_p$ is chosen uniformly at random (after the polynomials P_i, P_j are defined). As a consequence, when considering all $O(m^2)$ pairs of polynomials (P_i, P_j) with $(i,j) \in [m]^2$ and $i \neq j$, Schwartz-Zippel tells us that the simulation fails with probability at most m^2/p which is negligible. This is the original proof idea of Shoup [Sho97] (for the plain discrete logarithm problem).

However, Bauer et al. observed that this technique does not suffice anymore in the OMDL game. Again, the strategy is to replace the n-OMDL challenge $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{Z}_p^n$ with independent indeterminates $\vec{X} = (X_1, \ldots, X_n) \in \mathbb{Z}_p[X_1, \ldots, X_n]$ from the n-dimensional ring

$\mathbb{Z}_p[X_1, \ldots, X_n]$. In contrast to the discrete logarithm (DL) game, the adversary has access to a discrete logarithm oracle $\mathsf{DL}_g()$ that it can query on (encodings $\xi_i = \Xi(a_i)$ of) group elements $A = g^{a_i}$ to obtain $a_i \in \mathbb{Z}_p$. In particular, if it queries the oracle $\mathsf{DL}_g()$ on, e.g., a challenge element g^{X_i}, the challenger is forced to return an actual value $x_i \in \mathbb{Z}_p$ and thus the adversary gains (partial) information on the challenge $\vec{x}$. Hence, the adversary can choose the polynomials P_i (after obtaining x_i) dependent on x_i and for this reason the Schwartz-Zippel lemma does not apply anymore, since the polynomials $P_1, \ldots, P_m$ may not be independent from $(x_1, \ldots, x_n)$ anymore. However, the idea is that even though the adversary gains information on $\vec{x}$ via its $\mathsf{DL}_g()$ queries, the total information can only encode a space of dimension $q < n$ in $\mathbb{Z}_p^n$ where q is the total number of queries to $\mathsf{DL}_g()$ (when we think of one $\mathsf{DL}_g()$ query giving the adversary one information on $\vec{x}$). Let us informally encode this information in the space $\mathcal{L} \subset \mathbb{Z}_p^n$. Following this, the intuition is that, when sampling $\vec{x}$ uniformly at random (when the adversary queries $\mathsf{DL}_g()$ on a newly g^{X_i}, the challenger simply returns a uniformly random $x_i \leftarrow \mathbb{Z}_p$), there is still one dimension in the vector left that is completely independent from all the polynomials $P_1, \ldots, P_m$ and thus by (some form of) Schwartz-Zippel the probability that a collision appeared somewhere outside of the space $\mathcal{L}$ should be at most $1/p$. Indeed, the authors provide a technical lemma that exactly captures the above intuition and can be seen as some form of the Schwartz-Zippel lemma that deals with polynomials of degree 1.

We will follow that idea. However, in our case, since there is a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ involved, we have to deal with polynomials of degree 2 that appear in the target group $\mathbb{G}_T$. Precisely, when the adversary obtains encodings of polynomials $P(\vec{X})$ via the group operation oracle GC_1 in the group $\mathbb{G}_1$, $Q(\vec{X})$ via GC_2 in $\mathbb{G}_2$ (recall that $\vec{X} = (X_1, \ldots, X_n)$ is the list of indeterminates), then it can query the pairing oracle on $(g^{P(\vec{X})}, h^{Q(\vec{X})})$ to obtain $e(g^{P(\vec{X})}, h^{(\vec{X})}) = e(g, h)^{(PQ)(\vec{X})}$ where PQ is now a polynomial of degree 2. We will deal with this issue by carefully keeping track of three different lists, each for one group $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, for the simulation to go through. We also provide a generalization of their technical lemma to incorporate polynomials of degree 2. However, the structure and simulation of our proof follows the one of Bauer et al. with some adaptions to deal with the pairing. Our proof progresses through a series of games that we describe now. In the following, we write $E_P := \{0,1\}^{\log(p)}$ for the space of strings and we let generators be $g_1 := g \in \mathbb{G}_1$, $g_2 := h \in \mathbb{G}_2$, and $g_T := e(g, h) \in \mathbb{G}_T$. For a set $L = \{P_1, \ldots, P_q\} \subset \mathbb{Z}_p[X_1, \ldots, X_n]$ of polynomials in n variables, we write $\mathsf{Span}(L)$ for the $\mathbb{Z}_p$-subspace spanned by $P_1, \ldots, P_q$, that is

$$\mathsf{Span}(L) := \{\alpha_1 P_1 + \ldots + \alpha_q P_q \mid \alpha_i \in \mathbb{Z}_p\}.$$

Game $\mathbf{G}_0$: This is the real game in the generic group model. First, the challenger C initializes the challenge counter $n := 0$, the discrete logarithm oracle counter $q := 0$, a group operation oracle counter $c_j := 0$ for the group $\mathbb{G}_j$ for each group index $j \in \{1, 2, T\}$, the pairing evaluation oracle counter $q_e := 0$, the challenge vector $\vec{x} := ()$, and the field element $a_{0,j} := 1$ corresponding to the generator $g_j \in \mathbb{G}_j$ for each $j \in \{1, 2, T\}$. The adversary has access to an OMDL challenge oracle Chal, a discrete logarithm oracle $\mathsf{DL}_g()$ in the group $\mathbb{G}_1$, a group operation oracles GC_j in the group $\mathbb{G}_j$ for each $j \in \{1, 2, T\}$, and a pairing oracle $e(-,-)$. For $j \in \{1, 2, T\}$, the challenger C has an encoding function Enc_j that implements an injective random function $\Xi_j : \mathbb{Z}_p \to E_p$ to return encodings of group elements (in the generic group model). For this, the challenger initializes lists $\mathcal{M}_j$ for $j \in \{1, 2, T\}$ that keeps track of already assigned encodings $\xi_{i,j} := \Xi_j(a_i)$ for a field element $a_i \in \mathbb{Z}_p$ corresponding to the

group element $g_j^{a_i} \in \mathbb{G}_j$. Each time the function Enc_j is called on a field element $a_i \in \mathbb{Z}_p$, it returns $\xi_{i,j}$ if a_i is already assigned an element $\xi_{i,j}$. Otherwise, it returns a uniformly random $\xi_{i,j} \leftarrow E_p \setminus \{\xi_{k,j}\}_{k<i}$ and updates $\mathcal{M}_j := \mathcal{M}_j \sqcup \{(\xi_{i,j}, a_i)\}$ where $\sqcup$ means that the pair $(\xi_{i,j}, a_i)$ is appended to the list $\mathcal{M}_j$ with associated index i. For each $j \in \{1, 2, T\}$, these lists $\mathcal{M}_j$ are initially empty and the challenger inserts uniformly at random chosen $\xi_{0,j} \leftarrow E_p$ strings for to represent the generators $g_j \in \mathbb{G}_j$, i.e., it updates $\mathcal{M}_j := \mathcal{M}_j \sqcup \{(\xi_{0,j}, 1)\}$ for each j. By abuse of notation, we may sometimes write $a_i \in \mathcal{M}_j$ or $\xi_{i,j} \in \mathcal{M}_j$ to mean $(\xi_{i,j}, a_i) \in \mathcal{M}_j$.

If the adversary calls the group operation oracle GC_j for an $j \in \{1, 2, T\}$ on input (ξ, ξ', b) where $b \in \{0, 1\}$ is a bit (which indicates if the actual group operation $A \cdot A'$ should be computed or the inverse $A \cdot A^{-1}$), the challenger returns $\perp$ if $\xi \notin \mathcal{M}_j$ or $\xi' \notin \mathcal{M}_j$ (because the group operation can only be queried on already known group elements). Otherwise, it updates the group operation counter $c_j := c_j + 1$ and returns $\mathsf{Enc}_j(a_k)$ where $a_k := a_i + (-1)^b a_i'$ where $k := c_j$ and a_i, a_i' are the representatives for ξ, ξ', respectively. This captures the idea that the challenger keeps track of already returned (encodings of) group elements so that it does not assign different values $\xi_{i,j} \neq \xi_{i,j}'$ to the same group element. Further, if the adversary calls the pairing oracle on input $(\xi_{i,1}, \xi_{j,2})$ corresponding to the pair $(g_1^{a_i}, g_2^{a_j}) \in \mathbb{G}_1 \times \mathbb{G}_2$, the challenger returns $\perp$ if $\xi_{i,1} \notin \mathcal{M}_1$ or $\xi_{j,2} \notin \mathcal{M}_2$ (one of the two input elements are not assigned and thus unknown to the adversary at this point). Otherwise, it updates $c_T := c_T + 1$ and returns $\mathsf{Enc}_T(a_i a_j)$ where the value $a_i a_j \in \mathbb{Z}_p$ is considered modulo p as usual. If the adversary calls the challenge oracle Chal, update the challenge counter $n := n + 1$, sample a challenge $x_n \leftarrow \mathbb{Z}_p$ uniformly at random, update the group operation counters $c_1 := c_1 + 1$ and $c_2 := c_2 + 1$ (since the challenge is provided in both source groups $\mathbb{G}_1$ and $\mathbb{G}_2$), return $\mathsf{Enc}_1(a_{c_1})$ and $\mathsf{Enc}_2(a_{c_2})$ where $a_{c_1} := x_n$ and $a_{c_2} := x_n$, and update the challenge vector $\vec{x} \sqcup \{x_n\}$ (append the element x_n at the end of the vector $\vec{x}$). This captures the idea that the challenger samples a new random challenge x_n (resulting in the total challenge $(x_1, \ldots, x_n)$ up to that point), and returns (an encoding of) the corresponding group elements in $\mathbb{G}_1$ and $\mathbb{G}_2$. Finally, if the adversary calls the discrete logarithm oracle $\mathsf{DL}_g()$ on input ξ (which operates in the group $\mathbb{G}_1$ only), return $\perp$ if $\xi \notin \mathcal{M}_1$ (that is, the element ξ is unknown to the adversary or not in the group $\mathbb{G}_1$). Otherwise, update the discrete logarithm oracle counter $q := q + 1$, set $v := a_i$ where $i = \min_{k \in [\![c_1]\!]}\{\xi = \xi_{k,1}\}$, and return v. This captures the idea that the challenger returns the to ξ already assigned value a_i.

The following game is where we (acting as the challenger) introduce polynomials and the set of polynomials $L \subset \mathbb{Z}_p[X_1, \ldots, X_n]$ that encodes the information the adversary gets through the discrete logarithm oracle queries. Initially, the set L is empty. For instance, if the adversary queries $\mathsf{DL}_g()$ the first time on input $\Xi_1(X_1)$ corresponding to the group element g^{X_1}, we return a uniformly random $x_1 \leftarrow \mathbb{Z}_p$ and update $L := L \cup \{X_1 - x_1\}$. If the adversary queries $\mathsf{DL}_g()$ the second time on input $\Xi_1(3X_1 + 4)$, we are supposed to return the element $3x_1 + 4$. For this, we simply consider the $\mathbb{Z}_p$-subspace $\mathsf{Span}(1, L) \subset \mathbb{Z}_p^n[X_1, \ldots, X_n]$ spanned by 1 and the polynomials in L. The composition $3X_1 + 4 = (3x_1 + 4) + 3(X_1 - x_1)$ lets us return the value $3x_1 + 4$ if $\mathsf{DL}_g()$ is queried on $3X_1 + 4$. This strategy just ensures that we do not sample a new uniform value $x_2 \leftarrow \mathbb{Z}_p$ to answer the discrete logarithm query on $\Xi_1(3X_1 + 4)$. In that case, the adversary would detect incorrect behavior directly. Recollecting the idea of the GGM proof for the discrete logarithm problem, the simulation aborts when there is a pair $P_i \neq P_j$ such that $P_i(\vec{x}) = P_j(\vec{x})$. For the OMDL problem, the simulation of Bauer et al. allow the existence of pairs $P_i \neq P_j$ for which the current knowledge of $\vec{x}$ allows to deduce $P_i(\vec{x}) = P_j(\vec{x})$ (this is necessary, since the adversary adaptively obtains information on $\vec{x}$ and thus can construct such

polynomials). However, if $P_i - P_j \notin \mathsf{Span}(L)$ and still $P_i(\vec{x}) = P_j(\vec{x})$, then the simulation should abort. On the other hand, if $P_i - P_j \in \mathsf{Span}(L)$, then obviously $P_i(\vec{x}) = P_j(\vec{x})$. Hence, the event $P_i(\vec{x}) = P_j(\vec{x})$ can be phrased as $P_i - P_j \in \mathsf{Span}(L)$ in the OMDL game. As a further consequence, if a new polynomial P_j (which replaces the scalar a_j that corresponds to the group element g^{a_j}) is getting encoded, their simulation sets $\xi_j := \xi_i$ if there is an index $i \in [\![j-1]\!]$ such that $P_j - P_i \in \mathsf{Span}(L)$ (since this corresponds to the event $P_i(\vec{x}) = P_j(\vec{x})$ in the plain discrete logarithm game). On the other hand, the simulation fails if there is a pair of polynomials $P_i \neq P_j$ such that $P_i - P_j \in \mathsf{Span}(L)$ but $\xi_i \neq \xi_j$. Since we have now three groups and a pairing, this argument adapts slightly. For this, we let $P_{1,1}, \ldots, P_{m_1,1}$ be the polynomials the adversary constructs in the group $\mathbb{G}_1$, and $P_{1,2}, \ldots, P_{m_2,2}$ the ones that it constructs in $\mathbb{G}_2$, and $P_{1,T}, \ldots, P_{m_T,T}$ the ones that it obtains in $\mathbb{G}_T$. Note that the $P_{i,1}$ and $P_{i,2}$ are polynomials of degree 1, but the $P_{i,T}$ are polynomials of degree 1 or 2 because of the pairing. In this case, the simulation fails if there is an $\ell \in \{1, 2, T\}$ with a pair of polynomials $P_{i,\ell} \neq P_{j,\ell}$ such that $P_{i,\ell} - P_{j,\ell} \in \mathsf{Span}(L)$ but $\xi_{i,\ell} \neq \xi_{j,\ell}$. This will be the idea for the final game. But before that, we will formally define an intermediate game that tells the simulation to abort when it finds a collision among some $P_{i,\ell} \neq P_{j,\ell}$ evaluated at $\vec{x}$ while $P_{i,\ell} - P_{j,\ell} \notin \mathsf{Span}(L)$. For this game, we will derive a technical lemma that bounds the probability of simulation failure and thus bounds the statistical distance between games $\mathbf{G}_0$ and $\mathbf{G}_1$. Afterwards, we will define the final game and show that the intermediate and final game are equally distributed. Having obtained that, we upper-bound the probability of the adversary in winning the final game.

Game $\mathbf{G}_1$: This is the intermediate game. We introduce polynomials $P_{i,j}$ that replace the scalars a_i, the set of polynomials $L \subset \mathbb{Z}_p[X_1, \ldots, X_n]$ that encodes the information the adversary gets through the discrete logarithm oracle queries, and the abort condition that tells the simulation to abort if it finds a collision among a pair of distinct polynomials whose difference does not lie in the span $\mathsf{Span}(L)$ of L. Apart from that, there is no difference to the previous game. For this reason, we will keep the following description short. The challenger C initializes as usual $n := 0$, $q := 0$, $c_j := 0$ for each $j \in \{1, 2, T\}$, $q_e := 0$, $\vec{x} := ()$. Additionally, the challenger initializes polynomials $P_{0,j} := 1$ for $j \in \{1, 2, T\}$ instead of scalars $a_{0,j}$ and an initially empty set $L := \emptyset$. The adversary has access to the oracles Chal, $\mathsf{DL}_g()$, GC_j for $j \in \{1, 2, T\}$, and $e(-,-)$. For the encoding functions Enc_j, the challenger has its bookkeeping lists $\mathcal{M}_j$ for $j \in \{1, 2, T\}$ that keep track of already assigned encodings $\xi_{i,j} := \Xi_j(P_{i,j})$ for a polynomial $P_{i,j} \in \mathbb{Z}_p[X_1, \ldots, X_n]$ corresponding to the group element $g_j^{P_{i,j}} \in \mathbb{G}_j$. Additionally, the challenger C checks for each new request $\mathsf{Enc}_j(P_{\ell,j})$ if there exists an already assigned polynomials $P_{i,j}$ with $i \in [\![\ell - 1]\!]$ such that $P_{\ell,j} - P_{i,j} \in \mathsf{Span}(L)$. In that case, it sets $\xi_{\ell,j} := \xi_{i,j}$. (Otherwise, it sets $\xi_{\ell,j} \leftarrow E_p \setminus \mathcal{M}_j$ to a uniformly random string that has not already been assigned). If the adversary calls the challenge oracle Chal, update $n := n + 1$, sample a new indeterminate X_n and a value $x_n \leftarrow \mathbb{Z}_p$ uniformly at random (this value should help the simulation to abort directly when it finds a collision), update $c_1 := c_1 + 1$ and $c_2 := c_2 + 1$ (since the challenge is provided in both source groups $\mathbb{G}_1$ and $\mathbb{G}_2$), return $\mathsf{Enc}_1(P_{c_1,1})$ and $\mathsf{Enc}_2(P_{c_2,2})$ where $P_{c_1,1} := X_n$ and $P_{c_2,2} := X_n$, and update $\vec{x} \sqcup \{x_n\}$. On the other hand, if there is a polynomial $P_{i,j}$ with $i \in [\![\ell - 1]\!]$ such that $P_{\ell,j} - P_{i,j} \notin \mathsf{Span}(L)$ but with a collision $P_{\ell,j}(\vec{x}) = P_{i,j}(\vec{x})$, then the simulation aborts the game. We will after the definition of this game bound the probability that this happens. For $j \in \{1, 2\}$ with groups $\mathbb{G}_1, \mathbb{G}_2$, the polynomials are of degree 1 so that the technical lemma of Bauer et al. applies. However, in the case of $\mathbb{G}_T$, these polynomials can be of degree 2, so that we have to come up with a new lemma that captures also quadratic polynomials. To upper-bound

the final probability that a collision occurs in any of the three groups, we simply sum up the individual probabilities.

Continuing with the game, if the adversary calls the group operation oracle GC_j for an $j \in \{1, 2, T\}$ on input (ξ, ξ', b), the challenger updates $c_j := c_j + 1$ and returns $\mathsf{Enc}_j(P_{k,j})$ where $P_{k,j} := P_{i,j} + (-1)^b P_{i',j}$ where $k := c_j$ and $P_{i,j}, P_{i',j}$ are the representatives for ξ, ξ', respectively. (In case one of the inputs is invalid/not already assigned, the challenger returns $\perp$ as usual). If the adversary calls the pairing oracle on input $(\xi_{i,1}, \xi_{j,2})$, the challenger returns $\perp$ or it updates $c_T := c_T + 1$ and returns $\mathsf{Enc}_T(P_{i,1}P_{j,2})$. We point out that this is exactly the point where the quadratic polynomials appear. Finally, if the adversary calls the discrete logarithm oracle $\mathsf{DL}_g()$ on input ξ (which operates in the group $\mathbb{G}_1$ only), return $\perp$ if the input is invalid and otherwise do the following. Let $i \in [\![c_1]\!]$ be such that $\xi = \xi_{i,1}$ with corresponding polynomial $P_{i,1}$. Set $v := P_{i,1}(\vec{x})$. If $P_{i,1} \in \mathsf{Span}(1, L)$, then let $P_{i,1} = \alpha_0 + \alpha_1 Q_1 + \ldots + \alpha_{q-1} Q_{q-1}$ be the decomposition of $P_{i,1}$ in the $\mathbb{Z}_p$-subspace $\mathsf{Span}(1, L)$, where the coefficients are $\alpha_k \in \mathbb{Z}_p$, and update $v := \alpha_0$. Return v, set $Q_q := P_{i,1} - v$ and $L := L \cup \{Q_q\}$. This captures the idea that the challenger should be consistent with its answers to discrete logarithm oracle queries in case the adversary queries for redundant information (if A asks for the discrete logarithm of X_1 to get x_1, and then asks for $3X_1$, the challenger should always return $3x_1$).

In the following, we will upper-bound the probability that the simulation aborts the game. That is, we will bound the probability that for an $j \in \{1, 2, T\}$ there is a pair of polynomials $P_{\ell,j} \neq P_{i,j}$ such that $P_{\ell,j} - P_{i,j} \notin \mathsf{Span}(L)$ but $P_{\ell,j}(\vec{x}) = P_{i,j}(\vec{x})$. For this, we will need two lemmas. We introduce some notation. For a polynomial $P \in \mathbb{Z}_p[X_1, \ldots, X_n]$, we let the corresponding calligraphic $\mathcal{P}$ denote the zero set of the polynomial in $\mathbb{Z}_p^n$, i.e., $\mathcal{P} := \{\vec{x} \in \mathbb{Z}_p^n \mid P(\vec{x}) = 0\}$.

Lemma 4.6.3. *Let $D_1, \ldots, D_m, Q_1, \ldots, Q_{q+1} \in \mathbb{Z}_p[X_1, \ldots, X_n]$ be polynomials of degree 1 (i.e., linear polynomials). Also let*

$$C := \Big(\bigcap_{i \in [q]} \mathcal{Q}_i\Big) \setminus \Big(\bigcup_{i \in [m]} \mathcal{D}_i\Big)$$

be the set of points at which all polynomials Q_i vanish but none of the polynomials D_i do. Assume that $\mathcal{Q}_{q+1} \cap C \neq \emptyset$ and that $(\mathcal{Q}_1 \cap \ldots \cap \mathcal{Q}_q) \not\subseteq \mathcal{Q}_{q+1}$. If $\vec{x} \in \mathbb{Z}_p^n$ is sampled uniformly at random from the set C, then

$$\frac{p-m}{p^2} \leq \Pr[Q_{q+1}(\vec{x}) = 0] \leq \frac{1}{p-m}.$$

Proof. See Bauer et al. [BFP21] on page 9. □

Lemma 4.6.4. *Let $Q_1, \ldots, Q_q \in \mathbb{Z}_p[X_1, \ldots, X_n]$ be polynomials of degree 1, and let $D_1, \ldots, D_m, Q_{q+1} \in \mathbb{Z}_p[X_1, \ldots, X_n]$ be polynomials of degree $d_i \in \{1, 2\}$ (i.e., linear or quadratic polynomials). Also let*

$$C := \Big(\bigcap_{i \in [q]} \mathcal{Q}_i\Big) \setminus \Big(\bigcup_{i \in [m]} \mathcal{D}_i\Big)$$

be the set of points at which all polynomials Q_i vanish but none of the polynomials D_i do. Assume that $\mathcal{Q}_{q+1} \cap C \neq \emptyset$ and that $(\mathcal{Q}_1 \cap \ldots \cap \mathcal{Q}_q) \not\subseteq \mathcal{Q}_{q+1}$. If $\vec{x} \in \mathbb{Z}_p^n$ is sampled uniformly at random from the set C, then

$$\frac{p-4m}{p^2} \leq \Pr[Q_{q+1}(\vec{x}) = 0] \leq \frac{4}{p-4m}.$$

Proof. Since $\vec{x}$ is sampled uniformly at random from the above non-empty set C, it is $\Pr[\vec{x} \in Q_{q+1}] = |Q_{q+1} \cap C|/|C|$. In the following, we bound both these set sizes. We begin with C. In the following, we write $\mathbb{F}_p := \mathbb{Z}_p$ for the field of p elements.

We let $Q := \cap_{i\in[q]} Q_i$ be the affine space with dimension d (note that the polynomials $Q_1, \ldots, Q_q$ are of degree 1). By assumption we know that Q is non-empty, otherwise C would be empty. In particular, Q contains p^d elements. We write the set C as

$$C = Q \setminus \Big(\bigcup_{i\in[m]} (Q \cap \mathcal{D}_i) \Big).$$

For an $i \in [m]$, we want to upper-bound the size of the set $Q \cap \mathcal{D}_i$ and at the end sum up over all $i \in [m]$ to obtain a bound on C. For this, we fix an $i \in [m]$ and consider $Q \cap \mathcal{D}_i$. There are two cases two consider: (i) the polynomial D_i is of degree 1, and (ii) the polynomial D_i is of degree 2. The first case is easy to handle, which we do now. Obviously, Q cannot be contained in $\mathcal{D}_i$, otherwise we would get $C = \emptyset$. Therefore, either $Q \cap \mathcal{D}_i = \emptyset$ or $Q \cap \mathcal{D}_i$ is of dimension $d - 1$ (since D_i defines a hyperplane in $\mathbb{F}_p^n$). In both cases, the space contains at most p^{d-1} elements. And as a result, we obtain by summing over all $i \in [m]$ the bound

$$p^d - mp^{d-1} \leq |C| \leq p^d. \tag{1}$$

The second case (ii) is more complicated, since D_i does not define a hyperplane anymore but a hypersurface of degree 2 and linear algebraic methods do not apply anymore (as was done above for the case where D_i is of degree 1). Additionally, since D_i defines a multivariate polynomial of degree 2, its vanishing set intersection with Q can contain more points if D_i was of degree 1. As a result, the union over the sets $Q \cap \mathcal{D}_i$ could be close to Q so that their set-theoretic difference C could be very small (e.g., of size 1). This, however, would result in a lower-bound $|C| \geq 1$ which would have a devastating effect on our probability bound, since that would yield $\Pr[\vec{x} \in Q_{q+1}] = |Q_{q+1} \cap C|/|C| \leq 1$, giving us no way to properly bound the probability of simulation failure in our security game. Our intuition, however, tells us that the set $Q \cap \mathcal{D}_i$ for a degree 2 polynomial D_i should only be about twice (or some other constant factor) as big as if D_i was of degree 1. Indeed, we back up this intuition by escaping the realm of linear algebra to enter the realm of algebraic geometry. In the following, let D_i be a polynomial of degree 2 and let $\overline{\mathbb{F}}_p$ be the algebraic closure of the field $\mathbb{F}_p$. We will pass to the n-dimensional projective space $\mathbb{P}^n(\overline{\mathbb{F}}_p)$ over the field $\overline{\mathbb{F}}_p$ to analyze the zero set $Q \cap \mathcal{D}_i$. In order to do so, the polynomials $Q_1, \ldots, Q_q$ and D_i are homogenized and considered in $\overline{\mathbb{F}}_p[X_1, \ldots, X_n, X_0]$ where X_0 is a new variable that homogenizes the polynomials. Now we can do algebraic geometry. We denote by $d \leq n - q$ the dimension of the variety Q which is simply a complete intersection of hyperplanes. Again, Q cannot be contained in $\mathcal{D}_i$, otherwise we would get $C = \emptyset$. Therefore, either it is $Q \cap \mathcal{D}_i = \emptyset$ or $Q \cap \mathcal{D}_i$ defines a projective variety of dimension $d - 1$. The reason for the latter is: Since $Q \not\subseteq \mathcal{D}_i$ and $Q \cap \mathcal{D}_i \neq \emptyset$, the set $Q \cap \mathcal{D}_i$ is a proper closed subvariety of Q and thus of dimension $< d$ (see any textbook on algebraic geometry, e.g., Hartshorne [Har77]). On the other hand, since any new equation cuts out at most one dimension, and $\mathcal{D}_i$ is defined by one equation, the intersection $Q \cap \mathcal{D}_i$ drops by at most one dimension (cf. Gathmann [Gat21]). Since we want to upper bound the size of $Q \cap \mathcal{D}_i$, we ignore the case $Q \cap \mathcal{D}_i = \emptyset$ of empty intersection (anyway, this gives the trivial bound $|Q \cap \mathcal{D}_i| \leq 0$). An important tool in the study of $\overline{\mathbb{F}}_p$-rational points on an algebraic variety is Bezout's inequality [Cat92; Ful98] which states

that for two (projective) varieties V and W, the following inequality holds:

$$\deg(V \cap W) \leq \deg(V) \cdot \deg(W).$$

For our variety $Q \cap \mathcal{D}_i$ this gives a degree of at most 2, since Q is defined by polynomials of degree 1 and $\mathcal{D}_i$ by a polynomial of degree 2. A long line of works in mathematics has studied estimates on the number of $\mathbb{F}_p$-rational points on a projective variety $V \subset \mathbb{P}^n$, which even goes back to the fundamental work of Deligne [Del74]. Extending classical results [LN96], Cafure and Matera [CM07] find an elementary bound on this number depending on the degree δ and dimension r of V.

Theorem 4.6.5. *Let $V \subset \mathbb{P}^n(\overline{\mathbb{F}}_p)$ be a projective variety of dimension r and degree δ. Then the following estimate on the number $V(\mathbb{F}_p)$ of $\mathbb{F}_p$-rational points on V holds:*

$$|V(\mathbb{F}_p)| \leq \delta \cdot |\mathbb{P}^r(\mathbb{F}_p)| = \delta(p^r + \ldots + p + 1).$$

In our case, we have for the degree $\delta = \deg(Q \cap \mathcal{D}_i) \leq 2$ and for the dimension $r = \dim(Q \cap \mathcal{D}_i) = d - 1$. Given the above bound, we find the resulting bound over $\mathbb{F}_p$,

$$|Q \cap \mathcal{D}_i| \leq \delta(p^{d-1} + \ldots + p + 1) \leq 2\delta p^{d-1} \leq 4p^{d-1}.$$

As a result, we obtain by summing over all $i \in [m]$ the bound

$$p^d - 4mp^{d-1} \leq |C| \leq p^d. \tag{2}$$

Next, we want to bound the size $|Q_{q+1} \cap C|$. We define the intersection set $Q' := Q \cap Q_{q+1}$. Again, there are two cases two consider: (i) the polynomial Q_{q+1} is of degree 1, and (ii) the polynomial Q_{q+1} is of degree 2. The first case is easy to handle, which we do now. By assumption $Q \not\subseteq Q_{q+1}$, and therefore Q_{q+1} cuts out one dimension of Q, i.e., $\dim(Q') = d - 1$ where as before $d := \dim(Q)$. For a fixed $i \in [m]$, using the same techniques as before applied to Q' now (instead of Q), we find the bound

$$p^{d-1} - 4mp^{d-2} \leq |Q_{q+1} \cap C| \leq p^{d-1}. \tag{3}$$

In the second case, we directly pass to the projective space $\mathbb{P}^n$ over the algebraic closure $\overline{\mathbb{F}}_p$. The only difference now is that we directly apply Theorem 4.6.5 to the variety Q' (which is now of degree 2 by Bezout) and then to the varieties $Q' \cap \mathcal{D}_i$ (which is now of degree 4 by Bezout). With the updated degrees of the respective varieties, the same calculation as before gives

$$4p^{d-1} - 8mp^{d-2} \leq |Q_{q+1} \cap C| \leq 4p^{d-1}. \tag{4}$$

Taking all bounds (1)-(4) together, we finally obtain

$$\begin{aligned} p^d - 4mp^{d-1} &\leq |C| \leq p^d, \\ p^{d-1} - 4mp^{d-2} &\leq |Q_{q+1} \cap C| \leq 4p^{d-1}. \end{aligned}$$

Combining these identities, we obtain

$$\begin{aligned} &\frac{p^{d-1} - 4mp^{d-2}}{p^d} \leq \frac{|Q_{q+1} \cap C|}{|C|} \leq \frac{4p^{d-1}}{p^d - 4mp^{d-1}} \\ \iff \quad &\frac{p - 4m}{p^2} \leq \frac{|Q_{q+1} \cap C|}{|C|} \leq \frac{4}{p - 4m}. \end{aligned}$$

This completes the proof of our technical lemma. □

We continue with our sequence of games. We now compare the statistical distance of games $\mathbf{G}_0$ and $\mathbf{G}_1$. Recall that the simulation in game $\mathbf{G}_1$ aborts when it finds a collision among a pair of distinct polynomials in one of the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$. More concretely, let us for $j \in \{1, 2, T\}$ define the event F_j as: There exists an $i \in [\![c_j - 1]\!]$ such that $P_{c_j,j}(\vec{x}) = P_{i,j}(\vec{x})$ and $P_{c_j,j} - P_{i,j} \notin \mathsf{Span}(L)$. As the group operation oracles are called at most m times, we get

$$\mathbf{Adv}^{\mathsf{A}}_{\mathbf{G}_0} \leq \mathbf{Adv}^{\mathsf{A}}_{\mathbf{G}_1} + m \Pr[F_1] + m \Pr[F_2] + (m + q_e) \Pr[F_3]. \tag{♠}$$

We upper-bound the probability $\Pr[F_j]$ that event F_j happens. Before a call to Enc_j, the oracle defines $P_{c_j,j}$. We fix $j \in \{1, 2, T\}$ and consider a fixed $i \in [\![c_j - 1]\!]$ and let $P_j := P_{c_j,j} - P_{i,j}$. We want to use our two technical lemmas to bound the probability that $P_j(\vec{x}) = 0$ while $P_j \notin \mathsf{Span}(L)$. Therefore, we let $L := \{Q_1, \ldots, Q_q\}$ with the Q_i's being defined as in $\mathbf{G}_1$ and $Q_{q+1} := P_j$. According to Bauer et al., we further observe that the adversary A knows $P_{i_1,j}(\vec{x}) \neq P_{i_2,j}(\vec{x})$ when $\xi_{i_1,j} \neq \xi_{i_2,j}$. By writing $D_{\vec{i}} := P_{i_1,j} - P_{i_2,j}$ for $\vec{i} \in I := \{(i_1, i_2) \in [\![c_j - 1]\!]^2 \mid \xi_{i_1,j} \neq \xi_{i_2,j}\}$, we know that A knows $D_{\vec{i}}(\vec{x}) \neq 0$. Using previously defined notation, we get that

$$\vec{x} \in C := \Big(\bigcap_{i \in [q]} \mathcal{Q}_i \Big) \setminus \Big(\bigcup_{i \in I} \mathcal{D}_i \Big).$$

The same argumentation as in [BFP21] on page 17 ff. allows us to use our technical lemmas. Since for $j \in \{1, 2\}$ in the group $\mathbb{G}_j$ all appearing polynomials are of degree 1, we can directly apply the more straightforward Lemma 4.6.3. In the case $j = T$ for the group $\mathbb{G}_T$ the appearing polynomials can be of degree 2, so that we need to apply the more complicated Lemma 4.6.4. We will now apply these lemmas. For $j \in \{1, 2\}$, we obtain

$$\Pr[P_j(\vec{x}) = 0] \leq \frac{1}{p - m^2},$$

since there are at most m^2 elements in the set I which define the polynomials $D_{\vec{i}}$. Since we fixed an $i \in [\![c_j - 1]\!]$ to define P_j, we get the following when running over all such indices $i \leq m$:

$$\Pr[F_1] \leq \frac{m}{p - m^2}, \qquad \Pr[F_2] \leq \frac{m}{p - m^2}.$$

On the other hand, for $j = T$ in the group $\mathbb{G}_T$ with quadratic polynomials, we obtain

$$\Pr[P_j(\vec{x}) = 0] \leq \frac{4}{p - 4(m + q_e)^2},$$

since the queries to the pairing operation give additional q_e polynomials (i.e., the set I is of size $(m + q_e)^2$). Running over all possible $i \leq (m + q_e)$, we obtain

$$\Pr[F_T] \leq \frac{4(m + q_e)}{p - 4(m + q_e)^2}.$$

Combining all these identities, (♠) then reduces to

$$\mathbf{Adv}^{\mathsf{A}}_{\mathbf{G}_0} \leq \mathbf{Adv}^{\mathsf{A}}_{\mathbf{G}_1} + \frac{2m^2}{p - m^2} + \frac{4(m + q_e)^2}{p - 4(m + q_e)^2}.$$

We end with the final game $\mathbf{G}_2$ which is described as $\mathbf{G}_4$ in Bauer et al.'s proof. In this game, we completely get rid of actual values for the challenge $\vec{x} = (x_1, \ldots, x_n)$ and only use polynomials. The only use of $\vec{x}$ in the previous game $\mathbf{G}_1$ was in the abort conditions in the encoding function and how to sample the answer v to an discrete logarithm query. However, as already noted previously, we can transfer these events to the discrete logarithm oracle by letting the simulation to abort when it finds a collision among a pair of polynomials $P_{i,\ell} \neq P_{j,\ell}$ such that $P_{i,\ell} - P_{j,\ell} \in \mathsf{Span}(L)$ while $\xi_{i,\ell} \neq \xi_{j,\ell}$ for an $\ell \in \{1, 2, T\}$. As observed by Bauer et al., the abort conditions in $\mathbf{G}_1$ and $\mathbf{G}_2$ are indeed equivalent from the view of the adversary. Furthermore, the authors also show why it is possible to sample $v \leftarrow \mathbb{Z}_p$ uniformly at random when there is no $\vec{x}$ involved anymore. Therefore, we can define the final game now.

Game $\mathbf{G}_2$: This is the final game. We will keep the following description short (since there is barely a difference to the previous game). The challenger C initializes as usual $n := 0$, $q := 0$, $c_j := 0$ for each $j \in \{1, 2, T\}$, $q_e := 0$, $\vec{x} := ()$. Additionally, the challenger initializes polynomials $P_{0,j} := 1$ for $j \in \{1, 2, T\}$ and $L := \emptyset$. The adversary has access to the oracles Chal, $\mathsf{DL}_g()$, GC_j for $j \in \{1, 2, T\}$, and $e(-, -)$. For the encoding functions Enc_j, the challenger has its bookkeeping lists $\mathcal{M}_j$ for $j \in \{1, 2, T\}$ that keep track of already assigned encodings $\xi_{i,j} := \Xi_j(P_{i,j})$ for a polynomial $P_{i,j}$. The challenger C checks for each new request $\mathsf{Enc}_j(P_{\ell,j})$ if there exists an already assigned polynomials $P_{i,j}$ with $i \in [\![\ell - 1]\!]$ such that $P_{\ell,j} - P_{i,j} \in \mathsf{Span}(L)$. In that case, it sets $\xi_{\ell,j} := \xi_{i,j}$. (Otherwise, it sets $\xi_{\ell,j} \leftarrow E_p \setminus \mathcal{M}_j$ to a uniformly random string that has not already been assigned). If the adversary calls the challenge oracle Chal, update $n := n + 1$, sample a new indeterminate X_n, update $c_1 := c_1 + 1$ and $c_2 := c_2 + 1$, return $\mathsf{Enc}_1(P_{c_1,1})$ and $\mathsf{Enc}_2(P_{c_2,2})$ where $P_{c_1,1} := X_n$ and $P_{c_2,2} := X_n$. Furthermore, if the adversary calls the group operation oracle GC_j for an $j \in \{1, 2, T\}$ on input (ξ, ξ', b), the challenger updates $c_j := c_j + 1$ and returns $\mathsf{Enc}_j(P_{k,j})$ where $P_{k,j} := P_{i,j} + (-1)^b P_{i',j}$ where $k := c_j$ and $P_{i,j}, P_{i',j}$ are the representatives for ξ, ξ', respectively. (In case one of the inputs is invalid/not already assigned, the challenger returns $\perp$ as usual). If the adversary calls the pairing oracle on input $(\xi_{i,1}, \xi_{j,2})$, the challenger returns $\perp$ or it updates $c_T := c_T + 1$ and returns $\mathsf{Enc}_T(P_{i,1}P_{j,2})$. Finally, if the adversary calls the discrete logarithm oracle $\mathsf{DL}_g()$ on input ξ (which operates in the group $\mathbb{G}_1$ only), return $\perp$ if the input is invalid and otherwise do the following. Let $i \in [\![c_1]\!]$ be such that $\xi = \xi_{i,1}$ with corresponding polynomial $P_{i,1}$. Sample $v \leftarrow \mathbb{Z}_p$ uniformly at random. If $P_{i,1} \in \mathsf{Span}(1, L)$, then let $P_{i,1} = \alpha_0 + \alpha_1 Q_1 + \ldots + \alpha_{q-1} Q_{q-1}$ be the decomposition of $P_{i,1}$ in the $\mathbb{Z}_p$-subspace $\mathsf{Span}(1, L)$ and update $v := \alpha_0$. Set $Q_q := P_{i,1} - v$ and $L := L \cup \{Q_q\}$. If there is a pair of polynomials $P_{i,\ell} \neq P_{j,\ell}$ for $(i, j) \in [\![c_\ell]\!]^2$ such that $P_{i,\ell} - P_{j,\ell} \in \mathsf{Span}(L)$ and $\xi_{i,\ell} \neq \xi_{j,\ell}$ for $\ell \in \{1, 2, T\}$, then abort the game. Otherwise, return the value v.

To end the proof, we bound the probability that the adversary wins the final game. To this end, we show that there is a component of $\vec{x}$ that is sampled uniformly at random after the adversary outputs its solution $\vec{y}$ to the co-OMDL game. After A outputs $\vec{y}$, the set L is of size q and thus $\dim(\mathsf{Span}(L)) \leq q < n$. Therefore, $\mathsf{Span}(1, L)$ is of dimension $\leq q + 1 \leq n$. On the other hand, $\dim(\mathsf{Span}(X_1, \ldots, X_n)) = n$ while $1 \notin \mathsf{Span}(X_1, \ldots, X_n)$. Having said that, we obtain

$$\mathsf{Span}(X_1, \ldots, X_n) \not\subseteq \mathsf{Span}(1, L).$$

In particular, there is an index $i \in [n]$ such that $X_i \notin \mathsf{Span}(1, L)$. Let $i \in [n]$ be the smallest such index. The discrete logarithm oracle $\mathsf{DL}_g()$ outputs a value x_i uniformly at random when

queried on $\xi_{j_i,1}$. However, this x_i is sampled uniformly at random after the i-th component of $\vec{y}$ output by the adversary A. Hence, $\Pr[\vec{x} = \vec{y}] \leq 1/p$. Finally, this yields the winning probability

$$\mathbf{Adv}^{\mathsf{A}}_{\mathbf{G}_2} \leq \frac{1}{p}.$$

Taking together the obtained distances between sequential games, gives us the desired final bound

$$\varepsilon \leq \frac{2m^2}{p - m^2} + \frac{4(m + q_e)^2}{p - 4(m + q_e)^2} + \frac{1}{p}.$$

Finally, we remark that the proof naturally extends to the setting in which the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ might be of different orders $p > q$ and the discrete logarithm oracle $\mathsf{DL}_g()$ is given in the larger group $\mathbb{G}_1$. This is clear, since the only time we rely on the groups having same order is when $x_i \leftarrow \mathbb{Z}_p$ is sampled to instantiate the indeterminate X_i. However, by sampling $x_i \leftarrow \mathbb{Z}_p$ and just reducing it modulo q so that $[x_i] \in \mathbb{Z}_q$, the simulation remains correct. □

5

DKG and Distributed Randomness Beacon with (Near-)Quadratic Communication Complexity

Details on this Chapter

This chapter is based on the conference publication [Bac+24a] and its full version [Bac+23b]. I contributed to it as the main author. The implementation for Section 5.6 was carried out by my co-authors Dimitrios Papachristoudis and Simon Ochsenreither, under my guidance. The original publication has been partially reordered, and editorial improvements have been made throughout the text.

A *randomness beacon* is a source of continuous and publicly verifiable randomness which is of crucial importance for many applications. Existing works on randomness beacons suffer from at least one of the following drawbacks: (i) security only against static (i.e., non-adaptive) adversaries, (ii) each epoch takes many rounds of communication, or (iii) computationally expensive tools such as proof-of-work (PoW) or verifiable delay functions (VDF). In this work, we introduce GRandLine, the first adaptively secure randomness beacon protocol that overcomes all these limitations while preserving simplicity and optimal resilience in the synchronous network setting.

We achieve our result in two steps. First, we design a novel distributed key generation (DKG) protocol GRand that runs in $O(\lambda n^2 \log n)$ bits of communication but, unlike most conventional DKG protocols, outputs both secret and public keys as group elements. Here, λ denotes the security parameter. Second, following termination of GRand, parties can use their keys to derive a sequence of randomness beacon values, where each random value costs only a single asynchronous round and $O(\lambda n^2)$ bits of communication. We implement GRandLine and evaluate it using a network of up to 64 parties running in geographically distributed AWS instances. Our evaluation shows that GRandLine can produce about 2 beacon outputs per second in a network of 64 parties. We compare our protocol to the state-of-the-art randomness beacon protocols OptRand (NDSS 2023), BRandPiper (ACM CCS 2021), and Drand, in the same setting and observe that it vastly outperforms them.

5.1 Introduction

Distributed randomness plays a crucial role in many cryptographic and distributed system applications. A randomness beacon [Bha+21; Das+22a; Bha+23] is a source of *public*, *unpredictable*, and *unbiased random* values that can be used by anyone in a secure manner. A well-designed randomness beacon protocol ensures that random values are generated in a decentralized and secure manner, preventing a threshold of t out of n collaborating parties from biasing or predicting the random outputs. Randomness beacon protocols have wide-ranging applications in both academia and industry. In many consensus protocols [Yin+19; HMW18] they are used to securely choose a leader or a committee among participating parties (e.g., through a verifiable random function [Gil+17a]) to perform specific tasks. In this manner, these protocols can achieve better efficiency and circumvent impossibility results that apply to their deterministic counterparts. In mix networks [DMS04], randomness beacons are used to shuffle messages and thus provide unlinkability between sender and receiver of a message. In privacy-oriented cryptocurrencies and voting systems [HMW18; Mö+18], randomness beacons provide user anonymity and unlinkability to their actions. In recent years, randomness beacons have attracted significant interest and numerous protocols have been proposed [CMB23; KWJ23]. However, existing randomness beacon protocols found in the literature suffer from at least one of the following drawbacks: (i) they achieve security only against static (i.e., non-adaptive) adversaries, (ii) each epoch takes many rounds of communication, or (iii) they rely on computationally expensive tools such as proof-of-work (PoW) or verifiable delay functions (VDF).

5.1.1 Our Contributions

Motivated by this unsatisfactory state of affairs, we give a novel randomness beacon protocol GRandLine which improves upon the state-of-the-art by combining, for the first time, *all* of the following properties:[1]

- *Adaptive Security.* We prove GRandLine secure in the presence of an adaptive adversary. Many other protocols [Syt+17; Sch+20; Gil+17a] are only proven statically secure.

- *One-Round Epoch.* Each epoch in GRandLine takes only a single asynchronous round of communication and is non-interactive. It only requires synchrony for its pre-processing phase. This sets GRandLine apart from all other protocols except Drand [Dra20].

- *Communication-Efficient.* GRandLine has a communication cost of $O(\lambda n^2)$ bits per epoch, where λ is the security parameter. This sets GRandLine apart from BRandPiper [Bha+21] and RandShare [Syt+17], which have (worst-case) cubic or higher communication cost per epoch.

- *Optimal Resilience.* GRandLine has optimal resilience threshold $t < n/2$ (i.e., corruption threshold) in the synchronous network. This sets GRandLine apart from SPURT [Das+22a] and RandShare [Syt+17], which tolerate only suboptimal $t < n/3$.

- *Lightweight Tools.* After its pre-processing phase, GRandLine only uses lightweight cryptography such as hash functions and pairings. Notably, it does not rely on tools such as PoW or VDF. This sets GRandLine apart from RandChain [HYL20] and RandRunner [Sch+21], which rely on PoW and VDF, respectively.

- *Quadratic Pre-Processing.* GRandLine has a pre-processing phase with only $O(\lambda n^2 \log n)$ bits communication cost. This sets GRandLine apart from Drand [Dra20], OptRand [Bha+23], and BRandPiper [Bha+21], which have cubic or higher communication cost for pre-processing.

We achieve our result in two steps. First, we design a novel distributed key generation (DKG) protocol GRand with $O(\lambda n^2 \log n)$ bits communication cost that, unlike most of the conventional DKG protocols, outputs both secret and public keys as group elements. It is the first DKG protocol in any network setting that achieves subcubic communication cost with optimal resilience threshold. Second, we give a simple construction that allows to use the keys output by GRand for a non-interactive and unique *locally verifiable* threshold signature[2] from which we naturally derive a one-round randomness beacon using a final hash operation. For a detailed comparison of existing work on randomness beacon and DKG protocols, we refer to Table 5.1 and Table 5.2, respectively. Further, recent work [CMB23; KWJ23] give an excellent systematization of knowledge (SoK) for the extensive literature on randomness beacons.

[1] For the sake of clarity, we only compare to adaptively secure protocols in this list.

[2] By "locally verifiable" we mean that the final threshold signature does not have an efficient (independent of n) verification algorithm, but the partial signatures have.

Table 5.1: Comparison table of distributed randomness beacon (DRB) protocols.

Protocol	Network	Resil	Adapt	Unpred	Resp	Rounds	Commun	Crypto Primitive	Setup	Preprocess
Cachin et al. [CKS05]	*async*	1/3	✗	✓	✓	1	$O(\lambda n^2)$	Unique Th-Sig	*CRS*	$O(\lambda n^3)$
RandHerd [Syt+17]	*async*	1/3	✗	✓	✓	ABA	$O(\lambda c^2 \log n)$	PVSS, Th-Schnorr	*CRS*	$O(\lambda n^3)$
Herb [CSS19]	*sync*	1/3	✗	✓	✗	$t+1$	$O(\lambda n^3)$	Th-ElGamal	*CRS*	$O(\lambda n^3)$
Drand [Dra20]	*sync*	1/2	✗	✓	✓*	1	$O(\lambda n^2)$	Th-BLS	*CRS*	$O(\lambda n^3)$
SPURT [Das+22a]	*part sync*	1/3	✗	✓	✓	9	$O(\lambda n^2)$	PVSS, Pairings	*CRS*	✗†
Algorand [Gil+17a]	*part sync*	1/3	✗	$\Omega(t)$	✗	BC	$O(\lambda cn)$	VRF	*Seed*	$O(\lambda n^3)$
HydRand [Sch+20]	*sync*	1/3	✗	$t+1$	✗	3	$O(\lambda n^2)$	PVSS	*Seed*	$O(\lambda n^3)$
OptRand [Bha+23]	*sync*	1/2	✗	✓	✓	11	$O(\lambda n^2)$	PVSS, Pairings	*PoT*	$O(\lambda n^3)$
RandShare [Syt+17]	*async*	1/3	✓	✓	✓	ABA	$O(\lambda n^4)$	Async VSS	*CRS*	✗
SPURT [BL23c]	*part sync*	1/3	✓	✓	✓	9	$O(\lambda n^2)$	PVSS, Pairings	*CRS*	✗†
RandChain [HYL20]	*sync*	1/2	✓	$O(\lambda)$	✓	Δ_{PoW}	$O(\lambda n)$	PoW, VDF	*CRS*	✗◇
RandRunner [Sch+21]	*sync*	1/2	✓	$t+1$	✗	Δ_{VDF}	$O(\lambda n^2)$	Trapdoor VDF	*Seed*	$O(\lambda n^3)$
BRandPiper [Bha+21]	*sync*	1/2	✓	✓	✗	11	$O(\lambda f n^2)$	Multi-VSS	*PoT*	$O(\lambda f n^3)$
OptRand [BL23c]	*sync*	1/2	✓	✓	✓	11	$O(\lambda n^2)$	PVSS, Pairings	*PoT*	$O(\lambda n^3)$
Drand [BL22a]	*sync*	1/2	✓	✓	✓*	1	$O(\lambda n^2)$	Th-BLS	*CRS*	$O(\lambda n^3)$
GRandLine (ours)	*sync*	1/2	✓	✓	✓*	1	$O(\lambda n^2)$	PVSS, Pairings	*CRS*	$O(\lambda n^2 \log n)$

Resil denotes the Byzantine resilience threshold. **Adapt** denotes adaptive adversary. **Unpred** denotes unpredictability. **Resp** denotes responsiveness, i.e., progresses at the actual speed of the network.*In Drand and GRandLine, this is achieved only after the pre-processing phase. OptRand is responsive only when there are $t < n/4$ corrupt parties in the system. Asynchronous protocols are by default responsive. **Rounds** denotes the number of (a)synchronous rounds per epoch. In RandHerd and RandShare, parties run n asynchronous Byzantine agreement (ABA) instances in parallel which leads to expected $O(\log n)$ rounds per epoch. Algorand assumes a broadcast channel BC which leads to $t+1$ rounds per epoch when implemented with an actual broadcast protocol. In RandChain and RandRunner, each epoch takes one computational round to evaluate the VDF or PoW which is much larger than a synchronous network round. **Commun** denotes the communication cost in bits. In RandHerd and Algorand, c denotes the average size of a randomly chosen committee. In BRandpiper, $f \leq t$ denotes the actual number of faults in the system. **Crypto Primitive** denotes the cryptographic primitives in usage. **Setup** denotes the setup assumption. *CRS* denotes a common reference string setup. *Seed* denotes an initial random seed used to run the protocol. *PoT* denotes the powers-of-tau setup (Nik+22). **Preprocess** denotes the communication cost for pre-processing. This can either be a DKG, an SMR, or some other distributed protocol that generates the initial random seed. Since the protocols with an initial seed do not specify how to obtain it (other than by trusted setup), we assume the most efficient DKG for this task. † SPURT guarantees only a weak form of liveness: t out of n consecutive epochs might fail to produce an output. ◇ RandChain uses Nakamoto consensus and also suffers from blockchain-related attacks.

5.2 Technical Overview

The idea of using a threshold signature scheme with unique signatures (per message m and public key pk) and a non-interactive signing procedure is a well-known approach to generate one-round distributed randomness. It is most commonly used in consensus protocols and dates back to the seminal work of Cachin et al. [CKS05]. For epoch $e \geq 1$, this works as follows:

- Each party P_i non-interactively creates a signature share σ_i on the message $m := e$ and sends σ_i to all other parties.
- Upon receiving $t+1$ valid shares $\{\sigma_i\}_{i \in \mathcal{S}}$, a party locally reconstructs the full signature σ. The randomness beacon value is then computed as hash $O_e := \mathsf{H}(\sigma)$.

In this manner, one obtains a simple and efficient randomness beacon protocol which is also used by many blockchain and consensus protocols [Yin+19; HMW18]. This construction relies on a setup in which a secret key sk is (t, n)-secret shared among all parties. In a fully distributed system, this is commonly established via a DKG protocol for field elements [Can+99].

Concretely, this means that at the end of the DKG protocol each party P_i holds a secret key share $sk_i \in \mathbb{Z}_p$ such that $sk_i = f(i)$ for a polynomial $f \in \mathbb{Z}_p[X]$ of degree t. Further, the public key shares $pk_i := \omega^{sk_i} \in \mathbb{G}$ are publicly known where $\mathbb{G}$ is a prime order p group with generator ω. Unfortunately, even the most efficient DKG protocols [Shr+21a; Abr+23] incur a communication cost of $O(\lambda n^3)$ to generate their keys. So what does that mean for us?

5.2.1 Challenges and Our Techniques

We discuss the challenges in achieving the goal and our techniques to resolve these.

Challenges in Subcubic DKG. A common approach to generate a secret key $sk \in \mathbb{Z}_p$ shared among a set of n parties with at most t of them being malicious is as follows. Each party P_i samples a random value $r_i \leftarrow_s \mathbb{Z}_p$ and shares it among all parties using a verifiable secret sharing (VSS) scheme where r_i lies on a polynomial $f_i \in \mathbb{Z}_p[X]$ of degree t. Then, parties agree on a subset $I \subset [n]$ of at least $t + 1$ dealers whose VSS sharings completed successfully. Finally, each party P_i combines the shares it received from dealers in I to obtain a share sk_i of the final secret sk. Crucially, we have the guarantee that the secret key sk can be reconstructed even when corrupt parties refuse to participate in the reconstruction. However, the best known VSS schemes have quadratic communication cost [Shr+21a; AVZ21], which leads to cubic communication cost in the overall protocol. One way to overcome this issue is to let a randomly sampled (and sometimes anonymous) committee of small size (e.g., in the range of $O(\lambda)$) perform the task of sharing a secret. This technique is commonly used in consensus protocols to boost its scalability [Gil+17a], most notably in the Algorand blockchain. However, in order to achieve security against an adaptive adversary, these protocols come with undesirable features such as sub-optimal resilience threshold of $t < n/4$, reliance on secure erasures of internal states, and inefficient primitives such as fully homomorphic encryption (FHE) [Gen+21b]. Further, to sample the committee in the first place, these protocols rely on an initial seed of common randomness, which creates a circularity (without assuming some form of trusted setup).

Starting Point: Aggregatable PVSS. Publicly verifiable secret sharing (PVSS) schemes [Sta96] are VSS schemes with the additional property that any third party can verify that the sharing has been done correctly. In particular, this avoids the need for a complaint phase as is required in regular VSS schemes, which greatly simplifies constructions based on PVSS schemes. Recently, Gurkan et al. [Gur+21b] introduced a PVSS scheme that supports aggregation of several PVSS transcripts while preserving security (called *aggregatable PVSS* or simply *APVSS*). From that the authors design a DKG protocol whose secret and public keys both are group elements in an underlying pairing group. Crucially, the authors leverage the property of aggregation from their APVSS scheme in order to reduce the communication and computation cost of parties by relying on gossiping techniques rather than all-to-all communication [TLP22b]. However, given a known lower bound of $\Omega(n^2)$ communicated messages for Byzantine broadcast [DR85] (without assuming shared randomness in the first place), their protocol still has cubic communication cost due to the invocation of n instances of Byzantine broadcast. Additionally, their techniques only work against a static adversary that corrupts a mere $\log n$ parties maliciously. So, is there a way to regain all the desirable features simultaneously?

Recursion in APVSS to the Rescue. To solve the problem, we take inspiration from the world of recursive algorithms. In a recursive algorithm, the function calls itself with smaller input

values in such a way that eventually a base case is reached which is easy to solve. The result of the function for the current input is then obtained from simple operations on the returned value for the smaller input. This technique has also found application in distributed protocols in order to improve communication cost. The recursive Phase-King protocol [LS22; MR21b] for Byzantine agreement is a well-known example for this. The standard, non-recursive Phase-King protocol [BGP89] runs over $t + 1$ phases with a different leader (called the king) in each phase. The protocol succeeds because at least one honest party is guaranteed to be a leader. Interestingly, the need to run over $t + 1$ phases can be avoided using recursion. In the recursive variant, the leader is replaced by one half of the entire system, whose value is generated by the recursive invocation of the protocol. Essentially, in this manner there are only two phases, with the first half of the system emulating the leader of the first phase and the second half of the system emulating the leader of the second phase. Since at least one of these halves has an honest majority and thus emulates an honest leader, the protocol terminates after these two phases. In this manner, the communication cost of the protocol can be brought down from cubic to only quadratic.

We explain how we use a similar idea to design a DKG protocol whose secret and public keys both are group elements. Concretely, we want to devise a recursive protocol that allows parties to agree on an aggregated PVSS transcript AT with contribution from at least one honest party. From this transcript AT each party can locally and without any further interaction derive its share of the secret by a simple decryption operation (we will explain this in more detail soon). In order to achieve our goal in an efficient way (with the hope to not exceed quadratic communication cost), we carefully put together aggregation properties of PVSS and recent techniques from the theory of verifiable information dispersal for communication-efficient broadcast protocols [Nay+20a]. On a high level, our protocol works as follows. We split the system of n parties into two halves and run the protocol recursively and in parallel in both halves separately, so that each half ends up with a single transcript. In the next step, for each half, the protocol emulates an efficient single-sender broadcast protocol for long messages to transmit the transcript to all parties. For this step, we use the techniques developed in [Nay+20a] that rely on erasure codes and cryptographic accumulators. Finally, all parties aggregate these two transcripts and end up with a single (aggregated) PVSS transcript AT. Crucially, contribution from at least one honest party to the aggregate AT provides secrecy guarantees as discussed in [BL23c; Bha+23]. Conversely, such honest contribution is guaranteed by an argument similar to [LS22; MR21b], since at least one of the two halves has honest majority (relying on the fact that $t < n/2$).

From Aggregated PVSS to DKG. For the following discussion, let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be an asymmetric pairing of (multiplicative) groups of prime order p with generators $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$. We recall that a PVSS transcript generated by some dealing party P_* consists of a vector of commitments $\mathbf{C} := (C_1, \ldots, C_n) \in \mathbb{G}_1^n$, a vector of encryptions $\mathbf{E} := (E_1, \ldots, E_n) \in \mathbb{G}_2^n$, and some auxiliary data π which usually includes some proof. The commitments are computed as $C_i := g^{f(i)}$ for all $i \in [n]$ where $f \in \mathbb{Z}_p[X]$ is some polynomial of degree t chosen randomly by the dealer P_*, while the encryptions are computed as $E_i := pk_i^{f(i)}$. Here, (pk_i, sk_i) is the key pair of party P_i from a plain PKI setup such that $pk_i = h^{sk_i} \in \mathbb{G}_2$. The distinctive property of a PVSS scheme is that any subset $\mathcal{S}$ of at least $t + 1$ parties can pool their decrypted shares $\{D_i\}_{i \in \mathcal{S}}$ to reconstruct the secret D_0 encoded in the transcript, while this remains infeasible with t or less such shares. By combining several PVSS transcripts with contribution from at

least one honest dealing party, we have the guarantee that the secret of the aggregated transcript AT remains hidden from the adversary. In most applications of PVSS, the aggregated transcript AT is used to obtain one-time randomness by direct reconstruction of the secret (and possibly subsequent hashing). This is also the case for several previous randomness beacon protocols such as OptRand [Bha+23] and SPURT [Das+22a]. We are taking a different route inspired by the work [Gur+21b]. Specifically, we think of the commitments $C_1, \ldots, C_n$ as public key shares and of the decryptions $D_1, \ldots, D_n$ as secret key shares. That is, each party P_i outputs a secret key share $SK_i := D_i \in \mathbb{G}_2$, a vector of public key shares $(PK_1, \ldots, PK_n)$ where $PK_j := C_j \in \mathbb{G}_1$ for all $j \in [n]$, and a public key $PK := C_0 \in \mathbb{G}_1$ computed by standard Lagrange interpolation in the exponent from $C_1, \ldots, C_n$.

Table 5.2: Comparison table of distributed key generation (DKG) protocols.

Protocol	Network	Resil	Adapt	Commun	Rounds	Field	Crypto Primitive	Setup
Kokoris et al. [KMS20]	*async*	1/3	✓	$O(\lambda n^4)$	$O(n)$	✓	Async VSS	*CRS*
Abraham et al. [Abr+21a; Gao+21]	*async*	1/3	✗	$O(\lambda n^3)$	$O(1)$	✗	PVSS, Pairings	*CRS*
Das et al. [Das+22c; Das+23]	*async*	1/3	✗	$O(\lambda n^3)$	$O(\log n)$	✓	Async VSS	*CRS*
Bingo [Abr+23]	*async*	1/3	✓	$O(\lambda n^3)$	$O(1)$	✓	AVSS, Pairings	*PoT*
Harts [Bac+24b]	*async*	1/3	✓	$O(\lambda n^3 \log n)$	$O(1)$	✓	Async VSS	*CRS*
Shrestha et al. [Shr+21a]	*sync*	1/2	✗	$O(\lambda n^3)$	$O(n)$	✓	VSS	*PoT*
Gurkan et al. [Gur+21b]	*sync*	$\log n$	✗	$O(\lambda n^3 \log n)$	$O(n)$	✗	PVSS, Pairings	*CRS*
NI-DKG$^\circ$ [Kat+23; Gro21; CD24]	*sync*	1/2	✗	$O(\lambda n^4)^\circ$	$O(n)$	✓	PVSS	*CRS*
Gennaro et al. [Gen+07]	*sync*	1/2	✓	$O(\lambda n^4)^\circ$	$O(n)$	✓	VSS	*CRS*
Canetti et al. [Can+99]	*sync*	1/2	✓	$O(\lambda n^4)^\circ$	$O(n)$	✓	VSS, Erasures	*CRS*
Jarecki et al. [JL00]	*sync*	1/2	✓	$O(\lambda n^4)^\circ$	$O(n)$	✓	PVSS	*CRS*
GRand (ours)	*sync*	1/2	✓	$O(\lambda n^2 \log n)$	$O(n)$	✗	PVSS, Pairings	*CRS*

Resil denotes the Byzantine resilience threshold. **Adapt** denotes adaptive adversary. The protocol (Gen+07) was proven adaptively secure in the AGM (BL22a). **Commun** denotes the communication cost in bits. $\diamond$ The protocols (Gen+07; Can+99; JL00) assume a broadcast channel, which we implement with the commonly-used Dolev-Strong broadcast protocol (DS83a). However, we note that it is possible to achieve cubic communication cost by using an optimal broadcast protocol. **Rounds** denotes the number of (a)synchronous rounds to terminate. For asynchronous protocols, this is the expected number of rounds (as these protocols are randomized). **Field** denotes if the secret key is a field element or not (group element). **Crypto Primitive** denotes the cryptographic primitives in usage. The protocol (Can+99) relies on secure erasure of secret states. **Setup** denotes the setup assumptions. $\circ$ The protocols (Kat+23; Gro21; CD24) follow the common technique (JL00), where each party broadcasts a PVSS transcript for a field element, and primarily focus on the concrete computational efficiency of the PVSS scheme (which are rather inefficient compared to PVSS schemes for group elements).

From One to Infinity: Towards a Simple Randomness Beacon. Clearly, GRand is different from most DKG protocols found in the literature that output secret keys in a field rather than a group. However, a delightful key insight in [Gur+21b] is that this setup is enough to generate a stream of one-round randomness values. The idea is inspired from the threshold BLS signature [Bol03] and its application to randomness generation as introduced by Cachin et al. Specifically, each party P_i non-interactively creates a threshold BLS signature share $\sigma_i := \mathsf{H}_1(m)^{sk_i}$ on the epoch number $m := e \in \mathbb{Z}_{\geq 1}$ with its secret key share sk_i and multicasts (i.e., sends it to all parties) it. Upon receiving $t + 1$ valid shares $\{\sigma_i\}_{i \in S}$ (which is checked by a pairing equation $e(g, \sigma_i) = e(pk_i, \mathsf{H}_1(m))$ from P_i's public key share pk_i), a party locally reconstruct the full signature $\sigma = \mathsf{H}_1(m)^{sk}$ by Lagrange interpolation in the exponent and

derives the randomness beacon value as another hash $O_e = \mathsf{H}_2(\sigma)$. With our DKG protocol GRand that generates keys (PK_i, SK_i) as group elements, the operation $\mathsf{H}_1(m)^{SK_i}$ is not possible. However, when we think of the operation of „raising the group element $\mathsf{H}_1(m)$ to a power of sk_i" as an abstract group action $sk_i \odot \mathsf{H}_1(m)$, we realize that the action[3] $SK_i \odot \mathsf{H}_1(m)$ defined as $e(\mathsf{H}_1(m), SK_i) \in \mathbb{G}_T$ is possible. Therefore, we let each party P_i non-interactively create a share σ_i as $e(\mathsf{H}_1(m), SK_i)$. And upon receiving $t+1$ valid such shares, each party can locally reconstruct the full signature by Lagrange interpolation in the exponent as $\sigma = e(\mathsf{H}_1(m), SK)$. Again, the randomness beacon value for epoch e is then derived as another hash $O_e := \mathsf{H}_2(\sigma)$. Intuitively, since the secret key SK is hidden from the adversary, the signature σ should remain unpredictable so that O_e gives a random and unbiased randomness value. However, there is one crucial issue with this approach: the verification of *beacon shares* σ_i. Previously, it was possible to verify such a (signature) share by a pairing check, but now the beacon share σ_i is an element in the target group $\mathbb{G}_T$ itself so that there is possibly no way to verify correctness of $\sigma_i = e(\mathsf{H}_1(m), SK_i)$. To resolve this issue, the authors in [Gur+21b] augment the public keys and shares σ_i with additional elements inspired by Escala-Groth non-interactive zero-knowledge (NIZK) proofs. While their construction in the appendix is reasonably efficient, it still requires a lot of pairings and elements. Further, it lacks an adaptive security proof.

A Simple Two-Step Trick. In order to regain efficiency, we use the following two-step approach. After the DKG setup, each party P_i locally samples an element $\alpha_i \leftarrow_\$ \mathbb{Z}_p^*$ uniformly at random and sends an ElGamal encryption $\mathsf{cm}_i := (g^{\alpha_i}, h^{-\alpha_i} SK_i)$ of its secret key SK_i to all parties. Correctness of its second component can be checked via a pairing equation. Crucially, this requires only a single round of communication (no *broadcast* in the sense of consensus is needed) and therefore does not add asymptotic overhead. These additional elements allow parties to verify received beacon shares. Concretely, each party computes $\vartheta_i := (\mathsf{H}_1(m)^{\alpha_i}, \sigma_i)$ along with an efficient Chaum-Pedersen NIZK proof of discrete logarithm equality $\pi_i := \mathsf{Dleq}(g, g^{\alpha_i}, \mathsf{H}_1(m), \mathsf{H}_1(m)^{\alpha_i})$ to prove correctness of $\mathsf{H}_1(m)^{\alpha_i}$. Upon receiving such a tuple (ϑ_i, π_i), any party can verify the correctness of the beacon share σ_i using a pairing equation that involves the element $\mathsf{cm}_{i,2}$. Having done this, each party can compute the randomness beacon value O_e for epoch $e = m$ as described before. Intuitively, security is preserved because the elements $g^{\alpha_i}, \mathsf{H}_1(1)^{\alpha_i}, \mathsf{H}_1(2)^{\alpha_i}, \ldots$ do not reveal too much information about α_i, thus making it hard for the adversary to compute SK_i from the (randomized) element $h^{-\alpha_i} SK_i$. Overall, an epoch takes only a single round of communication in which each party P_i sends two group elements $\vartheta_i := (\mathsf{H}_1(m)^{\alpha_i}, \sigma_i)$ and a simple NIZK proof π_i to the other parties. Further, (PK_i, cm_i) can be thought of as an updated public key share of P_i which is three group elements, and verification of a beacon share σ_i takes two pairing operations (the same as threshold BLS!) and verification of the NIZK proof π_i.

Concurrent Work. Concurrent with or subsequent to our work, two other constructions for shared randomness generation have been proposed [FLT24; Das+24]. The first one [Das+24] focuses on the weighted setting for DKG and threshold verifiable unpredictable function (VUF), but the authors do not consider subcubic DKG protocols. The second one [FLT24] focuses on communication-efficiency in DKG and achieves: (i) a DKG protocol with $O(\lambda n^{2.5} \log n)$ bits of communication cost that outputs secret keys as group elements, and (ii) a DKG protocol with

[3] Note that this does not define a group action in the mathematical sense. Here, we use the term *group action* only informally to convey the intuition.

$O(\lambda^2 n^{2.5} \log n)$ bits of communication cost that outputs secret keys as field elements. None of these works consider adaptive adversaries. Further, we note that generic transformations can be applied to our DKG protocol to generate secret keys as field elements at the cost of $O(\lambda^2 n^2)$ added communication and reliance on secure erasures. At a high level, each party generates a PVSS transcript for a field element and erases the underlying secrets. Then, it disperses the transcript using linear erasure codes. A common coin (provided by our DKG protocol) then elects a random committee of $\Theta(\lambda)$ parties, whose transcripts are reconstructed jointly. Finally, the parties execute $\Theta(\lambda)$ parallel instances of a quadratic-communication consensus protocol to agree on these transcripts, from which the secret key shares are obtained. Note that each step incurs only quadratic communication cost.

5.2.2 More on Related Work

In this section, we provide a detailed discussion on existing work of randomness beacon and distributed key generation protocols.

Randomness Beacons. We categorize existing randomness beacons according to their assumptions and reliance on cryptographic tools. Essentially, there are two types of designs: the first employs threshold cryptography, while the second employs specialized tools such as proof-of-work or VDF.

Threshold Cryptography. The protocol of this type employ threshold cryptography in order to generate randomness. For this, there are two approaches. In the first one [Dra20; CKS05; Syt+17; Abr+18; CSS19], parties generate a (t, n)-threshold key $(sk_1, \ldots, sk_n)$ by running a DKG protocol from which the randomness beacon value is derived as a unique threshold signature on some message (typically the hash of the current epoch number). In more detail, there are the following protocols. Cachin et al. [CKS05] works in asynchrony, but does not specify the threshold signature nor the DKG protocol. Dfinity works in partial synchrony and uses the threshold BLS signature together with the non-interactive DKG protocol of Groth [Gro21] (it assumes a broadcast channel). Drand [Dra20] works in synchrony and uses threshold BLS together with Gennaro et al.'s DKG [Gen+07]. Herb works in synchrony and uses the threshold ElGamal signature, and RandHerd works in asynchrony and uses the threshold Schnorr signature [SS01]. The setup phase of these protocols have a communication complexity of $O(\lambda n^3)$ or higher due to the use of a DKG protocol for field elements. On the other hand, once the setup phase has terminated, these protocols achieve an improved communication complexity of $O(\lambda n^2)$ per beacon output within optimal one round. The second approach works through (P)VSS [Syt+17; HYL20; Sch+21; Bha+21; Bha+23; Sch+20; Das+22a; CD17; Dav+18]. Notable randomness beacons here are SPURT [Das+22a], BRandPiper [Bha+21], and OptRand [Bha+23]. The idea of this approach is to generate a new random value at each epoch by combining secret sharings from at least $t + 1$ parties. This ensures that the combined secret has contribution from at least one honest party that chose its secret uniformly at random so the randomness beacon value also inherits that property. Let us elaborate in more detail on some of the protocols. SPURT works in partial synchrony and has a communication complexity of $O(\lambda n^2)$ bits per beacon output. It relies on a pairing-based PVSS scheme. BRandPiper is a protocol from the family of RandPiper protocols. It works in synchrony and achieves a communication complexity of $O(\lambda f n^2)$ bits per beacon output, where $f \leq t$ is the actual number of faults in the system. It relies on an efficient VSS scheme and the RandPiper SMR protocol. In

BRandPiper, the leader of an epoch shares n secrets at once and for the beacon output parties reconstruct a random value accumulating secrets from $t + 1$ different (previous) leader parties. Most of these protocols assume a setup phase that when actually implemented incurs cubic or higher communication cost. Further, protocols of this type have a computation-heavy epoch where most of the computation is carried by one single party (epoch leader). Some protocols here have a communication complexity of $O(\lambda n^2)$ per epoch, while others have cubic or higher. Finally, HashRand [Ban+24] is a recent randomness beacon in asynchrony that achieves adaptive security without the use of a threshold cryptographic setup. Their randomness beacon is based on a (small) committee selection from which the secret shares of an AVSS scheme are reconstructed. Interestingly, the committee requires the presence of only one honest party to preserve security. However, their techniques rely on secure erasures of secret states and without that their protocol has a communication complexity of $O(\lambda n^3 \log n)$ per epoch. Further, it assumes resilience $t < n/3$. An advantage of their protocol is its post-quantum security.

Specialized Tools. The protocols in this category employ verifiable delay functions (VDFs) [Bon+18; Dra18] or proof-of-work (PoW) [Nak08] in order to generate randomness [HYL20; CD20; Sch+21; Cho+23; BGB17; Dra18]. VDFs are functions that require a certain amount of time to compute but can be verified quickly. Solana uses VDFs in its proof-of-history consensus protocol to establish a global source of time and generate random values. While VDF-based protocols can offer strong security guarantees and quadratic communication complexity, they require specialized hardware to compute the VDFs efficiently, which might not be accessible to all participants. The same applies to PoW-based protocols that rely on the assumption that the adversary has less computational hash power than the honest parties. In general, these primitives are computationally expensive tools with specialized hardware and are highly energy-consuming. We elaborate on some of these protocols. RandRunner [Sch+21] works in synchrony and uses a trapdoor VDF. Such a trapdoor VDF can generate unique function values efficiently with the knowledge of the trapdoor, but takes some high specified time T otherwise. RandRunner achieves a communication complexity of $O(\lambda n^2)$ bits per beacon output. However, it only achieves $(t + 1)$-unpredictability, since an adaptive adversary can simply corrupt the next t leaders and thus learn the beacon values for the next t epochs. RandChain [HYL20] uses a combination of PoW, VFD, and Nakamoto Consensus, and achieves a communication complexity of $O(\lambda n)$ bits per beacon output. One crucial drawback is that the beacon output is only guaranteed to be $1/5$-fair. Further, it suffers from blockchain-related attacks.

Distributed Key Generation. Most of the DKG protocols found in the literature are in synchrony [Can+99; Gro21; Gur+21b; JL00; Sch+19; Shr+21a]. Among these protocols, only the ones of Canetti et al. [Can+99] and Jarecki and Lysyanskaya [JL00] provide adaptive security. All of these synchronous DKG protocols with the exception of Shrestha et al. [Shr+21a] assume the existence of a one-round broadcast channel. When instantiating the broadcast channel with a state-of-the-art Byzantine broadcast protocol [Bac+23a; DY83], all these protocols have cubic or higher communication cost. More importantly, these DKG protocols require each party to broadcast a message of size at least λ, which inevitably results in $\Omega(\lambda n^3)$ bits communication cost by a known lower bound for Byzantine broadcast [DR85] (without assuming shared randomness in the first place). DKG protocols in asynchrony have only gained attention very recently by the works in [Bac+24b; KMS20; Abr+23; Abr+21a; Das+23; Das+22c]. All these constructions have cubic or higher communication cost, and the one of Abraham et al. [Abr+23] relies on a powers-of-tau setup.

5.2.3 Outline of this Chapter

The rest of this chapter is structured as follows. In Section 5.3, we define the preliminaries that are only relevant for this chapter. In Section 5.4, we present our new distributed key generation protocol and give a security and complexity analysis. In Section 5.5, we present our new distributed randomness beacon protocol and give a security and complexity analysis. In Section 5.6, we implement our distributed randomness beacon and compare it to the state-of-the-art distributed randomness beacons. In Section 5.7, we give a brief conclusion for the chapter. In Appendix 5.7, we give additional preliminaries relevant for the chapter. In Appendix 5.7, we give additional figures relevant for our distributed key generation protocol.

5.3 Preliminaries for this Chapter

In addition to the general preliminaries in Chapter 2, we introduce here preliminaries that are only relevant for this chapter.

5.3.1 Assumptions and Model

We specify our assumptions and the model for this chapter.

Network and Adversarial Model. In this chapter, we assume a synchronous network as defined in Chapter 2. The adversary is Byzantine, strongly adaptive, and can corrupt up to $t < n/2$ parties. The adversary is in full control over message delays (subject to the network delay Δ) and rushing.

Idealized Models and Computational Assumptions. We assume the random oracle model and the algebraic group model. We rely the co one-more discrete logarithm (co-OMDL) assumption [BL23c] for our security proofs. This assumption is also defined in Section 4.5.

Cryptographic Groups. Let λ denote the security parameter. Throughout this chapter, we assume that global parameters $par := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, h, e)$ are fixed and known to all parties. Here, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a type 3 asymmetric pairing of prime order p cyclic groups with generators $g \in \mathbb{G}_1, h \in \mathbb{G}_2$. Further, we use $\mathbb{G}$ to denote a group specified by par.

5.3.2 Cryptographic Primitives

In this section, we define the relevant cryptographic primitives for the chapter.

Aggregatable Publicly Verifiable Secret Sharing. In a verifiable secret sharing (VSS) scheme, a dealer distributes shares of a secret among a group of parties such that it can be reconstructed only if a threshold of these parties collaborate. In a publicly verifiable secret sharing (PVSS) scheme, any third party can verify the correctness of the sharing, thus avoiding the need for a complaint phase as in VSS schemes. Henceforth, we consider PVSS schemes that support aggregation of several sharings while preserving public verifiability (called *aggregatable* PVSS scheme).

Definition 5.3.1 (Aggregatable PVSS Scheme)**.** Let $\mathbb{G}$ be a cyclic group of prime order p specified by par. A (t, n)*-threshold aggregatable PVSS (APVSS) scheme over* $\mathbb{G}$ is a tuple of algorithms $\mathsf{APVSS} = (\mathsf{Keys}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Dist}, \mathsf{Agg}, \mathsf{OwnId}, \mathsf{Ver}, \mathsf{Rec})$ such that:

- Keys: The randomized *key generation algorithm* takes as input system parameters par and an index $i \in [n]$. It outputs a public key pk_i and a secret key sk_i.
- Enc: The randomized *encryption algorithm* takes as input a public key pk_i and a message m. It outputs a ciphertext c.
- Dec: The deterministic *decryption algorithm* takes as input a secret key sk_i and a ciphertext c. It outputs a message m (optionally with a proof of correct decryption). We require that for all messages m, $\Pr[\mathsf{Dec}_{sk_i}(\mathsf{Enc}_{pk_i}(m)) = m] = 1$.
- Dist: The randomized *secret sharing algorithm* takes as input a secret key sk_i and public keys $pk_1, \ldots, pk_n$. It outputs a vector of encrypted shares $\mathbf{E} = (\mathsf{Enc}_{pk_1}(S_1), \ldots, \mathsf{Enc}_{pk_n}(S_n))$ and a proof π, where $S_1, \ldots, S_n$ are shares of a secret $S \in \mathbb{G}$. We refer to $T := (\mathbf{E}, \pi)$ as a *PVSS transcript*.
- Agg: The deterministic *aggregation algorithm* takes as input PVSS transcripts $(\mathbf{E}_1, \pi_1), \ldots, (\mathbf{E}_k, \pi_k)$ (for any $k \in \mathbb{N}$). It outputs an *(aggregated) PVSS transcript* $T := (\mathbf{E}, \pi)$.
- OwnId: The deterministic *contributor identifier algorithm* takes as input an (aggregated) PVSS transcript $T = (\mathbf{E}, \pi)$ and a public key pk_i. It outputs 1 (accept) or 0 (reject). In the first case, we refer to P_i as a *contributor to* T.[4]
- Ver: The deterministic *verification algorithm* takes as input public keys $pk_1, \ldots, pk_n$, and an (aggregated) PVSS transcript $T = (\mathbf{E}, \pi)$. It outputs 1 (accept) or 0 (reject). In the first case, we call the transcript T *valid*; otherwise we call it *invalid*.
- Rec: The deterministic *reconstruction algorithm* takes as input $t + 1$ secret shares $S_1, \ldots, S_{t+1}$. It outputs a secret $S \in \mathbb{G}$.

Discussion. We defer formal definitions for correctness and secrecy of an APVSS scheme to Appendix 5.7. Our definition above (and those in the appendix) are based on the ones from [BL23c]. Essentially, the only difference to their definitions is the following. Their aggregation algorithm takes exactly $t + 1$ transcripts as input, in contrast to ours that can take any finite number of transcripts as input, even a single one. In particular, our definition generalizes theirs and we do not need to explicitly separate anymore between a standard PVSS transcript and an aggregated one. However, when we want to emphasize that the transcript was formed by aggregation of several (possibly themselves aggregated) transcripts, we will make this explicit and call the transcript aggregated. Having said this, we appropriately adapted some other algorithms and security notions to our generalized setting. Further, for an APVSS scheme, we require the secrecy notion of *aggregated unpredictability* as defined in [BL23c] (slightly adapted). This notion captures malleability attacks and prohibits any t-bounded (i.e., corrupting at most t parties) adversary from learning the secret of an aggregated transcript that has contribution from at least one honest party, even if the adversary is allowed to contribute to the aggregation itself.

Linear Erasure and Error Correcting Codes. We use standard (q, b)-Reed-Solomon (RS) codes [RS60]. This primitive allows to encode b data symbols into code words of q symbols (using the algorithm Encode) such that b elements of the code word suffice to recover the original

[4] We remark that OwnId could return 1 on an invalid transcript.

data (using the algorithm Decode). In our DKG construction, we will use Reed-Solomon codes with varying (q, b). Concretely, we use codes with q being the number of parties in some designated subset of parties $Q \subseteq \mathcal{P} = \{P_1, \ldots, P_n\}$ (called a *committee*) and b being $\lceil q/2 \rceil$. In the special case $Q = \mathcal{P}$, we have $(q, b) := (n, t + 1)$. For a formal definition, see Chapter 2.

Cryptographic Accumulator. A cryptographic accumulator scheme [Ngu05] allows to accumulate several elements from some set D into an accumulated value z (using the algorithm Eval). Further, for each element in D it allows to generate a compact proof of membership in D (using the algorithm Wit) called a witness. The standard security notion of collision-resistance requires that it is hard for an adversary to create invalid proofs of membership. An example of cryptographic accumulators are Merkle trees, where the root is the accumulation value and the authentication paths are membership proofs (i.e., witnesses) for the leaves. In this chapter, we use an accumulator scheme with membership proofs and accumulation value each of size $O(\lambda)$. This can be implemented using the accumulator scheme of [BBF19] built upon class groups of unknown order. Alternatively, we could use Merkle trees at the cost of $O(\log n)$ multiplicative overhead in the communication complexity. For a formal definition, see Chapter 2.

5.3.3 Consensus Primitives

In this section, we define the relevant consensus primitives for the chapter.

Byzantine Agreement. A Byzantine agreement (BA) protocol [LSP82a] allows a set of parties, each holding an input $v_i \in V$ from a value set V with $|V| \geq 2$, to agree on a common output value $v \in V$ that was input from at least one honest party. In our definition, we also account for some probability of failure ε which corresponds to the adversary's ability in breaking underlying cryptographic tools.

Definition 5.3.2 (Byzantine Agreement)**.** Let Π be a protocol executed by n parties $P_1, \ldots, P_n$, where each party P_i holds an input value $v_i \in V$. We define the following properties for Π which each holds with probability at least $1 - \varepsilon$ in the presence of an adversary corrupting at most t parties:

- *Validity.* Π is (t, ε)-*valid* if the following holds: if every honest party has the same input value v as input, then every honest party outputs this value v.
- *Consistency.* Π is (t, ε)-*consistent* if the following holds: every honest party that outputs a value outputs the same value v.
- *Termination.* Π is (t, ε)-*terminating* if the following holds: every honest party terminates with an output value $v \in V$.

We say that Π is a (t, ε)-*secure* Byzantine agreement protocol if it is (t, ε)-valid, (t, ε)-consistent, and (t, ε)-terminating.

Distributed Randomness Beacon. A distributed randomness beacon (DRB) is a distributed protocol that allows a set of n parties to generate a sequence of unpredictable and unbiased random values, one for each epoch. Each party P_i has a local log that is defined as a write-once array $\Sigma_i = (\Sigma_i[1], \Sigma_i[2], \ldots)$ with $\Sigma_i[\ell]$ being its beacon output at epoch $\ell \geq 1$. Initially, each value is set to $\perp$. We say that party P_i outputs a beacon value in epoch ℓ if it writes a value on

$\Sigma_i[\ell]$. A secure randomness beacon has to satisfy the properties of consistency, availability, bias-resistance, and d-unpredictability. We elaborate on these security notions. Consistency and availability guarantee that each honest party outputs the same value $\sigma_e \in \{0,1\}^\lambda$ in each epoch $e \geq 1$. Bias-resistance guarantees that the beacon outputs are indistinguishable from uniformly random numbers. This property ensures that the adversary has no power in biasing the beacon output, even when controlling up to t parties in the system. On the other hand, this notion does not prohibit the adversary from learning the beacon output some epochs ahead of the honest parties. That is ensured by the notion of d-unpredictability, which states that the adversary does not learn the beacon output d epochs before the honest parties. Conversely, an adversary could predict the beacon output some epochs ahead of the honest parties, e.g., by corrupting the next t leaders whose previously committed values determine the next t beacon outputs (see GRandPiper [Bha+21] and HydRand [Sch+20]) without actually having the power to bias it. We defer formal definitions to Appendix 5.7.

5.4 Distributed Key Generation

In this section, we design a novel distributed key generation (DKG) protocol whose secret and public keys both are group elements. This is different from most DKG protocols that output secret keys in a field rather than a group. However, as demonstrated delightfully by Gurkan et al. [Gur+21b], this is enough for applications such as efficient randomness beacons. We first formally define a DKG protocol.

Definition 5.4.1 (DKG Protocol)**.** Let Π be a protocol executed by n parties $P_1, \ldots, P_n$, where for all $i \in [n]$, P_i outputs a *secret key share* SK_i, a vector of *public key shares* $(PK_1, \ldots, PK_n)$, and a *public key* PK. We define the following properties for Π which each holds with probability at least $1 - \varepsilon$ in the presence of an adversary corrupting at most t parties:

- *Consistency.* Π is (t, ε)-*consistent* if the following holds: all honest parties output the same public key PK and the same vector of public key shares $(PK_1, \ldots, PK_n)$.
- *Correctness.* Π is (t, ε)-*correct* if the following holds: there exists a deterministic algorithm Rec that on input any set of $t+1$ secret key shares $\{SK_i\}_{i \in I}$ outputs the same unique secret key SK. Further, SK is a valid secret key for PK.
- *Secrecy.* Π is (t, ε, T)-*secret* if for all algorithms A that run in time at most T, its success probability in the following experiment is at most ε.
  - *Offline Phase.* Initialize a corruption index set $C := \emptyset$ and let $\mathcal{H} := [n] \setminus C$. Run A on input par.
  - *Corruption Queries.* At any point of the experiment, A may corrupt a party by submitting an index $i \in \mathcal{H}$. In this case, return the internal state of P_i and update $C := C \cup \{i\}$. Henceforth, A has full control over P_i.
  - *Online Phase.* Initiate an execution of Π with A having full control over parties in C. Let $y := PK \leftarrow \Pi$ be the public key output by honest parties, and $x := SK$ the secret key.
  - *Winning Condition.* Let S^* denote the output of A. Then, A is considered successful iff $|C| \leq t$ and $S^* = x$.

We say that Π is a (t, ε, T)-*secure* DKG protocol if it is (t, ε)-consistent, (t, ε)-correct, and (t, ε, T)-secret.

5.4.1 Building Blocks

In this section, we describe the building blocks that will be used in the construction of our DKG protocol GRand. Although we instantiate these building blocks with specific schemes, our construction of GRand in the next section is done in an abstract way such that these building blocks can also be instantiated with other such schemes.

Aggregatable PVSS Scheme. We will make use of an APVSS scheme in the construction of GRand. Concretely, we instantiate this with our APVSS scheme given in Figure 5.6 (see Appendix 5.7). Our APVSS scheme is similar to the one of Bhat et al. [Bha+23] which is essentially SCRAPE [CD17] augmented with a signed NIZK proof of knowledge of discrete logarithm for $\zeta = g^{\alpha}$ where $\alpha := f(0)$ for a randomly chosen degree-t polynomial $f \in \mathbb{Z}_p[X]$. The only difference is that the reconstructed secret in our scheme is $S := h^{\alpha}$, whereas the one in their scheme is $S' := e(\hat{g}, h^{\alpha})$ where $\hat{g} \in \mathbb{G}_1$ is an additional generator. This choice is motivated by their security analysis, which is a reduction from the co-decisional bilinear squaring (co-DBS) problem. However, since in our randomness beacon we never explicitly reconstruct the secret, it does not make much of a difference for our security proof and thus we can sidestep the need for further generators in the source groups. Finally, aggregated unpredictability of our APVSS scheme follows directly from aggregated unpredictability of their APVSS scheme (see [BL23c] for a proof of the latter), since any prediction $S^* := h^{\alpha}$ for the former gives a prediction $S'^* := e(\hat{g}, S^*)$ for the latter.

Byzantine Agreement Protocol. We will make use of a Byzantine agreement (BA) protocol in the construction of GRand. Concretely, we instantiate this with the BA protocol given in Appendix 5.7.1. Essentially, this is merely a variant of the BA protocol of Momose and Ren [MR21b]. Their protocol has optimal resilience $t < n/2$ and achieves a communication complexity of $O(\lambda n^2)$ bits assuming threshold signatures of size $O(\lambda)$ from a trusted setup. The threshold signatures are used to prove knowledge of a threshold of signatures from other parties on the same message. Instead, we implement these threshold signatures with the recent transparent-setup threshold signatures of Attema et al. [ACR21] at the cost of multiplicative logarithmic overhead in the communication complexity.[5] Importantly, this scheme has the following desirable features: (i) It does not require a trusted setup phase, i.e., all public parameters are random coins. (ii) The k-aggregation algorithm can be evaluated by any party with input at least k valid signatures from distinct signers, and it only takes the signatures and public values as input. (iii) It allows for any threshold $k \leq n$ which can be chosen by the aggregator at aggregation time independent of the setup phase. (iv) It is non-interactive, correct, and unforgeable against an adaptive adversary. Having said this, our resulting Byzantine agreement protocol has a communication complexity of $O(\lambda n^2 \log n)$ and terminates in a linear number of rounds. For more details on the transparent threshold signatures, we refer to the original work [ACR21]. Further, we briefly discuss the security guarantees of the resulting BA protocol in Appendix 5.7.1.

[5] When the network of parties is of small size $n \in O(\lambda)$, we can instead directly use an aggregated BLS signature augmented with the n-bit long vector of signers.

Deliver Protocol. We will make use of a protocol in the construction of GRand that allows parties to efficiently broadcast a long message. Concretely, we instantiate this with the protocol Deliver given in Figure 5.5 (see Appendix 5.7). The protocol design was introduced in [Bha+21; Nay+20a] and is based on erasure codes and cryptographic accumulators. Deliver is a two-round protocol that is invoked by a party P_i that wants to efficiently broadcast a long message m to all parties in some set Q. In contrast to [Bha+21; Nay+20a], we make use of Deliver for sets of varying sizes. We parameterize Deliver by a set Q of q parties among which it is executed. It is invoked by a party $P_i \in Q$ and takes as input a long message m, the accumulation value z for an encoding of m, and Q along with implicit parameters $q = |Q|$ and $b = \lceil q/2 \rceil$. For this, P_i first splits m into b data symbols and encodes these into q code words using an (q, b)-erasure code RS. Then, P_i sends the j-th code word, the accumulation value z for the set of q code words along with a witness to $P_j \in Q$. Upon receiving a valid triple of this type, P_j forwards it to all other parties in Q. Finally, upon receiving b valid code words corresponding to the accumulation value z, P_j reconstructs the full message m using the decoding algorithm. In this way, message m can efficiently reach all honest parties in Q when the sender P_i was honest. Finally, we note that correctness of Deliver is implied by collision-resistance of the underlying cryptographic accumulator scheme, since the only way to reconstruct a different message $m' \neq m$ is by receiving a witness for non-membership. For more details, we refer to [Nay+20a].

5.4.2 Our Design

In this section, we present our novel DKG protocol GRand. At its heart lies a recursive protocol GenAPVSS that allows parties to aggregate several PVSS transcripts with contribution from at least one honest party in an efficient manner. From such an aggregated PVSS transcript parties can locally derive their secret key shares without any further interaction between them.

Recursive PVSS Aggregation. We give an informal description of the protocol GenAPVSS (cf. Figure 5.1). We parameterize GenAPVSS by a set Q of q parties among which it is executed. Upon termination of the protocol, all parties output a common, single PVSS transcript AT which is obtained by aggregation of two transcripts. The high-level idea of the protocol is to split the system Q of all parties into two disjoint sets (called *committees*) Q_1, Q_2 of roughly equal size, let each committee Q_i, $i \in [2]$, run the protocol among themselves, and then broadcast the resulting PVSS transcript T_i to the other committee Q_{1-i}. All parties then terminate with the aggregation $AT := \mathsf{Agg}(T_1, T_2)$. In more detail, this works as follows.

Let $Q = Q_1 \cup Q_2$ be a (deterministic) partition of Q into two disjoint sets called committees. For each $i \in [2]$, run the protocol GenAPVSS among parties in Q_i and let T_i denote the common output. Now, instead of directly sending the whole transcript T_i to all parties in the opposite committee Q_{1-i}, each party sends only a much shorter accumulation value z_i for T_i. To counteract malicious behavior, all parties in Q establish consensus on both accumulation values z_1, z_2 via two separate instances of a Byzantine agreement protocol BA. Note that at this stage, parties in Q_i do not know anything about the opposite committee Q_{1-i}'s transcript T_{1-i} other than the accumulation value z_{1-i} and vice versa. Therefore, for each $i \in [2]$, parties in Q_i next broadcast their transcript T_i to all parties in Q using the protocol Deliver on input (Q, T_i, z_i). This ensures efficient delivery of the large transcript T_i to all other parties. Since an adversarial-controlled committee could simply refuse to deliver its transcript to some of the honest parties, we introduce two further steps to maintain consistency. First, parties in Q

decide on whether the previous step succeeded via two separate instances of binary Byzantine agreement, one to decide for each committee. Second, parties proceed with another invocation of Deliver to guarantee that all honest parties obtain the transcripts among $\{T_1, T_2\}$ for which the respective Byzantine agreement execution output 1. Parties conclude the protocol with aggregation of these transcripts and terminate with $AT := \mathsf{Agg}(T_1, T_2)$ as output.

Our DKG Protocol. A formal description of our DKG protocol GRand is given in Figure 5.2. The protocol consists of two simple steps. First, all parties in $\mathcal{P}$ execute the protocol GenAPVSS to establish consensus on an aggregated PVSS transcript $AT := \{C_j, E_j, \pi\}_{j\in[n]}$ (which has contribution from at least one honest party by design and thus is secure from the adversary). Then, each party P_i computes its secret share $D_i := \mathsf{Dec}_{sk_i}(E_i)$ and terminates. The public key shares of GRand are defined as $(PK_1, \ldots, PK_n) := (C_1, \ldots, C_n)$ with the secret key shares being $(SK_1, \ldots, SK_n) := (D_1, \ldots, D_n)$. Using the specific APVSS scheme described in Figure 5.6, the public key shares of GRand are $PK_i = g^{f(i)}$ and the secret key shares are $SK_i = h^{f(i)}$, where $f \in \mathbb{Z}_p[X]$ is the hidden polynomial of degree t encoded in the APVSS transcript AT. We note that even though P_i knows $h^{f(i)}$, it does not know $f(i)$ itself. In particular, this is different from many DKG protocols in the literature where the hidden polynomial itself is distributed among the parties.

5.4.3 Security and Complexity Analysis

In this section, we give a security and complexity analysis of our DKG protocol GRand. In the following, let APVSS be an aggregatable PVSS scheme, let BA be a Byzantine agreement protocol, and let AC be a cryptographic accumulator scheme. Then, assuming aggregated unpredictability of APVSS, security of BA, and collision-resistance of AC, this implies security of GRand. Further, it has log-quadratic communication complexity and linear round complexity when instantiated with our components from Section 5.4.1.

Theorem 5.4.1. *If* APVSS *is* $(t, \varepsilon_A, T_A, q_s)$*-aggregated unpredictable, if* BA *is* (t, ε_B, T_B)*-secure, and if* AC *is* (n, ε_C, T_C)*-collision-resistant, then* GRand *(cf. Figures 5.1 and 5.2) is a* (t, ε, T)*-secure DKG protocol, where*

$$\varepsilon \leq 2n\left(\varepsilon_A + 2\varepsilon_B + n\varepsilon_C\right), \quad T \geq T_A + T_B + T_C + O(n^2).$$

Using the components in Section 5.4.1, GRand *has a communication complexity of* $O(\lambda n^2 \log n)$ *bits and terminates in* $O(n)$ *rounds.*

Subsequently, we give a proof of Theorem 5.4.1. We split our proof into two parts. First, we show security of GRand in the sense of Definition 5.4.1, assuming security of the underlying components. Second, we compute the communication and round complexity of GRand when using our concrete components. In this context, we note that security of GRand with our concrete components follows from the first part, since the security of these components was already discussed in previous sections of this paper.

Proof. Before we do the analysis, we briefly recall the high-level idea of the DKG protocol design. For the following discussion, we assume for the sake of presentation that the number of all parties n is a power of two $n := 2^k$, $k \in \mathbf{N}$k, and recall that $\mathcal{P} := \{P_1, \ldots, P_n\}$. For the protocol description, we follow the direct down-top approach. First, each party P_{2i-1}, $i \in [n/2]$,

Let $\mathcal{Q} \subseteq \mathcal{P}$ be a set of q parties and let $b := \lceil q/2 \rceil$. Further, let $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$ be a (deterministic) partition of $\mathcal{Q}$ into two disjoint subsets (each called a *committee*) of size $q_1 := \lceil q/2 \rceil$ and $q_2 := \lfloor q/2 \rfloor$, respectively. Hereafter, we use the notation $\langle m \rangle := \mathsf{Encode}(m_1, \ldots, m_b)$ for a (q, b)-Reed-Solomon code $\mathsf{RS} = (\mathsf{Encode}, \mathsf{Decode})$ and where $(m_1, \ldots, m_b)$ is a deterministic partition of m. We describe the protocol from the view of party P_i and let $l \in \{1, 2\}$ be such that $P_i \in \mathcal{Q}_l$.

- **Initialization.** Initialize empty lists $\mathsf{C}, \mathsf{T}, \mathsf{Z}$ of length 2 and set $\mathsf{Z}[j] := \perp$ (default value) for $j \in \{1, 2\}$. *// Variables are defined.*
- **Recursive Execution.** Run $\mathsf{GenAPVSS}(\mathcal{Q}_l)$ among parties in $\mathcal{Q}_l$ and let T_l denote the output. If $|\mathcal{Q}_l| = 1$, then obtain T_l by locally executing $T_l \leftarrow \mathsf{Dist}(sk_i, (pk_1, \ldots, pk_n))$. *// Both committees run the protocol recursively and output a PVSS transcript each.*
- **Accumulator Delivery.** Compute an accumulation value z_l for the encoding $\langle T_l \rangle$ and send z_l to all parties in $\mathcal{Q}$. Upon receiving the same value z_j from $\lfloor q_j/2 \rfloor + 1$ distinct parties in $\mathcal{Q}_j$ (i.e., a majority set of parties), update $\mathsf{Z}[j] := z_j$ for each $j \in \{1, 2\}$ at most once. *// Parties only accept the majority value z_j received from each committee $\mathcal{Q}_j$.*
- **Accumulator Agreement.** For each $j \in \{1, 2\}$, run Byzantine agreement BA_j on input $\mathsf{Z}[j]$ among parties in $\mathcal{Q}$. Let z_j denote the output, and update $\mathsf{Z}[j] := z_j$. *// Parties establish consensus on both accumulation values z_1, z_2 (one from each committee).*
- **Transcript Delivery.** If $\mathsf{Z}[l] \neq \perp$, then invoke $\mathsf{Deliver}$ on input $(\mathcal{Q}, T_l, \mathsf{Z}[l])$ among parties in $\mathcal{Q}$. Further, only participate in another instance of $\mathsf{Deliver}$ with respective accumulation value $z \neq \perp$ if $z \in \mathsf{Z}[\cdot]$. Upon decoding a message T_j (with accumulation value $\mathsf{Z}[j]$), update $\mathsf{T}[j] := T_j$ for each $j \in \{1, 2\}$ at most once. *// Parties efficiently broadcast their T_j to all parties in $\mathcal{Q}$.*
- **Committee Selection.** For each $j \in \{1, 2\}$, update the list $\mathsf{C}[j] := \mathsf{Ver}(\mathsf{T}[j], (pk_1, \ldots, pk_n)) \in \{0, 1\}$. For each $j \in \{1, 2\}$, run (binary) Byzantine agreement BA_j on input $\mathsf{C}[j]$ among parties in $\mathcal{Q}$. Let b_j denote the output bit, and update $\mathsf{C}[j] := b_j$. *// Parties decide on which committee(s) have correctly delivered a valid PVSS transcript T_j with resp. accumulation value z_j.*
- **Transcript Agreement.** For each $j \in \{1, 2\}$ such that $\mathsf{C}[j] = 1$, invoke $\mathsf{Deliver}$ on input $(\mathcal{Q}, \mathsf{T}[j], \mathsf{Z}[j])$ among parties in $\mathcal{Q}$. Upon decoding a message T_j (with respective index j) such that $\mathsf{C}[j] = 1$ and $\mathsf{Ver}(T_j, (pk_1, \ldots, pk_n)) = 1$, update $\mathsf{T}[j] := T_j$ at most once. *// This ensures that T_j reaches all honest parties in $\mathcal{Q}$.*
- **Final Aggregation.** Compute the aggregation $AT := \mathsf{Agg}(\mathsf{T}[1], \mathsf{T}[2])$ and output. *// Aggregate both transcripts and terminate.*

Figure 5.1: Description of GenAPVSS for the set $\mathcal{Q} \subseteq \mathcal{P}$ from the view of party P_i.

Let $\mathcal{P} := \{P_1, \ldots, P_n\}$ be the n parties. The protocol outputs a vector of secret key shares $(SK_1, \ldots, SK_n) \in \mathbb{G}_2^n$ where SK_j is known only to P_j, a vector of public key shares $(PK_1, \ldots, PK_n) \in \mathbb{G}_1^n$, and a public key $PK \in \mathbb{G}_1$.

- **Transcript Generation.** Run $\mathsf{GenAPVSS}(\mathcal{P})$ among all parties in $\mathcal{P}$ and obtain a PVSS transcript $AT := \{\mathbf{C}, \mathbf{E}, \pi\}$ from the execution. *// This generates a common (aggregated) PVSS transcript AT for all parties in $\mathcal{P}$.*
- **Key Derivation.** Compute the decryption $D_i := \mathsf{Dec}_{sk_i}(E_i)$. Terminate with output $(PK_1, \ldots, PK_n) := (C_1, \ldots, C_n)$ and $SK_i := D_i$. *// Parties derive their secret key shares non-interactively from the PVSS transcript AT. In particular, these key shares interpolate a degree-t polynomial $f \in \mathbb{Z}_p[X]$ in the exponent.*

Figure 5.2: Description of our DKG protocol GRand from the view of party P_i.

generates a random (t, n)-threshold PVSS transcript T_{2i-1} by itself, which encodes a degree-t polynomial $f_{2i-1} \in \mathbb{Z}_p[X]$ in the exponent. Then, each party P_{2i-1} interacts with its neighbor P_{2i} with the goal to exchange their transcripts and aggregate them to a single (aggregated) transcript $AT_i := \mathsf{Agg}(T_{2i-1}, T_{2i})$. Then, we split $\mathcal{P}$ into disjoint sets of neighboring parties (that we call *committees*) $Q_i := \{P_{2i-1}, P_{2i}\}$ for all $i \in [n/2]$, so that

$$Q_1 = \{P_1, P_2\},\ Q_2 = \{P_3, P_4\},\ \ldots,\ Q_{n/2} = \{P_{n-1}, P_n\}.$$

We consider committee Q_1 and its interaction with Q_2, but note that each committee Q_j, $j \in [n/2]$, executes the same instructions. Let AT_1 and AT_2 denote the transcripts established by committees Q_1 and Q_2, respectively. First, each party $P_i \in Q_1$ generates an accumulation value z_1 for AT_1 and sends it to all parties in $Q := Q_1 \cup Q_2$. Then, parties in Q collectively execute two instances of Byzantine agreement BA in order to have consensus on the values z_1, z_2. Following this, each party $P_i \in Q_1$ broadcasts AT_1 to all parties in Q using $\mathsf{Deliver}$ on input (AT_1, z_1). In the next step, parties in Q collectively execute two instances of (binary) Byzantine agreement BA in order to decide which committee(s) delivered its transcript correctly. Finally, each party $P_i \in Q$ broadcasts AT_j, $j \in \{1, 2\}$, to all parties in Q using $\mathsf{Deliver}$ on input (AT_j, z_j) in case the committee Q_j was determined to be successful in the previous step. This ensures that the transcripts reach all honest parties. Parties conclude by locally aggregating $AT := \mathsf{Agg}(AT_1, AT_2)$ the valid PVSS transcripts and progress to the next level of iteration. This high-level idea applies to all levels $\ell \in [\log n]$ of the iteration/recursion: the two neighboring committees Q_{2i-1} and Q_{2i} of size $q/2 := 2^{\ell-1}$ each interact with each other to exchange their newly generated PVSS transcripts from the previous level and finally aggregate them. At the end of the protocol, all parties obtain the same, aggregated PVSS transcript AT from which they directly derive the key shares without any further interaction.

Security Analysis. Let A be an adversary that (t, ε, T)-breaks security of GRand. Using this adversary, we build a reduction against the aggregated unforgeability of the underlying aggregatable PVSS scheme APVSS. For this, we split our proof into two parts. First, we provide a simulation of the aggregated unforgeability experiment to A via a sequence of games. In particular, we interpolate between some games using reductions against the security of the

Byzantine agreement protocol BA and the security of the cryptographic accumulator scheme AC. Second, we bound A's winning probability in the final game by providing an efficient reduction against the aggregated unforgeability of APVSS. We consider the following sequence of games with A as adversary. Throughout the analysis, we denote by $C \subset [n]$ the set of corrupt parties and by $\mathcal{H} := [n] \setminus C$ the set of honest parties.

GAME $\mathbf{G}_0$: This is the real game. In particular, the game samples system parameters *par* and initializes a corruption set $C := \emptyset$ and updates $\mathcal{H} := [n] \setminus C$ throughout the game. Then, the game runs A on input *par* with access to a corruption oracle. Whenever A decides to corrupt a party $P_i \in \mathcal{H}$, the game faithfully returns the internal state of party P_i to A and updates $C := C \cup \{i\}$. Henceforth, A gets full control over P_i. Further, all honest parties follow the protocol instructions for the DKG protocol as specified in Figure 5.1. In particular, at the beginning of the protocol, each party P_i honestly samples a polynomial $f_i \in \mathbb{Z}_p[X]$ of degree t and computes a PVSS transcript T_i. After the protocol execution, each party P_i outputs a transcript AT and derives the public key PK, the vector of public key shares, and its secret key share SK_i. At the end of the game, A outputs a secret S^*. It wins the game if $|C| \leq t$ and $S^* = SK$. Clearly, A's advantage in winning the game is given by

$$\Pr[\mathbf{G}_0 = 1] = \varepsilon.$$

GAME $\mathbf{G}_1$: This game is identical to the previous game, except that we add an abort condition. The idea of this hybrid is to rule out failure of the protocol Deliver. Namely, whenever an instance of Deliver fails to output the correct message, the game aborts. At each level $r \in [\log n]$ of the recursion, there are $2n \cdot 2^{r-1}$ instances of Deliver. Summing these up over all levels, we obtain

$$\sum_{r=1}^{\log n} 2n \cdot 2^{r-1} = 2n \cdot \sum_{r=1}^{\log n} 2^{r-1} \leq 2n^2.$$

It is easy to see that Deliver only fails when an invalid proof of membership is received. As such, the probability of failure of Deliver is directly given by the probability of finding a collision for the cryptographic accumulator scheme AC underlying Deliver. As this probability is given by ε_C, we can bound the winning probability of this game by

$$\Pr[\mathbf{G}_1 = 1] \geq \Pr[\mathbf{G}_0 = 1] - 2n^2\varepsilon_C.$$

GAME $\mathbf{G}_2$: This game is identical to the previous game, except that we add another abort condition. The idea of this hybrid is to rule out failure of the consensus protocol BA. Namely, whenever an instance of the Byzantine agreement protocol BA fails to establish consensus, the game aborts. At each level $r \in [\log n]$ of the recursion, there are $4 \cdot 2^{r-1}$ instances of the protocol (each of the 2^{r-1} committees executes two instances of BA for accumulation value agreement and two instances of BA for committee selection). Summing these up over all levels, we obtain

$$\sum_{r=1}^{\log n} 4 \cdot 2^{r-1} = 4 \cdot \sum_{r=1}^{\log n} 2^{r-1} \leq 4n.$$

As each instance of BA fails with probability ε_B, we can bound the winning probability of this game by

$$\Pr[\mathbf{G}_2 = 1] \geq \Pr[\mathbf{G}_1 = 1] - 4n\varepsilon_B.$$

GAME $\mathbf{G}_3$: This game is identical to the previous game, except that we add another abort condition. So far we have rule out failure of the distributed protocols BA and Deliver. From Lemma 5.4.2, it follows that the protocol execution then established the same aggregated transcript AT for all parties that has contribution from at least one honest party. As such, the idea of this hybrid is to guess this special party $P^* \in [n]$ that contributes to the aggregate AT and that remains honest until the end of the game. Concretely, at the beginning of the game, the game makes a random guess by sampling $i^* \leftarrow_{\$} [n]$ and executes the game as in $\mathbf{G}_2$. At the end of the game, the game aborts if $P_{i^*} \neq P^*$ or $P_{i*} \notin \mathcal{H}$. Since the choice of i^* remains information-theoretically hidden from A's view, we can bound the winning probability of this game by

$$\Pr[\mathbf{G}_3 = 1] \geq \Pr[\mathbf{G}_2 = 1]/(2n).$$

Note that the factor n comes from the condition $P_{i^*} = P^*$, while the factor 2 comes from the condition $P_{i^*} \in \mathcal{H}$. It remains to bound the probability that the final game $\mathbf{G}_3$ outputs 1. For that, we build an efficient reduction R against the aggregated unforgeability of APVSS. The design should be straightforward.

Building a Reduction. At the beginning of the aggregated unforgeability experiment, R submits a submits a request $(\mathsf{givePVSS}, i^*)$ and obtains a PVSS transcript T_{i^*}. Then it simulates the game $\mathbf{G}_3$ to A by sampling random degree-t polynomials $f_i \in \mathbb{Z}_p[X]$ for all $i \in \mathcal{H} \setminus \{i^*\}$ and computing corresponding PVSS transcripts T_i. For party P_{i^*}, however, it uses T_{i^*}. Whenever A decides to corrupt a party $P_i \in \mathcal{H}$, the reduction R simply forwards this query to its own challenger for the aggregated unforgeability experiment and returns the output to the adversary A. In this manner, R can correctly answer all corruption queries of A. At the end of the simulation to A, the adversary A outputs a secret x^* to R. We assume that the adversary outputs a correct forgery x^*, so that x^* is the secret of the final aggregated transcript AT which has contribution from P_{i^*}. Now, the reduction R outputs the tuple (AT, T_{i^*}, x^*) to the challenger of the aggregated unforgeability experiment. In particular, the winning conditions are satisfied: (i) $|C| \leq t$ is clear, since the same holds true for the adversary A. (ii) $\mathsf{Ver}((pk_1, \ldots, pk_n), AT) = 1$ is clear, since the DKG protocol execution succeeded by assumption. (iii) The existence of an index $i \in \mathcal{H}$ such that $\mathsf{OwnId}(AT, pk_i) = 1$ is clear, since i^* is this index. Having said this, it follows immediately that we can bound the winning probability of the final game by

$$\Pr[\mathbf{G}_3 = 1] \leq \varepsilon_A.$$

Overall, we obtain the final bound

$$\varepsilon_A \geq \frac{1}{2n}\left(\varepsilon - 2n^2\varepsilon_c - 4n\varepsilon_B\right) \iff \varepsilon \leq 2n\left(\varepsilon_A + 2\varepsilon_B + n\varepsilon_C\right).$$

We proceed with the proof for the statement that the DKG protocol is complete assuming no failure of the distributed protocols BA and Deliver. Here, complete means that all honest parties at the end of the protocol execution output the same transcript AT that has contribution from at least one honest party.

Lemma 5.4.2. *If the protocols* BA *and* Deliver *are perfectly secure, then* GRand *(cf. Figures 5.1 and 5.2) is a complete DKG protocol in the sense that all honest parties output the same transcript AT with contribution from at least one honest party at the end of the protocol.*

Proof. We will show this lemma using an iterative argument. Concretely, we show that the recursive protocol GenAPVSS executed among a set of parties $Q := Q_1 \cup Q_2$ succeeds under the assumption that Q has honest majority. By this we mean that after termination, all honest parties output a common transcript AT that has contribution from at least one subcommittee among $\{Q_1, Q_2\}$ with honest majority itself (the existence of such a subcommittee is clear, otherwise Q itself would not have honest majority). From this, it then follows that GRand is complete, since at least one of the two committees $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$ from the first recursion step has honest majority (as we assume $t < n/2$), say $\mathcal{P}_1$ for example. Then, the same argument also applies to $\mathcal{P}_1$ and its splitting $\mathcal{P}_1 = \mathcal{P}_{1,1} \cup \mathcal{P}_{1,2}$, and so on. Since at the bottom level, there is a pair of neighboring parties that form a committee, we know that both their initially sampled PVSS transcript will be included in all subsequent levels and thus also in the final aggregate AT.

We start with the proof. Consider a committee $Q = Q_1 \cup Q_2$ with honest majority among which the protocol GenAPVSS is executed. It follows that one of the two committees Q_1, Q_2 also has honest majority, and we assume w.l.o.g. Q_1 to be this committee. Further, we assume that parties in Q_1 have already established a common PVSS transcript T_1. We will show that at the end, all parties in Q obtain a common PVSS transcript AT that has contribution T_1. In the first step of the protocol, all parties in Q_1 generate an accumulation value z_1 for T_1 and then send it to all parties in Q. Since Q_1 has honest majority and we assume that all messages are signed by a party before it sends it, all parties in Q will set their local accumulation value list as $\mathsf{Z}[1] := \{z_1\}$ and have agreement on it (without possibly knowing this). Then, parties run two instances of the Byzantine agreement protocol, which by consistency establishes the same accumulation values z_1, z_2 for all parties in Q. By validity of BA, we know that z_1 is the correct accumulation value for T_1, which was sent by an honest majority of parties in Q. In the next step, each party in Q_1 broadcasts T_1 to all parties in Q using Deliver. Since this protocol uses Reed-Solomon codes with a reconstruction threshold of $q/2 + 1$ (majority threshold) and there are at least $q/2 + 1$ honest parties in Q, all parties in Q will reconstruct the correct transcript T_1 after this step. On the other hand, we cannot say anything regarding a hypothetical transcript T_2 with accumulation value z_2 coming from the other committee Q_2. It could be that corrupt parties in Q_2 (which can form the majority in that committee) send a valid transcript T_2 to only some honest parties in Q or even none so that not all honest parties might be able to reconstruct the full message. Therefore, in the next step, parties execute two instances $2\mathsf{BA}_1, 2\mathsf{BA}_2$ of binary Byzantine agreement on input 1 if it was able to reconstruct a transcript T_i with accumulation value z_i and on input 0 otherwise. The security guarantees of Byzantine agreement now have the following implications. If the output for say B_i is 1, then there must have been at least one honest party that provided the input 1 into the protocol. In particular, that honest party was able to reconstruct a transcript T_i with accumulation value z_i from the previous step. If the output for BA_i is 0, then we cannot say anything further (as it could be that some honest parties have input 0 and others 1). But since all honest parties in Q were able to reconstruct the full transcript T_1 coming from the (honest) committee Q_1, we know that all honest parties input 1 into BA_1 and therefore by validity of Byzantine agreement they all output 1.

In the final step of the protocol, all parties that have a transcript T_i with accumulation value z_i send it to all other parties in Q using the Deliver protocol. If the output of BA_i was 0, then parties simply ignore any transcript T_i delivered by any party. If the output of BA_i was 1, then all honest parties will reconstruct T_i, as there was at least one honest party that invoked the deliver protocol on T_i by the properties of BA as already clarified. As BA_1 has output 1, we

know that all honest parties will have the same transcript T_1 at the end of this step. Further, if BA_2 has also output 1, then likewise all honest parties will have the same transcript T_2 at the end of this step. If BA_2 has output 0, then we know that all honest parties will simply ignore any transcript T_2 delivered by any party and thus all honest parties will agree on $T_2 = \emptyset$ simply. Having said all this, we have shown that in case the committee Q has honest majority, then all honest parties will end up with the same transcripts T_1 and T_2 (coming from the children committees Q_1 and Q_2) where at least one of them is a valid and true PVSS transcript $T_i \neq \emptyset$. As a result, after the local aggregation step of $\{T_i\}_{i\in[2]}$, all honest parties obtain the same transcript $AT := \mathsf{Agg}(T_1, T_2)$. This concludes our discussion on our initial goal to show that all honest parties terminate GenAPVSS with the same aggregated transcript that has at least one honest contribution. Termination of the protocol is clear, as all building blocks are deterministic and run in a finite number of rounds. □

Communication Complexity. In the following, we measure the communication complexity of GRand. We begin by focusing on one particular level of the recursive protocol GenAPVSS and then sum over the total number of $\log n$ levels. For this, let us consider the level $\ell \in [\log n]$ of the recursion tree ($\ell = 1$ denotes the bottom level in which parties form committees of size 2) with Q_1 and Q_2 each of size $2^{\ell-1}$ that merge into Q which is of size $q := 2^\ell$. We count the communicated bits among honest parties in the committee Q and then multiply this with the number n/q of parent committees at that particular level ℓ. However, in contrast to the previous analysis we do not now assume that Q has honest majority. The reason for this is that even though our protocol is deterministic, it could be possible that in a corrupt majority committee the honest parties communicate much more bits (e.g., $O(\lambda q^3)$ bits) than in an honest majority committee which could blow up the overall communication complexity. Thus, we do not make any assumptions on Q and its children committees Q_1, Q_2.

We begin with the analysis assuming the worst-case scenario in which both committees are corrupt and all honest parties in both committees start each with a different transcript. In the first stage, each party in Q multicasts an accumulation value of size $O(\lambda)$ to all other parties in Q. Thus, this step takes a total communication of $O(\lambda q^2)$ bits, as the number of parties in Q is q. In the next stage, the parties execute two instances of the Byzantine agreement protocol BA which itself has a communication complexity of $O(\lambda q^2 \log q)$ bits. Here, we assume the scenario where the adversary lets the honest parties agree all on a different accumulation value which is the one from its own local PVSS transcript. Afterwards, parties invoke the deliver protocol on their PVSS transcript which is of size $O(\lambda n)$. By definition of the deliver, each party splits its transcript into chunks of size $O(\lambda n/q)$ and sends its chunk to one particular party. Therefore, the total communication complexity of this step is $q \cdot O(\lambda n/q \cdot q) = O(\lambda nq)$ bits. In the next step of the deliver protocol, each party multicasts a chunk it received from a different party only in case the augmented accumulation value corresponds to its own accumulation value and it does so only once in total. Hence, this step also incurs the same number of communicated bits which is $O(\lambda nq)$. Following this, the next two steps of the recursive protocol are identically to the preceding two steps: two instances of (binary) Byzantine agreement followed by an invocation of the deliver protocol. As a result, we obtain $O(\lambda nq + \lambda q^2 \log q)$ bits for the total communication complexity of honest parties in the committee Q of size q. Since there are n/q such parent committees, we get a communication complexity of $n/q \cdot O(\lambda nq + \lambda q^2 \log q) = O(\lambda n^2 + \lambda nq \log q)$ bits for that particular level. By summing over all levels for $\ell \in [\log n]$ with $q = 2^\ell$, we obtain a total

communication complexity of

$$\sum_{\ell=1}^{\log n} O(\lambda n^2 + \lambda n q \log q) = O(\lambda n^2 \log n) + \sum_{\ell=1}^{\log n} O(\lambda n 2^\ell \ell)$$
$$\leq O(\lambda n^2 \log n) + \sum_{\ell=1}^{\log n} O(\lambda n 2^\ell \log n) \leq O(\lambda n^2 \log n) + O(\lambda n^2 \log n).$$

bits, which is identical $O(\lambda n^2 \log n)$ as claimed. This concludes our discussion on the communication complexity of GRand.

Round Complexity. We proceed with the round complexity of GRand which is the same as the one of the recursive protocol GenAPVSS. For this, we first give a formula for the round complexity of our Byzantine agreement protocol in Appendix 5.7.1. Denote by $r(n)$ the round complexity of the protocol that consists of two sequential executions of the four-round graded Byzantine agreement protocol GBA with two rounds of proposal phases and two sequential executions of the Byzantine agreement protocol recursively on committees of size $n/2$. From this observation, we easily derive the recursive formula

$$r(n) = (4 + r(n/2) + 1) + (4 + r(n/2) + 1) = 2r(n/2) + 10,$$

where at the lowest level of the recursion we have $r(1) = 0$, since a Byzantine agreement protocol involving a single party is trivial. It can easily be seen that $r(n) = 10n - 10$ is the correct solution for this recursive formula. We use this now to establish a formula for the round complexity of our recursive protocol GenAPVSS. The protocol consists of the following steps: two parallel executions of the protocol itself with committees of size $n/2$, a one-round accumulator proposal step, two parallel executions of Byzantine agreement protocol, an invocation of the two-round deliver protocol, again two parallel executions of Byzantine agreement, and finally again an invocation of the two-round deliver protocol. From this observation, we derive for the round complexity $R(n)$ of GenAPVSS the following recursive formula

$$\begin{aligned} R(n) &= (R(n/2) + 1) + (r(n) + 2) + (r(n) + 2) \\ &= R(n/2) + 2r(n) + 5 \\ &= R(n/2) + 20n - 15, \end{aligned}$$

where at the lowest level of the recursion we trivially have $R(1) = 0$. Again this can be solved using standard techniques, which gives us $R(n) = 40n - 15\log(n-2) - 40$. This concludes our security and complexity analysis of GRand. □

Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a pairing with generators $g \in \mathbb{G}_1, h \in \mathbb{G}_2$. Let $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{G}_1$ and $\mathsf{H}_2 : \mathbb{G}_T \to \{0,1\}^\lambda$ be two cryptographic hash functions modeled as random oracle. Hereafter, let $g_r := \mathsf{H}_1(r)$ for all $r \in \mathbb{N}$.

- **Setup Phase.** Parties execute the DKG protocol GRand and obtain a vector of secret key shares $(SK_1, \ldots, SK_n) \in \mathbb{G}_2^n$, a vector of public key shares $(PK_1, \ldots, PK_n) \in \mathbb{G}_1^n$, and a public key $PK \in \mathbb{G}_1$. *// This serves as setup for the randomness beacon which starts following a one-time, one-round commitment phase.*
- **Commitment Phase.** Initialize an empty local set $\mathcal{G} := \emptyset$. Sample $\alpha_i \leftarrow_\$ \mathbb{Z}_p^*$ uniformly at random and multicast (i.e., send to all parties) the commitment $\mathsf{cm}_i := (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Upon receiving $\mathsf{cm}_j = (\mathsf{cm}_{j,1}, \mathsf{cm}_{j,2})$ from party P_j, check if equality $e(PK_j, h) = e(\mathsf{cm}_{j,1}, h)e(g, \mathsf{cm}_{j,2})$ holds. Only if this equality holds, update the set $\mathcal{G} := \mathcal{G} \cup \{P_j\}$. *// This step is done only once, and each party stores the commitments* cm_j *it received from other parties.*
- **Beacon Epoch r.** Compute the share $\sigma_i := (g_r^{\alpha_i}, e(g_r, SK_i))$ along with the NIZK proof $\pi_i := \mathsf{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$, and multicast (σ_i, π_i). Upon receiving (σ_j, π_j) from any party $P_j \in \mathcal{G}$, check if π_j verifies using $\mathsf{cm}_{j,1}$ and $\sigma_{j,1}$. Further, check if equality $\sigma_{j,2} = e(g_r, \mathsf{cm}_{j,2})e(\sigma_{j,1}, h)$ holds.
- **Reconstruction Phase.** Upon receiving $t+1$ valid tuples $\{(\sigma_j, \pi_j)\}_{j \in \mathcal{S}}$ from distinct parties in $\mathcal{G}$, compute $\sigma := e(g_r, SK)$ by Lagrange interpolation in the exponent from $\{\sigma_{j,2} = e(g_r, SK_j)\}_{j \in \mathcal{S}}$. *// Only local computation without interaction.*
- **Beacon Output.** Upon reconstruction of σ in epoch r, output the beacon value as $\varrho_r := \mathsf{H}_2(\sigma) \in \{0,1\}^\lambda$. *// The beacon value is output as soon as $t+1$ valid tuples are received.*

Figure 5.3: Description of our DRB protocol GRandLine from the view of party P_i.

5.5 Distributed Randomness Beacon

In this section, we design an efficient and simple randomness beacon protocol. The construction is inspired by the threshold VUF design of Gurkan et al. [Gur+21b] and can be thought of as an optimized version of their protocol that comes with an adaptive security proof. As such, our protocol is comparable with the threshold BLS signature.

5.5.1 Our Design

In this section, we present our randomness beacon GRandLine. For a formal description of the protocol, we refer to Figure 5.3 below. Let H_1 and H_2 be hash functions modeled as random oracle and denote $g_r := \mathsf{H}_1(r)$ for all $r \in \mathbb{N}$. Parties begin by executing GRand upon which every party P_i obtains a public-secret key pair $(PK_i, SK_i) := (g^{f(i)}, h^{f(i)})$ for a hidden polynomial $f \in \mathbb{Z}_p[X]$ of degree t. The idea now is to use $\vartheta_i := e(g_r, SK_i) \in \mathbb{G}_T$ as a partial signature on the epoch number $r \in \mathbb{N}$, obtain the full signature $\vartheta := e(g_r, SK)$ via Lagrange interpolation in

the exponent from enough shares, and derive the randomness beacon value as $\mathsf{H}_2(\vartheta) \in \{0,1\}^\lambda$. However, the problem with a naive implementation of this approach is that no party can verify the correctness of a received share $\vartheta_i = e(g_r, h)^{f(i)}$. In order to resolve this issue, we augment the signature shares with additional elements from which its correctness can be checked via pairing equations. For this, we follow an economical two-step approach. After DKG setup, each party P_i locally samples an $\alpha_i \leftarrow_\$ \mathbb{Z}_p^*$ uniformly at random and multicasts (i.e., sends to all parties) $\mathsf{cm}_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Correctness of its second component can be checked via a pairing equation. After this commitment phase, the actual randomness beacon starts. For epoch $r \geq 1$, each party computes $\sigma_i := (g_r^{\alpha_i}, \vartheta_i)$ along with a Chaum-Pedersen NIZK proof of discrete logarithm equality [CP93] as $\pi_i := \mathsf{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$ to prove correctness of $g_r^{\alpha_i}$. Upon receiving such a tuple, any party can verify the correctness of the partial signature ϑ_i using a pairing equation. Having done this, each party can compute the randomness beacon value for epoch r as described above. Overall, partial signatures consist of two group elements along with a simple proof of discrete logarithm equality. And verification of a share takes a single pairing equation with two pairing operations (as for the regular BLS signature).

5.5.2 Security and Complexity Analysis

In this section, we give a security and complexity analysis of our randomness beacon protocol GRandLine. On an high level, consistency and availability follow from uniqueness of the signature $\vartheta := e(\mathsf{H}_1(r), SK)$ (per message r and public key PK) and soundness of the Chaum-Pedersen NIZK proof system for discrete logarithm equality. Unpredictability follows from unforgeability of ϑ, and the final hash operation $\mathsf{H}_2(\vartheta)$ guarantees uniformity in the random oracle model. Essentially, these are the standard arguments for the transformation from unique threshold signatures to randomness beacons [CKS05], but with the additional argument of soundness of NIZK proofs for correctness of the threshold signature. Using techniques from [BL22a] to handle adaptive corruptions, we give a security reduction from the hardness of n-co-OMDL to the unforgeability of the threshold signature.

Intuitively, the adversary essentially has three options to successfully forge a signature $\vartheta^* = e(\mathsf{H}_1(r^*), SK)$ on some message (i.e., epoch number) r^*. It either finds the secret key SK, the encryption secret α_i for an honest party's $i \in [n]$, or the discrete logarithm of the element $\mathsf{H}_1(r^*)$. But this should be infeasible given secrecy of the underlying DKG protocol and the ElGamal encryption used for commitments $\mathsf{cm}_i := (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Further, the output BLS signatures $\mathsf{H}_1(1)^{\alpha_i}, \mathsf{H}_1(2)^{\alpha_i}, \ldots$ do not reveal additional information. We make this intuition sound by building a reduction that embeds the n-co-OMDL challenge ξ in either the PVSS transcript of some party (that remains honest at the end), in the ElGamal encryption secrets $\alpha_1, \ldots, \alpha_n$ of parties, or in the random oracle outputs $\mathsf{H}_1(\cdot)$, a choice that remains hidden from the adversary. Leveraging the algebraic group model, we are able to solve ξ using the polynomial equations that come from the forgery and additional data output by the adversary.

Theorem 5.5.1. *If* n**-co-OMDL** *is* (ε_A, T_A)*-hard in the AGM and* BA *is* (t, ε_B, T_B)*-secure, then* GRandLine *(cf. Figure 5.3) is a* $(t, \varepsilon, T, L, q_h, 1)$*-secure randomness beacon protocol in the AGM+ROM, where*

$$\varepsilon \leq Ln\left(12\varepsilon_A + 4n\varepsilon_B + \frac{q_h^2 + 4q_h}{2p}\right), \quad T \geq T_A + T_B + O(Ln^2).$$

Further, GRandLine *has a communication complexity of* $O(\lambda n^2)$ *bits per epoch, and each epoch takes one asynchronous round.*

Subsequently, we give a proof for Theorem 5.5.1. Since the claim on the communication and round complexity of GRandLine is trivial to verify, we will only focus on the security analysis.

Proof. In the following, we prove 1-unpredictability and bias-resistance of our randomness beacon GRandLine. We do this by showing that it is hard for an algebraic adversary to output a future randomness beacon value that is valid. Since our randomness beacon values are derived as a unique (deterministic) threshold signature from threshold keys output by GRand, it is enough to show that the adversary cannot produce a forged signature for a future epoch. Since we hash the final epoch signature through a random oracle H_2, the 1-unpredictability and bias-resistance of GRandLine follows.

Having said that, let A be an algebraic algorithm that $(t, \varepsilon, T, L, q_h, 1)$-breaks unpredictability of GRandLine, and let $\xi = (\xi_1, \ldots, \xi_n) \in (\mathbb{G}_1 \times \mathbb{G}_2)^n$ be the co-one-more discrete logarithm challenge of degree n with corresponding oracle $\mathsf{DL}_g()$ where $\xi_i = (\xi_{i,1}, \xi_{i,2}) = (g^{z_i}, h^{z_i})$ for all $i \in [n]$. Without loss of generality, we assume that A queries the random oracle H_2 before producing its prediction $\varrho_r := \mathsf{H}_2(\sigma)$ for some round $r \in [L]$. Further, we assume that all parties are honest prior to the execution of the protocol. It is straightforward how to adjust the proof to the general case. Hereafter, let $C \subset [n]$ be the dynamically changing set of corrupt parties and let $\mathcal{H} := [n] \setminus C$ be the set of honest parties. Initially, we have $C = \emptyset$. We consider the following sequence of games with A as adversary.

Game $\mathbf{G}_0$: This is the real game. Generate the system parameters $par = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, h, e)$ where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an asymmetric type 3 pairing of prime order p cyclic groups with generators $g \in \mathbb{G}_1, , h \in \mathbb{G}_2$. For all indices $i \in [n]$, generate the key pairs as $(pk_i, sk_i) \leftarrow \mathsf{Keys}(par, i)$ such that $pk_i = h^{sk_i}$. Whenever A decides to corrupt a party P_i, return the internal state of that party and set $C := C \cup \{i\}$. Thereafter, A gets full control over P_i. Execute the DKG protocol GRand on behalf of the honest parties and let $(PK_1, \ldots, PK_n)$ and $(SK_1, \ldots, SK_n)$ be the vector of public and secret key shares, respectively. For all $i \in \mathcal{H}$, execute the commitment phase by sampling $\alpha_i \leftarrow_\$ \mathbb{Z}_p^*$ uniformly at random and publishing $\mathsf{cm}_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Answer random oracle queries r_i to H_1 by sampling $\gamma_i \leftarrow_\$ \mathbb{Z}_p$ and returning $H_1[r_i] := g^{\gamma_i} \in \mathbb{G}_1$. Answer random oracle queries s_i to H_2 by sampling $\varrho_i \leftarrow_\$ \{0,1\}^\lambda$ and returning $H_2[s_i] := \varrho_i$. For all $i \in \mathcal{H}$ and epoch numbers $r \geq 1$, compute the beacon share σ_i of party P_i as $(g_r^{\alpha_i}, e(g_r, SK_i))$ along with a proof of discrete logarithm equality $\pi_i := \mathsf{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$ certifying the correctness of $g_r^{\alpha_i}$ and publish it. Output the beacon value $\varrho_r := e(g_r, SK)$ for epoch r (either by Lagrange interpolation in the exponent from beacon shares $\{e(g_r, SK_i)\}_{\mathcal{S}}$ or directly from the knowledge of the secret key SK). At any point of the game, say in epoch r, A outputs a prediction (ϱ_ℓ^*, ℓ) for an epoch $\ell \in [L]$. The adversary wins the game if $\ell > r$ and $\varrho_\ell^* = \varrho_\ell$ where $\varrho_\ell := \mathsf{H}_2(e(g_\ell, SK))$. Clearly, A's advantage in winning

the game is per definition given by

$$\Pr[\mathbf{G}_0 = 1] = \varepsilon.$$

GAME $\mathbf{G}_1$: This game is identical to the previous game, except that we add an abort condition. Before the execution of the game, sample a guess $\ell^* \leftarrow_\$ [L]$ uniformly at random. Then, execute the game as before and abort the game if $\ell \neq \ell^*$. Since the choice of ℓ^* remains hidden from A and does not affect the subsequent execution of the game, we bound the winning probability of this game by

$$\Pr[\mathbf{G}_1 = 1] \geq \Pr[\mathbf{G}_0 = 1]/L.$$

GAME $\mathbf{G}_2$: This game is identical to the previous game, except that we reprogram the random oracle H_1 on input ℓ differently (note that $\ell = \ell^*$ is the epoch for which A provides a prediction, i.e., a forgery for the underlying threshold signature). To this end, program H_1 on input r_i as follows. For $r_i \neq \ell$, sample $\gamma_i \leftarrow_\$ \mathbb{Z}_p$ uniformly at random and return $H_1[r_i] := g^{\gamma_i}$. For $r_i = \ell$, however, return $H_1[\ell] := \xi_{1,1}$ where $\xi_{1,1} \leftarrow_\$ \mathbb{G}_1$ is some randomly sampled group element. Clearly, this game is indistinguishable from the previous one so that there is no change in the winning probability, i.e.,

$$\Pr[\mathbf{G}_2 = 1] = \Pr[\mathbf{G}_1 = 1].$$

GAME $\mathbf{G}_3$: This game is identical to the previous game, except that we add another abort condition. The idea of this hybrid is to guess a party $P_{i^*} \in [n]$ that contributes to the keys generated from GRand and that remains honest until the end of the game. Note that the keys output by GRand are directly derived from the final APVSS transcript $AT \leftarrow \mathsf{GenAPVSS}$ which is just an aggregation of several initially sampled PVSS transcripts with contribution from at least one honest party P^*. Before the execution of the game, make a guess by sampling $i^* \leftarrow_\$ [n]$. Then, execute the game as before and let $P^* \in \mathcal{H}$ be the honest party whose initial PVSS transcript is included in the final aggregated transcript AT during the execution of GRand. At the end, abort the game if $P_{i^*} \neq P^*$. Since the choice of i^* remains information-theoretically hidden from A's view, we bound the winning probability of this game by

$$\Pr[\mathbf{G}_3 = 1] \geq \Pr[\mathbf{G}_2 = 1]/n.$$

GAME $\mathbf{G}_4$: This game is identical to the previous game, except that we add another abort condition. Namely, whenever an instance of the Byzantine agreement protocol BA fails to establish consensus, the game aborts. At each level $r \in [\log n]$ of the recursive setup phase, there are $4 \cdot 2^{r-1}$ instances of the consensus protocol (each of the 2^{r-1} committees executes two instances of BA to agree on accumulation values and two instances of BA to agree on which PVSS transcripts of the children committees to aggregate). Summing these up over all levels, we obtain

$$\sum_{r=1}^{\log n} 4 \cdot 2^{r-1} = 4 \cdot \sum_{r=1}^{\log n} 2^{r-1} \leq 4n.$$

As each instance of BA fails with probability ε_B, we can bound the winning probability of this game by

$$\Pr[\mathbf{G}_4 = 1] \geq \Pr[\mathbf{G}_3 = 1] - 4n\varepsilon_B.$$

Game $\mathbf{G}_5$: This game is identical to the previous game, except that we add another abort condition. Namely, whenever the adversary can forge a NIZK proof of knowledge of discrete logarithm θ for one of its PVSS transcripts, the game aborts. As the NIZK proof of our PVSS scheme is a regular Schnorr proof with statistical soundness, we may bound the soundness error simply by $1/p$. Since the adversary makes a total of q_h random oracle queries, we can bound the winning probability of this game by

$$\Pr[\mathbf{G}_5 = 1] \geq \Pr[\mathbf{G}_4 = 1] - \frac{q_h}{p}.$$

Game $\mathbf{G}_6$: This game is identical to the previous game, except that we add another abort condition. Namely, whenever the adversary can forge a NIZK proof of discrete logarithm equality π for one of his partial signatures during a randomness beacon epoch, the game aborts. As the NIZK proof of discrete logarithm equality is a standard Chaum-Pedersen proof with statistical soundness, we may bound the soundness error simply by $1/p$. Since the adversary makes a total of q_h random oracle queries, we can bound the winning probability of this game by

$$\Pr[\mathbf{G}_6 = 1] \geq \Pr[\mathbf{G}_5 = 1] - \frac{q_h}{p}.$$

Game $\mathbf{G}_7$: This game is identical to the previous game, except that we add another abort condition. Namely, whenever the adversary finds a collision among the random oracle queries to the hash function $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{G}_1$, the game aborts. As the adversary has a total of q_h tries and can run the birthday paradox algorithm, we bound the winning probability of this game by

$$\Pr[\mathbf{G}_7 = 1] \geq \Pr[\mathbf{G}_6 = 1] - \frac{q_h^2}{2p}.$$

As A is an algebraic adversary, at the end of the game it returns the forgery (ϱ_ℓ, ℓ) where $\varrho_\ell = \mathsf{H}_2(e(g_\ell, SK))$ together with an algebraic representation (w.l.o.g. we assume that the adversary queries the random oracle on $e(g_\ell, SK)$ before outputting ρ_ℓ)

$$\begin{aligned}\Big(&\{a_0, a_{i,1}, \ldots, b_{i,2}\}_{i=1}^{n}, \{b_{i,3}\}_{i=1}^{q_h}, \{r_{i,1}, \ldots, r_{i,n}\}_{i=1}^{q_s}, \{c_{i,j,1}, \ldots, c_{i,j,3}\}_{i,j=1}^{n},\\ &\{d_{i,1,2}, \ldots, d_{i,n,2}\}_{i=1}^{q_h}, \{e_{i,1,2}, \ldots, e_{i,n,2}\}_{i=1}^{q_h}, \{f_{i,1,2}, \ldots, f_{i,n,2}\}_{i=1}^{q_h},\\ &\{d_{i,j,1}, e_{i,j,1}, f_{i,j,1}\}_{i,j=1}^{n}, \{s_{i,1,1}, \ldots, s_{i,n,n}\}_{i=1}^{q_s}, \{t_{i,1,1}, \ldots, t_{i,n,n}\}_{i=1}^{q_s},\\ &\{u_{i,1,1}, \ldots, u_{i,n,n}\}_{i=1}^{q_s}, \{v_{i,1}, \ldots, v_{i,n}\}_{i=1}^{q_s}\Big)\end{aligned}$$

of elements in $\mathbb{Z}_p$ such that

$$\begin{aligned}e(g_\ell, SK) \stackrel{!}{=}\ & e(g,h)^{a_0} \cdot \prod_{i=1}^{n} e(g, Y_i)^{a_{i,1}} \cdot e(g, pk_i)^{a_{i,2}} \cdot e(g, \mathsf{cm}_{i,2})^{a_{i,3}}\\ & \cdot \prod_{i=1}^{n} e(C_i, h)^{b_{i,1}} \cdot e(\mathsf{cm}_{i,1}, h)^{b_{i,2}} \cdot \prod_{i=1}^{q_h} e(h_{1,i}, h)^{b_{i,3}} \cdot \prod_{i=1}^{q_s}\prod_{j=1}^{n} e(\sigma_{i,j}, h)^{r_{i,j}}\\ & \cdot \prod_{i,j=1}^{n} e(C_i, Y_j)^{c_{i,j,1}} \cdot e(C_i, pk_j)^{c_{i,j,2}} \cdot e(C_i, \mathsf{cm}_{j,2})^{c_{i,j,3}}\\ & \cdot \prod_{i,j=1}^{n} e(\mathsf{cm}_{i,1}, Y_j)^{d_{i,j,1}} \cdot \prod_{i=1}^{q_h}\prod_{j=1}^{n} e(h_{1,i}, Y_j)^{d_{i,j,2}}\\ & \cdot \prod_{i=1}^{q_s}\prod_{j,k=1}^{n} e(\sigma_{i,j}, Y_k)^{s_{i,j,k}} \cdot \prod_{i,j=1}^{n} e(\mathsf{cm}_{i,1}, pk_j)^{e_{i,j,1}}\\ & \cdot \prod_{i=1}^{q_h}\prod_{j=1}^{n} e(h_{1,i}, pk_j)^{e_{i,j,2}} \cdot \prod_{i=1}^{q_s}\prod_{j,k=1}^{n} e(\sigma_{i,j}, pk_k)^{t_{i,j,k}}\\ & \cdot \prod_{i,j=1}^{n} e(\mathsf{cm}_{i,1}, \mathsf{cm}_{j,2})^{f_{i,j,1}} \cdot \prod_{i=1}^{q_h}\prod_{j=1}^{n} e(h_{1,i}, \mathsf{cm}_{j,2})^{f_{i,j,2}}\\ & \cdot \prod_{i=1}^{q_s}\prod_{j,k=1}^{n} e(\sigma_{i,j}, \mathsf{cm}_{k,2})^{u_{i,j,k}} \cdot \prod_{i=1}^{q_s}\prod_{j=1}^{n} e(g_i, SK_j)^{v_{i,j}}. \end{aligned} \tag{1}$$

Here, the representation is split (from left to right) into powers of pairing evaluations on combinations of the generators g, h, the polynomial commitments $C_1, \ldots, C_n$ and encrypted shares $Y_1, \ldots, Y_n$ of the aggregated transcript output by $\mathsf{GenAPVSS}$ (which has contribution from the designated party P^*), the public keys $pk_1, \ldots, pk_n$ of parties (these constitute the setup phase), the auxiliary commitments $\mathsf{cm}_1, \ldots, \mathsf{cm}_n$ (these constitute the commitment phase), the answers to hash queries $h_{1,i} := \mathsf{H}_1(m_i), i \in [q_h]$, returned by the random oracle, and the beacon value shares (seen as partial signatures of the underlying threshold signature scheme) $e(g_i, SK_j)$, where $i \in [q_s]$ and $j \in [n]$, along with the correctness shares $\sigma_{i,j} := g_i^{\alpha_j}$ (recall that $g_i := \mathsf{H}_1(i)$ by definition). Here, q_s is defined as the current epoch in which the adversary outputs its prediction and thus $q_s < \ell$ by assumption (as the protocol is deterministic after the setup phase and parties do output the beacon values in sequence). Further, we assume w.l.o.g.

that $m_i = i$ for all $i \in [q_s]$. In the following, we let Q_h denote the set $[q_h] \setminus \{\ell\}$ (recall that ℓ is the index where the forgery happens). Then the above equation over $\mathbb{G}_T$ to base $e(g,h)$ yields

$$\begin{aligned}
\gamma_\ell f(0) \stackrel{!}{=}\ & a_0 + \sum_{i=1}^{n} a_{i,1} f(i) sk_i + a_{i,2} sk_i + a_{i,3}(-\alpha_i + f(i)) \\
& + \sum_{i=1}^{n} b_{i,1} f(i) + b_{i,2}\alpha_i + \sum_{i \in Q_h} b_{i,3}\gamma_i + \sum_{i=1}^{q_s}\sum_{j=1}^{n} r_{i,j}\gamma_i\alpha_j \\
& + \sum_{i,j=1}^{n} c_{i,j,1} f(i) f(j) sk_j + c_{i,j,2} f(i) sk_j + c_{i,j,3} f(i)(-\alpha_j + f(j)) \\
& + \sum_{i,j=1}^{n} d_{i,j,1}\alpha_i f(j) sk_j + \sum_{i \in Q_h}\sum_{j=1}^{n} d_{i,j,2}\gamma_i f(j) sk_j \\
& + \sum_{i=1}^{q_s}\sum_{j,k=1}^{n} s_{i,j,k}\gamma_i\alpha_j f(k) sk_k + \sum_{i,j=1}^{n} e_{i,j,1}\alpha_i sk_j \\
& + \sum_{i \in Q_h}\sum_{j=1}^{n} e_{i,j,2}\gamma_i sk_j + \sum_{i=1}^{q_s}\sum_{j,k=1}^{n} t_{i,j,k}\gamma_i\alpha_j sk_k \\
& + \sum_{i,j=1}^{n} f_{i,j,1}\alpha_i(-\alpha_j + f(j)) + \sum_{i \in Q_h}\sum_{j=1}^{n} f_{i,j,2}\gamma_i(-\alpha_j + f(j)) \\
& + \sum_{i=1}^{q_s}\sum_{j,k=1}^{n} u_{i,j,k}\gamma_i\alpha_j(-\alpha_k + f(k)) + \sum_{i=1}^{q_s}\sum_{j=1}^{n} v_{i,j}\gamma_i f(j) \\
& + \gamma_\ell\Big(b_{\ell,3} + \sum_{j=1}^{n} d_{\ell,j,2} f(j) sk_j + \sum_{j=1}^{n} e_{\ell,j,2} sk_j + \sum_{j=1}^{n} f_{\ell,j,2}(-\alpha_j + f(j))\Big).
\end{aligned}$$

Note that we have split the terms into those that contain the ℓ-th term and those that do not. As a result, the terms on the right-hand side of the equation other than the last one are independent from the variable γ_ℓ. By rewriting, we get the simplified identity

$$\gamma_\ell\left(b_{\ell,3} - f(0) + \sum_{j=1}^{n}\left[d_{\ell,j,2} f(j) sk_j + e_{\ell,j,2} sk_j + f_{\ell,j,2}(\ldots)\right]\right) = A, \qquad (\spadesuit)$$

where A is the negative/minus of the appropriate rest term. Let us write this equation as $\gamma_\ell \cdot B = A$ for the appropriate B. We consider the following five events:

- E_1 defined by the identity $B = 0$.
- E_2 defined by: there is no index $i \in \mathcal{H}$ such that $f_i \neq 0$ that are known polynomials in $f_{\ell,1,2}, \ldots, f_{\ell,n,2}$.
- E_3 defined by: there is no index $i \in \mathcal{H}$ such that $e_i \neq 0$ that are known polynomials in the coefficients output by A.

- E_4 defined by: there is no index $i \in \mathcal{H}$ such that $d_{\ell,i,2} \neq 0$ and $f_{\ell,i,2} + s_i \neq 0$ for known coefficients s_i.

- E_5 defined by: there is no index $i \in \mathcal{H}$ s.t. $r'_i \neq 0$ and $t'_i \neq 0$ that are known polynomials in the coefficients output by A.

With this, we obtain the following technical lemma.

Lemma 5.5.2. *Let* $\mathbf{G}_7$ *and events* E_i *for* $i \in [5]$ *be defined as above. Then there exist (algebraic) algorithms* A_j *for* $j \in [6]$ *playing in game* n**-co-OMDL** *that run in time at most* T *such that:*

$$\begin{aligned}
\Pr[n\text{-}\mathbf{co\text{-}OMDL}^{\mathsf{A}_1} = 1] &= \Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge \neg E_1],\\
\Pr[n\text{-}\mathbf{co\text{-}OMDL}^{\mathsf{A}_2} = 1] &= \Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge E_1 \wedge \neg E_2],\\
\Pr[n\text{-}\mathbf{co\text{-}OMDL}^{\mathsf{A}_3} = 1] &= \Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge E_1 \wedge E_2 \wedge \neg E_3],\\
\Pr[n\text{-}\mathbf{co\text{-}OMDL}^{\mathsf{A}_4} = 1] &= \frac{1}{2}\Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge \ldots \wedge E_3 \wedge \neg E_4],\\
\Pr[n\text{-}\mathbf{co\text{-}OMDL}^{\mathsf{A}_5} = 1] &= \frac{1}{2}\Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge \ldots \wedge E_4 \wedge \neg E_5],\\
\Pr[n\text{-}\mathbf{co\text{-}OMDL}^{\mathsf{A}_6} = 1] &= \Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge E_1 \wedge \ldots \wedge E_5].
\end{aligned}$$

Moreover, $T \leq T' + O(Ln^2)$.

Proof. Let $\xi = (\xi_1, \ldots, \xi_n) \in \mathbb{G}^n$ with $\xi_i = (g^{z_i}, h^{z_i})$ for $i \in [n]$ be the co-OMDL instance of degree n. Algorithms A_i for $i \in [6]$ have access to a (perfect) discrete logarithm oracle $\mathsf{DL}_g()$ in $\mathbb{G}_1$ (to base g) which they can query at most $n-1$ times. When we say algorithm A_i queries the discrete logarithm oracle on ξ_j, we mean that it queries $\mathsf{DL}_g()$ on the first component of ξ_j which is a group element in $\mathbb{G}_1$. Before we start with the description of the algorithms A_i, we describe four different algorithms Sim_i for $i \in [4]$ that give a perfect simulation of the game $\mathbf{G}_7$ to the adversary A.

Simulator $\mathsf{Sim}_1(\xi, par)$: On input ξ, Sim_1 queries the discrete logarithm oracle $\mathsf{DL}_g()$ on $\xi_2, \ldots, \xi_n$ and gets $(z_2, \ldots, z_n)$. It generates the public-secret key pairs of honest parties by sampling $sk_i \leftarrow_\$ \mathbb{Z}_p$ uniformly at random and publishes $pk_i := h^{sk_i}$. Sim_1 executes the DKG protocol GRand on behalf of the honest parties by sampling degree-t polynomials $f_i \in \mathbb{Z}_p[X]$ uniformly at random for all $i \in \mathcal{H}$. At any point of the simulation, Sim_1 answers corruption queries by returning the internal state of the respective party faithfully. After this setup phase, it executes the commitment phase by sampling $\alpha_i \leftarrow_\$ \mathbb{Z}_p^*$ uniformly at random for all $i \in \mathcal{H}$ and publishes $\mathsf{cm}_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Further, for all $i \neq \ell$ it answers random oracle queries r_i to H by sampling $\gamma_i \leftarrow_\$ \mathbb{Z}_p$ uniformly at random and returning $H_1[r_i] := g^{\gamma_i}$. For $r_i = \ell$, however, it returns $H_1[\ell] := \xi_1$. It answers random oracle queries to H_2 by lazy sampling as usual. After the setup phase, the sequential randomness beacon phase begins. For all epochs $r < \ell$, Sim_1 computes the beacon share $\sigma_i = (g_r^{\alpha_i}, e(g_r, SK_i))$ along with the proof of discrete logarithm equality $\pi_i = \mathsf{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$ for honest party P_i by simple computation from the knowledge of α_i and SK_i. It is clear that this simulation is perfect, since the simulator follows all steps of the protocol instructions faithfully and only changes the way how the random group element $g_\ell = \mathsf{H}_1(\ell)$ is generated (on which the forgery is produced).

Simulator $\mathsf{Sim}_2(\xi, par)$: On input ξ, Sim_2 generates the public-secret key pairs of honest parties by sampling $sk_i \leftarrow_\$ \mathbb{Z}_p$ uniformly at random and publishes $pk_i := h^{sk_i}$. Sim_2 executes

the DKG protocol GRand on behalf of the honest parties by sampling degree-t polynomials $f_i \in \mathbb{Z}_p[X]$ uniformly at random for all $i \in \mathcal{H}$. After this setup phase, it executes the commitment phase as follows. For all $i \in \mathcal{H}$, it generates the commitment cm_i of party P_i as $\mathsf{cm}_i := (\xi_{i,1}, \xi_{i,2}^{-1} SK_i)$ and publishes it. Further, for all i it answers random oracle queries r_i to H by sampling $\gamma_i \leftarrow_\$ \mathbb{Z}_p$ uniformly at random and returning $H_1[r_i] := g^{\gamma_i}$. It does so also for the random oracle query $H_1[\ell]$. It answers random oracle queries to H_2 by lazy sampling as usual. After the setup phase, the sequential randomness beacon phase begins. For all epochs $r < \ell$, Sim_2 computes the beacon share $\sigma_i = (g_r^{\alpha_i}, e(g_r, SK_i))$ along with the proof of discrete logarithm equality $\pi_i = \mathsf{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$ for honest party P_i by the identity $g_r^{\alpha_i} = \xi_{i,1}^{\gamma_i}$, by computation $e(g_r, SK_i)$ from the knowledge of SK_i, and π_i by honest-verifier zero-knowledge (HVZK) simulation. At any point of the simulation, Sim_2 answers corruption queries $i \in \mathcal{H}$ by calling its discrete logarithm oracle $\mathsf{DL}_g()$ on input ξ_i to obtain $\alpha_i := \mathsf{DL}_g(\mathsf{cm}_i)$ and returning α_i along with other internal data such as sk_i (note that α_i is the only value for party P_i that was not generated honestly). Note that this is different from the previous simulator in that it was able to return the full internal state without calling its discrete logarithm oracle. It is clear that this simulation is perfect, since the simulator only changes the way how the commitments cm_i for $i \in \mathcal{H}$ are generated (indistinguishable from an honest generation) and can output beacon shares via random oracle programming and HVZK simulation.

Simulator $\mathsf{Sim}_3(\xi, par)$: On input ξ, Sim_3 generates the public-secret key pairs of honest parties as $pk_i := \xi_{i,2}$ and publishes them. This implicitly fixes the secret keys as $sk_i = z_i$ which is the discrete logarithm value of ξ_i. Sim_3 executes the DKG protocol GRand on behalf of the honest parties by sampling degree-t polynomials $f_i \in \mathbb{Z}_p[X]$ uniformly at random for all $i \in \mathcal{H}$. After this setup phase, it executes the commitment phase by sampling $\alpha_i \leftarrow_\$ \mathbb{Z}_p^*$ uniformly at random for all $i \in \mathcal{H}$ and publishes $\mathsf{cm}_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Further, for all i it answers random oracle queries r_i to H by sampling $\gamma_i \leftarrow_\$ \mathbb{Z}_p$ uniformly at random and returning $H_1[r_i] := g^{\gamma_i}$. It does so also for the random oracle query $H_1[\ell]$. It answers random oracle queries to H_2 by lazy sampling as usual. After the setup phase, the sequential randomness beacon phase begins. For all epochs $r < \ell$, Sim_3 computes the beacon share $\sigma_i = (g_r^{\alpha_i}, e(g_r, SK_i))$ along with the proof of discrete logarithm equality $\pi_i = \mathsf{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$ for honest party P_i as follows: generation of $g_r^{\alpha_i}$ and π_i are by simple computation from the knowledge of α_i, and generation of $e(g_r, SK_i)$ is done via the identity

$$e(g_r, SK_i) = e(g, SK_i)^{\gamma_i} = e(g^{f(i)}, h)^{\gamma_i} = e(PK_i, h)^{\gamma_i}.$$

At any point of the simulation, Sim_3 answers corruption queries $i \in \mathcal{H}$ by calling its discrete logarithm oracle $\mathsf{DL}_g()$ on input ξ_i to obtain sk_i and returning sk_i along with other internal data such as α_i (note that sk_i is the only value for party P_i that was not generated honestly). It is clear that this simulation is perfect, since the simulator follows all steps of the protocol instructions faithfully and only changes the bulletin board keys are generated.

Simulator $\mathsf{Sim}_4(\xi, par)$: On input ξ, Sim_4 queries the discrete logarithm oracle $\mathsf{DL}_g()$ on $\xi_{t+2}, \ldots, \xi_n$ and gets $(z_{t+2}, \ldots, z_n)$. It generates the public-secret key pairs of honest parties by sampling $sk_i \leftarrow_\$ \mathbb{Z}_p$ uniformly at random and publishes $pk_i := h^{sk_i}$. Sim_4 executes the DKG protocol GRand on behalf of the honest parties by sampling degree-t polynomials $f_i \in \mathbb{Z}_p[X]$ uniformly at random for all $i \in \mathcal{H} \setminus \{P^*\}$. For the designated party P^* (that contributes to the final aggregated PVSS transcript and remains honest), however, it generates the degree-t polynomial $f_{i^*} = d_0 + d_1 X + \ldots + d_t X^t \in \mathbb{Z}_p[X]$ such that $g^{d_j} = \xi_{j+1}$ for all $j \in [\![t]\!]$ (i.e., the

$t+1$ coefficients of the polynomial are given by the discrete logarithm values of $\xi_1, \ldots, \xi_{t+1}$). From this, it can generate the commitments and encrypted shares of party P^*'s PVSS transcript by Lagrange interpolation in the exponent and knowledge of the secret keys sk_j of all parties. In this context, it is important to note the following crucial and subtle observation: since Sim_4 generates the public keys pk_i of honest parties faithfully and the adversary A is algebraic, it outputs the (updated) public keys of corrupt parties as a linear combination of known values which enables Sim_4 to compute the respective secret keys from this linear combination along with the algebraic representation.

At any point of the simulation, Sim_4 answers corruption queries by returning the internal state of the respective party faithfully. Note that by assumption P^* remains honest and for any other party the simulator follows the protocol instructions honestly so that it can return the internal state of the respective party without any discrete logarithm query. After this setup phase, it executes the commitment phase by sampling $\alpha_i \leftarrow_\$ \mathbb{Z}_p^*$ uniformly at random for all $i \in \mathcal{H}$ and publishes $\mathsf{cm}_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Further, for all i it answers random oracle queries r_i to H by sampling $\gamma_i \leftarrow_\$ \mathbb{Z}_p$ uniformly at random and returning $H_1[r_i] := g^{\gamma_i}$. It does so also for the random oracle query $H_1[\ell]$. It answers random oracle queries to H_2 by lazy sampling as usual. After the setup phase, the sequential randomness beacon phase begins. For all epochs $r < \ell$, Sim_4 computes the beacon share $\sigma_i = (g_r^{\alpha_i}, e(g_r, SK_i))$ along with the proof of discrete logarithm equality $\pi_i = \mathsf{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$ for honest party P_i by simple computation from the knowledge of α_i and SK_i. It is clear that this simulation is perfect, since the simulator follows all steps of the protocol instructions faithfully for all parties except P^* and only changes the way the designated party generates its PVSS transcript which is indistinguishable from an honestly generated transcript.

Having defined the simulators Sim_i for $i \in [4]$, we can now describe the algorithms A_i and how they convert the forgery output by A (winning game $\mathbf{G}_7$) into a valid solution to the co-OMDL instance ξ with high probability, conditioned on some event happening. Recall equation (♠) defined as $B\gamma_\ell = A$ for the appropriate terms A and B. We now describe the algorithms A_i that simulate the game $\mathbf{G}_7$ to the adversary.

Algorithm $\mathsf{A}_1(\xi, par)$: Algorithm A_1 works identical as the simulator Sim_1. In particular, the simulation is perfect and the only change is the way how the random group element $g_\ell = \mathsf{H}_1(\ell)$ is generated on which the forgery is produced. Suppose that A_1 wins the game $\mathbf{G}_7$ and that event $\neg E_1$ happens, i.e., $B \neq 0$. Then equation (♠) is a non-trivial equation of degree one in $\gamma_\ell = \mathsf{DL}_g(\xi_1)$ (as clarified before, the terms A and B are completely independent from the variable γ_ℓ). By simple algebra, this allows A_1 to solve for $\gamma_\ell = A \cdot B^{-1}$ and thus efficiently output the discrete logarithm values of the co-OMDL challenge ξ. Overall, we obtain

$$\Pr[n\textbf{-co-OMDL}^{\mathsf{A}_1} = 1] = \Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge \neg E_1].$$

The bound on the running time of A_1 is obvious.

Algorithm $\mathsf{A}_2(\xi, par)$: Algorithm A_2 works identical as the simulator Sim_2. In particular, the simulation is perfect and the only change is the way how the commitments cm_i are generated. In this case, we have $\mathsf{cm}_i = (\xi_{i,1}, \xi_{i,2}^{-1} SK_i)$ for all $i \in \mathcal{H}$ (here, $\mathcal{H} \subset [n]$ is the set of all honest parties up to the point in which parties output these commitments) and thus $\alpha_i = \mathsf{DL}_g(\xi_i)$. Suppose that A_2 wins the game $\mathbf{G}_7$ and that event $E_1 \wedge \neg E_2$ happens, i.e., $B = 0$ and also there is an index $i \in \mathcal{H}$ such that $f_i \neq 0$ (we will define these very soon). We let $F := f_{\ell,1,2}\alpha_1 + \ldots + f_{\ell,n,2}\alpha_n$

and from $B = 0$ get the identity

$$F = b_{\ell,3} - f(0) + \sum_{j=1}^{n} \left[d_{\ell,j,2} f(j) sk_j + e_{\ell,j,2} sk_j + f_{\ell,j,2} f(j) \right], \qquad (\heartsuit)$$

and so all terms on the right-hand side of the equation are independent from the variables $\alpha_1, \ldots, \alpha_n$ and can also be concretely computed by A_2 as they are known. We consider the defining equation of F in more detail now.

Since the adversary A is algebraic, it outputs its commitments cm_i as (known) linear combinations of the commitments of the honest parties, since all previously generated elements output by the simulator Sim_2 are generated honestly and thus independent from the unknown ξ. For simplicity, we assume for the remainder of this paragraph that $C = [t]$ and $\mathcal{H} = [t+1, n]$. As a result, there are linear polynomials $F_i \in \mathbb{Z}_p[X]$ for all $i \in C$ such that $\alpha_i = F_i(\alpha_{t+1}, \ldots, \alpha_n) = F_i(z_{t+1}, \ldots, z_n)$ that only depend on the outputs by the honest parties. Therefore, the defining equation for F can be transformed into

$$F = f_0 + f_{t+1}\alpha_{t+1} + f_{t+2}\alpha_{t+2} + \ldots + f_n\alpha_n$$

for some appropriately defined coefficients $f_0, f_{t+1}, \ldots, f_n \in \mathbb{Z}_p$. By assumption we suppose that A_2 wins the game $\mathbf{G}_7$ and that event $\neg E_2$ happens, that means there is an index $i \in \mathcal{H}$ such that $f_i \neq 0$. Since the value of F by equation ($\heartsuit$) can be concretely computed by A_2, in combination with the above equation for F linear in the variables $\alpha_{t+1}, \ldots, \alpha_n$ the algorithm A_2 proceeds as follows. It computes all the values $\alpha_j = \mathsf{DL}_g(\xi_j)$ by calling its discrete logarithm oracle on input ξ_j for $j \in \mathcal{H} \setminus \{i\}$ and then solves for the value α_i in the above linear equation for F. By simple algebra, this allows A_2 to solve for $\alpha_1, \ldots, \alpha_n$ and thus efficiently output the discrete logarithm values of the co-OMDL challenge ξ. Overall, we obtain

$$\Pr[n\textbf{-co-OMDL}^{\mathsf{A}_2} = 1] = \Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge E_1 \wedge \neg E_2].$$

The bound on the running time of A_2 is obvious.

Algorithm $\mathsf{A}_3(\xi, par)$: Algorithm A_3 works identical as the simulator Sim_3. In particular, the simulation is perfect and the only change is the way the bulletin board keys are generated. In this case, we have $pk_i = \xi_{i,2}$ for all $i \in \mathcal{H}$ and thus implicitly $sk_i = z_i$ which is the discrete logarithm value of ξ_i. Suppose that A_3 wins the game $\mathbf{G}_7$ and the event $E_1 \wedge E_2 \wedge \neg E_3$ happens. Recall the E_1 defining equation

$$\sum_{j=1}^{n} \left[d_{\ell,j,2} f(j) sk_j + e_{\ell,j,2} sk_j - f_{\ell,j,2}\alpha_j + f_{\ell,j,2} f(j) \right] = f(0) - b_{\ell,3}. \qquad (\blacklozenge)$$

Again, let $F := f_{\ell,1,2}\alpha_1 + \ldots + f_{\ell,n,2}\alpha_n$ as before. In contrast to the previous paragraph, now the values α_i for all $i \in C$ (here, $C \subset [n]$ is the set of all corrupt parties up to the point in which parties output their commitments cm_i) are known and independent of $z_1, \ldots, z_n$ that are the discrete logarithm values of ξ_i. The reason for this is that since the adversary A is algebraic, it outputs the commitments $\mathsf{cm}_{i,1} = g^{\alpha_i} \in \mathbb{G}_1$ with an algebraic representation of elements in the prime field $\mathbb{Z}_p$. However, as the simulator Sim_3 (that defines the algorithm A_3) only uses the elements $\xi_{i,2} \in \mathbb{G}_2$ in the second source group for its simulation of the protocol, A is agnostic of the elements $\xi_{i,1} \in \mathbb{G}_1$ and therefore has to explicitly provide knowledge of the α_i for $i \in C$

through the algebraic representation. This argument relies on the fact that the underlying pairing is of type 3 and there is no efficient way for A to transform elements from one source group to the other source group. The same argument also applies to the evaluations of the hidden degree-t polynomial $f(X) \in \mathbb{Z}_p[X]$, as the adversary outputs the commitments of its chosen polynomials in the first source group $\mathbb{G}_1$ and therefore independent of the $\xi_{i,2}$ elements. As a result, the values $f(0), f(1), \ldots, f(n)$ are known to A_3 and independent from the variables $z_1, \ldots, z_n$. Overall, the only variables in the above equation (♦) that depend on $z_1, \ldots, z_n$ are the secret keys $sk_1, \ldots, sk_n$. We simplify as

$$\sum_{j=1}^{n} \hat{e}_j sk_j = f(0) - b_{\ell,3} + \sum_{j=1}^{n} \left[f_{\ell,j,2}\alpha_j - f_{\ell,j,2} f(j) \right] \qquad (\blacklozenge)$$

where $\hat{e}_j = d_{\ell,j,2} f(j) + e_{\ell,j,2}$ for all $j \in [n]$. Since A is algebraic, it outputs its (updated) keys pk_i for $i \in C$ (up to the point in which parties publish their keys on the bulletin board) as linear combinations of the honest parties' keys. For simplicity, we assume that $C = [t]$ and $\mathcal{H} = [t+1, n]$. Thus, let us write

$$sk_i = \lambda_{0,i} + \lambda_{t+1,i} z_{t+1} + \ldots + \lambda_{n,i} z_n$$

for all $i \in C$ and some coefficients $\lambda_{0,i}, \lambda_{t+1,i}, \ldots, \lambda_{n,i} \in \mathbb{Z}_p$, since $sk_i = z_i$ for $i \in \mathcal{H}$. If we plug in these identities into (♦), we obtain

$$\begin{aligned} &\sum_{j \in C} \hat{e}_j \lambda_{0,j} + \sum_{j \in \mathcal{H}} z_j \left(\hat{e}_j + \sum_{i \in C} \lambda_{j,i} \right) = [\ldots] \\ \iff &\sum_{j \in C} \hat{e}_j \lambda_{0,j} + \sum_{j \in \mathcal{H}} z_j \left(\hat{e}_j + \lambda_{j,1} + \ldots \lambda_{j,t} \right) = [\ldots] \end{aligned}$$

where $[\ldots]$ is simply identical to the right-hand side of (♦). From this equation we see that it defines a polynomial of degree one in the variables $z_{t+1}, \ldots, z_n$. We define the corresponding coefficients of z_j for $j \in \mathcal{H}$ as e_j, that is $e_j := \hat{e}_j + \sum_{i \in C} \lambda_{j,i}$. By assumption we suppose that A_3 wins the game $\mathbf{G}_7$ and that event $\neg E_3$ happens, that means there is an index $i \in \mathcal{H}$ such that $e_i \neq 0$. Having said that, algorithm A_3 proceeds as follows. It computes all the values $z_j = \mathsf{DL}_g(\xi_j)$ by calling its discrete logarithm oracle on input ξ_j for $j \in \mathcal{H} \setminus \{i\}$ and then solves for the remaining variable z_i in the above linear equation. By simple algebra, this allows A_3 to solve for $sk_1, \ldots, sk_n$ and thus efficiently output the discrete logarithm values of the co-OMDL challenge ξ that it received. Overall, we obtain

$$\Pr[n\textbf{-co-OMDL}^{\mathsf{A}_3} = 1] = \Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge E_1 \wedge E_2 \wedge \neg E_3].$$

The bound on the running time of A_3 is obvious.

Algorithm $\mathsf{A}_4(\xi, par)$: Algorithm A_4 works identical as the simulator Sim_4. In particular, the simulation is perfect and the only change is the way the final aggregated PVSS transcript is formed. In this case, the simulator generates the degree-t polynomial $f_{i^*} = d_0 + d_1 X + \ldots + d_t X^t \in \mathbb{Z}_p[X]$ such that $g^{d_j} = \xi_{j+1}$ for all $j \in [\![t]\!]$ (i.e., the $t+1$ coefficients of the polynomial are given by the discrete logarithm values of $\xi_1, \ldots, \xi_{t+1}$). Everything else is generated honestly (in particular, the bulletin board keys and the PVSS transcripts of other parties). Without loss of generality, we

assume that the adversary chooses all PVSS transcripts that contribute to the final aggregated transcript $AT \leftarrow \mathsf{GenAPVSS}$ output from the DKG execution except the one from party P^*. Further, we assume for simplicity that the final transcript only consists of two single PVSS transcript (i.e., the contribution vector $b \in \{0,1\}^n$ is of weight two $|b| = 2$). The general case in which there is more than one PVSS transcript chosen by A works analogously (intuitively, it does not make a difference if A outputs its PVSS transcripts separately or aggregated).

Now there are two cases that can happen: (i) A chooses its PVSS transcript dependent from the transcript T^* of the honest party P^*, and (ii) A generates its PVSS transcript honestly and independent from T^*. However, since parties augment the PVSS transcript with a (non-interactive) proof od knowledge θ, the idea is that the reduction can extract the evaluation $f_i(0)$ of the polynomial $f_i \in \mathbb{Z}_p[X]$ chosen by A (since the proof of knowledge comes with an algebraic representation of the respective hash query for the challenge in the Fiat-Shamir heuristic) and thereby solve for $f_{i^*}(0)$ in case A chose f_i dependent on f_{i^*}. This allows the reduction to obtain the discrete logarithm value $z_1 = f_{i^*}(0)$ and thus solve the co-OMDL challenge. This intuition is made formal and explicit in the security reduction of [BL23c]. We omit it here and directly assume that the adversary chooses its polynomial f_i honestly and independent of the honest party's P^* polynomial f_{i^*}. On the other hand, the adversary has also the possibility to choose its commitments cm_i dependent of the elements output by the transcript T^* (or equivalently the aggregated transcript AT), since the transcript includes terms in both source groups $\mathbb{G}_1$ and $\mathbb{G}_2$ to base g and h, respectively. The bulletin board keys, however, remain independent from the transcript or challenge ξ, as already clarified in the description of Sim_4. We will take these facts into consideration in our following analysis. Suppose that A_4 wins the game $\mathbf{G}_7$ and that the event $E_1 \wedge E_2 \wedge E_3 \wedge \neg E_4$ happens. From event E_1 we recall the equation

$$\sum_{j=1}^{n} \left[d_{\ell,j,2} f(j) sk_j + e_{\ell,j,2} sk_j - f_{\ell,j,2} \alpha_j + f_{\ell,j,2} f(j) \right] = f(0) - b_{\ell,3}.$$

From event E_3 we know that $e_j = 0$ for all $j \in \mathcal{H}$ where $e_j = \hat{e}_j + \sum_{i \in C} \lambda_{j,i} = \hat{e}_j$ (observe that now $\lambda_{j,i} = 0$ for all $i \in C$ and $j \in \mathcal{H}$ as the secret keys are independent of the challenge ξ). It follows that $0 = \hat{e}_j = d_{\ell,j,2} f(j) + e_{\ell,j,2}$. The easy case now is if there is an $j \in \mathcal{H}$ such that $d_{\ell,j,2} \neq 0$, as this allows us to rewrite $f(j) = -e_{\ell,j,2}/d_{\ell,j,2}$ and obtain a non-trivial equation

$$-\frac{e_{\ell,j,2}}{d_{\ell,j,2}} = z_1 + z_2 j + \ldots + z_{t+1} j^t \qquad (\heartsuit)$$

in the variables $z_1, \ldots, z_{t+1}$ (recall that the reduction A_4 chooses the polynomial f_{i*} such that its coefficients are the discrete logarithm values of $\xi_1, \ldots, \xi_{t+1}$). Further, by calling its discrete logarithm oracle $\mathsf{DL}_g()$ on inputs $\xi_2, \ldots, \xi_{t+1}$, it can solve for z_1 in the above equation ($\heartsuit$) and thus efficiently output the discrete logarithm values of the co-OMDL challenge ξ. This case in the definition of event $\neg E_4$ is settled and thus for the remainder of this paragraph we assume $d_{\ell,j,2} = 0$ for all $j \in \mathcal{H}$. Having said that, the only unknown terms dependent on $z_1, \ldots, z_{t+1}$ in $d_{\ell,j,2} f(j) sk_j + e_{\ell,j,2} sk_j$ are the $f(j)$ for $j \in C$. Since $|C| \leq t$, the algorithm A_4 proceeds as follows. It calls its discrete logarithm oracle $\mathsf{DL}_g()$ on inputs $g^{f(j)}$ for all $j \in C$, thus obtaining t new linearly independent equations $f(j) = z_1 + z_2 j + \ldots + z_{t+1} j^t$ (the equations written in matrix form give a Vandermonde matrix which is well-known to have full rank) with known

values $f(j)$. In particular, we can rewrite the above equation from event E_1 as

$$\sum_{j=1}^{n} \left[d_{\ell,j,2} f(j) sk_j + e_{\ell,j,2} sk_j - f_{\ell,j,2} \alpha_j + f_{\ell,j,2} f(j) \right] = f(0) - b_{\ell,3},$$

which is equivalent to

$$\sum_{j=1}^{n} \left[-f_{\ell,j,2} \alpha_j + f_{\ell,j,2} f(j) \right] = f(0) + D \quad \Longleftrightarrow \quad \sum_{j=1}^{n} \left[f(j) - \alpha_j \right] f_{\ell,j,2} = f(0) + D \quad (2)$$

where $D \in \mathbb{Z}_p$ is the appropriate (known) rest term. Note that the terms $d_{\ell,j,2} f(j) sk_j$ vanish for $j \in \mathcal{H}$, the values $e_{\ell,j,2} sk_j$ and $b_{\ell,3}$ are known, and finally the terms $d_{\ell,j,2} f(j) sk_j$ are also now known for $j \in C$. On the other hand, when the adversary A outputs its commitments $\mathsf{cm}_{i,2}$ in the second group $\mathbb{G}_2$ it outputs them as a (known) linear combination of the elements h, $pk_1, \ldots, pk_n$, $\mathsf{cm}_{t+1,2}, \ldots, \mathsf{cm}_{n,2}$, and $Y_1, \ldots, Y_n$. As a result, this gives an identity for all $i \in C$ as follows

$$f(i) - \alpha_i = r_{0,i} + \sum_{j=1}^{n} r_{j,i} sk_j + \sum_{j \in \mathcal{H}} s_{j,i} \left(f(j) - \alpha_j \right) + \sum_{j=1}^{n} t_{j,i} f(j) sk_j, \quad (4)$$

where the $r_{j,i}, s_{j,i}, t_{j,i} \in \mathbb{Z}_p$ are known coefficients. We take the sum of $f_{\ell,i,2}(f(i) - \alpha_i)$ over all $i \in [n]$ and also use the equations given by (4) and the one given by (2) above. This results in

$$\begin{aligned} f(0) + D &= \sum_{i=1}^{n} f_{\ell,i,2}(f(i) - \alpha_i) \\ &= \sum_{i \in \mathcal{H}} f_{\ell,i,2}(f(i) - \alpha_i) + r_0 + \sum_{j=1}^{n} r_j sk_j + \sum_{j \in \mathcal{H}} s_j (f(j) - \alpha_j) + \sum_{j=1}^{n} t_j f(j) sk_j \\ &= r_0 + \sum_{j=1}^{n} r_j sk_j + \sum_{j=1}^{n} t_j f(j) sk_j + \sum_{j \in \mathcal{H}} (f_{\ell,j,2} + s_j)(f(j) - \alpha_j), \quad (\clubsuit) \end{aligned}$$

where we define the symbols

$$r_0 := \sum_{i \in C} f_{\ell,i,2} r_{0,i}, \quad r_j := \sum_{i \in C} f_{\ell,i,2} r_{j,i}, \quad s_j := \sum_{i \in C} f_{\ell,i,2} s_{j,i}, \quad t_j := \sum_{i \in C} f_{\ell,i,2} t_{j,i}.$$

Now the other case of $\neg E_4$ applies, which means that there is an index $i \in \mathcal{H}$ such that $f_{\ell,i,2} + s_i \neq 0$. In that case, we could let A_4 instead work identical as simulator Sim_2. In particular, the simulation is perfect and the only change is the way how the commitments cm_i are generated. In this case, we have $\mathsf{cm}_i = (\xi_{i,1}, \xi_{i,2}^{-1} SK_i)$ for all $i \in \mathcal{H}$ and thus $\alpha_i = \mathsf{DL}_g(\xi_i)$. Analogously, we can derive the same (general) equation ($\clubsuit$), where D only depends on $f(j)$ and sk_j for $j \in [n]$ and is independent of α_j for $j \in \mathcal{H}$. As a result, this equation gives a linear polynomial in the variables $\{\alpha_j\}_{j \in \mathcal{H}}$ and there is a non-zero coefficient $f_{\ell,i,2} + s_i \neq 0$ of α_i for that particular i given by event $\neg E_4$ happening. As for algorithm A_2, the algorithm A_4 proceeds as follows. It computes all the values $\alpha_j = \mathsf{DL}_g(\xi_j)$ by calling its discrete logarithm oracle on input ξ_j for $j \in \mathcal{H} \setminus \{i\}$ and then solves for the value α_i given the above linear equation

(♣). By simple algebra, this allows A_4 to solve for $\alpha_1, \ldots, \alpha_n$ and thus efficiently output the discrete logarithm values of the co-OMDL challenge ξ. Finally, we let algorithm A_4 choose the simulation strategies Sim_2 and Sim_4 each with probability $1/2$ at the beginning of its execution. Since, this choice remains hidden from the adversary A, we obtain

$$\Pr[n\text{-co-OMDL}^{\mathsf{A}_4} = 1] = \frac{1}{2}\Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge \ldots \wedge E_3 \wedge \neg E_4].$$

The bound on the running time of A_4 is obvious.

Algorithm $\mathsf{A}_5(\xi, par)$: Before we proceed with the description of algorithm A_5, we give a brief summary of the identities we have so far. From event E_1 in its very plain form, we have the equation

$$\sum_{j=1}^{n} \left[d_{\ell,j,2} f(j) sk_j + e_{\ell,j,2} sk_j - f_{\ell,j,2} \alpha_j + f_{\ell,j,2} f(j) \right] = f(0) - b_{\ell,3}.$$

Together with event E_4, we have as before

$$f(0) + D = r_0 + \sum_{j=1}^{n} r_j sk_j + \sum_{j=1}^{n} t_j f(j) sk_j$$

where by event E_3 now

$$D = -\left(b_{\ell,3} + \sum_{j \in C} \left[d_{\ell,j,2} f(j) sk_j + e_{\ell,j,2} sk_j \right] \right).$$

Taking these together, we get

$$f(0) - (r_0 + b_{\ell,3}) = \sum_{j=1}^{n} r'_j sk_j + \sum_{j=1}^{n} t'_j f(j) sk_j \qquad (\diamond)$$

for appropriate (known) coefficients r'_j and t'_j. This equation has the same form as the one that algorithm A_3 started with. From this observation, we let algorithm A_5 choose the simulation strategies Sim_3 and Sim_4 each with probability $1/2$ at the beginning of its execution. This choice remains completely hidden from the adversary A's view. Therefore, the same calculations as for A_3 and A_4 with our new coefficients and just adjusted event $\neg E_5$ (which is basically just an adaption of events $\neg E_3$ and the first case of event $\neg E_4$), A_5 can derive a solution for the co-OMDL challenge ξ with probability $1/2$. We omit the whole calculation here again and simply proceed with the next event. Overall, we obtain

$$\Pr[n\text{-co-OMDL}^{\mathsf{A}_5} = 1] = \frac{1}{2}\Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge \ldots \wedge E_4 \wedge \neg E_5].$$

The bound on the running time of A_5 is obvious.

Algorithm $\mathsf{A}_6(\xi, par)$: Algorithm A_6 works identical as the simulator Sim_4. In particular, the simulation is perfect and the only change is the way the final aggregated PVSS transcript is formed. In this case, the simulator generates the degree-t polynomial $f_{i^*} = d_0 + d_1 X + \ldots + d_t X^t \in \mathbb{Z}_p[X]$ such that $g^{d_j} = \xi_{j+1}$ for all $j \in [\![t]\!]$ (i.e., the $t+1$ coefficients of the polynomial are given by the

discrete logarithm values of $\xi_1, \ldots, \xi_{t+1}$). Everything else is generated honestly (in particular, the bulletin board keys and the PVSS transcripts of other parties). As clarified before, we make the assumptions as in the fourth algorithm description A_4. That is, we assume that the adversary chooses its polynomial independent of the honest party P^*'s transcript and thus we may also assume directly that the aggregated transcript hides the polynomial $f = f_{i^*} \in \mathbb{Z}_p[X]$ (i.e., we ignore the shift caused by the adversary's polynomials). We suppose that A_6 wins the game $\mathbf{G}_7$ and that the event $E_1 \wedge \ldots \wedge E_5$ happens. Then the above equation ($\diamond$) simplifies into the following

$$\begin{aligned} f(0) - (r_0 + b_{\ell,3}) &= \sum_{j \in C} r'_j sk_j + \sum_{j \in C} t'_j f(j) sk_j \\ \iff f(0) &= r_0 + b_{\ell,3} + \sum_{j \in C} r'_j sk_j + \sum_{j \in C} t'_j f(j) sk_j. \end{aligned}$$

Having computed that, the only unknown terms dependent on $z_1, \ldots, z_{t+1}$ on the right-hand side of this equation are the $f(j)$ for $j \in C$. Since $|C| \leq t$, the algorithm A_6 proceeds as follows. It calls its discrete logarithm oracle $\mathsf{DL}_g()$ on inputs $g^{f(j)}$ for all $j \in C$, thus obtaining t new linearly independent equations $f(j) = z_1 + z_2 j + \ldots + z_{t+1} j^t$ (the equations written in matrix form give a Vandermonde matrix which is known to have full rank) with known values $f(j)$. Finally, the above equation allows A_6 to compute $f(0) = z_1$ and thus obtain in total $t + 1$ points on the polynomial $f \in \mathbb{Z}_p[X]$ of degree t. Hence, it can efficiently solve for the discrete logarithm values of the co-OMDL challenge ξ. Overall, we obtain

$$\Pr[n\textbf{-co-OMDL}^{\mathsf{A}_6} = 1] = \Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge \ldots \wedge E_4 \wedge E_5].$$

The bound on the running time of A_6 is obvious. □

To end the proof, consider the following algorithm B playing in n-**co-OMDL**. B samples $i^* \leftarrow_{\$} [6]$ and then internally emulates A_{i^*}. Clearly, B is an algebraic algorithm running in time at most T (the running time of A_i, $1 \leq i \leq 6$). An application of the law of total probability yields the following

$$\begin{aligned} \Pr[n\textbf{-co-OMDL}^{\mathsf{B}} = 1] &= \frac{1}{6} \sum_{i=1}^{6} \Pr[n\textbf{-co-OMDL}^{\mathsf{A}_i} = 1] \\ &\geq \frac{1}{12} \cdot \Pr[\mathbf{G}_7^{\mathsf{A}} = 1] \\ &\geq \frac{1}{12} \left(\frac{\varepsilon}{Ln} - 4n\varepsilon_B - \frac{2q_h}{p} - \frac{q_h^2}{2p} \right). \end{aligned}$$

By rearrangement, we conclude the proof of Theorem 5.5.1 with the final bound

$$\varepsilon \leq Ln \left(12\varepsilon_A + 4n\varepsilon_B + \frac{q_h^2 + 4q_h}{2p} \right).$$

□

5.6 Implementation and Evaluation

In this section, we evaluate the performance of our distributed randomness beacon protocol GRandLine for various network sizes. Concretely, we evaluate its throughput (i.e., the number of beacon values output per second) and compare it to existing state-of-the-art randomness beacons in the same setting: OptRand [Bha+23], BRandPiper [Bha+21], and Drand [Dra20]. Although we developed our code to be agnostic to the choice of a pairing-friendly curve, we have used BLS12-381 for our instantiation. In particular, we have used the implementation of BLS12-381 by arkworks [con22] for primitive elliptic curve operations.

Table 5.3: Computation cost per epoch for non-leader, leader parties.

Protocol	Balanced	Rounds	Pairings	NIZKs (Ver)	NIZKs (Gen)	Exponents	Public Ver
Drand [Dra20]	✓	1	$2n$	0	0	$t+2$	2, 0
OptRand [Bha+23]	✗	11	$4n+1$, $2n+1$	n, n^2+n	$n+1$	$2n+t+2$	$2n+3$, $3t+3$
BRandPiper [Bha+21]	✗	11	$4n$, $2n$	0	0, n^2	n, n^2	$2n$, $t+1$
GRandLine (ours)	✓	1	$2n+1$	n	1	$t+2$	$2t+2$, $4n+t+1$

Balanced denotes balanced computation cost across all parties. **Rounds** denotes the number of rounds per epoch. **Pairings** denotes the number of pairing operations performed by each non-leader, leader party. **NIZKs (Ver)** denotes the number of NIZK verifications performed by each non-leader, leader party. OptRand and GRandLine employ Chaum-Pedersen proofs of discrete logarithm equality for the NIZKs. **NIZKs (Gen)** denotes the number of NIZKs generated by each non-leader, leader party. BRandPiper employs KZG proofs for the NIZKs whose verification uses pairings. **Exponents** denotes the number of group exponentiations performed by each non-leader, leader party. For Drand and GRandLine, this can be done using one multi-exponentiation and one regular exponentiation. For OptRand, this can be done using one multi-exponentiation and $2n+1$ regular exponentiations. For BRandPiper, this can be done using one resp. n multi-exponentiation(s) by a non-leader resp. leader party. **Public Ver** denotes the number of pairings, group exponentiations performed to publicly verify an epoch beacon output. Here, we directly incorporate the NIZK verifications into the group exponentiation count.

5.6.1 Implementation Details

We have implemented our prototype of GRandLine using the Rust programming language and the arkworks ecosystem [con22]. We use a custom, optimized APVSS scheme implementation for our underlying cryptographic operations [Pap23a] and tokio [con23] for networking. The implementation follows strictly the description in Figure 5.3 and it is publicly available at our GitHub repository [Pap23b].

An important note: We emphasize that we have not implemented our DKG protocol and instead manually set up the keys of parties for our experiments. The same is also true for the other randomness beacon protocols of interest, BRandPiper and OptRand. Their publicly available implementations [Shr22; Luo22] have the output generated from a potential pre-processing phase already configured. Without major modifications on their codes, it is not possible for us to run and evaluate their pre-processing phase. That being said, we likewise decided to not implement our DKG protocol.

Instantiation. We instantiate pairings with the BLS12-381 pairing-friendly family of elliptic curves. For efficiency, we use in our implementation $\mathbb{G}_1$ as the group for encrypted shares of

the underlying APVSS scheme and $\mathbb{G}_2$ as the group for commitment shares. For our group generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, we use fixed generators of unknown exponent. We simulate our protocol's setup phase by precomputing and using config files with the PVSS public keys and BLS12-381 public keys for Schnorr digital signatures.

5.6.2 Experimental Setup

We demonstrate the efficiency of GRandLine by evaluating our implementation with a varying total number of nodes, i.e., 4, 8, 16, 32, and 64. All experiments were conducted over Amazon EC2 where each replica was executed on a t3.medium instance. Each instance has 2 vCPUs, all cores sustained a Turbo CPU clock speed of up to 3.1GHz. The machines have up to 5 Gbps bandwidth, 4GB memory and run Ubuntu 22.04 LTS.

Network. To simulate execution over the Internet and to ensure comparability with other proposals, all of our experiments were conducted over Amazon EC2 where each replica was executed on a t3.medium instance across 8 regions: N. Virginia (us-east-1), Ohio (us-east-2), N. California (us-west-1), Oregon (us-west-2), Stockholm (eu-north-1), Frankfurt (eu-central-1), Tokyo (ap-northeast-1), Sydney (ap-southeast-2). For any choice of total number of nodes, we distribute the nodes evenly across all eight regions. For our runs with 4 nodes we used the following regions: N. Virginia (us-east-1), N. California (us-west-1), Frankfurt (eu-central-1), Tokyo (ap-northeast-1). In all cases, we create an overlay network among nodes where all nodes are pairwise connected in a complete graph.

Baselines. We compare the performance of our implementation to three state-of-art publicly available implementations: BRandPiper [Luo22], Drand [Dra20] and OptRand [Shr22]. Our choice is motivated by the fact that all of these schemes are adaptively secure, have optimal resilience threshold $t < n/2$, and do not use cryptographically heavy tools such as proof-of-work or (trapdoor) VDFs. For a comparison of the computation costs, we refer to Table 5.3.

5.6.3 Evaluation Results

Similar to previous work [Das+22a], we run each experiment three times for about 10 minutes each and took the average over these three runs. Additionally, we note that many other previous works [Bha+21; Bha+23; Sch+20; Ban+24] do not specify the number of runs for their experiments (which could therefore potentially be only a single run). We report the throughput of GRandLine and the comparative randomness beacons as the number of beacon outputs per second in Figure 5.4.

GRandLine. Compared to all three baselines, GRandLine outputs beacons at significantly higher rates. In particular, our evaluation results show that with 4, 8, 16, 32, and 64 nodes, GRandLine generates on average 11, 8, 7, 4, and 2 beacons per second, respectively. The average time between generating two consecutive beacons is 87.19ms, 117.37ms, 133.29ms, 249.65ms, and 489.89ms, respectively. We recall that each experiment was run for about 10 minutes.

OptRand. The protocol proceeds through epochs of up to 11 rounds, where each epoch $e \geq 1$ is delegated by a different leader L_e chosen in a round-robin fashion. In each epoch $e \geq 1$, the leader L_e is instructed to collect and aggregate $t + 1$ valid PVSS transcripts that other parties send to it at the beginning of the epoch. Later, parties collectively reconstruct the secret S_e of

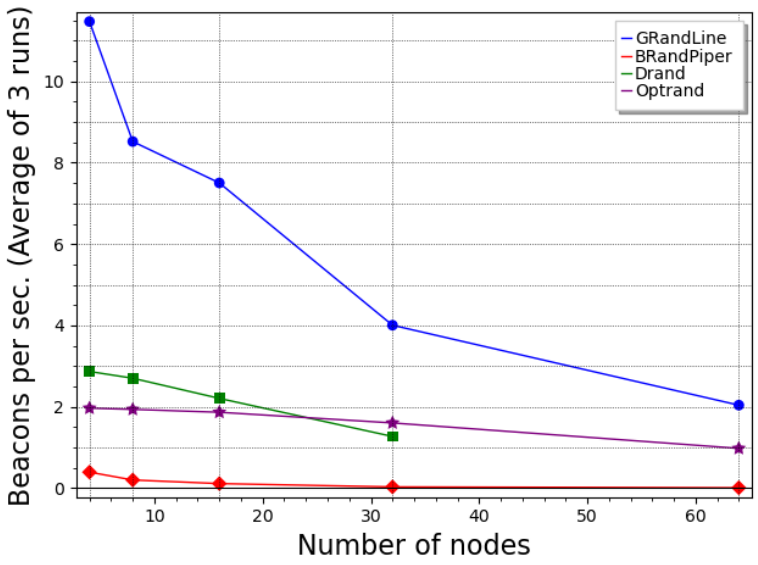

Figure 5.4: Performance graph for the randomness beacons: OptRand, BRandPiper, Drand, GRandLine.

the aggregated transcript and compute the beacon value O_e for epoch e as hash $O_e := \mathsf{H}(S_e)$. Further, the protocol relies on an initial setup where parties start with agreed-upon buffers $\mathcal{B}(P_i)$ for all $i \in [n]$ that contain random PVSS transcripts each. In their public implementation [Shr22] this phase is skipped and already configured. Finally, the protocol has two modes of operation: an optimistically-responsive mode and a non-optimistic mode. The former allows for responsive progress when there are $t < n/4$ actual corrupt parties in the system, which results in much higher throughput compared to the latter.

We test only against OptRand's optimistically-responsive variant, since its non-optimistic variant performs comparably to BRandPiper (see [Bha+23] for a discussion on that). OptRand is leader-based, has many rounds of communication per epoch, and the epoch leader has to carry most of the computation which leads to a bottleneck for higher values of n. This results in significantly lower throughput per second compared to GRandLine where the computation cost is balanced across all nodes (cf. Table 5.3). Further, OptRand's reference implementation is instantiated with the BN128 pairing-friendly curve from libff [Lab21]. Unfortunately, while this curve allows OptRand to benefit from more efficient modular operations, it is below acceptable security standards [Sak+22]. Concretely, it provides only 100-bit security level. By instantiating GRandLine with a similar curve, specifically arkworks' ark-bn254 crate [Ark], we estimate that our protocol is capable of generating roughly twice the number of beacons compared to what is shown in Figure 5.4.

BRandPiper. The protocol proceeds through epochs of 11 rounds, where each epoch $e \geq 1$ is delegated by a different leader L_e chosen in a round-robin fashion. In each epoch $e \geq 1$, the leader L_e shares n randomly chosen secrets $\mathbf{s}_e = (s_{e,1}, \ldots, s_{e,n}) \in \mathbb{Z}_p^n$ among all parties via an efficient multi-secret VSS scheme. Later, parties collectively reconstruct the ephemeral randomness R_e as a sum of n secrets $s_{e,i}, \ldots, s_{e-n+1,i}$ from n consecutive leaders (i.e., one secret from each leader). The randomness beacon value O_e for epoch e is then computed as

hash $O_e := \mathsf{H}(R_e)$. Further, the protocol relies on an initial setup where parties start with agreed-upon buffers $\mathcal{B}(P_i)$ for all $i \in [n]$ that contain elements from a multi-secret VSS sharings. In their public implementation [Luo22] this phase is skipped and already configured. Finally, the protocol is not responsive.

BRandPiper's throughput depends heavily on an estimate for the synchronous network delay parameter Δ. A higher value for Δ leads to increased security but reduces the performance and vice versa. For BRandPiper, we first look for the smallest value for Δ that does not break their implementation and then measure throughput with this value for the delay parameter. Like OptRand's non-optimistic variant, BRandPiper outputs beacons every 11Δ. BRandPiper suffers from increased overheads incurred by both synchronization as well as cryptographic operations due to its round-robin leader election and the use of multi-secret VSS. In particular, our results shown in Figure 5.4 confirm that BRandPiper's performance is severely limited by the presence of a single slow node in the system.

Drand. The protocol uses the non-interactive and unique threshold BLS signature to generate the beacon value. Concretely, in each epoch $e \geq 1$, parties collectively reconstruct the full signature σ_e on the message $m := e$ and compute the beacon value O_e for epoch e as hash $O_e := \mathsf{H}(\sigma_e)$. Further, the protocol relies on an initial key setup realized through Pedersen's DKG protocol [Ped92], which essentially runs n parallel instances of Feldman's VSS with some additional verification steps. It has a communication cost of $O(\lambda n^4)$ bits and outputs secret keys as field elements.

Although the threshold BLS signature allows for asynchronous communication, the actual deployment of Drand relies on a period parameter that determines the time after which a beacon value is output. Concretely, in each period of 30 seconds only a single beacon value is output. Finally, we note that we were only able to evaluate Drand's throughput for up to 32 nodes, as in our experiments, Drand's DKG initialization step keeps failing for 64 or more nodes, even for large estimates of the network delay. The very same issue was already reported in previous works [Das+22a; Bha+21]. For a fair comparison, we measure Drand's throughput only after its DKG setup by computing the time from the start of the epoch until the beacon is reconstructed. We observe that GRandLine outperforms Drand despite their similarity in computational cost (cf. Table 5.3). We suspect that there may be implementation inefficiencies in Drand that hindered its throughput. Another reason could be the choice of programming language. Our protocol is implemented in Rust which is highly optimized for fast execution and has a better run-time performance due to the lack of garbage collection On the other hand, Drand is implemented in Go-lang whose garbage collector can mess up consistency of performance (especially, at these high throughput rates).

5.7 Conclusion

In this work, we presented a novel distributed key generation (DKG) protocol GRand and a novel distributed randomness beacon protocol GRandLine. Our DKG protocol GRand has a communication complexity of $O(\lambda n^2 \log n)$ bits while preserving optimal Byzantine resilience threshold $t < n/2$ in the synchronous network setting. This gives the first DKG protocol in any network setting that achieves subcubic communication complexity (cf. Table 5.2). Our randomness beacon protocol GRandLine has a communication complexity of $O(\lambda n^2)$ bits per

epoch, where each epoch takes only a single asynchronous round of communication and is non-interactive (cf. Table 5.1). In each epoch, GRandLine employs only lightweight cryptography such as hash functions and pairings. Further, both our protocols are secure in the presence of a strongly adaptive adversary. Finally, we have implemented GRandLine in Rust (with manually configured key setup) and found that it vastly outperforms previous randomness beacons in the same setting.

While our DKG protocol has a low communication complexity, it only terminates after a linear number $O(n)$ of rounds. Especially in large-scale systems, a lower number of rounds is highly desirable, preferably independent of the number n of all parties. Therefore, an intriguing question is how to lower the number of rounds from linear $O(n)$ to expected constant $O(1)$ while preserving quadratic communication complexity. On the other hand, our randomness beacon has low communication and round complexity, but has slightly worse computational complexity compared to the randomness beacon derived from threshold BLS. Concretely, each party needs to compute two pairing evaluations along with a Chaum-Pedersen NIZK (for discrete logarithm equality) verification to verify other parties' beacon shares. In contrast, the threshold BLS scheme only requires one pairing evaluation for this step. Finally, our randomness beacon protocol requires $O(n)$ pairing evaluations and NIZK verifications for public verifiability of an epoch beacon value. This is where it significantly falls behind threshold BLS, but is comparable to the other protocols (cf. Table 5.3). However, using standard batch verification techniques [CHP07; Tom+20], this can be reduced to only six multi-exponentiations and two pairing evaluations.

Appendix 5A: Additional Definitions

In this section, we provide the omitted definitions from Section 5.3.

Aggregatable PVSS Scheme. We define the security notions for an APVSS schemes (cf. Definition 5.3.1). We start with correctness and public verifiability.

- *Correctness.* We say that APVSS is *correct* if for all keys $(pk_1, sk_1), \ldots, (pk_n, sk_n) \in \mathsf{Keys}(par)$ and all $i \in [n]$,

$$\Pr[\mathsf{Ver}((pk_j)_{j\in[n]}, T) = 1 \wedge \mathsf{OwnId}(pk_i, T) = 1] = 1,$$

  where the probability is taken over all transcripts T output by $\mathsf{Dist}(sk_i, (pk_i)_{i\in[n]})$.

- *Public Verifiability.* We say that APVSS is *publicly verifiable* if for all key pairs $(pk_1, sk_1), \ldots, (pk_n, sk_n) \in \mathsf{Keys}(par)$ and all $(\mathbf{E}, \pi)$ such that $\mathsf{Ver}((pk_1, \ldots, pk_n), (\mathbf{E}, \pi)) = 1$, there exists a unique $S \in \hat{\mathbb{G}}$ such that

$$\mathsf{Rec}(\{\mathsf{Dec}_{sk_i}(\mathbf{E}_i)\}_{i\in\mathcal{I}}) = S \quad \forall \mathcal{I} \subset [n],\ |\mathcal{I}| = t + 1.$$

We require that the secret from an aggregated transcript $T = \mathsf{Agg}(T_1, \ldots, T_k)$ corresponds to the sum of the secrets S_i that can be reconstructed from the T_i individually. Further, we require that the set of contributors to T consists of the contributors to the single transcripts T_i. Formally, we define this as follows.

Definition 5.7.1 (Correctness of Aggregation)**.** Let $\mathsf{APVSS} = (\mathsf{Keys}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Dist}, \mathsf{Agg}, \mathsf{OwnId}, \mathsf{Ver}, \mathsf{Rec})$ be a publicly verifiable APVSS scheme over $\hat{\mathbb{G}}$. We say that APVSS is *correctly aggregatable* if for all $(pk_1, sk_1), \ldots, (pk_n, sk_n) \in \mathsf{Keys}(par)$, all $k \in \mathbb{N}$, and all transcripts $(\mathbf{E}_1, \pi_1), \ldots, (\mathbf{E}_k, \pi_k)$ the following is true. If for all $i \in [k]$, we have $\mathsf{Ver}((pk_1, \ldots, pk_n), (\mathbf{E}_i, \pi_i)) = 1$, then for all $\mathcal{I} \subset [n]$, $|\mathcal{I}| = t + 1$, the (aggregated) transcript $(\mathbf{E}', \pi') := \mathsf{Agg}((\mathbf{E}_1, \pi_1), \ldots, (\mathbf{E}_k, \pi_k))$ satisfies

$$\mathsf{Rec}(\{\mathsf{Dec}_{sk_i}(\mathbf{E}'_i)\}_{i\in\mathcal{I}}) = \prod_{j\in[k]} \mathsf{Rec}(\{\mathsf{Dec}_{sk_i}(\mathbf{E}_{j,i})\}_{i\in\mathcal{I}}),$$

where we write $\mathbf{E}_j = (\mathbf{E}_{j,1}, \ldots, \mathbf{E}_{j,n})$. Additionally, we require that $\mathsf{OwnId}(pk_i, T) = 1$ for an $i \in [n]$ if and only if there is an $j \in [k]$ such that $\mathsf{OwnId}(pk_i, T_j) = 1$.

We recall the security notion for APVSS schemes called *aggregated unpredictability* as introduced in [BL23c]. Intuitively, it captures malleability attacks and prohibits any *t-bounded* (i.e., corrupting at most t parties) adversary from learning the secret of an aggregated transcript that has contribution from at least one honest party.

Definition 5.7.2 (Aggregated Unpredictability of APVSS)**.** Let $\mathsf{APVSS} = (\mathsf{Keys}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Dist}, \mathsf{Agg}, \mathsf{OwnId}, \mathsf{Ver}, \mathsf{Rec})$ be a publicly verifiable APVSS scheme over $\hat{\mathbb{G}}$. For an algorithm A, we define the *aggregated unpredictability* experiment $\mathbf{AggPred}^{\mathsf{A}}_{\mathsf{APVSS},t}$ as follows:

- *Offline Phase.* Initialize $\mathcal{T} := \emptyset$. For all $i \in [n]$, run Keys on input (par, i) to generate keys $(pk_i, sk_i) \leftarrow \mathsf{Keys}(par, i)$. On input par and $\{pk_i\}_{i\in[n]}$, A returns an index set $C \subset [n]$ of initially corrupted parties along with updated public keys $\{\hat{pk}_i\}_{i\in C}$. Set $pk_i := \hat{pk}_i$ for all $i \in C$.

- *Corruption Queries.* At any point of the experiment, A may corrupt a party by submitting an index $i \in [n] \setminus C$. In this case, return the secret key sk_i and set $C := C \cup \{i\}$. Further, let $\mathcal{H} := [n] \setminus C$ denote the set of honest parties (updated adaptively as C changes).
- *Transcript Queries.* At any point of the experiment, A gets access to an oracle of the following type: When A submits a request $(\mathsf{givePVSS}, i)$ for an $i \in \mathcal{H}$, do the following. On behalf of dealer P_i, run Dist on input sk_i and $pk_1, \ldots, pk_n$. Return the output $T = (\mathbf{E}, \pi)$ and set $\mathcal{T} := \mathcal{T} \cup \{(T, i)\}$.
- *Output Determination.* When A outputs an aggregated transcript $(\mathbf{E}', \pi')$ and an element $S^* \in \hat{\mathbb{G}}$, proceed as follows:
  - Return 1 if the following hold: $|C| \leq t$, $\mathsf{Ver}((pk_1, \ldots, pk_n), (\mathbf{E}', \pi')) = 1$, $S^* = \mathsf{Rec}(\{\mathsf{Dec}_{sk_i}(\mathbf{E}'_i)\}_{i \in [t+1]})$, and there exists an index $i \in \mathcal{H}$ such that $\mathsf{OwnId}((\mathbf{E}', \pi'), pk_i) = 1$.
  - Return 0 otherwise.

We say that APVSS is (t, ε, T, q_s)-*aggregated unpredictable* if for all algorithms A that run in time at most T and make at most q_s transcript queries, $\Pr[\mathbf{AggPred}^{\mathsf{A}}_{\mathsf{APVSS},t} = 1] \leq \varepsilon$. On the other hand, we say that A (t, ε, T, q_s)-*breaks aggregated unpredictability* of APVSS if it runs in time at most T, makes at most q_s transcript queries, and we have $\Pr[\mathbf{AggPred}^{\mathsf{A}}_{\mathsf{APVSS},t} = 1] > \varepsilon$.

Distributed Randomness Beacon. We give a formal definition of a secure distributed randomness beacon. For that, we use the definition in [BL23c] verbatim.

Definition 5.7.3 (d-Secure Randomness Beacon)**.** Let $\mathcal{RB}$ be an epoch-based protocol executed by n parties $P_1, \ldots, P_n$. We define the following security properties for $\mathcal{RB}$:

- *Consistency.* $\mathcal{RB}$ is (t, L)-consistent if the following holds whenever at most t parties are corrupted: if an honest party outputs a value $\sigma_\ell \in \{0,1\}^\lambda$ in epoch $\ell \in [L]$, then all honest parties output σ_ℓ in epoch ℓ.
- *Availability.* $\mathcal{RB}$ is (t, L)-available if the following holds whenever at most t parties are corrupted: for each $\ell \in [L]$, every honest party outputs a value $\sigma_\ell \in \{0,1\}^\lambda$ in epoch ℓ.
- *Bias-Resistance.* $\mathcal{RB}$ is (ε, T, t, L)-bias-resistant if it is (t, L)-available, (t, L)-consistent, and the following holds for all algorithms A, D s.t. A corrupts at most t parties and both A and D run in time at most T. Denote by $\Sigma_{\mathsf{A},L}$ the probability distribution induced by the outputs of an honest party in an execution of $\mathcal{RB}$ until epoch L with A as adversary. Then
$$\Big| \Pr_{\sigma \leftarrow \Sigma_{\mathsf{A},L}}[\mathsf{D}(\sigma) = 1] - \Pr_{u \leftarrow U_L}[\mathsf{D}(u) = 1] \Big| \leq \varepsilon,$$
where U_L denotes the uniform distribution over the L-fold Cartesian product of $\{0,1\}^\lambda$ with itself.
- *d-Unpredictability.* $\mathcal{RB}$ is $(\varepsilon, T, t, L, q_h, d)$-unpredictable if it is (t, L)-available, (t, L)-consistent, and for all $\ell \in [L]$ and algorithms A that run in time at most T and make at most q_h random oracle queries, the following experiment outputs 1 with probabillity at most ε :

- *Offline Phase.* For all $i \in [n]$, run Keys on input (par, i) to generate keys $(pk_i, sk_i) \leftarrow \mathsf{Keys}(par, i)$. On input par and $\{pk_i\}_{i\in[n]}$, A returns an index set $C \subset [n]$ of initially corrupted parties along with updated public keys $\{\hat{pk}_j\}_{j\in C}$. Set $pk_j := \hat{pk}_j$ for all $j \in C$. Initiate an execution of $\mathcal{RB}$ with A controlling parties in C.
- *Random Oracle Queries.* At any point of the experiment, A gets access to an oracle that answers queries of the following type: When A submits a query m, check if $H[m] = \perp$. If so, set $H[m] \leftarrow \{0,1\}^\lambda$. Return $H[m]$.
- *Online Phase.* Run $\mathcal{RB}$ with A. When A outputs a tuple (σ'_e, e) for an $e > \ell$, the experiment ends with output 0 if there is an honest party that has output a value $\sigma_{\ell+1}$ for epoch $\ell + 1$. Continue the execution of $\mathcal{RB}$ for another $e - \ell$ epochs.
- *Corruption Queries.* During the online phase, A may corrupt a party P_i by submitting an index $i \in [n] \setminus C$. In this case, return the internal state of P_i and set $C := C \cup \{i\}$. Henceforth, A controls P_i.
- *Output Determination.* Return 1 if $|C| \leq t$, $e \geq \ell + d$, $L \geq e$, and $\sigma'_e = \sigma_e$. Otherwise, return 0.

We say that $\mathcal{RB}$ is a $(\varepsilon, T, t, L, q_h, d)$-secure randomness beacon protocol if it is (ε, T, t, L)-bias-resistant, $(\varepsilon, T, t, L, q_h, d)$-unpredictable, (t, L)-available, and (t, L)-consistent.

Appendix 5B: Additional Figures

In this section, we provide figures for components of our DKG protocol. In Figure 5.5, we give a description of the deliver function whose purpose is to efficiently broadcast a long message to all parties in some designated set. In Figure 5.6, we give a description of the aggregatable PVSS scheme which allows for public verifiability of a sharing with the additional feature of secure aggregation of several sharings. For the description, we denote by $\mathcal{LC}$ the linear vector space over $\mathbb{Z}_p$ of length n and dimension $t + 1$ defined as

$$\mathcal{LC} := \{(f(1), \ldots, f(n)) \mid f \in \mathbb{Z}_p[X]_{(t)}\},$$

where $\mathbb{Z}_p[X]_{(t)}$ denotes the set of all polynomials in $\mathbb{Z}_p[X]$ of degree at most t. Its dual space $\mathcal{LC}^\perp$ is defined as

$$\mathcal{LC}^\perp := \{(\mu_1 r(1), \ldots, \mu_n r(n)) \mid r \in \mathbb{Z}_p[X]_{(n-t)}\},$$

where $\mu_i := \prod_{j\in[n]\setminus\{i\}} 1/(i - j)$. Further, we denote by $\langle m \rangle_i$ the tuple consisting of message m and a signature σ_i of P_i on m.

5.7.1 Byzantine Agreement Protocol

In this section, we present the Byzantine agreement protocol designed by Momose and Ren [MR21b] in which we implement the threshold signatures with the ones of Attema et al. [ACR21]. On a high level, the protocol recursively calls itself two times on each half and uses threshold signatures (with variable thresholds) to prove knowledge of a threshold of signatures from different parties on the same message. We emphasize that the transparent threshold signatures [ACR21] we use are not unique and thus not suitable for randomness beacons. We

Let $Q \subseteq \mathcal{P}$ be a set of q parties and let $b := \lceil q/2 \rceil$ be the decoding threshold for a (q, b)-Reed-Solomon code $\mathsf{RS} = (\mathsf{Encode}, \mathsf{Decode})$. Further, let $\mathsf{AC} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Wit}, \mathsf{Ver})$ be an accumulator scheme, and let $z \leftarrow \mathsf{Eval}(ak, \mathsf{Encode}(m))$ be the accumulation value for a deterministic encoding of message m.

- **Round 1.** Split m into b data symbols $(m_1, \ldots, m_b)$ as per a predefined policy. Run Encode on input $(m_1, \ldots, m_b)$ to obtain q code words $(s_1, \ldots, s_q)$. For all $j \in [q]$, compute a witness $w_j \leftarrow \mathsf{Wit}(ak, z, s_j)$ and send the (signed) tuple $\langle s_j, w_j, z \rangle_i$ to party $P_j \in Q$.
- **Round 2.** Any party $P_j \in Q$ does the following. Upon receiving the *first* valid code word $\langle s_j, w_j, z \rangle_*$ for z, forward this code word to all parties in Q. *// Valid here means that the tuple* (s_j, w_j, z) *verifies according to* $\mathsf{Ver}(ak, z, w_i, s_i) = 1$.
- **Local Output.** Upon receiving b valid code words for z, run Decode on these code words to obtain m.

Figure 5.5: Description of the $\mathsf{Deliver}$ protocol on input (Q, m, z) invoked by party P_i.

provide a formal description of the BA protocol in Figure 5.7. As a building block, it uses the protocol in Figure 5.8. For more details, we refer directly to the original work [MR21b].

Transparent Threshold Signatures. In the following, we elaborate on the setup of the transparent threshold signatures of Attema et al. [ACR21]. Concretely, it requires a plain PKI and additional $O(n)$ random group elements for a vector commitment scheme which can for example be derived from random oracles. Essentially, the latter defines the public parameters of the commitment scheme, which is an unstructured public random string. In the recursive Byzantine agreement protocol, threshold signatures are used within sets of parties that are determined at the onset of the protocol execution (as the partition of the set of parties is deterministic). Therefore, the setup can be run for each such set of parties specified by the protocol, once the set of all parties along with their PKI keys is fixed. By the recursion, the total number of such sets is $1 + 2 + 4 + \ldots + n \leq 2n$ with exponentially decreasing sizes. Thus, the total number of additional random group elements required for setup is still $O(n)$ with essentially no overhead in the running time (compared to when the setup is only run for the set of all parties).

Security. The authors show that their BA protocol [MR21b] is secure against an adaptive adversary, assuming perfect security of the threshold signature. Conversely, if the threshold signature has only computational security, we can build a straightforward reduction from the security of the Byzantine agreement protocol to the security of the threshold signature. For this, observe that a total of $O(n \log n)$ threshold signatures are exchanged in an execution of the BA protocol, and any violation of security in the BA protocol is directly incurred by a threshold signature. Therefore, a guess with success probability more than $1/n^2$ suffices to build an efficient reduction against the security of the threshold signature. In particular, the security of the threshold signature of Attema et al. implies the security of our Byzantine agreement protocol.

Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a pairing with generators $g \in \mathbb{G}_1, h \in \mathbb{G}_2$, and let (pk_i, sk_i) be the key pair of party P_i where $pk_i = h^{sk_i}$.
The *secret sharing algorithm* Dist (invoked by some dealing party P_L) takes as input a secret key sk_L and public keys $pk_1, \ldots, pk_n$. It outputs a PVSS transcript $T_L := \{\mathbf{C}, \mathbf{E}, \pi\}$ as follows.

1. Sample a polynomial $f(X) := \alpha + \ldots + \alpha_t X^t \leftarrow \mathbb{Z}_p[X]$ of degree t. For all $i \in [n]$, set the commitments $C_i := g^{f(i)} \in \mathbb{G}_1$ and encrypted shares $E_i := pk_i^{f(i)} \in \mathbb{G}_2$.

2. Compute a Schnorr NIZK proof of knowledge $\theta = (c, r)$ of discrete logarithm for $\zeta := g^{\alpha}$, and set $\pi := \langle \zeta, \theta \rangle_L$.

The *aggregation algorithm* Agg takes as input $k \geq 1$ transcripts $\{\mathbf{C}_i, \mathbf{E}_i, \pi_i\}_{i \in [k]}$ and outputs a transcript $T := \{\mathbf{C}, \mathbf{E}, \pi\}$. Let $\mu_1, \ldots, \mu_{t+1}$ denote the Lagrange coefficients for the set $\{1, \ldots, t+1\}$ at the point $x = 0$.

3. Set $C_i := \mathbf{C}_{1,i} \cdot \ldots \cdot \mathbf{C}_{k,i}$ and $E_i := \mathbf{E}_{1,i} \cdot \ldots \cdot \mathbf{E}_{k,i}$ for all $i \in [n]$. Further, set $\pi := (\pi_1, \ldots, \pi_k)$. Return the transcript $T := \{\mathbf{C}, \mathbf{E}, \pi\}$ where $\mathbf{C} = (C_1, \ldots, C_n)$ and $\mathbf{E} = (E_1, \ldots, E_n)$.

The *contributor identifier algorithm* OwnId takes as input a transcript $T := \{\mathbf{C}, \mathbf{E}, \pi\}$ and a public key pk_i.

4. Parse π as $(\pi_1, \ldots, \pi_k)$. For all $j \in [k]$, check if the signature on π_j verifies using pk_i. Output 1 if and only if one of these checks succeeds.

The *verification algorithm* Ver takes as input public keys $pk_1, \ldots, pk_n$ and a transcript $T = \{\mathbf{C}, \mathbf{E}, \pi\}$. It outputs 1 (accept) or 0 (reject). Recall the linear space $\mathcal{LC}$ and its dual $\mathcal{LC}^{\perp}$ as defined before.

5. Sample $(\nu_1, \ldots, \nu_n) \leftarrow_{\$} \mathcal{LC}^{\perp}$ and check if $C_1^{\nu_1} \cdot \ldots \cdot C_n^{\nu_n} = 1$. For all $i \in [n]$, check if $e(g, E_i) = e(C_i, pk_i)$.

6. Parse π as $(\pi_1, \ldots, \pi_k)$ and check if $\zeta_1 \cdot \ldots \cdot \zeta_k = C_0$ (obtained by Lagrange interpolation in the exponent from $\mathbf{C}$). Further, check if the NIZK proofs θ_i and signatures on π_i (using pk_i) for all $i \in [k]$ verify.

7. Output 1 if and only if all the above checks verify.

The *decryption algorithm* Dec (on input encrypted shares $\mathbf{E} = (E_1, \ldots, E_n)$) and the *reconstruction algorithm* Rec.

8. On input a secret key sk_i, compute the secret share $S_i = E_i^{1/sk_i}$. A secret share S_j is valid if $e(C_j, h) = e(g, S_j)$. Otherwise, it is invalid.

9. On input $t + 1$ valid secret shares $\{S_i\}_{i \in \mathcal{I}}$, reconstruct the secret S by Lagrange interpolation in the exponent. The secret has the form $S = h^{f(0)} \in \mathbb{G}_2$.

Figure 5.6: Description of the algorithms of our aggregatable PVSS scheme.

Let $Q \subseteq \mathcal{P}$ be a set of q parties and let $b := \lfloor q/2 \rfloor + 1$. Partition Q into two disjoint subsets $Q = Q_1 \cup Q_2$ (each called a *committee*) of size $q_1 := \lceil q/2 \rceil$ and $q_2 := \lfloor q/2 \rfloor$, respectively. Let $l \in \{1, 2\}$ be such that $P_i \in Q_l$, and also fix some small number $const \in \mathbb{Z}_{\geq 1}$. The protocols uses subprotocol $\mathsf{GBA}()$ defined in Figure 5.8.

- **Graded Consensus.** Execute $\mathsf{GBA}(Q, v_i)$ among all parties in Q. Let (v_i, g_i) denote the output.
- **First Recursion.** If $l = 1$, execute $\mathsf{BA}(Q_1, v_i)$ among parties in Q_1. Let v denote the output. In case $|Q_1| \leq const$, parties can execute any Byzantine agreement protocol to obtain v.
- **Proposal Phase.** If $l = 1$, send $\langle \mathsf{propose}, v \rangle_i$ to all parties in Q. Upon receiving the same proposal value v from $\lfloor q_1/2 \rfloor + 1$ different parties in Q_1 (i.e., a majority value) and if $g_i = 0$, update $v_i = v$.
- **Graded Consensus.** Execute $\mathsf{GBA}(Q, v_i)$ among all parties in Q. Let (v_i, g_i) denote the output.
- **Second Recursion.** If $l = 2$, execute $\mathsf{BA}(Q_2, v_i)$ among parties in Q_2. Let v denote the output. In case $|Q_2| \leq const$, parties can execute any Byzantine agreement protocol to obtain v.
- **Proposal Phase.** If $l = 2$, send $\langle \mathsf{propose}, v \rangle_i$ to all parties in Q.
- **Output Generation.** Upon receiving the same proposal value v from $\lfloor q_2/2 \rfloor + 1$ different parties in Q_2 (i.e., a majority value) and if $g_i = 0$, update $v_i = v$. Finally, terminate with output v_i.

Figure 5.7: Recursive Byzantine agreement protocol BA described from the view of party P_i on input v_i.

Let $\mathcal{Q} \subseteq \mathcal{P}$ be a set of q parties among which the protocol is executed and let $b := \lfloor q/2 \rfloor + 1$. We denote by $\pi_b(m)$ a threshold signature on m with threshold b, which is a proof of knowledge of b signatures on m from different parties.

- **Echo Phase.** Initialize sets $W, C_1, C_2 := \emptyset$, grade $g := 0$, and variable $\mathsf{sent} := 0$. Send $\langle \mathsf{echo}, v_i \rangle_i$ to all parties.
- **Forward Phase.** Upon receiving b valid echo messages $\langle \mathsf{echo}, v \rangle_j$ on the same value v from different parties, update $W := W \cup \{(v, \pi_b(\mathsf{echo}, v))\}$. Once $W \neq \emptyset$, send $\langle v, \pi_b(v) \rangle_i \in W$ to all parties and update $\mathsf{sent} = 1$.
- **First Vote Phase.** If $\mathsf{sent} = 1$ and did not receive a valid tuple $(\tilde{v}, \pi_b(\tilde{v}))$ for a different value $\tilde{v} \neq v$, send $\langle \mathsf{vote}_1, v \rangle_i$ to all parties. Upon receiving b valid votes $\langle \mathsf{vote}_1, w \rangle_j$ on the same value w from different parties, update $C_1 := C_1 \cup \{(w, \pi_b(\mathsf{vote}_1, w))\}$.
- **Second Vote Phase.** Once $C_1 \neq \emptyset$ is non-empty, send $\langle \mathsf{vote}_2, w \rangle_i$ along with $\langle w, \pi_b(\mathsf{vote}_1, w) \rangle_i \in C_1$ to all parties.
- **Output Generation.** Upon receiving b valid votes $\langle \mathsf{vote}_2, u \rangle_j$ on the same value u from different parties, update $C_2 := C_2 \cup \{(u, \pi_b(\mathsf{vote}_2, u))\}$. If $C_1 \neq \emptyset$, set $v_i := \tilde{v} \in C_1$. If further $\tilde{v} \in C_2$, set $g := 1$. Terminate with (v_i, g).

Figure 5.8: Graded Byzantine agreement protocol GBA described from the view of party P_i on input $(\mathcal{Q}, v_i)$.

6

Network-Agnostic Security Comes (Almost) for Free in DKG

Details on this Chapter

This chapter is based on the conference publication [Bac+23a] and its full version [Bac+22]. The sections on DKG to which I contributed as the main author are included, whereas the sections on MPC are not. The original publication has been partially reordered, and editorial improvements have been made throughout the text.

Distributed key generation (DKG) protocols are an essential building block for threshold cryptosystems. Many DKG protocols tolerate up to $t_s < n/2$ corruptions assuming a well-behaved synchronous network, but become insecure as soon as the network delay becomes unstable. On the other hand, solutions in the asynchronous model operate under arbitrary network conditions, but only tolerate $t_a < n/3$ corruptions, even when the network is well-behaved. In this work, we ask whether one can design a protocol that achieves security guarantees in either scenario. We give a complete characterization of *network-agnostic* DKG protocols, showing that the tight bound is given by the identity $t_a + 2t_s < n$. Our protocol incurs comparable communication complexity as state-of-the-art DKG protocols with optimal resilience in their respective purely synchronous and asynchronous settings, thereby showing that network-agnostic security comes *(almost) for free*.

6.1 Introduction

The problem of *distributed key generation* (DKG) has been extensively studied in the cryptographic literature and is a fundamental building block for threshold cryptosystems. It allows a set of n parties to compute a uniform sharing of a secret key such that a sufficiently large threshold of $t + 1 < n$ parties must cooperate to reconstruct the secret or compute some function of it. As such, DKG has met several applications, including key escrow services, password-based authentication, threshold signing and encrypting, and many more. Many existing protocols solve DKG for up to $t < n/2$ malicious corruptions, assuming that the network is *synchronous* [Ped91; Gen+99; Shr+21a]. In the synchronous model, message delays are upper bounded by some known finite delay Δ and parties are assumed to have synchronized clocks. These protocols, however, provide no security guarantees when the network is *asynchronous*. Therefore, a more recent line of work has aimed at solving DKG in the asynchronous network model [KG09; Abr+21a; Das+22c]. However, asynchronous protocols inherently tolerate at most $t < n/3$ malicious parties, even when the network is synchronous. This poses a vexing dilemma for a protocol designer who can not predict the behaviour of the network. On the one hand, she can choose a synchronous protocol that tolerates the maximum number of $t < n/2$ malicious parties. However, such a protocol might lose all security guarantees if the network ever becomes asynchronous. On the other hand, she can opt for an asynchronous protocol. While this type of protocol remains secure under arbitrary network conditions, it tolerates only $t < n/3$ corrupted parties even if the network behaves synchronously.

Motivated by the above discussion, we ask the following question: *Is it possible to design a network-agnostic DKG protocol that achieves security guarantees in either scenario? Moreover, can we achieve network-agnostic security with no efficiency overhead, i.e., with the same efficiency as state-of-the-art purely synchronous and asynchronous DKG protocols?*

6.1.1 Our Contributions

We answer these questions in the affirmative. Our contributions are motivated by a series of recent works on network-agnostic protocols for various types of consensus [BKL19; BKL21] and multi-party computation [BLZL20; Ale+22b; ACC22] (MPC). Existing protocols, however, strongly rely on trusted setup, particularly in the form of threshold cryptosystems [BLZL20; DHLZ21]. Thus, the import of our work lies within replacing this setup at essentially no cost.

In more detail, we show the following results:

We propose the first network-agnostic DKG protocol. Our protocol tolerates $n/3 < t_s < n/2$ corrupted parties in the synchronous model and $t_a < n/3$ parties in the asynchronous model where t_a and t_s can be chosen arbitrarily subject to $t_a + 2 \cdot t_s < n$. Our protocol is resilience-optimal since we also prove $t_a + 2 \cdot t_s < n$ is *necessary* for network-agnostic DKG. It works in the plain PKI model[1] and allows parties to agree on a *field element* x for the public key $y = g^x$ with only $O(\lambda n^3)$ communication complexity. This matches the best known results in synchrony [Shr+21a] and asynchrony [Das+22c]. Thus, our DKG protocol can be used to efficiently bootstrap trusted key generation for network-agnostic consensus and MPC protocols.

In summary, our protocol incurs no additional setup assumptions or asymptotic overhead over state-of-the-art communication-efficient protocols for the synchronous and asynchronous network models. This shows that network-agnostic DKG essentially come 'for free'.

6.2 Technical Overview

We provide an overview of the challenges and our novel techniques.

6.2.1 Background and Starting Point

We consider n parties $P_1, \ldots, P_n$ that communicate over pairwise authenticated channels. Moreover, we assume that parties share a public key infrastructure (PKI), and denote P_i's secret and public key as sk_i and pk_i, respectively. We do not make any assumption on the distributions of corrupted parties' keys and assume that they can be maliciously generated. Throughout, we fix thresholds $0 < t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ such that $t_a + 2 \cdot t_s < n$. Our model assumptions can now be characterized as follows:

- If the model is synchronous, parties are assumed to have synchronized clocks and messages sent by parties are delivered within some known finite upper bound Δ. At most t_s parties can be maliciously corrupted.
- Otherwise, if the network is asynchronous, messages can be arbitrarily delayed, as long as they are never dropped and are delivered within finite time. Moreover, parties' clocks can be arbitrarily out of synch. In this case, at most t_a parties may be maliciously corrupted. Note that the network can never become asynchronous once t_a or more parties have been corrupted.
- Parties do not know a priori how the network might behave.

Our goal is to design a distributed key generation (DKG) protocol for parties to securely distribute a uniform *field* element x corresponding to some public key $y = g^x$. Since parties cannot be sure whether the network is synchronous or not in general, we require that the secret reconstruction threshold be $\ell = t_s + 1$. With the aim of matching the best known DKG protocols for the synchronous and asynchronous model, we aim for our DKG to run in $O(\lambda n^3)$ communication complexity and be statically secure. In Table 6.1, we compare the existing state-of-the-art with our proposed DKG which, as we show, satisfies these aims.

[1] In this model, the public keys of corrupted parties can be generated arbitrarily.

Protocol	Network	Adv	Commun	Rounds	Setup
Shrestha et al. [Shr+21a]	*sync*	static	$O(\lambda n^3)$	$O(n)/O(1)$	PKI, ROM, SRS
Das et al. [Das+22c]	*async*	static	$O(\lambda n^3)$	$O(\log n)$	PKI, ROM, URS
Abraham et al. [Abr+21a]	*async*	static	$\tilde{O}(\lambda n^3)$	$O(1)$	PKI, SRS
Zhang et al. [Zha+23]	*async*	static	$O(\lambda n^4)$	$O(1)$	URS
Abraham et al. [Abr+22b]	*async*	adapt	$\tilde{O}(\lambda n^3)$	$O(1)$	PKI, SRS
Our work (Section 6.6)	*fallback*	static	$O(\lambda n^3)$	$O(n)$	PKI, ROM, SRS

Table 6.1: Comparison table of state-of-the-art DKG protocols. **Network** denotes the network model, which is synchronous (sync), asynchronous (async) or either synchronous or asynchronous (fallback). **Advers** denotes the adversarial model, which is either static or adaptive. **Commun** denotes communication complexity in bits. **Rounds** denotes the (expected) round complexity, where Shrestha et al. (Shr+21a) provide a deterministic and a randomized protocol (deterministic/randomized). **Setup** denotes the setup assumptions, which include a bulletin board PKI (PKI), a random oracle (ROM), a structured reference string (SRS) or a uniform random string (URS). We note that (Abr+21a) constructs a shared *group* element (rather than a *field* element) and originally achieves $\tilde{O}(\lambda n^3)$ communication complexity, which can be reduced to $O(\lambda n^3)$ as detailed in (Gao+22) using their approach. The protocols (Shr+21a; Abr+22b; Abr+21a) require a powers-of-tau setup as SRS.

We stress that this goal cannot be achieved by simply running a generic, network-agnostic MPC protocol [BLZL20; DHLZ21], since these protocols require trusted setup in the form of shared keys (which is exactly the goal we are trying to achieve).

Network-Agnostic Protocols: A Blueprint. To design an efficient network-agnostic DKG protocol, a natural approach is to follow the template of previous works [BKL19; BLZL20; BKL21]. Here, the protocol is divided into two components, a synchronous component Π_s and an asynchronous component Π_a. Parties begin by running Π_s which performs securely, given that up to t_s parties are corrupted. In this case, parties pass the output v_s obtained from Π_s to Π_a. The final output of the protocol is the value v_a output by Π_a. Note, however, that Π_a achieves security only against t_a corruptions, as it is asynchronous. Thus, the key challenge is to prevent Π_a from simply overwriting the output v_s in a synchronous network, as this would degrade the overall corruption threshold of the protocol to t_a.

To prevent this outcome, the idea is to design Π_s and Π_a with two special properties. First, suppose that the network is synchronous and parties agree on the intermediate value v_s passed to the asynchronous component Π_a. In this case, Π_a should simply relay the correct output v_s (rather than recomputing it), *even if t_s parties are corrupted*. Second, if the network is asynchronous, Π_a should be able to compute the correct output v_a on its own in the presence of t_a corruptions. In addition to this, Π_s must prevent parties from computing a catastrophically incorrect intermediate output v_s that might violate the overall security properties of the protocol, given an asynchronous network with t_a corrupted parties.

Background: Synchronous DKG. The above discussion shows why naively running a synchronous DKG and then an asynchronous agreement protocol back-to-back would not produce a network-agnostic protocol. For the setting of DKG, this might lead to the resulting secret key not having enough 'contributions' from honest parties. Our starting point is the synchronous New-DKG protocol of Gennaro et al. [Gen+07], which we make amenable to our network model. Loosely speaking, New-DKG is divided into two phases as follows:

- **Sharing Phase.** In the first phase, each party performs verifiable secret sharing (VSS) using Pedersen's VSS scheme [Ped92] to share two random polynomials f and f' of the same degree. Parties then execute a public complaint management protocol, since some parties may try to misbehave and, e.g., not send (correct) shares to some parties. As a result, they agree on a set of parties Q that honestly executed Pedersen's VSS.

- **Reconstruction Phase.** In the second phase, parties then reconstruct their share of the final secret. To do so, they perform the reconstruction phase of Feldman's VSS [Fel87b] from the sharing phase with respect to the polynomial f. The shares of misbehaving parties are reconstructed publicly to ensure termination.

By committing to a second polynomial f', Pedersen's VSS ensures that shared secrets are unconditionally hiding and parties can efficiently blindly evaluate f in the exponent to verify their shares. Then, Q is sufficiently large to ensure that at least one party must have honestly performed VSS. Thus, the adversary has no information about the secret when Q is decided.

One may be tempted to design a simpler and more efficient DKG protocol where parties, as in the so-called Joint-Feldman DKG [Gen+07], run Feldman's VSS in parallel. However, Gennaro et al. [Gen+07] highlighted that the adversary can bias the distribution of the public key by manipulating the set Q, thus precluding security. In general, a rushing adversary can choose to include or exclude the contributions of parties in the final secret which thus precludes any 'one-round' DKG protocol from outputting a uniformly random secret.[2] In any case, it is not hard to see that the complaint management protocols in New-DKG fail in asynchrony. This is because the complaints of honest parties may be arbitrarily delayed on the network, precluding either correctness or liveness. Dealing with this issue creates additional challenges which we address in the following section.

6.2.2 Our DKG Construction

A natural idea is to replace Pedersen's VSS in the first phase with publicly verifiable secret sharing (PVSS) [Cho+85b; Sta96]. The key property of PVSS is that all parties can *non-interactively* verify whether a given sharing is correct. We begin by presenting a secure, but excessively expensive strawman solution which will serve as our starting point.

A Strawman Solution. To ensure that parties agree on PVSS sharings, each party (acting as the dealer) would first synchronously broadcast their PVSS sharing. In the second phase, parties could then use the asynchronous common subset (ACS) protocol of Blum et al. [BKL21] to agree on a common subset of such sharings. In their protocol, each party provides an input v and the protocol lets them agree on a common subset of $n - t_a$ outputs, given t_a corruptions in an asynchronous network. In addition, their protocol guarantees that if all honest parties start with the *same* input v, then the protocol will remain secure for up to t_s corruptions and the output will be the singleton set $\{v\}$. These properties have made their protocol a staple building block in many network-agnostic protocols.

In our scenario, parties would input their common view of all sharings after the broadcast phase to ACS. Given that the synchronous phase succeeded (i.e., the network is synchronous), all parties would input the *same view* and hence ACS would allow them to (re-)agree on this view,

[2] Note that a possibly biased secret can still be sufficient for applications like threshold signatures, as highlighted in [Gen+07; BL22a].

given at most t_s corrupted parties. If parties have not obtained any output from the synchronous phase, they would simply input their PVSS dealing as an input to ACS directly. Even in case the network is asynchronous, parties would still be able to agree on a subset of $n - t_a$ dealings in this manner. From this, they could securely derive a common secret key.[3]

Unfortunately, existing network-agnostic ACS protocols [BKL21; Ale+22a] execute n instances of binary consensus and consequently require $O(n)$ distributed coin flips, and adapting ACS to the network-agnostic setting without this requirement is an open problem. As the best known protocol to flip coins without trusted setup requires $O(\lambda n^3)$ communication complexity [Gao+22], this step alone incurs at least $O(\lambda n^4)$ overhead. In addition, the above solution requires all parties to broadcast $O(\lambda n)$-sized sets of ciphertexts containing the parties PVSS dealings. Using existing broadcast protocols, this step would incur an additional communication overhead of $O(\lambda^2 n^4)$. Towards building a network-agnostic DKG with $O(\lambda n^3)$ communication complexity, we introduce novel techniques to overcome the above challenges.

From ACS to Intrusion-Tolerant Consensus. Recall that under synchrony, all parties are guaranteed to output at least $n - t_s$ values from the parallel broadcast in our above strawman protocol. However, if the network is asynchronous, parties are not guaranteed agreement or termination of sufficiently many broadcast instances. To cope, our idea is to let parties execute an asynchronous agreement protocol with an *intrusion tolerance* validity property [MR10]. Intrusion tolerance guarantees that a decided value is either one that is proposed by an honest party or a default value $\perp$. In addition, we will require that our agreement protocol satisfies a special *validity with termination property* for up to t_s corruptions. This property ensures that if all parties input the *same value* v to the protocol, they all terminate with this value. (This property was first formalized by Blum et al. [BKL19].) Given this building block, our high-level strategy (from the view of a party P) is as follows.

- If P correctly outputs in at least $n - t_s$ broadcasts, it inputs its set of values to the intrusion-tolerant consensus protocol Π_{IT}. Otherwise, it inputs a default value $\perp'$.
- If a set of values is decided upon by Π_{IT}, P continues with the protocol. Otherwise, P participates in an execution of an asynchronous DKG protocol with $O(\lambda n^3)$ complexity and reconstruction threshold of $t_s + 1$; the protocol of Das et al. [Das+22c] satisfies these requirements.

In synchrony, by the security of broadcast, all parties will propose the same set to Π_{IT} and, by the validity of consensus Π_{IT} under t_s corruptions, this set will be decided. In this case, parties do not fall back to the asynchronous path and can cheaply agree on a t_s-sharing of a field element x.

In asynchrony, the synchronous path might fail. In this case, however, agreement and intrusion tolerance of Π_{IT} ensure that all parties securely continue execution of the synchronous path with the same view or collectively fall back to asynchronous DKG. In case parties do not fall back, their common view on the protocol state allows them to securely emulate the synchronous protocol path. In either scenario, parties agree on a t_s-sharing of a field element x, *even if the network behaves asynchronously*. The following paragraphs describe how, for each phase of our protocol, we manage to keep communication below $O(\lambda n^3)$.

[3] This discussion omits a minor technical detail: the adversary must not be able to send incorrect messages on behalf of honest parties, even in asynchrony. But ensuring this is easy with signatures.

Protocol	Resil	Adapt	Commun	Rounds	Input	Setup
Abraham et al. [Abr+19a]	$1/2 - \epsilon$	✓	$\tilde{O}(\lambda n + \ell n)$	$O(1)$	ℓ-bit	*trusted*
Momose-Ren [MR21c]	1/2	✓	$O(\lambda n^2)$	$O(n)$	1-bit	*trusted*
Chan et al. [CPS20]	$1 - \epsilon$	✓	$O(\lambda^2 n^2)$	$O(\lambda)$	1-bit	*trusted*
Dolev-Strong [DS83b]	1	✓	$O(\lambda n^3 + \ell n)$	$O(n)$	ℓ-bit	*plain*
Momose-Ren [MR21c]	$1/2 - \epsilon$	✓	$O(\lambda n^2)$	$O(n)$	1-bit	*plain*
Tsimos et al. [TLP22a]	$1 - \epsilon$	✗	$O(\lambda^2 n^2)$	$O(n)$	1-bit	*plain*
Our work (Section 6.4)	$1 - \epsilon$	✓	$O(n\ell + \lambda n^2)$	$O(n)$	ℓ-bit	*plain*

Table 6.2: Comparison table of existing synchronous broadcast protocols. **Resil** denotes the Byzantine corruption threshold as a fraction of the total number of parties, where $\epsilon \in (0, 1)$ is a constant. **Adapt** denotes whether the adversary is adaptive or not. **Commun** denotes communication complexity in bits for messages of length ℓ when relevant. **Rounds** denotes the round complexity. **Input** denotes the length of the input value (either binary or multivalued). **Setup** denotes the setup assumption regarding the keys, either trusted or plain PKI.

An Efficient Broadcast Protocol. The first ingredient we propose is an efficient multivalued synchronous broadcast protocol assuming $t < (1 - \epsilon) \cdot n$ corruptions for any constant $\epsilon \in (0, 1)$. Tsimos, Loss and Papamanthou [TLP22a] propose an efficient *binary* broadcast protocol, BulletinBC, that is statically secure. BulletinBC requires $O(\lambda^2 n^2)$ communication, and is very similar to the classic Dolev-Strong broadcast protocol [DS83b] except to reduce communication complexity, parties *gossip* instead of *multicast* sets of signatures during the protocol. We modify this protocol and an extension protocol from Nayak et al. [Nay+20b] in Section 6.4 to build a multivalued synchronous broadcast protocol with $O(n\ell + \lambda n^2)$ communication complexity where ℓ is the length of the input message. The best prior known protocol had a communication cost of $O(n\ell + n^3)$ and so this construction may be of independent interest. In Table 6.2, we compare our protocol to other synchronous broadcast protocols from the literature. We note that the extension protocol from Nayak et al. in combination with the Byzantine agreement protocol from Momose and Ren [MR21c] yields a synchronous broadcast protocol with quadratic communication complexity in the honest majority setting, but assumes trusted setup in the form of threshold signatures. In their paper, Momose and Ren also present a second efficient Byzantine agreement protocol that does not require trusted setup. However, that protocol only works with a sub-optimal resilience of $t < (\frac{1}{2} - \epsilon) \cdot n$.

We use our extension protocol for the first phase of DKG. More precisely, each party P_i broadcasts (1) n Pedersen commitments corresponding to random polynomials f_i and f_i', (2) n ciphertexts corresponding to each party P_j's share of P_i's secret, namely $f_i(j)$ and $f_i'(j)$, and (3) n NIZK proofs that proves ciphertext j, for each $j \in [1, n]$, contains encryptions of values $f_i(j)$ and $f_i'(j)$. This obviates the need for a complaint management protocol as each party can determine the well-formedness of each message broadcast themselves. With $O(\lambda)$-sized NIZKs [Par+13; Can+20], each party invokes broadcast with an $O(\lambda n)$-sized message, and consequently this step incurs $O(\lambda n^3)$ communication.

Our Intrusion-Tolerant Consensus Protocol. We adapt a multivalued Byzantine agreement protocol from Mostéfaoui and Raynal [MR17] to ensure intrusion tolerance and validity under t_s corruptions in synchrony. We show in Section 6.5 that the protocol has $O((\ell + \lambda)n^3 + \lambda n^3)$ communication complexity. As such, parties cannot simply propose their $O(\lambda n^2)$-sized set of

sharings (recall that such a set contains $n - t_s$ sharings each of size $O(\lambda n)$) to consensus within DKG without incurring super-cubic complexity.

Efficiently Reconstructing the Final Output. To keep the communication complexity below $O(\lambda n^3)$, we observe that each party does not require the entire contents of the $O(\lambda n^2)$-sized set to reconstruct their share and the public key of the final secret. To this end, a party accumulates n 'personalised' values, one per party and each of size $O(\lambda n)$, into an accumulation value z that they propose to consensus. An accumulation value for party P_i contains a description of the qualified parties Q, the $|Q|$ ciphertexts P_i needs to reconstruct their share of the secret $\sum_{q \in Q} f_q(i)$ (alongside $\sum_{q \in Q} f'_q(i)$), and a Pedersen commitment corresponding to these two summations. By intrusion tolerance, if a non-trivial value is decided by consensus, the honest party (or parties) who proposed such a z can send the relevant part and proof of membership in z to each party. By using an accumulator with accumulation value z of size at most $O(\lambda)$ [BP97; Lip12], we thus achieve $O(\lambda n^3)$ complexity for this step. We emphasize that without the intrusion tolerance property it would be possible for parties to decide a value from consensus that does not correspond to an 'honest' accumulation value z. One could bypass intrusion tolerance using a consensus protocol with $O(n\ell + \lambda n^3)$ complexity that ensures *external validity* on decided values [Cac+01]. However, it appears difficult to design such a protocol using erasure codes (as is typical) as parties cannot feasibly evaluate an external validity function on a message until it is reconstructed.

From each party's personalized value, they can reconstruct their share of the secret key but not yet the public key. To reconstruct the public key, it is tempting to replace the reconstruction phase of New-DKG with another round of broadcast and agreement. However, this would allow an adversary to bias the distribution of the shared secret by deciding to fallback to asynchronous DKG depending on, e.g., the first bit of the reconstructed public key. We therefore avoid this by publicly reconstructing the public key using the approach of Shrestha et al. in [Shr+21a]. More precisely, each party P_i computes and multicasts the value $G = g^{\sum_{q \in Q} f_q(i)}$ and their accumulation value that they prove is consistent with their Pedersen commitment via an efficient Fiat-Shamir based NIZK [Can+99; Shr+21a]. Parties can thus collect $t_s + 1$ valid points in the exponent of g and then reconstruct the public key by Lagrange interpolation in the exponent and terminate.

6.2.3 More on Related Work

In [MR21a], Momose and Ren initiate the study of the network-agnostic setting where the thresholds for safety and liveness properties are considered separately, and construct corresponding state machine replication protocols.

Distributed Key Generation. Many synchronous DKG protocols assume the existence of broadcast channels, i.e., that essentially abstract away secure broadcast and consensus, including the seminal protocol of Gennaro et al. [Gen+07]. In a recent work, Shrestha et al. [Shr+21a] consider when broadcast is no longer assumed (as in our work), and propose a protocol with $O(\lambda n^3)$ complexity which is the state-of-the-art. Canetti et al. [Can+99] propose an adaptively-secure DKG protocol, but almost all other work, including ours, consider static security. Das et al. [Das+22c] propose an asynchronous DKG protocol with $O(\lambda n^3)$ communication complexity. In order to bypass the need for direct coin flipping (which incurs $O(\lambda n^3)$ overhead), they perform a clever reduction to n instances of binary consensus which uses $O(\lambda n^2)$ for coin flips from honest

parties. Abraham et al. use a so-called aggregatable DKG protocol [Gur+21a] to also build a protocol with $O(\lambda n^3)$ overhead that only requires an efficient Byzantine agreement primitive like [Gao+22]. However, the only efficient construction of aggregatable DKG we are aware of allows parties to agree on a shared *group* element as a secret which can thus be applied only to less standard cryptosystems. The DKG of Zhang et al. [Zha+23] does not require a PKI, CRS or the ROM, but incurs $O(\lambda n^4)$ overhead. Recently, Abraham et al. construct asynchronous DKG with an adaptive security proof [Abr+22b], although they require a powers-of-tau trusted setup which, using the best known asynchronous protocol [DXR22], implies $\tilde{O}(\lambda n^3)$ communication overhead overall.

6.2.4 Outline of this Chapter

The rest of this chapter is structured as follows. In Section 6.3, we define the preliminaries that are only relevant for this chapter. In Section 6.4, we present our new Byzantine broadcast protocol and give a security and complexity analysis. In Section 6.5, we present our new intrusion-tolerant consensus protocol and give a security and complexity analysis. In Section 6.6, we present our network-agnostic distributed key generation protocol and give a security and complexity analysis. In Appendix 6.6.2, we give a graded consensus protocol along with security proofs relevant for our intrusion-tolerant consensus protocol. In Appendix 6.6.4, we give a binary consensus protocol along with security proofs relevant for our intrusion-tolerant consensus protocol.

6.3 Preliminaries for this Chapter

In addition to the general preliminaries in Chapter 2, we introduce here preliminaries that are only relevant for this chapter.

General Notation. Let λ denote the security parameter. In this chapter, we assume that global parameters $par = (\mathbb{G}, p, g, h)$ are fixed and known to all parties. Here, $\mathbb{G}$ is a cyclic group of prime order p with generators g and h. We sometimes use maps or key-value stores, which are data structures of the form $\mathsf{map}[k] = v$ for lookup key k that outputs value v.

Setup and Adversarial Model. In this chapter, we will make direct use of the established public key infrastructure (PKI) between parties. Concretely, each party P_i has an encryption-decryption key pair $(\mathsf{ek}_i, \mathsf{dk}_i)$ for a public-key encryption scheme and a verification-signing key pair (vk_i, sk_i) for a digital signature scheme, where only ek_i and vk_i are known to all parties. We also let $\mathcal{VK}(Q)$ denote the function that takes as input a list of parties $P' = (P_{i(1)}, \ldots, P_{i(k)})$ and outputs the list of their verification keys $(vk_{i(1)}, \ldots, vk_{i(k)})$. We assume a Byzantine adversary that can corrupt up to t parties, and it is rushing. Contrary to the previous chapters, the adversary in this chapter is static.

Network Model. Our network has two possible states: it is either synchronous with delay parameter $\mathsf{Deliver}$, or it is fully asynchronous. Importantly, honest parties do not know a priori how the network behaves, i.e., in which type of network they are in. We stress that the adversary is in full control over message delays, subject to the constraints given by the network type. In particular, the network could behave synchronous in the view of an honest party, while it behaves asynchronous in the view of another honest party.

Idealized Models and Computational Assumptions. We assume the random oracle model. Contrary to the previous chapters, we do not assume the algebraic group model. We will work with the discrete logarithm assumption.

6.3.1 Cryptographic Primitives

Definitions and properties that we introduce hereafter are only required to hold with overwhelming probability $1 - \mathsf{negl}(\lambda)$.

Non-Interactive Zero-Knowledge Proof. A zero-knowledge proof is an interactive protocol between a prover Prove and a verifier Ver which enables the prover to convince the verifier that a statement x is in an NP language $\mathcal{L}$ without leaking any information besides the fact that the statement is true. A non-interactive zero-knowledge (NIZK) proof is a class of zero-knowledge proofs, where no interaction is required. The prover Prove outputs only one message, called a proof, which convinces the verifier Ver of the truth of the statement.

Definition 6.3.1 (Non-interactive zero-knowledge proof (NIZK) [Gro06])**.** Let R be an NP relation. For pairs $(X, \omega) \in R$ we call X the statement and ω the witness. Let L be the language consisting of statements in R. A non-interactive zero-knowledge proof is a tuple of PPT algorithms $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Ver})$ such that:

- Gen: This is a parameter generation algorithm that takes as input the security parameter λ. It outputs parameters par, implicitly input to other algorithms.
- Prove: This is a probabilistic proving algorithm that takes as input a statement X to be proven and the corresponding witness ω where $(X, \omega) \in R$. It outputs a proof π, denoted as $\pi \leftarrow \mathsf{Prove}(X, \omega)$.
- Ver: This is a deterministic verification algorithm that takes as input a statement X and a proof π. It outputs an acceptance bit b, denoted as $b \leftarrow \mathsf{Ver}(X, \pi)$.

For security, we require perfect completeness, zero-knowledge, and simulation-soundness. We formally define these following the literature [Gro06]. Let $\mathcal{R}$ and $\mathcal{L}$ be as specified above.

Definition 6.3.2 (Perfect Completeness)**.** Let $\Sigma = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Ver})$ be a non-interactive proof system as defined above. For an algorithm A, define the perfect completeness experiment $\mathbf{PerfComp}^{\mathsf{A}}_{\Sigma}$ as follows:

- *Offline Phase.* Run the parameter generation algorithm on input λ to generate parameters $par \leftarrow \mathsf{Gen}(\lambda)$. Then, run A on input par.
- *Online Phase.* When A outputs (x, w), generate the proof $\pi \leftarrow \mathsf{Prove}(x, w)$.
- *Output Determination.* Return 1 if $(x, w) \in \mathcal{R}$ and $\mathsf{Ver}(x, \pi) = 1$. Otherwise, return 0.

We say that Σ has perfect completeness if for all algorithms A, $\Pr[\mathbf{PerfComp}^{\mathsf{A}}_{\Sigma} = 1] = 1$.

Definition 6.3.3 (Zero-Knowledge)**.** Let $\Sigma = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Ver})$ be a non-interactive proof system as defined above. Let $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ be a pair of PPT algorithms (called the simulator). Furthermore, let algorithm Sim' be such that $\mathsf{Sim}'(par, \tau, x, w) = \mathsf{Sim}_2(par, \tau, x)$

if $(x, w) \in \mathcal{R}$ and $\mathsf{Sim}'(par, \tau, x, w) = 0$ otherwise. For an algorithm A, we define the zero-knowledge advantage of A as

$$\begin{aligned}\textbf{Adv-zk}_{\Sigma}^{\mathsf{A},\mathsf{Sim}} = |\Pr[par \leftarrow \mathsf{Gen}(\lambda) : \mathsf{A}^{\mathsf{Prove}(par,\cdot,\cdot)}(par) = 1] \\ - \Pr[(par, \tau) \leftarrow \mathsf{Sim}_1(\lambda) : \mathsf{A}^{\mathsf{Sim}'(par,\tau,\cdot,\cdot)}(par) = 1]|.\end{aligned}$$

We say that Σ is zero-knowledge if there exists a simulator Sim as above such that for all non-uniform PPT algorithms A, its zero-knowledge advantage is negligible in λ.

Definition 6.3.4 (Simulation-Soundness)**.** Let $\Sigma = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Ver})$ be a non-interactive proof system as defined above. Let $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ be a pair of PPT algorithms. For an algorithm A, we define the simulation-soundness advantage of A as

$$\begin{aligned}\textbf{Adv-ss}_{\Sigma}^{\mathsf{A},\mathsf{Sim}}(\lambda) = \Pr[(par, \tau) \leftarrow \mathsf{Sim}_1(\lambda), (x, \pi) \leftarrow \mathsf{A}^{\mathsf{Sim}_2(par,\tau,\cdot)}(par) : \\ w \leftarrow \mathsf{Ver}(par, x, \pi) = 1, (x, \pi) \notin Q, x \notin \mathcal{L}],\end{aligned}$$

where Q is the list of simulation queries and responses. We say that Σ has simulation-soundness if there exists a simulator Sim as above such that for all non-uniform PPT algorithms A, its simulation-soundness advantage is negligible in λ.

Cryptographic Accumulator. A cryptographic accumulator scheme [Ngu05] allows to accumulate several elements from some set D into an accumulated value z (using the algorithm Eval). Further, for each element in D it allows to generate a compact proof of membership in D (using the algorithm Wit) called a witness. The standard security notion of collision-resistance requires that it is hard for an adversary to create invalid proofs of membership. An example of cryptographic accumulators are Merkle trees, where the root is the accumulation value and the authentication paths are membership proofs (i.e., witnesses) for the leaves. In this chapter, we use an accumulator scheme with membership proofs and accumulation value each of size $O(\lambda)$. This can be implemented using the accumulator scheme of [BBF19] built upon class groups of unknown order. Alternatively, we could use Merkle trees at the cost of $O(\log n)$ multiplicative overhead in the communication complexity. For a formal definition, we refer to Chapter 2. We will use the shorthand notation $(w_1, \ldots, w_n) \leftarrow \mathsf{Wits}(ak, z, D)$ for set $D = \{d_1, \ldots, d_n\}$ to mean the n invocations $(\mathsf{Wit}(ak, z, d_1), \ldots, \mathsf{Wit}(ak, z, d_n))$ of Wit. Further, we stress that our protocols could also use vector commitments (with constant-sized openings) [CF13] instead of accumulators, which can also be built without trusted setup.

Linear Erasure and Error Correcting Codes. We use standard (q, b)-Reed-Solomon (RS) codes [RS60]. This primitive allows to encode b data symbols into code words of q symbols (using the algorithm Encode) such that b elements of the code word suffice to recover the original data (using the algorithm Decode). In our construction, we will use Reed-Solomon codes with $(q, b) = (n, n - t)$. For a formal definition, see Chapter 2.

Public Key Encryption. A public key encryption scheme allows to encrypt messages towards a given public key pk. Only the holder of the corresponding secret key sk can then decrypt the encryption and recover the message. Further, for security one requires that the encryption does not leak any information about the encrypted message, even when an encryption oracle for messages is given. We now formally define a public key encryption (PKE) scheme and the security notion of CPA-security for it.

Definition 6.3.5 (Public key encryption)**.** A public key encryption scheme is a tuple of PPT algorithms $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ such that:

- KGen: This is a key generation protocol that takes as input the security parameter λ. It outputs a public-secret key pair $(\mathsf{ek}, \mathsf{dk})$, denoted $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KGen}(\lambda)$.
- Enc: This is a probabilistic encryption algorithm that takes as input a public key ek and a message $m \in \{0,1\}^*$. It outputs a ciphertext c, denoted $c \leftarrow \mathsf{Enc}(\mathsf{ek}, m)$.
- Dec: This is a deterministic decryption algorithm that takes as input a decryption key dk and a ciphertext c. It outputs a message m, denoted as $m \leftarrow \mathsf{Dec}(\mathsf{dk}, c)$, with the possibility of $m = \perp$ denoting failure.

Definition 6.3.6 (CPA-Security of PKE)**.** Let $\Pi_{\mathsf{PKE}} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be a public key encryption scheme as above. For $b \in \{0,1\}$ and an algorithm A, we define the experiment $\mathbf{CPA}^{\mathsf{A}}_{\Pi_{\mathsf{PKE}},b}$ as follows:

1. Run the key generation algorithm and get $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KGen}(\lambda)$.
2. Run A on input $(\mathsf{ek}, \mathsf{dk})$ and get two messages m_0, m_1 of the same length.
3. Compute the ciphertext $c \leftarrow \mathsf{Enc}(\mathsf{ek}, m_b)$. Then run A on input c.
4. When A returns $b' \in \{0,1\}$, the experiment returns b'.

We say that Π_{PKE} has indistinguishable encryptions against chosen plaintext attacks (CPA-security) if for all PPT algorithms A, we have

$$|\Pr[\mathbf{CPA}^{\mathsf{A}}_{\Pi_{\mathsf{PKE}},0} = 1] - \Pr[\mathbf{CPA}^{\mathsf{A}}_{\Pi_{\mathsf{PKE}},1} = 1]| \leq \mathsf{negl}(\lambda).$$

Aggregatable Signatures. In an aggregate signature scheme, a party can use its secret key to sign a message individually. All parties can also call algorithm Comb to combine several (possibly already aggregated) signatures with respect to the same message to form a new signature on the same message. As usual, signatures can also be verified using algorithm Ver. We emphasize that Comb and Ver are non-interactive algorithms in this work.

Definition 6.3.7 (Aggregate Signatures)**.** An aggregate signature scheme is a tuple of PPT algorithms $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Comb}, \mathsf{Ver})$ such that:

- KGen: This is a key generation protocol that takes as input the security parameter λ and outputs a public-secret key pair (vk_i, sk_i).
- Sign: This is a probabilistic signing algorithm that takes as input a secret key sk_i and a message $m \in \{0,1\}^*$. It outputs a signature σ_i, denoted as $\sigma_i \leftarrow \mathsf{Sign}(sk_i, m)$.
- Comb: This is a deterministic signature combining algorithm that takes as input a sequence of signatures $\Sigma = (\sigma_{i(1)}, \ldots, \sigma_{i(k)})$, the corresponding sequence of verification keys $VK = (vk_{i(1)}, \ldots, vk_{i(k)})$, and a message m. It outputs either an aggregate signature σ, denoted as $\sigma \leftarrow \mathsf{Comb}(\Sigma, VK, m)$, or $\perp$.

- Ver: This is a deterministic signature verification algorithm that takes as input a message m, an aggregate signature σ, and a set of verification keys $VK = \{vk_1, \ldots, vk_k\}$. It outputs an acceptance bit $b \in \{0, 1\}$.

Note that signatures can be sequentially combined, i.e., Comb can take signatures previously output from Sign or Comb as input. We sometimes write $\langle m \rangle_i$, which denotes the pair (m, σ) where $\sigma \leftarrow \mathsf{Sign}(sk_i, m)$. For a single verification key vk (resp. signature σ), we also write vk (resp. σ) instead of $\{vk\}$ (resp. $\{\sigma\}$) as input to algorithms.

Remark 6.3.1. We can instantiate aggregate signatures of size $O(\lambda) + n$ in the random oracle model, where n is the number of signers [Bon+03]. Similarly, we require individual (non-aggregated) signatures to be of size $O(\lambda)$. We implicitly assume domain separation when signing messages in protocols we introduce.

Definition 6.3.8 (Unforgeability under Chosen Message Attack)**.** Let $\Pi_{\mathsf{AggSgn}} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Comb}, \mathsf{Ver})$ be an aggregate signature scheme. For an algorithm A, we define the experiment $\mathbf{UF\text{-}CMA}^{\mathsf{A}}_{\Pi_{\mathsf{AggSgn}}}$ as follows:

1. Run the key generation algorithm and get a public-secret key pair (vk_1, sk_1). Then, A is given the public key vk_1.
2. At any time of the experiment, A gets access to a signing oracle as follows: When A submits a message $m \in \{0, 1\}^*$, return $\sigma_1 \leftarrow \mathsf{Sign}(sk_1, m)$.
3. A outputs $k - 1$ additional public keys $vk_2, \ldots, vk_k$ for some $k \geq 1$ along with a message m^*. A outputs an aggregate signature σ^* with respect to public keys $VK = \{vk_1, \ldots, vk_k\}$ and message m^*.
4. If $\mathsf{Ver}(VK, \sigma^*, m^*) = 1$ and m^* was not queried previously by A, the experiment returns 1. Otherwise it returns 0.

We say that Π_{AggSgn} is unforgeable under chosen message attack (UF-CMA) or simply secure if for all PPT algorithms A, we have $\Pr[\mathbf{UF\text{-}CMA}^{\mathsf{A}}_{\Pi_{\mathsf{AggSgn}}} = 1] \leq \mathsf{negl}(\lambda)$.

6.3.2 Distributed Primitives

When relevant, our primitives take input from a value set V with $|V| \geq 2$; we assume a default value $\perp \notin V$. We distinguish between algorithms that *generate output* (also called liveness), and algorithms that additionally *terminate*. In particular, an algorithm may be live but not terminating, since it may need to still remain online and send more messages to help other parties output. Our treatment of liveness and termination varies between the primitives we introduce below. Note that $\perp$ is considered as a valid output in each protocol. We first introduce intrusion-tolerant Byzantine agreement and secure broadcast, the two main building blocks we use to build DKG.

Intrusion-Tolerant Byzantine Agreement. Byzantine agreement is a classic primitive that allows parties which each input a value to agree on a common output value. We define liveness (generating output) and termination in two separate properties below. We emphasize that our definition captures the standard Byzantine agreement problem.

Definition 6.3.9 (Byzantine Agreement)**.** Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i begins holding input $v_i \in V$.

- ***Validity.*** Π is *t-valid* if the following holds whenever at most t parties are corrupted: if every honest party's input is equal to the same value v, then every honest party outputs v.
- ***Consistency.*** Π is *t-consistent* if whenever at most t parties are corrupted, every honest party that outputs a value outputs the same value v.
- ***Liveness.*** Π is *t-live* if whenever at most t parties are corrupted, every honest party outputs a value $v \in V \cup \{\perp\}$.
- ***Termination.*** Π is *t-terminating* if whenever at most t parties are corrupted, every honest party terminates.
- ***Intrusion tolerance.*** Π is *t-intrusion tolerant* if whenever at most t parties are corrupted, every honest party that outputs a value either outputs an honest party's input v or $\perp$.
- ***Validity with termination.*** Π is *t-valid with termination* if the following holds whenever at most t parties are corrupted: if every honest party's input is equal to the same value v, then every honest party outputs v and terminates.

If Π is t-valid, t-consistent, t-live, and t-terminating, we say it is *t-secure*.[4] If Π is t-secure and is t-intrusion tolerant, we say it is *t-secure with intrusion tolerance*.

Byzantine Broadcast. In Byzantine broadcast (or just broadcast), parties aim to agree on a value which is either the value chosen by the designated sender or a default value (in case the sender is dishonest). Our definition handles termination directly, even in asynchrony (where we only guarantee weak validity). As for Byzantine agreement, the following definition captures the standard broadcast primitive.

Definition 6.3.10 (Byzantine Broadcast)**.** Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where a designated party P begins holding input $v \in V$.

- ***Validity.*** Π is *t-valid* if whenever at most t parties are corrupted: if party P is honest and inputs v, then all honest parties P_j output v.
- ***Consistency.*** Π is *t-consistent* if whenever at most t parties are corrupted, every honest party outputs the same value v'.
- ***Liveness.*** Π is *t-live* if whenever at most t parties are corrupted, every honest party outputs a value $v' \in V \cup \{\perp\}$.
- ***Termination.*** Π is *t-terminating* if whenever at most t parties are corrupted, every honest party terminates.
- ***External validity.*** Π is *t-externally valid* if the following holds whenever at most t parties are corrupted: if honest party P_i outputs v', then for validity predicate Q, $Q(v)$ is true.

[4] We emphasise that t-security does not imply t-intrusion tolerance.

- ***Weak validity.*** Π is *t-weakly valid* if whenever at most t parties are corrupted: if P is honest and inputs v, then all honest parties P_i output either v or $\perp$ and terminate upon generating output.

If Π is t-valid, t-consistent, t-live and t-terminating, we say it is *t-secure*.

Note that weak validity was also defined in [BKL19], and external validity was introduced in [Cac+01] for Byzantine agreement.

Distributed Key Generation. Following prior work, we introduce a property-based definition of distributed key generation (DKG). In a DKG protocol, a set of parties collaborate to share a uniformly random secret. Each party outputs the public key corresponding to the secret, its own share of the secret, and a list of public key shares that parties can use to prove ownership of their share. We restrict our definition to the case where parties share a uniform secret *field element* x with corresponding public key $y = g^x$ (where g is a generator of the underlying cyclic group). We emphasize that the literature also considers DKG protocols where a secret group element is shared. Further, there are also other definitions for DKG to capture other settings.

Definition 6.3.11 (Distributed Key Generation)**.** Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i outputs a secret key share sk_i, a vector of public key shares $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, a public key pk, and parties terminate upon generating output.

- ***Correctness.*** Π is *(t, d)-correct* for $d > t$ if whenever at most t parties are corrupted, there exists a polynomial $f \in \mathbb{Z}_p[X]$ of degree $d - 1$ such that for all $i \in [n]$, $\mathsf{sk}_i = f(i)$ and $\mathsf{pk}_i = g^{\mathsf{sk}_i}$. Moreover, $pk = g^{f(0)}$.

- ***Consistency.*** Π is *t-consistent* if whenever at most t parties are corrupted, all honest parties output the same public key pk and the same vector of public key shares $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$.

- ***Secrecy.*** Π is *t-secret* if the following holds whenever at most t parties are corrupted: For every (PPT) adversary A, there exists a (PPT) simulator Sim with the following property. On input an element $y \in \mathbb{G}$ and a set of corrupted parties C with $|C| \leq t$, Sim generates a transcript whose distribution is computationally indistinguishable from A's view of a run of Π with corrupted set C in which all honest parties output y as their public key.

- ***Uniformity.*** Π is *t-uniform* if the following holds whenever at most t parties are corrupted: Fix $y \in \mathbb{G}$. Then, for every (PPT) adversary A, for every honest party that outputs public key pk, $pk = y$ holds with probability negligibly close to $1/p$, where the probability is taken over A's randomness (and not the coins used in setup).

If Π is (t, d)-correct, t-consistent, t-secret, and t-uniform, we say it is *(t, d)-secure*.

Our definition is adapted from that of Bacho and Loss [BL22c] except we only require a standard secrecy notion akin to that of Gennaro et al. [Gen+07]. As we consider static security, our simulator is parametrised by the set of corrupted parties B chosen by the adversary. Apart from our additional uniformity property, the main difference is that we allow the secret threshold to be a value d that exceeds the number of corruptions t by more than 1. Looking forward, our DKG protocol will satisfy (t_s, d)-security in synchrony and (t_a, d)-security in asynchrony for $d = t_s + 1$. In particular, our protocol achieves t_a-secrecy in asynchrony. The definition of

secrecy is not well-defined in asynchrony when considering more than t_a corruptions, because in particular not all parties may output y (or worse yet they may output different keys). One could define a variant of secrecy that guarantees 'secrecy with abort' but its usefulness is less clear given only a subset of honest parties could output a secret share.

6.4 Efficient Synchronous Broadcast

In this section, we construct a synchronous secure broadcast protocol with $O(\ell n + \lambda n^2)$ communication complexity that tolerates $t < (1-\epsilon) \cdot n$ corruptions with $\epsilon \in (0,1)$ for messages of length ℓ. To do so, we adapt the extension protocol proposed by Nayak et al. [Nay+20b]. Their protocol, however, relies on a λ-bit broadcast module with the same corruption tolerance and communication complexity $O(\lambda n^2)$. We therefore first construct such a protocol.

6.4.1 Short Message Broadcast Module

We present our protocol $\Pi_{\mathsf{BC}}^{t,\epsilon}$ in Figure 6.1 that allows λ-bit messages to be broadcast with $O(\lambda n^2)$ communication complexity. We assume the existence of an aggregate signature scheme as. Let $R = O(\log n)$ and $q = O(1/\epsilon)$ be two constants that we use in the protocol and the proof.

Our protocol is similar to BulletinBC [TLP22a] (see Figure 2 there), which in turn is similar to the well-known Dolev-Strong broadcast protocol. Whereas in Dolev-Strong signatures are multicast to all parties, in BulletinBC signatures are sent to each party only with probability q/n (the *gossiping* technique). We emphasise that gossiping does not require a common coin, but only a local source of randomness: parties each locally sample the set of parties to gossip messages to. To ensure security, BulletinBC thus requires an additional $R = O(\log n)$ rounds to ensure that the 'gossiped' message propagates to all parties except with negligible probability. Notably, we extend BulletinBC to support *multivalued* broadcast and improve upon the communication complexity by using an aggregate signature scheme $\mathsf{as} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Comb}, \mathsf{Ver})$.

In $\Pi_{\mathsf{BC}}^{t,\epsilon}$, each party P_i manages two local maps $\mathsf{sent}, \mathsf{detect} : M \to \{\mathsf{false}, \mathsf{true}\}$ with initialization $\mathsf{sent}[m] = \mathsf{detect}[m] = \mathsf{false}$ for all $m \in M$ (where M denotes the message space). In the first step of the protocol, the sender P^* multicasts its signed input value m_i and sets $\mathsf{sent}[m_i] = \mathsf{detect}[m_i] = \mathsf{true}$. The protocol then runs $t + R$ rounds as follows. In rounds $1 \le r \le t + R$, for each $m \in M$, if P_i has 1) received a signature on m signed by the sender P^*; 2) can form a valid aggregate signature with $\min\{r-1, t\}$ signers;[5] and 3) they have previously gossiped/multicast at most one message $m' \neq m$ (i.e., $|\{m' \in M : \mathsf{sent}[m'] = \mathsf{true}\}| \le 1$, P_i sets $\mathsf{sent}[m] = \mathsf{true}$, computes an aggregate signature on it and sends this plus P^*'s signature to each party with probability q/n.

Note if we simply replace the (deterministic) multicast from Dolev-Strong with probabilistic sending, then consistency may not hold if P^* signs more than two messages above, since condition 3) above implies that honest parties do not relay all messages. To deal with this, parties keep track of P^*'s signatures separately. In particular, when P_i receives a signature σ of P^* on m, if $\mathsf{detect}[m] = \mathsf{false}$ and $|\{m' : \mathsf{detect}[m] = \mathsf{true}\}| < 2$, P_i *multicasts* (not gossips) m and

[5] For rounds $r \ge t + 1$ we only require $t + 1$ signatures including the sender's.

$\Pi_{\mathsf{BC}}^{t,\epsilon}(m^*)$

1. At time 0:
   - Set $\mathsf{sent}[m] = \mathsf{detect}[m] = \mathsf{false}$ for all $m \in M$ and $v = \perp$.
   - If $P_i = P^*$: set $\sigma^* \leftarrow \mathsf{as.Sign}(sk_i, m^*)$ and $\mathsf{sent}[m^*] = \mathsf{detect}[m^*] = \mathsf{true}$, then multicast $(\sigma^*, \Sigma = (), \mathbf{P} = (), m^*)$.
2. For $r = 1, \ldots, t + R$: At time $r \cdot \Delta$: for each $m \in M$:
   - If received messages $(\sigma, \Sigma_1, \mathbf{P_1}, m), \ldots, (\sigma, \Sigma_j, \mathbf{P_j}, m)$ for distinct $\mathbf{P_1}, \ldots, \mathbf{P_j}$ from distinct parties such that for $\mathbf{P} = \cup_{k \in [1,j]} \mathbf{P_j}$
     (a) For each $k \in [1, j]$, $\mathsf{as.Ver}(\mathcal{VK}(\mathbf{P_k}), \Sigma_k, m) = 1$;
     (b) $\mathsf{as.Ver}(\mathcal{VK}(P^*), \sigma, m) = 1$;
     (c) $|\mathbf{P}| \geq \min\{r - 1, t\}$; and
     (d) $\{P_i, P^*\} \nsubseteq \mathbf{P}$:
     - If $\mathsf{sent}[m] = \mathsf{false}$ and $|\{m' \in M : \mathsf{sent}[m'] = \mathsf{true}\}| < 2$:
       * Set $\mathsf{sent}[m] = \mathsf{true}$.
       * Compute $\mathsf{as.Sign}(sk_i, m) = \sigma$.
       * Compute $\mathsf{as.Comb}(\Sigma', VK', m) = \sigma'$, where $\Sigma' = (\Sigma_1, \ldots, \Sigma_j, \sigma)$ and $VK' = (\mathcal{VK}(\mathbf{P_1}), \ldots, \mathcal{VK}(\mathbf{P_j}), vk_i)$.
       * For all $k \in [1, n]$, send $(\sigma_j, \sigma', \mathbf{P} \cup \{P_i\}, m)$ to P_k with probability q/n.
       * If $\mathsf{detect}[m] = \mathsf{false}$: set $\mathsf{detect}[m] = \mathsf{true}$ and multicast $(\sigma, (), (), m)$.
   - If received message $(\sigma, \cdot, \cdot, m)$ such that $\mathsf{as.Ver}(\mathcal{VK}(P^*), \sigma, m) = 1$:
     - If $\mathsf{detect}[m] = \mathsf{false}$ and $|\{m' \in M : \mathsf{detect}[m'] = \mathsf{true}\}| < 2$:
       * Set $\mathsf{detect}[m] = \mathsf{true}$ and multicast $(\sigma, (), (), m)$.
3. At time $(t + R + 1) \cdot \Delta$:
   - If $\mathsf{sent} = \mathsf{detect}$ and $|\{m' \in M : \mathsf{sent}[m'] = \mathsf{true}\}| = 1$: set $v = m'$. Output v and terminate.

Figure 6.1: Synchronous broadcast (BC) protocol with sender P^* for $t < (1 - \epsilon) \cdot n$ and $\epsilon \in (0, 1)$ from the view of party P_i.

σ and then sets $\mathsf{detect}[m] = \mathsf{true}$ so that all parties subsequently receive it.[6]

Finally, in step 3 of the protocol in round $t + R + 1$, if the maps sent and detect are equal and there is only one value $m' \in M$ such that $\mathsf{sent}[m'] = \mathsf{true}$, then P_i outputs this value and terminates; otherwise it outputs $\perp$ and terminates. Note if there are two or more messages m such that some honest party set $\mathsf{sent}[m] = \mathsf{true}$, then these honest parties will broadcast the signer's signature on each m which all honest parties will process and thus terminate with

[6]We conjecture that the protocol without these extra messages also satisfies consistency, but the protocol as written has the same asymptotic complexity and therefore we leave it as future work.

- Input: Set of n nodes W, disjoint subsets $S_2, S_3 \subset W$, $S \subset W \setminus (S_2 \cup S_3)$, integer $q \leq n$.
- Output: The graph G.

1. Let G be the empty graph with node set W.
2. For every $u \in S$ and $v \in W$, add an edge $\{u, v\}$ to G with probability q/n.
3. Return graph G.

Figure 6.2: Description of the AddRndEdges(W, S_2, S_3, S, q) procedure.

$|\mathsf{detect}| = 2$ and output $\perp$.

Communication Complexity. Each party gossips at most two messages of size $O(\lambda + n + \ell)$ and multicasts at most two messages of size $O(\lambda + \ell)$. Since $q = \Theta(\lambda)$, each party sends an expected $O(\lambda)$ messages in each gossip step. Thus, communication complexity is overall $O(\lambda n^2 + n\lambda^2 + \ell(\lambda + n^2))$ which, when $\ell = O(\lambda)$, is $O(\lambda n^2 + \lambda^2 n)$. For $n \geq \lambda$, we have $O(\lambda n^2 + \lambda^2 n) = O(\lambda n^2)$. For $n < \lambda$, there is a trivial solution to achieve $O(\lambda n^2)$ communication complexity. Namely, one can run standard Dolev-Strong broadcast [DS83b] with multi-signatures which has communication complexity $O(n^3 + \lambda n^2 + \ell n^2)$. Since $n < \lambda$, we have $n^3 < \lambda n^2$, and since we have $\ell = O(\lambda)$, it follows that $O(n^3 + \lambda n^2 + \ell n^2) = O(\lambda n^2)$.

Theorem 6.4.1. *Let n, t be such that $t < (1 - \epsilon) \cdot n$ for some constant $\epsilon \in (0, 1)$. Then $\Pi_{\mathsf{BC}}^{t,\epsilon}$ (cf. Figure 6.1) is t-secure when run on a synchronous network and n-weakly valid when run on an asynchronous network.*

Proof. Towards the goal of proving the theorem, we begin by modeling probabilistic dissemination by the procedure AddRndEdges (cf. Figure 6.2) and prove some results used to prove t_s-security of our BC protocol. The techniques that follow are from [TLP22a], where however they use binary input instead of multivalued one.

AddRndEdges is defined over a set of nodes W that is partitioned into three disjoint subsets $S_1, S_2, S_3 \subset W$ with $S_1 = W \setminus (S_2 \cup S_3)$. The way AddRndEdges works is as follows. At the beginning, we have an empty graph G with node set W. Now given $S \subset S_1$, AddRndEdges adds the edge $\{u, v\}$ to the graph G with probability q/n for every pair of nodes $u \in S$ and $v \in W$. At the end, the resulting graph with all the added edges is output. In the context of our protocol, S will be the set of parties that send a message m at a specific round r and S_2 will be the set of parties that have not received m in a previous round. An edge from $u \in S$ to $v \in W$ represents that party u sends m to party v in round r. Our goal is to determine how many parties in S_2 receive message m for the first time during round r. For this, we define the following indicator random variables.

Definition 6.4.1. Let $G \leftarrow \mathsf{AddRndEdges}(W, S_2, S_3, S, q)$ (cf. Figure 6.2) be a graph. For all $u \in S_2$, let $Z_u \in \{0, 1\}$ with $Z_u = 1$ if and only if u has nonzero degree in G.

We find that the number of nodes in S_2 with nonzero degree in G is at least twice the number of nodes in S. This will allow us to show that messages propagate exponentially fast in our

broadcast protocol. The proof of the following lemma can be found in the appendix of [TLP22a] (proof of Lemma 1).

Lemma 6.4.2. *Let (S_1, S_2, S_3) be a partition of n nodes into (disjoint) sets with $\tau = |S_1| \leq \epsilon n/3$, $|S_2| = \epsilon n - |S_1|$, $|S_3| = n - \epsilon n$, where $\epsilon \in (0,1)$ is a constant. Let $S \subset S_1$ with $|S| \geq 2\tau/3$, and let $\{Z_u\}_{u \in S_2}$ be the random variables defined above. Then for $q \geq 15/\epsilon$, we have*

$$\Pr\left[\sum_{u \in S_2} Z_u \geq 2\tau\right] = 1 - p, \quad \textit{where} \quad p = \max\left\{\epsilon n \cdot e^{-\epsilon q/9}, \left(\frac{e}{2}\right)^{-\epsilon q/4}\right\}.$$

For our proof of t_s-security and t_a-weak validity, we define the following sets of parties w.r.t. a value $m \in V$ and a round r:

1. $S(m, r)$: honest parties P_i that set $\mathsf{sent}[m] = \mathsf{true}$ at round r.
2. $S_1(m, r)$: honest parties P_i that set $\mathsf{sent}[m] = \mathsf{true}$ by round r.
3. $S_2(m, r)$: honest parties P_i that still have $\mathsf{sent}[m] = \mathsf{false}$ in round r.

Additionally, we let S_3 be the set of malicious parties (in particular, $|S_3| = n - \epsilon n$). Before we start we our proof of security for the Core BC Protocol, we show that the number of parties that receive a message at round $\tilde{r}$ that was sent at round $r < \tilde{r}$ increases exponentially with $\tilde{r} - r$ (with overwhelming probability). The proof of the following theorem can be found in the appendix of [TLP22a] (proof of Lemma 2) with the difference that their set V is of order 2. However, this difference does not have any impact on the proof itself and therefore the same proof applies for our case of V being of order at least 2.

Lemma 6.4.3. *For a specific value $m \in V$, let r be the first round of the Core BC Protocol $\Pi_{\mathsf{BC}}^{t,\epsilon}$ where an honest party P_i sets $\mathsf{sent}[m] = \mathsf{true}$. Let $R = \lceil \log_3(\epsilon n) \rceil$, and let p be as in Lemma 6.4.2. Then we have the following bounds:*

1. *For all rounds ρ such that $r \leq \rho \leq r + R$ and $|S_1(m, \rho - 1)| \leq \epsilon n/3$, we have with probability at least $(1-p)^{\rho - r}$ that*

$$|S(m, \rho)| \geq 2/3 \cdot |S_1(m, \rho)| \quad \textit{and} \quad |S_1(m, \rho)| \geq 3^{\rho - r}.$$

2. *Let $\tilde{r} > r$ be a round such that $|S_1(m, \tilde{r} - 1)| > \epsilon n/3$. Then $|S_1(m, \tilde{r})| = \epsilon n$ with probability at least $(1 - \tilde{p})(1-p)^{\tilde{r} - r - 1}$ where $\tilde{p} = \epsilon n \cdot e^{-2\epsilon q/9}$.*

For the proofs hereafter, we let $R = \lceil \log_3(\epsilon n) \rceil$ and $q = \Theta(\lambda)$, where λ is the security parameter. Furthermore, all our statements hold with probability $1 - \mathsf{negl}(\lambda)$.

Lemma 6.4.4. *Let $t_s < (1-\epsilon) \cdot n$. Then $\Pi_{\mathsf{BC}}^{t_s,\epsilon}$ achieves t_s-consistency when run in a synchronous network.*

Proof. Let $\mathcal{M} = \{m_1, \ldots, m_k\}$ be the set of messages for which at least one honest party sets $\mathsf{sent}[m_i] \leftarrow \mathsf{true}$ during one run of the protocol. We consider three cases separately, namely when $|\mathcal{M}| = 0$, $|\mathcal{M}| = 1$ and $|\mathcal{M}| > 1$. Suppose first that $|\mathcal{M}| = 0$. Then all honest parties will never update v and consequently output $\perp$ at step 3.

Suppose $|\mathcal{M}| = 1$. We show that if an honest party P_i sets $\mathsf{sent}[m] = \mathsf{true}$ for some value $m \in V$ at some round r, then by the end of the protocol all honest parties have set $\mathsf{sent}[m] = \mathsf{true}$ with probability $1 - \mathsf{negl}(\lambda)$. We consider the following two cases.

(i) Suppose $r < t_s + 1$. For this, we distinguish the two cases $S_1(m, r) > \epsilon n/3$ and $S_1(m, r) \leq \epsilon n/3$. If $S_1(m, r) > \epsilon n/3$, then by item 2 of Lemma 6.4.3 all ϵn honest parties set $\mathsf{sent}[m] = \mathsf{true}$ by the next round with probability at least

$$(1 - \tilde{p})(1 - p)^{(r+1)-r-1} = 1 - \epsilon n \cdot e^{-2\epsilon q/9}.$$

Since $n = \mathsf{poly}(\lambda)$, this probability is $1 - \mathsf{negl}(\lambda)$. On the other hand, if $S_1(m, r) \leq \epsilon n/3$, let $r_0 := r + R - 1$. In case $S_1(m, r_0) > \epsilon n/3$, again the previous case applies. Otherwise, item 1 of Lemma 6.4.3 tells us that at round $r_0 + 1 = r + R$ we get

$$S_1(m, r + R) \geq 3^R = 3^{\lceil \log_3(\epsilon n) \rceil} \geq 3^{\log_3(\epsilon n)} = \epsilon n$$

with probability at least $(1 - p)^R$. Since $-p \geq -1$ and $R \geq 1$, Bernoulli's inequality applies and gives $(1 - p)^R \geq 1 - pR$. Since $n = \mathsf{poly}(\lambda)$ and $q = \Theta(\lambda)$, this probability is $1 - \mathsf{negl}(\lambda)$.

(ii) Suppose $r \geq t_s + 1$. Suppose an honest party P_i sets $\mathsf{sent}[m] = \mathsf{true}$ at some round $r \geq t_s + 1$. Then, P_i has received a valid multi-signature on m of degree at least $t_s + 1$ and $|\{m' \in M : \mathsf{sent}[m'] = \mathsf{true}\}| \leq 1$. In particular, an honest party P_j already set $\mathsf{sent}[m] = \mathsf{true}$ at some round $r' < t_s + 1$. Hence, former case (i) applies to honest party P_j. Ultimately, all honest parties set $\mathsf{sent}[m] = \mathsf{true}$ by the end of the protocol with probability $1 - \mathsf{negl}(\lambda)$.

Finally, suppose that $|\mathcal{M}| \geq 2$. By the above logic, $\mathsf{sent}[m]$ was set to true for an honest party P_i in round $r < t_s + 1$ for each $m \in M$. By construction of Π_{BC}, P_i will set $\mathsf{detect}[m] \leftarrow \mathsf{true}$ and multicast $(\sigma, (), (), m)$ if not already done, where σ is the signature of the designated sender P^* on message m. Thus, in the next round (given $R \geq 1$), all honest parties will receive $(\sigma, (), (), m)$, which correctly verifies. If $|\mathcal{M}| = 2$, then all honest parties will set $\mathsf{detect}[m] = \mathsf{true}$ for both messages, and consequently at step 3 all output $\perp$ by construction of the protocol. If $|\mathcal{M}| > 2$, then it follows that for the first two messages that each honest party receives, they will set $\mathsf{detect}[m] \leftarrow \mathsf{true}$, and similarly output $\perp$. □

Lemma 6.4.5. *Let $t_s < (1 - \epsilon) \cdot n$. Then $\Pi_{\mathsf{BC}}^{t_s,\epsilon}$ achieves t_s-liveness and t_s-termination when run in a synchronous network.*

Proof. This follows trivially from the fact that all parties will terminate at step 3 after some finite amount of time regardless of whether or not they change the value v they output. □

Lemma 6.4.6. *Let $t_s < (1 - \epsilon) \cdot n$. Then $\Pi_{\mathsf{BC}}^{t_s,\epsilon}$ achieves t_s-validity when run in a synchronous network.*

Proof. In order to prove t_s-validity, we show that if the sender P^* is honest and inputs $m \in V$, then all honest parties output $v = m$. This directly follows from the proof of t_s-consistency and the fact that no honest party sets $\mathsf{sent}[m] \leftarrow \mathsf{true}$ on a message not signed by P^*. After $R = \lceil \log_3(\epsilon n) \rceil$ rounds, all honest parties have set $\mathsf{sent}[m] = \mathsf{true}$ with probability $1 - \mathsf{negl}(\lambda)$ and will output $v = m$. □

Lemma 6.4.7. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{BC}}^{t_s,\epsilon}$ achieves t_a-weak validity even when run in an asynchronous network.*[7]

Proof. In order to prove t_a-weak validity, we show that if the sender P^* is honest and inputs $m = m_j \in V$, then all honest parties output either $v = m_j$ or $v = \perp$. This directly follows from the proof of t_s-validity in the synchronous case: in case an honest party does not receive a valid multisignature on m_j by the end of the protocol run, it just outputs $v = \perp$. □

This completes the proof of Theorem 6.4.1. □

6.4.2 Broadcast Extension Protocol

Here, we present our broadcast extension protocol $\Pi_{\mathsf{BC\text{-}Ext}}^{t,\epsilon}$ which, for $t < (1-\epsilon) \cdot n$, allows for broadcast with $O(n\ell/\epsilon + \lambda n^2)$ communication complexity. In particular, this implies a $O(n\ell + \lambda n^2)$ honest majority broadcast algorithm by setting $\epsilon = \frac{1}{2}$.

Let $\Pi_{\mathsf{BC}}^{t,\epsilon}$ be a broadcast protocol (e.g., the one in Section 6.4) that terminates after T time with default value $\perp_{bc}$. we explicitly specify the Core protocol in Figure 6.3 which makes use of several helper functions defined in Figure 6.4. Our protocol is very close to the dishonest majority broadcast protocol of Nayak et al. [Nay+20b] (see Figure 3 there). The only notable difference (modulo presentation differences) is that parties gossip signatures similar to Π_{BC} rather than multicasting.

$\Pi_{\mathsf{BC\text{-}Ext}}^{t,\epsilon}$ works as follows. In the first step, the sender P^* splits up its input message m^* into n blocks $D = \{D_1, \ldots D_n\}$ using the erasure coding scheme rs ($t+1$ blocks suffice to reconstruct message m^*), where block D_j corresponds to party P_j. Then P^* accumulates D into an accumulation value z using acc except that D_j is accumulated as (j, D_j). P^* then invokes the broadcast protocol $\Pi_{\mathsf{BC}}^{t,\epsilon}$ on input z; note that z is of size $O(\lambda)$. Suppose that value $z \neq \perp$ is output by each honest party; if $z = \perp$, all honest parties output $\perp$. Similar to Π_{BC} from the previous subsection, the protocol proceeds in $t + R$ rounds. Let $z \neq \perp$ be the output from Π_{BC}. Supposing that P^* is honest, once Π_{BC} terminates, P^* first multicasts a signature on message HAPPY (after setting their own variable happy to true). Then, P^* computes witnesses w_j corresponding to block D_j for $j \in [1, n]$, and sends each party their respective block and the witness w_j (step 2(a)). On receipt of their witness (which they can verify is valid), P_i multicasts their block and share (step 2(b)). Note that P_i multicasts a block at most once. P_i then waits Δ time and collects blocks from all parties. On receipt of the first message (wit, s_j, w_j) from P_j, P_i can determine whether it is valid (and was accumulated in z) via $\mathsf{acc.Ver}(ak, z, w_j, (j, s_j))$. Finally, after Δ time, if P_i has received enough valid blocks, P_i can reconstruct a message m (step 2(c)). Then, by splitting up m, re-accumulating the blocks and comparing the output with z (output by Π_{BC}), P_i can determine whether they reconstructed a message that all other parties are able to or not. If so, they set variable $\mathsf{happy} = 1$. Suppose an honest party sets $\mathsf{happy} = 1$. Then, that party splits up the message it learned (or input to $\Pi_{\mathsf{BC\text{-}Ext}}$, if it is the sender P^*) into blocks, propagates signatures via gossip, and propagates blocks and their corresponding witnesses to each party individually. Our proofs show in particular that all parties will eventually output the message, even when using gossip in step 2(a).

[7] Note that n-weak validity trivially follows.

$\Pi^{t,\epsilon}_{\text{BC-Ext}}(m^*)$

1. At time 0:
   - Set $o = \perp$ and $\mathsf{happy} = 0$.
   - Set $\mathsf{dist} = \mathsf{share} = \mathsf{recon} = \mathsf{false}$ and $\Sigma = VK = \mathbf{P} = \mathbf{D} = \perp$.
   - If $P_i = P^*$: set $o = m^*$ and $\mathsf{happy} = 1$. Compute $\mathbf{D} = \mathsf{EncBlocks}(m^*, n-t) = (D_1, \ldots, D_n)$ and $z^* = \mathsf{acc.Eval}(ak, \mathbf{D})$. Invoke protocol $\Pi^{t,\epsilon}_{\mathsf{BC}}$ with sender P^* using the input z^*.
2. At time T: If $z \neq \perp_{bc}$ is output by $\Pi^{t,\epsilon}_{\mathsf{BC}}$: for $r = 1, \ldots, t+R$:
   (a) At time $T + (2r-2)\cdot\Delta$:
      - If $\mathsf{happy} = 1$ and $\mathsf{dist} = \mathsf{false}$:
        - Set $\mathsf{dist} = \mathsf{true}$.
        - If $P_i = P^*$: multicast $(\mathsf{as.Sign}(sk_i, \mathsf{HAPPY}), \{P_i\})$. Otherwise, compute $\sigma = \mathsf{MS}(\Sigma, VK)$, then for all $j \in [1,n]$, send $(\sigma, \mathbf{P} \cup \{P_i\})$ to P_j with probability q/n.
        - Compute $(w_1, \ldots, w_n) = \mathsf{acc.Wits}(ak, z, \mathbf{D})$.
        - For all $j \in [1,n]$, send (wit, s_j, w_j) to P_j (where the j-th element of $\mathbf{D}$ is (j, s_j)).
   (b) At time $T + (2r-1)\cdot\Delta$:
      - If $\mathsf{share} = \mathsf{false}$ and received (wit, s_i, w_i) such that $\mathsf{acc.Ver}(ak, z, w, (i, s_i)) = 1$: Set $\mathsf{share} = \mathsf{true}$ and multicast (wit, s_i, w_i).
   (c) At time $T + 2r\cdot\Delta$:
      - If $\mathsf{recon} = \mathsf{false}$:
        - For $j \in [1,n]$, let (wit, s_j, w_j) be the first message $(\mathsf{wit}, \cdot, \cdot)$ received from P_j (set to $(\mathsf{wit}, \perp, \perp)$ if not received).
        - Set $S = (((1, s_1), w_1), \ldots, ((n, s_n), w_n))$.
        - Compute $M = \mathsf{Recon}(S, ak, z, t)$ and $\mathbf{D} = \mathsf{EncBlocks}(M, n-t)$.
        - If $\mathsf{acc.Eval}(ak, \mathbf{D}) = z$, $\mathsf{recon} = \mathsf{false}$, $\mathsf{HappyCheck}(r) = \mathsf{true}$ and $\mathsf{Valid}(M)$ ($\mathsf{HappyCheck}$ checks r signatures were received):
          * Set $\mathsf{happy} = 1$, $o = M$ and $\mathsf{recon} = \mathsf{true}$.
          * Invoke $\mathsf{Agg}(r)$.
3. At time $T + 2\cdot(t+R)\cdot\Delta$: Output o and terminate.

Figure 6.3: BC extension protocol with sender P^* for $t < (1-\epsilon)\cdot n$ and $\epsilon \in (0,1)$ from the view of party P_i with external validity predicate Valid.

Helper functions for $\Pi^{t,\epsilon}_{\text{BC-Ext}}$

- $\underline{\mathsf{MS}(\Sigma, VK)}$: Compute $\mathsf{as.Sign}(sk_i, \mathsf{HAPPY}) = \sigma_i$ and $\mathsf{as.Comb}(\Sigma', VK', \mathsf{HAPPY}) = \sigma'$, where $\Sigma' = (\Sigma, \sigma)$ and $VK' = (VK, vk_i)$. Output σ'.
- $\underline{\mathsf{HappyCheck}(r)}$: Output true if and only if received messages $(\Sigma^1, \mathbf{P^1}), \ldots, (\Sigma^j, \mathbf{P^j})$ for distinct $\mathbf{P^1}, \ldots, \mathbf{P^j}$ from distinct parties such that for $\mathbf{P} = \cup_{k\in[1,j]}\mathbf{P^k}$
  1. For each $k \in [1, j]$, $\mathsf{as.Ver}(\mathcal{VK}(\mathbf{P^k}), \Sigma^k, \mathsf{HAPPY}) = 1$;
  2. $|\mathbf{P}| \geq \min\{r, t+1\}$; and
  3. $P_i \notin \mathbf{P}$.
- $\underline{\mathsf{Agg}(r)}$: Consider the messages $(\Sigma_1, \mathbf{P_1}), \ldots, (\Sigma_j, \mathbf{P_j})$ that were received and lead to $\overline{\mathsf{HappyCheck}(r)}$ outputting true.
  - Set $\Sigma = \mathsf{as.Comb}(\Sigma = (\Sigma_1, \ldots, \Sigma_j), (\mathcal{VK}(\mathbf{P_1}), \ldots, \mathcal{VK}(\mathbf{P_j})), \mathsf{HAPPY})$, $VK = \cup_{k\in[1,j]}\mathcal{VK}(\mathbf{P_k})$, and $\mathbf{P} = \cup_{k\in[1,j]}\mathbf{P_k}$.
- $\underline{\mathsf{EncBlocks}(m, b)}$: Divide m into b blocks $m_1, \ldots, m_b$ each of length $\lceil \frac{\ell a}{b} \rceil$, where $\ell = \log_2 m$ and $a = \min\{i : n \leq 2^i - 1\}$. Compute $(s_1, \ldots, s_n) = \mathsf{rs.Encode}(m_1, \ldots, m_b)$. Output $((1, s_1), \ldots, (n, s_n))$.
- $\underline{\mathsf{Recon}((((1, s_1), w_1), \ldots, ((n, s_n), w_n)), ak, z, d_0)}$: For $j \in [1, n]$: If $\mathsf{acc.Ver}(ak, z, w_j, (j, s_j)) = 0$, set $s_j = \perp$. Compute $m_1, \ldots, m_b = \mathsf{rs.Decode}(s_1, \ldots, s_n)$ and $m = (m_1 \mid \cdots \mid m_b)$. Output m.

Figure 6.4: Helper functions for the BC extension protocol from the view of party P_i.

Communication Complexity. Each party participates in Π_{BC} where the sender's input is of size $O(\lambda)$[8], which, using $\Pi_{\mathsf{BC}}^{t,\epsilon}$ from Figure 6.1, requires $O(\lambda n^2)$ communication. Each party sends at most one wit message to each party at step 2(b), which costs $O(n(\ell/b + \lambda) = O(\ell/\epsilon + \lambda n)$ for each party (recall $b = n - t$). The sender multicasts an $O(\lambda)$-sized message, and all parties gossip at most one message of size $O(\lambda + n)$, which costs an expected $O(\lambda^2 + \lambda n)$ per party. Thus the protocol incurs $O(n\ell/\epsilon + \lambda n^2 + \lambda^2 n) = O(n\ell/\epsilon + \lambda n^2)$ communication.

Theorem 6.4.8. *Let n, t be such that $t < (1-\epsilon) \cdot n$ for constant $\epsilon \in (0,1)$. Then $\Pi_{\mathsf{BC\text{-}Ext}}^{t,\epsilon}$ (cf. Figures 6.3 and 6.4) is t-secure when run on a synchronous network and n-weakly valid when run on an asynchronous network.*

Proof. We provide our proof of the theorem via a sequence of lemmata, from which the result directly follows. Hereafter, we let $R = \lceil \log_3(\epsilon n) \rceil$ and $q = \Theta(\lambda)$, where λ is the security parameter. Further, all our statements hold with probability $1 - \mathsf{negl}(\lambda)$.

Lemma 6.4.9. *Let $t_s < (1-\epsilon) \cdot n$. Then $\Pi_{\mathsf{BC\text{-}Ext}}^{t_s,\epsilon}$ achieves t_s-liveness when run in a synchronous network.*

Proof. In order to prove t_s-liveness, we show that every honest party outputs a value v' and terminates. But this is clear from the protocol $\Pi_{\mathsf{BC\text{-}Ext}}^{t_s,\epsilon}$, especially step 3. □

Lemma 6.4.10. *If an honest party P_i reaches item (i) of step 2 of $\Pi_{\mathsf{BC\text{-}Ext}}^{t_s,\epsilon}$ by setting* happy $= 1$ *and* dist $=$ false *and invokes the instructions of this step with input message $m \in V$, then every honest party P_j outputs $\sigma_j = m$.*

Proof. By t_s-consistency of the Core BC Protocol $\Pi_{\mathsf{BC}}^{t_s,\epsilon}$, the output z_i of $\Pi_{\mathsf{BC}}^{t_s,\epsilon}$ is the same at every honest party. If an honest party executes item (i) of step 2 with message m, then by definition $z_i = \mathsf{acc.Eval}(ak, \mathsf{EncBlocks}(m, n-t))$. If another honest party P_j sets $\sigma_j = m'$ after initialization, then it has to satisfy $\mathsf{acc.Eval}(ak, \mathsf{EncBlocks}(m', n-t)) = z_j = z_i$. By the properties of the Reed-Solomon code, the same codewords correspond to the same message. So if $m \neq m'$, then $\mathbf{D} \neq \mathbf{D'}$. In particular, there exists a component $D_i = (i, s_i) \in \mathbf{D}$ such that $D_i \notin \mathbf{D'}$. But a witness for $D_i \notin D'$ with respect to the accumulation value $z_i = \mathsf{acc.Eval}(ak, \mathbf{D}) = \mathsf{acc.Eval}(ak, \mathbf{D'})$ exists, which happens only with probability $\mathsf{negl}(\lambda)$ by security of the accumulator. Therefore, we may assume $m = m'$ and only need to show that every other honest party P_j sets σ_j to a value.

Suppose that P_i executes item (i) of step 2 in some round r. In that case, P_i computes witnesses $(w_1, \ldots, w_n) = \mathsf{acc.Wits}(ak, z, \mathbf{D})$ and sends (wit, s_j, w_j) to each party P_j (where the j-th element of $\mathbf{D}$ is (j, s_j)). In item (ii) of step 2 of the same round r, every honest party can verify and multicast the valid (wit, s_j, w_j) to all the other parties if it has not done that already in a previous round. Note that verification of the tuples (wit, s_j, w_j) can be done safely because of the security of the accumulator. In item (iii) of step 2, every honest party gets at least $n - t_s$ correct coded values, since there are at least $n - t_s$ honest parties. In particular, every party P_j can identify the corrupted values and remove them, of which there are at most t_s. By the properties of the Reed-Solomon code, P_j with happy $= 0$ is able to recover the message m. Now, the exact same analysis as in the proof of t_s-consistency of the Core BC Protocol $\Pi_{\mathsf{BC}}^{t,\epsilon}$

[8] To tame the communication complexity, parties should disregard messages signed by a (dishonest) sender in Π_{BC} that are larger than the prescribed size $O(\lambda)$.

shows that with probability $1 - \mathsf{negl}(\lambda)$ after sufficiently many rounds in the domain $[t + R]$, every honest parties P_j has set its value $\mathsf{happy} = 1$ and $\sigma_j = m$. Note that an honest party does not set its output again in future rounds, since the multi-signature already contains its signature. Once P_j sets σ_j, it will skip item (ii) of step 2 in all future round and the value of σ_j will not be changed. Therefore, in step 3 all honest parties output m and terminate. □

Lemma 6.4.11. *Let $t_s < (1-\epsilon) \cdot n$. Then $\Pi^{t_s,\epsilon}_{\text{BC-Ext}}$ achieves t_s-validity when run in a synchronous network.*

Proof. In order to prove t_s-validity, we show that if the sender P_j is honest and inputs $m = m_j \in V$, then all honest parties output $v = m_j$. In round $r = 1$, the sender executes item (i) of step 2 with message m_j. By the previous lemma, every honest party P_i outputs $\sigma_i = m_j$ by the end of the protocol and terminates. □

Lemma 6.4.12. *Let $t_s < (1 - \epsilon) \cdot n$. Then $\Pi^{t_s,\epsilon}_{\text{BC-Ext}}$ achieves t_s-consistency when run in a synchronous network.*

Proof. In order to prove t_s-consistency, we show that every honest party outputs the same value σ'. If all honest parties output $\perp$, then they trivially output the same value $\sigma' = \perp$. Therefore, assume some honest party P_i outputs $\sigma_i = m \neq \perp$. In case P_i is the sender, t_s-validity of $\Pi^{t_s,\epsilon}_{\text{BC-Ext}}$ tells us that all honest parties output m. So assume that P_i is not the sender. If P_i sets $\sigma_i = m \neq \perp$ in item (iii) of step 2 of round $1 \leq r \leq t_s$, then P_i will execute the distribution item (i) of step 2 with message m in the next round $r + 1$. By Lemma 6.4.10, all honest parties output the same value $\sigma' = m$. On the other hand, if P_i sets $\sigma_i = m \neq \perp$ in round $r \geq t_s + 1$, then it receives a multisignature of degree at least $t_s + 1$. In particular, one of these signatures comes from an honest party $P_j \neq P_i$ that has sent its signature (with probability q/n) and executed item (i) of step 2 in some previous round $1 \leq r' \leq t_s$. Again by Lemma 6.4.10, all honest parties (including P_i) output m' and thus $m' = m$. So every honest party outputs the same $\sigma' = m$. □

Lemma 6.4.13. *Let $t_s < (1 - \epsilon) \cdot n$. Then $\Pi^{t_s,\epsilon}_{\text{BC-Ext}}$ is t_s-secure when run in a synchronous network.*

Proof. This is clear from the above lemmata. Lemma 6.4.12 gives t_s-consistency, Lemma 6.4.11 gives t_s-validity, and Lemma 6.4.9 gives t_s-liveness. □

Lemma 6.4.14. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi^{t_s,\epsilon}_{\text{BC-Ext}}$ achieves t_a-weak validity even when run in an asynchronous network.*

Proof. In order to prove t_a-weak validity, we show that if the sender P_j is honest and inputs $m = m_j \in V$, then all honest parties output either $v = m_j$ or $v = \perp$. This is clear from the proof of t_s-validity in the synchronous case and the construction of the protocol $\Pi^{t_s,\epsilon}_{\text{BC-Ext}}$. □

This completes the proof of Theorem 6.4.8. □

6.5 Multivalued Intrusion-Tolerant Consensus

In this section, we design an intrusion-tolerant Byzantine agreement protocol with $O((\ell + \lambda)n^3)$ communication complexity. We start with the construction and proceed then with a security and complexity analysis.

6.5.1 Our Design

We construct our intrusion-tolerant Byzantine agreement protocol from intrusion-tolerant *graded consensus* and a binary Byzantine agreement protocol. Graded consensus is a relaxation of Byzantine agreement/consensus where parties input a value v and output a value/grade pair (v, g) where $g \in \{0, 1, 2\}$ (other choices of the grade set are possible). Apart from validity, liveness and intrusion tolerance, graded consensus satisfies *graded consistency* which ensures that 1) the grades of all honest parties never differs by more than 1; and 2) all honest parties output the same v given they output grade $g \geq 1$ (note parties may output $(\perp, 0)$). In Appendix 6.6.2, we formally define and construct a graded consensus protocol with a high validity threshold and $O(\ell n^3)$ communication complexity. Our overall protocol is a modification Mostéfaoui and Raynal's asynchronous protocol for $t < n/3$ [MR17] which is not framed in terms of graded consensus.

Let $\Pi_{\mathsf{BA}}^{t_a,t_s}$ be a Byzantine agreement protocol with input domain $\{0, 1\}$ that is t_a-secure and t_s-valid with termination. In Appendix 6.6.4, we present a modified version of the protocol from [BKL19] with expected communication complexity $O(\lambda n^3)$ *without trusted setup* that satisfies these requirements. Let $\Pi_{\mathsf{GC}}^{t_a,t_s}$ be a t_a-secure and t_s-graded valid multivalued graded consensus protocol. We present intrusion-tolerant Byzantine agreement protocol $\Pi_{\mathsf{IT}}^{t_a,t_s}$ in Figure 6.5.

$\Pi_{\mathsf{IT}}^{t_a,t_s}(v_i)$

1. Set $\mathsf{aux}_i = \perp, \mathsf{bp} = 0$.
2. Run $\Pi_{\mathsf{GC}}^{t_a,t_s}$ using input v_i.
3. Upon receiving output (v, g) from $\Pi_{\mathsf{GC}}^{t_a,t_s}$: set $\mathsf{aux}_i = v$. If $g = 2$, set $\mathsf{bp} = 1$. Run $\Pi_{\mathsf{BA}}^{t_a,t_s}$ using input bp.
4. Upon receiving output b from $\Pi_{\mathsf{BA}}^{t_a,t_s}$: if $b = 1$, then multicast $\langle \mathsf{commit}, \mathsf{aux}_i \rangle_i$. Otherwise, multicast $\langle \mathsf{commit}, \perp \rangle_i$.
5. Upon receiving $t_s + 1$ signatures of $(\mathsf{commit}, \mathsf{aux})$ from distinct parties: multicast $t_s + 1$ such signatures, output aux and terminate.

Figure 6.5: Intrusion-tolerant multivalued Byzantine agreement protocol from the view of party P_i.

We describe the protocol from the perspective of a party P_i with initial value $v_i \in V$. First, P_i runs the multivalued graded consensus protocol $\Pi_{\mathsf{GC}}^{t_a,t_s}$ on input v_i and outputs (v, g) (step 2). Then, if $g = 2$, P_i proposes 1 to $\Pi_{\mathsf{BA}}^{t_a,t_s}$ and otherwise proposes 0 (step 3). In particular, if an honest party P_i proposes 1, then by graded consistency, all honest parties output (v, g)

from Π_{GC} with $g \in \{1, 2\}$, and so if bit 1 is decided in $\Pi_{\mathsf{BA}}^{t_a,t_s}$ then all parties can safely output v. Given this occurs, P_i then signs and multicasts the value v (received previously from the output of the graded consensus protocol) along with the signature, otherwise it multicasts $\perp$ together with a signature (step 4). In the final phase of the protocol, P_i outputs a value aux (and terminates) if it received that value with at least $t_s + 1$ valid signatures, ensuring that at least one signature on aux is from an honest party (step 5).

6.5.2 Security and Complexity Analysis

In this section, we provide a security and complexity analysis of our multivalued intrusion-tolerant Byzantine agreement protocol.

Communication Complexity. $\Pi_{\mathsf{GC}}^{t_a,t_s}$ (step 2) has a communication complexity bounded by $O(\ell n^3)$ (note it is signature-free). $\Pi_{\mathsf{BA}}^{t_a,t_s}$ (step 3) has an expected complexity of $O(\lambda n^3)$. The multicast of commit messages in steps 4 and 5 an additional complexity of $O(\lambda n^3 + \ell n^2)$ using regular signatures or $O(n^3 + (\lambda + \ell)n^2)$ using aggregate signatures. Thus, the overall expected complexity of $\Pi_{\mathsf{IT}}^{t_a,t_s}$ is given by $O((\lambda + \ell)n^3)$.

Theorem 6.5.1. *Let n, t_s, t_a be such that $0 \leq t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ and $t_a + 2 \cdot t_s < n$. Then, the Byzantine agreement protocol $\Pi_{\mathsf{IT}}^{t_a,t_s}$ (cf. Figure 6.5) is t_s-valid and t_a-secure with intrusion tolerance.*

Proof. We provide our proof of the theorem via a sequence of lemmata, from which the result directly follows.

Lemma 6.5.2. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{IT}}^{t_a,t_s}$ achieves t_s-validity with termination. It follows that $\Pi_{\mathsf{IT}}^{t_a,t_s}$ achieves t_a-validity.*

Proof. In order to prove t_s-validity with termination, we show that if every honest party's input is equal to the same value v, then every honest party outputs w and terminates. First, suppose that all honest parties receive output b from $\Pi_{\mathsf{BA}}^{t_a,t_s}$ and output v from $\Pi_{\mathsf{GC}}^{t_a,t_s}$ before terminating (lines 3 and 4, Figure 6.5). Since all parties input v into $\Pi_{\mathsf{GC}}^{t_a,t_s}$, by t_s-graded validity of $\Pi_{\mathsf{GC}}^{t_a,t_s}$ all parties eventually receive output $(v, 2)$ from $\Pi_{\mathsf{GC}}^{t_a,t_s}$ (line 3). By construction of $\Pi_{\mathsf{IT}}^{t_a,t_s}$, all honest parties then propose bp $= 1$ to $\Pi_{\mathsf{BA}}^{t_a,t_s}$ (line 3). By the t_s-validity of $\Pi_{\mathsf{BA}}^{t_a,t_s}$, all honest parties eventually output $b = 1$ from $\Pi_{\mathsf{BA}}^{t_a,t_s}$ (line 4). All $n - t_s$ honest parties P_i then multicast $\langle \mathsf{commit}, v \rangle_i$: since $n - t_s \geq t_s + 1$, all honest parties eventually deliver enough valid (commit, v) signatures (line 5). Moreover, since $t_s + 1 > t_s$, no honest party receives a valid commit message for any other $v' \neq v$.

Suppose now that some honest P_i terminates before both outputting from $\Pi_{\mathsf{GC}}^{t_a,t_s}$ and outputting b from $\Pi_{\mathsf{BA}}^{t_a,t_s}$. That is, P_i delivered $t_s + 1$ valid $\langle \mathsf{commit}, v \rangle_i$ messages from distinct parties before multicasting $t_s + 1$ signatures to all honest parties (line 5), since by the t_s-validity of $\Pi_{\mathsf{BA}}^{t_a,t_s}$, bit $b = 0$ is never output by $\Pi_{\mathsf{BA}}^{t_a,t_s}$.[9] Similarly, honest parties only multicast their individual commit message upon $\Pi_{\mathsf{GC}}^{t_a,t_s}$ outputting a value (which must be $(v, 2)$ as argued above). It follows that all honest parties output the same v and terminate either on receipt of P_i's signatures or otherwise (line 5). □

[9] Note that this argument still holds even though not every party may actually input a value v to $\Pi_{\mathsf{BA}}^{t_a,t_s}$ (thus precluding t_s-validity). This is because these executions where some parties terminate before inputting v are indistinguishable from executions where these same parties input v and then halt for an indefinite period of time. Thus, the safety property contained in t_s-validity holds.

Lemma 6.5.3. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{IT}}^{t_a,t_s}$ achieves t_a-consistency.*

Proof. Suppose that all honest parties that terminate receive output from $\Pi_{\mathsf{BA}}^{t_a,t_s}$ and $\Pi_{\mathsf{GC}}^{t_a,t_s}$ before terminating; we argue similarly to above Lemma 6.5.2 if this is not the case. Suppose that some honest party outputs $(v, 2)$ from graded consensus (line 3). Then, by t_a-graded consistency, all honest parties output (v, g) with the same v where $g \in \{1, 2\}$, which all honest parties eventually do by t_a-liveness. Consider the first honest party who outputs b from $\Pi_{\mathsf{BA}}^{t_a,t_s}$ (line 4); by t_a-agreement and t_a-liveness all honest parties output the same b. Thus, all honest parties eventually multicast $\langle \mathsf{commit}, v \rangle_i$ if $b = 1$ or $\langle \mathsf{commit}, \perp \rangle_i$ if $b = 0$ (line 4). It follows that all honest parties who terminate output the same v by a similar argument to above. Otherwise, if no party outputs $(v, 2)$, then by t_a-graded consistency no honest party proposes $\mathsf{bp} = 1$ to $\Pi_{\mathsf{BA}}^{t_a,t_s}$, and so by t_s-validity all parties output $b = 0$. Thus, as no honest party P_i multicasts $\langle \mathsf{commit}, v \rangle_i$ for $v \neq \perp$ and $t_s + 1 > t_a$ (line 4), all parties who terminate agree on output $\perp$. □

Lemma 6.5.4. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{IT}}^{t_a,t_s}$ achieves t_a-liveness and t_a-termination.*

Proof. Note that, in $\Pi_{\mathsf{IT}}^{t_a,t_s}$, t_a-liveness holds if and only if t_a-termination holds since all parties terminate directly after outputting. Suppose that all honest parties that terminate receive output from $\Pi_{\mathsf{BA}}^{t_a,t_s}$ and $\Pi_{\mathsf{GC}}^{t_a,t_s}$ before terminating; we argue similarly to above Lemma 6.5.2 if this is not the case. By the t_a-liveness of $\Pi_{\mathsf{GC}}^{t_a,t_s}$, all honest parties that output eventually output (v, g) (line 3), and by t_a-agreement and t_a-liveness of $\Pi_{\mathsf{BA}}^{t_a,t_s}$, all honest parties output the same b (line 4). All honest parties then multicast the same signed (commit, v) pair (line 4), and then since $n - t_a > t_s + 1$ and similarly to before all honest parties eventually output a value and terminate (line 5). □

Lemma 6.5.5. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{IT}}^{t_a,t_s}$ achieves t_s-intrusion tolerance.*

Proof. This follows similarly from the proof of t_s-validity with termination, Lemma 6.5.2. Suppose that some honest party P_i outputs $(v, 2)$ from the graded consensus, i.e., $\Pi_{\mathsf{GC}}^{t_a,t_s}$ (line 3). By its t_s-intrusion tolerance, v must have been input to $\Pi_{\mathsf{GC}}^{t_a,t_s}$, and thus $\Pi_{\mathsf{IT}}^{t_a,t_s}$, by an honest party. Then by t_s-consistency of $\Pi_{\mathsf{GC}}^{t_a,t_s}$, all honest parties who output will output (v, g) with $g \geq 1$ (line 3). Thus, if 1 is output by all honest parties in $\Pi_{\mathsf{BA}}^{t_a,t_s}$ (by properties of $\Pi_{\mathsf{BA}}^{t_a,t_s}$), it follows that $\mathsf{aux}_i = v$ will be output by all honest parties; otherwise, $\perp$ will be output. Thus, t_s-intrusion tolerance holds in this case. Suppose now that no honest party outputs (v, g) from $\Pi_{\mathsf{GC}}^{t_a,t_s}$ such that $g = 2$. By construction of $\Pi_{\mathsf{IT}}^{t_a,t_s}$, no honest party sets $\mathsf{bp} = 1$ (note all parties output from $\Pi_{\mathsf{GC}}^{t_a,t_s}$ by its t_s-liveness) and thus all honest parties propose $\mathsf{bp} = 0$ to $\Pi_{\mathsf{BA}}^{t_a,t_s}$. By t_s-validity of $\Pi_{\mathsf{BA}}^{t_a,t_s}$, it follows that all honest parties decide 0, and thus by construction of $\Pi_{IT}^{t_a,t_s}$ will eventually output $\perp$. That is, they will not output a value proposed by a dishonest party. □

This completes the proof of Theorem 6.5.1. □

6.6 Network-Agnostic DKG

In this section, our communication-efficient network-agnostic distributed key generation protocol $\Pi_{\mathsf{DKG}}^{t_a,t_s}$ with threshold $d = t_s + 1$. We prove it t_s-secure when run over a synchronous network and t_a-secure when run over an asynchronous network. We start with the construction and proceed then with a security and complexity analysis.

6.6.1 Our Design

We present our network-agnostic DKG protocol Π_{DKG} in Figure 6.6 which uses two helper functions that are defined in Figure 6.7. We recall public parameters $par = (\mathbb{G}, p, g, h)$ introduced in Section 6.3, where g and h are independent generators of the cyclic group $\mathbb{G}$ of prime order p. Π_{DKG} relies on the following underlying protocols:

- Π_{ADKG}: an asynchronous DKG protocol. We assume that Π_{ADKG} is (t_a, d)-secure with threshold $d = t_s + 1$ and has $O(\lambda n^3)$ communication complexity. The protocol from Das et al. [Das+22b] satisfies these requirements.
- $\Pi_{\mathsf{BC\text{-}Ext}}$: a broadcast protocol with default value $\perp_{bc}$. We assume that $\Pi_{\mathsf{BC\text{-}Ext}}$ is t_s-secure when run on a synchronous network, t_a-weakly valid on an asynchronous network, and t_s-externally valid. For a message of length ℓ, we require that $\Pi_{\mathsf{BC\text{-}Ext}}$ has communication complexity $O(\ell n + \lambda n^2)$. The protocol $\Pi_{\mathsf{BC\text{-}Ext}}$ defined in Figure 6.3 satisfies these requirements.
- Π_{IT}: a multivalued Byzantine agreement protocol with default value $\perp_{it}$. We assume that Π_{IT} is t_a-secure with intrusion tolerance and t_s-valid with termination, and has $O(\lambda n^3)$ communication complexity. The protocol Π_{IT} defined in Figure 6.5 satisfies these requirements.

We also assume the existence of a public-key encryption scheme $\mathsf{pke} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$, an accumulator acc, and a linear erasure coding scheme rs. Finally, we require two NIZK proof systems nizk_1 and nizk_2 which define the following relations:

- nizk_1: Statements X_1 and witnesses $(s_{ij}, u_{ij}) \in \mathbb{Z}_p^2$, where X_1 is the statement that $\prod_{k=0}^{t_s}(C_{ik})^{j^k} = g^{s_{ij}} h^{u_{ij}}$ and c_{ij} is an encryption of (s_{ij}, u_{ij}) under ek_j, where variables C_{ik} and c_{ij} are as defined in step 1 of Π_{DKG}.
- nizk_2: Statements X_2 and witnesses $(x_i, x_i') \in \mathbb{Z}_p^2$, where X_2 is the statement, given (public) values A and B, that $A = g^{x_i}$ and $B = g^{x_i} h^{x_i'}$.

Description of our Protocol. In the following, we give a step-by-step description of Π_{DKG} (cf. Figure 6.6).

Step 1. Let P_i be an honest party executing Π_{DKG}. P_i chooses two random polynomials f_i, f_i' of degree t_s with coefficients a_{ik} and b_{ik} in $\mathbb{Z}_p$ for $k \in [0, t_s]$. In this step, P_i will share points $(j, f_i(j))$ and $(j, f_i'(j))$ with each party P_j, $j \in [n]$, using public-key encryption scheme pke. As in Pedersen's verifiable secret sharing scheme [Ped92], P_i will also compute Pedersen commitments $C_{ik} = g^{a_{ik}} h^{b_{ik}}$ that allow parties to evaluate the polynomials in the exponents g and h together. In particular, the inclusion of polynomial f' blinds f such that values that contribute to the final secret are hidden from the adversary until after it has been decided, preventing the adversary from biasing the secret. In order for all parties to verify that all parties have received correct sharings, P_i will further compute a NIZK π_{ij} via nizk_1 for each P_j that verifies that the encrypted values under P_j's key are exactly $f_i(j)$ and $f_i'(j)$. All n parties then invoke $\Pi_{\mathsf{BC\text{-}Ext}}$ (broadcast), inputting a message to the i-th instance containing these Pedersen commitments, encryptions for all n parties and the corresponding NIZKs.

$\Pi_{\mathsf{DKG}}^{t_a,t_s}$

1. Set $\mathsf{it} = \mathsf{ready} = \mathsf{false}$. Then, sample two uniformly random polynomials $f_i, f_i' \in \mathbb{Z}_p[z]$ of degree t_s: $f_i(z) = a_{i0} + a_{i1}z + \ldots + a_{it_s}z^{t_s}$, $f_i'(z) = b_{i0} + b_{i1}z + \ldots + b_{it_s}z^{t_s}$, and let $z_i := a_{i0}$. Then do the following steps:

   (a) For all $k \in [0, t_s]$, compute commitments $C_{ik} = g^{a_{ik}}h^{b_{ik}}$.

   (b) For all $j \in [n]$, compute shares $s_{ij} = f_i(j)$, $u_{ij} = f_i'(j)$, and encrypted shares $c_{ij} = \mathsf{pke.Enc}(\mathsf{ek}_j, (s_{ij}, u_{ij}))$.

   (c) For all $j \in [n]$, compute proofs $\pi_{ij} = \mathsf{nizk}_1.\mathsf{Prove}(X_1, \omega_1 = (s_{ij}, u_{ij}))$.

   Let $M_i = \big(C_i = (C_{i0}, \ldots, C_{it_s}), c_i = (c_{i1}, \ldots, c_{in}), \pi_i = (\pi_{i1}, \ldots, \pi_{in})\big)$. Run n instances of $\Pi_{\mathsf{BC\text{-}Ext}}^{t_s,1/2}$ with senders $P_1, \ldots, P_n$ using external validity predicate Valid with input M_i in instance i.

2. At time T: Let $(M_1', \ldots, M_n')$ be the output from the n instances of $\Pi_{\mathsf{BC\text{-}Ext}}^{t_s,\frac{1}{2}}$ where M_j' was output from instance $j \in [n]$:

   (a) If $|\{j \in [1,n] : M_j' \neq \perp_{bc}\}| \geq n - t_s$, then do: Compute $(L_1, \ldots, L_n) = \mathsf{Split}(M_1', \ldots, M_n')$, $\mathsf{acc.Eval}(ak, (L_1, \ldots, L_n)) = z$, and $(w_1, \ldots, w_n) = \mathsf{acc.Wits}(ak, z, ((1, L_1), \ldots, (n, L_n)))$. Run $\Pi_{\mathsf{IT}}^{t_a,t_s}$ using input z.

   (b) Otherwise, run $\Pi_{\mathsf{IT}}^{t_a,t_s}$ using input $\perp_{dkg}$.

3. Upon receiving output z_{it} from $\Pi_{\mathsf{IT}}^{t_a,t_s}$: if $z_{it} \in \{\perp_{it}, \perp_{dkg}\}$, run Π_{ADKG}. Otherwise, set $\mathsf{it} = \mathsf{true}$.

4. Upon setting $\mathsf{it} = \mathsf{true}$: if $z = z_{it}$, for $j \in [1,n]$, send $(\mathsf{part}, L_j, w_j)$ to P_j.

5. Upon receiving $(\mathsf{part}, L_i = (c_i^*, C_i^*, Q), w_i)$: wait until $\mathsf{it} = \mathsf{true}$ (never satisfied if Π_{ADKG} is invoked). Then, if $z_{it} \neq \perp$ and $\mathsf{acc.Ver}(ak, z_{it}, w_i, (i, L_i)) = 1$, set $\mathsf{ready} = \mathsf{true}$.

6. Upon setting $\mathsf{ready} = \mathsf{true}$: let $x_i = \sum_{j \in Q} s_{ji}$ and $x_i' = \sum_{j \in Q} u_{ji}$ where s_{ji} and u_{ji} are decrypted from c_i^* using $\mathsf{pke.Dec}$. Compute $D_i = g^{x_i}$, $C_i'' = g^{x_i}h^{x_i'}$ and $\pi_i' = \mathsf{nizk}_2.\mathsf{Prove}(X_2, \omega_2 = (x_i, x_i'))$ where X_2 is defined with respect to $A = D_i$ and $B = C_i''$. Multicast $(\mathsf{recon}, D_i, \pi_j', L_j = (c_j^*, C_j^*, Q), w_j)$.

7. Upon receiving $(\mathsf{recon}, D_j, \pi_j', L_j = (c_j^*, C_j^*, Q), w_j)$ for $t_s + 1$ distinct values j with 1) $\mathsf{acc.Ver}(ak, z_{it}, w_j, (j, L_j)) = 1$ and 2) $\mathsf{nizk}_2.\mathsf{Ver}(X_2, \pi_i') = 1$ using $A = D_j$ and $B = C_j^*$:

   (a) Wait until $\mathsf{ready} = \mathsf{true}$.

   (b) Consider $g^{F(x)}$, where $F(x)$ is the polynomial defined by Lagrange interpolation of the $t_s + 1$ distinct values $D_j = g^{x_j} = g^{F(j)}$ in the exponent. Compute $y = g^{F(0)}$. For $j \in [1,n]$, compute $\mathsf{pk}_j = g^{F(j)}$. Output $(x_i, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), y)$ and terminate.

Figure 6.6: DKG protocol with threshold $d = t_s + 1$ from the view of party P_i. T denotes the time taken by $\Pi_{\mathsf{BC\text{-}Ext}}^{t_s,\frac{1}{2}}$ to terminate when run in synchrony.

Helper functions for $\Pi_{\mathsf{DKG}}^{t_a,t_s}$

- $\mathsf{Valid}(M_j)$: Return true if and only if the message M_j can be parsed as $M_j = \big(C_i = (C_{j0}, \ldots, C_{jt_s}), c_j = (c_{j1}, \ldots, c_{jn}), \pi_j = (\pi_{j1}, \ldots, \pi_{jn})\big)$ and, for all $k \in [n]$, $\mathsf{nizk}_1.\mathsf{Ver}(X_1, \pi_{jk}) = 1$.

- $\mathsf{Split}(M_1, \ldots, M_n)$: Let $Q = [q_1, \ldots, q_s] \subseteq [n]$ be the maximal set such that $M_j \neq \perp$. Then, for $j \in [1, n]$: let $c_j^* = (c_{q_1 j}, \ldots, c_{q_s j})$ and $C_j^* = \prod_{q \in Q} C_{j,q}^*$ where $C_{j,q}^* = \prod_{k=0}^{t_s} (C_{mk})^{j^k}$ for each $q \in Q$, and then let $L_j = (c_j^*, C_j^*, Q)$. Finally, output $(L_1, \ldots, L_n)$.

Figure 6.7: DKG helper functions from the view of party P_i.

Steps 2 and 3. If the network is synchronous, then by t_s-security of $\Pi_{\mathsf{BC\text{-}Ext}}$ and since at least $n - t_s$ honest parties broadcast, all parties will agree on the same set of values of size $\geq n - t_s$ once all instances of $\Pi_{\mathsf{BC\text{-}Ext}}$ terminate at the same time T. By t_s-external validity of $\Pi_{\mathsf{BC\text{-}Ext}}$, only messages that are Valid (cf. Figure 6.7) – namely, those which are well-formed and contain n valid NIZKs – can be output. Note in asynchrony that $\Pi_{\mathsf{BC\text{-}Ext}}$ does not satisfy consistency, so honest parties could output different messages. To resolve this, it would be natural for parties to execute consensus on the output of $\Pi_{\mathsf{BC\text{-}Ext}}$ that ensures t_s-validity in synchrony and t_a-security in asynchrony. However, not all parties may output $n - t_s$ values from $\Pi_{\mathsf{BC\text{-}Ext}}$, so parties require a mechanism to 'abort' if not enough values are obtained from consensus.

We use intrusion-tolerant consensus Π_{IT} to efficiently solve this problem. Rather than proposing the entire $O(\lambda n^2)$-sized output of $\Pi_{\mathsf{BC\text{-}Ext}}$ to consensus, P_i instead proposes an accumulated value z to Π_{IT}. Intuitively, z accumulates n values (one per party) each of size $O(\lambda n)$ corresponding to the information that each party 'needs' to eventually reconstruct their secret share and the common public key; we describe these values further below. If an honest party does not output enough values from $\Pi_{\mathsf{BC\text{-}Ext}}$, they instead propose $\perp_{dkg}$ to Π_{IT}. Π_{IT} guarantees that a decided value is either one proposed by an honest party or $\perp$. Consequently, if Π_{IT} outputs $v \in \{\perp, \perp_{dkg}\}$, all honest parties fallback to Π_{ADKG}. This will not occur in synchrony and may (not) occur in asynchrony. Otherwise, all honest parties output the same accumulated value z.

Steps 4 and 5. If $z \notin \{\perp, \perp_{dkg}\}$ is decided by Π_{IT}, then z must have been proposed by an honest party, say P_j. Assuming this is true, P_j (plus any other honest party that output z) sends each party their 'value' accumulated in z alongside a proof of membership. Party P_i obtains their value L_i this way, where L_i is computed using Split (cf. Figure 6.7). More precisely, L_i contains:

- The same Q for all n parties, corresponding to the 'qualified' set of parties of size $\geq n - t_s$ from which P_j received values from $\Pi_{\mathsf{BC\text{-}Ext}}$;
- $|Q|$ ciphertexts encrypting $f_q(i)$ and $f'_q(i)$ to P_i for all $q \in Q$; and
- Commitment $C_j^* = g^{\sum_{q \in Q} f_q(i)} h^{\sum_{q \in Q} f'_q(i)}$.

These messages allow each party to reconstruct a sharing of a secret $\sum_{q \in Q} f_q(0)$. After deciding

z from Π_{IT}, P_j sends the relevant part message to all parties, which parties verify is correct using algorithm acc.Ver.

Steps 6 and 7. At this point, P_i has received a valid message of the form (part, $L_i = (c_i^*, C_i^*, Q)$, w_i). By decrypting values in c_i^*, P_i can deduce its own secret share $x_i = \sum_{j \in Q} f_j(i)$ but not necessarily the corresponding public shares $g^{F(1)}, \ldots, g^{F(n)}$ and public key $g^{F(0)}$. Thus, parties will collaborate to compute g^x by reconstructing the polynomial $F(\cdot) = \sum_{j \in Q} f_j(\cdot)$ in the exponent of g. To this end, parties will reveal their share g^{x_i} and then compute a proof with nizk_2 that shows that it is consistent with the sharings of polynomials $f_j(\cdot)$ and $f'_j(\cdot)$ in step 1 of the protocol (which were previously hidden). More precisely, P_i computes $D_i = g^{x_i}$, $x'_i = \sum_{j \in Q} f'_j(i)$ (by decryption of c_i^*), $C_i'' = g^{x_i} h^{x'_i}$ and $\pi'_j = \mathsf{nizk}_2.\mathsf{Prove}(X_2, (x_i, x'_i))$. Then, P_i multicasts a reconstruction message recon containing D_i, the proof π'_j and P_i's value L_i alongside w_i, the proof of inclusion in z.

On receipt of a recon message from P_j, P_i can verify that (1) L_j was accumulated in z (using acc.Ver), and (2) the NIZK π'_j is correct and, in particular, is consistent with the value $C_i^* = g^{x_i^*} h^{x_i'^*}$ contained in L_j. Because these checks pass, the value C_i^* must be of the form $g^{x_i} h^{x'_i}$ computed by a honest party that output z from Π_{IT}, and thus the value D_j contained in the recon message must be of the form $g^{\sum_{k \in Q} f_k(j)}$, i.e., it must be a valid share. When P_i receives $t_s + 1$ such values, P_i evaluates $F(0)$ in the exponent of g to derive public key g^x and $F(j)$ for $j \in [1, n]$ to derive the n public shares. At this point, P_i terminates.

6.6.2 Security and Complexity Analysis

In this section, we provide a security and complexity analysis of our network-agnostic DKG protocol.

Communication Complexity. At step 1, each party invokes secure broadcast with $O(\lambda n)$-sized input (assuming NIZKs are size $O(\lambda)$, each which costs $O(n\ell + \lambda n^2)$, so this step incurs $O(\lambda n^3)$ overhead. Apart from using generic NIZKs, one can instantiate nizk_1 with $O(\lambda)$-sized proofs in a suitable Paillier group under the decisional composite residuosity assumption [Can+20]. At step 2, Π_{IT} takes $O(\lambda n^3)$ communication. If parties invoke Π_{ADKG}, then steps 4 to 7 are ignored, and Π_{ADKG} costs $O(\lambda n^3)$ itself. At step 4, $O(n)$ parties send n part messages, each of size $O(\lambda n)$, so this incurs $O(\lambda n^3)$ overhead. At step 5, $O(n)$ parties multicast a recon message of size $O(\lambda n)$, incurring $O(\lambda n^3)$ overhead, again assuming nizk_2 has $O(\lambda)$-sized proofs. nizk_2 can be instantiated using the efficient NIZK used in [Shr+21b] in the random oracle model in any cryptographic group $\mathbb{G}$. Thus, Π_{DKG} has a communication complexity of $O(\lambda n^3)$.

Theorem 6.6.1. *Let n, t_s, t_a be such that $0 \leq t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ and $t_a + 2 \cdot t_s < n$, and let $d = t_s + 1$. Assuming a plain PKI, ROM and a CRS, the distributed key generation protocol $\Pi_{\mathsf{DKG}}^{t_a, t_s}$ (cf. Figures 6.6 and 6.7) is (t_s, d)-secure when run on a synchronous network and (t_a, d)-secure when run on an asynchronous network.*

Proof. We provide our proof of the theorem via a sequence of lemmata, from which the result directly follows.

Lemma 6.6.2. *Let $0 \leq t_a < n/3 \leq t_s < n/2$, $t_a + 2 \cdot t_s < n$ and $d = t_s + 1$. Then distributed key generation protocol $\Pi_{\mathsf{DKG}}^{t_a, t_s}$ (cf. Figure 6.6) achieves (t_s, d)-correctness and t_s-consistency*

when run in a synchronous network and (t_a, d)-correctness and t_a-consistency when run in an asynchronous network.

Proof. First, suppose that at most t_s parties are corrupted and that the network is synchronous. Note that all parties terminate the n instances of $\Pi_{\mathsf{BC\text{-}Ext}}$ at time T (t_s-liveness) with the same values $(M'_1, \ldots, M'_n)$ (t_s-consistency) at step 2. By t_s-validity and t_s-external validity of $\Pi_{\mathsf{BC\text{-}Ext}}$ and since at least $n - t_s$ parties are honest, at least $n - t_s$ instances of $\Pi_{\mathsf{BC\text{-}Ext}}$ terminate with valid input from honest parties. Thus, all honest parties satisfy the condition at step 2(a) and invoke Split with the same input. Since acc.Eval and acc.Wits are deterministic, all honest parties invoke $\Pi_{\mathsf{IT}}^{t_a,t_s}$ with the same input z. Thus by t_s-validity of Π_{IT} all honest parties output z. By n-external validity of $\Pi_{\mathsf{BC\text{-}Ext}}$, each value $M'_i \neq \perp_{bc}$ output by instance i is of the form $(C_i = (C_{i0}, \ldots, C_{it_s}), c_i = (c_{i1}, \ldots, c_{in}), \pi_i = (\pi_{i1}, \ldots, \pi_{in}))$. By the completeness and simulation-soundness of nizk_1, each c_{ij} is an encryption of $(s_{ij}, u_{ij}) = (f_i(j), f'_i(j))$ under ek_j for polynomials f_i, f'_i defined by the values in C_i in the exponent of g, h. Note that Split is defined such that each message L_j contains:

- The same set of qualified parties Q;
- c^*_j, i.e., encryptions of $f_q(j)$ under pk_j for $q \in Q$; and
- $C^*_j = \prod_{q \in Q} C^*_{j,q}$ where $C^*_{j,q}$ is $f_q(j)$ and $f'_q(j)$ evaluated in the exponent of g and h, respectively. Thus, C^*_j is $\sum_{q \in Q} f'_q(j), \sum_{q \in Q} f'_q(j)$ evaluated in the exponent of g, h.

By construction of steps 4 and 5 and the security of acc, all honest parties eventually reach step 6 on receipt of their valid part message derived from the output of the n instances of $\Pi_{\mathsf{BC\text{-}Ext}}$ which all parties agree on. Each party then multicasts a recon message. Similarly to the above, by the security and correctness of acc and nizk_2, all honest parties eventually reach step 7. By construction, each honest party performs Lagrange interpolation in the exponent of g with respect to $t_s + 1$ valid shares with respect to the polynomial $F(\cdot) = \sum_{q \in Q} f_q(\cdot)$. Now, t_s-consistency follows since all honest parties evaluate $F(\cdot)$ at 0 in the exponent to derive the same y and at $[1, n]$ to derive the same sequence sequence $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$. (t_s, d)-correctness follows where the polynomial F in the definition of DKG (Definition 6.3.11) is as described above.

Suppose now that the network is asynchronous and at most t_a parties are corrupted. By n-external validity of $\Pi_{\mathsf{BC\text{-}Ext}}$, all non-bottom messages that honest parties output at the beginning of step 2 satisfy Valid(), and by construction all parties then invoke Π_{IT} with some input.

- Suppose that honest party P_i outputs $v \in \{\perp_{it}, \perp_{dkg}\}$ from Π_{IT}. Then all honest parties eventually (t_a-consistency) output the same value v (t_a-validity). It follows that no honest party sets ready = true. Since all honest parties thus invoke Π_{ADKG}, (t_a, d)-correctness and t_a-consistency follows from the (t_a, d)-security of Π_{ADKG}.
- Otherwise, by t_a-intrusion tolerance (and t_a-security) of Π_{IT}, all parties eventually output the same value z_{it} which is such that z_{it} was proposed by an honest party, say P_i. Similarly to the synchronous case, P_i must have output at least $n - t_s$ well-formed values M'_i at step 2, and then at step 4 sent valid part messages to all parties. Thus, all honest parties eventually set ready = true. Since there are at least $t_s + 1$ honest parties and by similar arguments to the synchronous case, it follows that all parties eventually reach step 7, from which the two claimed properties similarly follow.

We note also that honest parties can terminate upon generating output since they do not send any new messages thereafter. □

Lemma 6.6.3. *Let $0 \leq t_a < n/3 \leq t_s < n/2$, $t_a + 2 \cdot t_s < n$. Then the distributed key generation protocol $\Pi_{\mathsf{DKG}}^{t_a,t_s}$ (cf. Figure 6.6) achieves t_s-secrecy and t_s-uniformity when run in a synchronous network and t_a-secrecy and t_a-uniformity when run on an asynchronous network.*

Proof. We first consider secrecy. We follow the same high-level strategy used in previous work [Gen+99; Gen+07; Shr+21b]. We construct a simulator S which takes as input a public key y, and the set of initially corrupted parties by adversary A (recall we consider static corruptions), which w.l.o.g. we write as $B = \{P_1, \ldots P_t\}$ with $t \leq t_s$. A controls the network and co-ordinates the actions of parties in B. To prove t_s-secrecy, we show that S can simulate interaction with A such that, conditioned on the output public key being y, the view of the run from A's perspective is indistinguishable from one where the specification of Π_{DKG} is exactly executed. We present the simulator S in Figure 6.8.

We first assume synchrony with at most t_s corruptions. Note that, as argued in Lemma 6.6.2, all honest parties in Π_{DKG} eventually output the same value z from Π_{IT}, and thus no honest party invokes Π_{ADKG}. Then, since S is specified to perfectly simulate steps 1 to 5 of Π_{DKG} (step 1 in Figure 6.8), A's view is identically distributed to that of a run of Π_{DKG} until the condition at step 2 in Figure 6.8 is satisfied. The simulation strategy after this point is essentially that of the simulator in Figure 10 of [Shr+21b]. The goal is to 'hit' (in the language of [Gen+99]) the input public key y by modifying the contribution of one or more simulated parties to the final secret from the view of A. Recall that at step 6 of Π_{DKG}, a party P_k's share of the final secret is computed as $\sum_{j \in Q} s_{jk}$ where each value s_{jk} corresponds to (the y-coordinate of) a point on a polynomial $f_j(\cdot)$ and is derived by decryption using dk_k. In particular, A knows, for each $j \in Q$, at most t_s points on $f_j(\cdot)$, since by the security of pke, except with neglgible probability, the other encryptions give A any information about their contents. Moreover, the simulator cannot 'lie' about these points since A decrypted cipertexts to learn this information. Thus, S's strategy is to modify the values $(\mathsf{recon}, \ldots)$ multicast by honest parties at line 6 of Π_{DKG} to force the public key to be $y = g^x$.

Consider the first party P_i to reach step 6 of Π_{DKG} with $\mathsf{ready} = \mathsf{true}$ (step 2 in Figure 6.8). By t_s-intrusion tolerance of Π_{IT}, there exists an honest party P_j that correctly executes step 4 of Π_{DKG}. Note that the simulator has the state of the $\geq n - t_s$ honest parties and thus can reconstruct all polynomials $f_k(\cdot)$ where $k \in Q$ (in particular using information that P_j sent in part messages as written in Figure 6.8). To 'hit' $y = g^x$, the simulator needs to send recon messages consistent with a polynomial $F(\cdot)$ such that $F(0) = x$. To this end, S reconstructs such a polynomial $F(\cdot)$ in the exponent of g by interpolating t points from the 'real' secret plus the point $(0, y)$. Note that at the end of step 6, each party $P_j \in G$ sends a message of the form $M_j = (\mathsf{recon}, D_j, \pi'_j, L_j = (c^*_j, C^*_j, Q), w_j)$, where L_j and w_j are contained in part messages sent at step 4 and (due to the security of accumulator acc and Π_{IT}) cannot be changed in the simulation. For each $P_j \in G$, we therefore set $D_j = g^{F(j)}$ and use the zero-knowledge property of nizk_2 to simulate a proof that is consistent with L_j. More precisely, S simulates a proof for nizk_2 that proves knowledge of $(F(j), x^*)$ such that $D_j = g^{F(j)}$ and $C^*_j = g^{F(j)} h^{x^*}$ for some x^*. By the same argument in [Shr+21b], recon messages are correctly distributed and moreover verification passes at step 7 for A's corrupted parties, from which secrecy follows.

Simulator $S(y, B)$ for $\Pi_{\mathsf{DKG}}^{t_a,t_s}$

1. Simulate for G by locally executing steps 1 to 5 of Π_{DKG} (cf. Figure 6.6) until *any* party $P_i \in G$ outputs a value $v \in \{\perp_{it}, \perp_{dkg}\}$ from $\Pi_{\mathsf{IT}}^{t_a,t_s}$. In this case, begin executing the simulator for Π_{ADKG} with input (y, B) (while continuing to simulate steps 1 to 5 of Π_{DKG}) and ignore steps 2 to 4 below; when Π_{ADKG} produces output o for $P_i \in G$, output o and terminate.

2. Upon the *first* party $P_i \in G$ reaching step 6 of Π_{DKG} with ready = true:

   (a) Let $P_j \in G$ be a party that executed step 4 in Figure 6.6 with it = bc = true and $z = z_{it}$. Consider the set of messages $\mathcal{M}$ of the form $(\mathsf{part}, L_k = (c_k^*, C_k^*, Q), w_k)$ that it sent to P_k for each $k \in [1, n]$.

   (b) For $q \in Q$, compute polynomials $f_q(y)$ from $\mathcal{M}$ by Lagrange interpolation of $t_s + 1$ points (s_{qk}, k) obtained by decrypting $t_s + 1$ ciphertexts c_{qk} using simulated secret keys dk_k (where $P_k \in G$).

   (c) For $k \in [1, t_s]$, compute $x_j = \sum_{j \in Q} f_j(k)$ using polynomials $f_j(y)$ derived above. Let $g^{F(y)}$ be the exponential w.r.t. g of the polynomial defined by Lagrange interpolation in the exponent of the $t_s + 1$ points $\{(0, y), (1, g^{x_1}), \ldots, (t_s, g^{t_s})\}$.

   (d) For $k \in [t + 1, n]$, compute $D_k = g^{F(k)}$, and compute C_k'' as in step 6 in Figure 6.6. Run the nizk_1 simulator with respect to (D_k, C_k'') which outputs π_k'. Compute $M_k = (\mathsf{recon}, D_k, \pi_k', L_k, w_k)$ where (L_k, w_k) is derived from $(\mathsf{part}, L_k, w_k) \in \mathcal{M}$.

3. Upon party $P_i \in G$ reaching step 6 in Figure 6.6 with ready = true: instead of executing step 6, simulate P_i multicasting M_i for M_i defined above.

4. Upon party $P_i \in G$ reaching step 7 in Figure 6.6 simulate locally.

Figure 6.8: Simulator for $\Pi_{\mathsf{DKG}}^{t_a,t_s}$ (cf. Figure 6.6) from the view of the simulator S interacting with adversary A, where $B = \{P_1, \ldots, P_t\}$ is the set of parties that A initially corrupts, where $t \leq t_s$ (resp. $t \leq t_a$) in synchrony (resp. asynchrony), and $G = \{P_{t+1}, \ldots, P_n\}$ (without loss of generality). We refer to a party P_i's local variables from Figure 6.6 directly when clear from context.

We now consider t_s-uniformity, where the aim is to show, for an a priori fixed $y' \in \mathbb{G}$, the probability that the protocol run outputs y' is negligibly close to $1/p$. By t_s-correctness and t_s-consistency, all parties output a share of the secret $x = \sum_{j \in Q} x_q$, where $x_j = f_j(0)$ sampled uniformly in step 1. Note that the final secret is build from the sharings of $|Q| \geq n - t_s$ parties. Since $n - 2t_s \geq 1$, at least one honest party's contribution is included in x, and as argued the adversary has no informatione except with negligible probability about the contributions of honest parties until after Q has been fixed. Uniformity then follows from the fact that $f_j(0)$ is uniformly sampled. Suppose now that the network is asynchronous and there are at most t_a corruptions. Suppose that one honest party outputs $z_{it} \notin \{\perp_{dkg}, \perp_{it}\}$. Then, since Π_{IT} is

t_a-secure with intrusion tolerance, all honest parties will eventually output z_{it}, where z_{it} was proposed by an honest party. Since we did not use the synchrony of the network in the proof follows from above in this case. Otherwise, by t_a-security, all honest parties will invoke Π_{ADKG}. The result then follows from the (t_a, d)-security of Π_{ADKG}. □

This completes the proof of Theorem 6.6.1. □

Corruption Thresholds. Our construction shows that $t_a + 2 \cdot t_s < n$ corruptions are sufficient to ensure (t_s, d)-security in synchrony and (t_a, d)-security in asynchrony for $d = t_s + 1$. We note that it is also *necessary*:

Lemma 6.6.4. *Let n, t_a, t_s be such that $t_a + 2 \cdot t_s \geq n$. If DKG protocol Π is t_s-uniform in a synchronous network, then it cannot also be t_a-consistent in an asynchronous network.*

Proof. Our proof (sketch) is very similar to that of [BKL19] for Byzantine agreement, except that we rely on t_s-uniformity instead of t_s-validity to reach a contradiction. Let $t_a + 2 \cdot t_s = n$. Partition the n parties into sets S_0, S_1, S_a with $|S_0| = |S_1| = t_s$ and $|S_a| = t_a$. Consider an experiment E where communication between S_0 and S_1 is blocked by the adversary but all messages are otherwise delivered in Δ time, and two virtual copies of S_a, namely S_a^0 and S_a^1, exist that interact only with parties in $S_0 \cup S_a^0$ and $S_1 \cup S_a^1$ respectively. We construct two executions:

1. In execution 1, the network is synchronous, and parties in S_1 are corrupted and abort immediately.

2. In execution 2, the network is asynchronous, parties in S_a are corrupted and execute two independent runs of the protocol as S_a^0 and S_a^1, and all communication between S_0 and S_1 is delayed indefinitely.

In execution 1, the view of honest parties $S_0 \cup S_a^0$ is distributed identically to the views of $S_0 \cup S_a^0$ in E. Then t_s-uniformity guarantees that all parties in S_0 output a uniformly random secret. Similarly, all parties in S_1 output a secret which is uniform, and since S_0 and S_1 are disjoint, they must be independent. In execution 2, the view of honest parties $S_0 \cup S_1$ is distributed identically to $S_0 \cup S_1$ in E. But this violates t_a-consistency because, except with negligible probability, the keys output by S_0 and S_1 are different (since they are independent and uniform). □

Appendix 6A: Graded Consensus from MV-Broadcast

We introduce two primitives that we use to build intrusion-tolerant Byzantine agreement in Section 6.5, both of which are weaker primitives and thus do not require additional assumptions in asynchrony to solve (i.e., coin flipping). Looking forward, neither primitive guarantees termination itself but will terminate when used in our Byzantine agreement protocol. We note that for our MV-broadcast and graded consensus protocols in this section we assume that at most one message per 'type' is accepted by a party (which is trivial to implement). This step is taken in order to ensure security of the protocols and bound their communication complexity.

6.6.3 Towards Intrusion Tolerance: MV-Broadcast

The first primitive is MV-broadcast which was defined in [MR17] for the asynchronous setting. The goal of MV-broadcast is to 'filter' messages for consensus: parties input a value v and output a set S such that each value inside was either MV-broadcasted by an honest party (validity 1) or is a default value. It guarantees a limited form of agreement on values (validity 2 and inclusion). For MV-broadcast, we consider default value $\perp_{mv} \notin V$.

Definition 6.6.1 (Multivalued Broadcast [MR17])**.** Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i begins holding input $v_i \in V$.

- ***Validity 1.*** Π achieves *t-validity 1* if whenever at most t parties are corrupted, if an honest party outputs a set S such that $v \in S$ and $v \neq \perp_{mv}$, then v was input by an honest party.
- ***Validity 2.*** Π achieves *t-validity 2* if the following holds whenever at most t parties are corrupted: if every honest party's input is equal to the same value v, then no honest party outputs a set containing $\perp_{mv}$.
- ***Inclusion.*** Π achieves *t-inclusion* if the following holds whenever at most t parties are corrupted: if honest P_i and P_j output sets S_i and S_j respectively, then $S_i = \{w\} \implies w \in S_j$ (note $w = \perp_{mv}$ is possible).
- ***Liveness.*** Π achieves *t-liveness* if whenever at most t parties are corrupted, every honest party outputs a set S where each $v \in S$ is such that $v \in V \cup \{\perp_{mv}\}$.
- ***Validity with liveness.*** Π achieves *t-validity with liveness* if the following holds whenever at most t parties are corrupted: if every honest party's input is equal to the same value v, every honest party outputs the set $S = \{v\}$.

In [MR17], validity 1 and validity 2 are denoted as justification and obligation, respectively. We additionally introduce the notion of validity with liveness which our protocol will satisfy with t_s corruptions in synchrony. We present our MV-broadcast construction in Figure 6.9. $\Pi_{\mathsf{MV}}^{t_a,t_s}$ works as follows.

Consider honest party P_i with input value $v_i \in V$. For each $v \in V$, P_i manages tracks two local initially empty sets $M_1(v)$ and $M_2(v)$. There are two types of messages multicast by P_i, $(\mathsf{mv1}, v)$ and $(\mathsf{mv2}, v)$ for $v \in V$. The distinction between $\mathsf{mv1}$ and $\mathsf{mv2}$ is simply to multicast $(\mathsf{mv2}, v)$ as soon as enough $(\mathsf{mv1}, v)$ messages (on the same value v) were received, which is tracked via the set $M_1(v)$. By definition, $M_1(v)$ is the set of parties from which P_i

$\Pi_{\mathsf{MV}}^{t_a,t_s}(v_i)$

1. Set output = ready = false and $M_2(v) = \emptyset$ for all $v \in V$.
2. For every $v \in V$, let $M_1(v)$ be the (initially empty) set of parties from which P_i has received $(\mathsf{mv1}, v)$.
3. Multicast $(\mathsf{mv1}, v_i)$.
4. Upon receiving $(\mathsf{mv1}, v)$ messages on the same value v (possibly $\perp_{mv}$) from $n - t_s$ distinct parties: if ready = false, set ready = true and multicast $(\mathsf{mv2}, v)$.
5. Upon receiving $(\mathsf{mv1}, v)$ messages on the same value v from $t_s + 1$ distinct parties: if $(\mathsf{mv1}, v)$ not yet multicast, multicast $(\mathsf{mv1}, v)$.
6. Upon receiving $(\mathsf{mv1}, v_{max})$ for v_{max} such that $v_{max} = \arg\max_{v' \in V} |M_1(v')|$: if $|\cup_{v' \in V} M_1(v')| - |M_1(v_{max})| \geq t_s + 1$ and $(\mathsf{mv1}, \perp_{mv})$ not yet multicast, multicast $(\mathsf{mv1}, \perp_{mv})$.
7. Upon having $|\cup_{v \in V} M_2(v)| \geq n - t_s$: if output = false, set output = true and output $\{v : |M_2(v)| > 0\}$.

Updating $M_2(v)$:

(i) Upon receiving $(\mathsf{mv2}, v)$ from P_j: if previously received messages $(\mathsf{mv1}, v)$ on the same value from $n - t_s$ distinct parties, set $M_2(v) = M_2(v) \cup \{j\}$.

(ii) Upon receiving $(\mathsf{mv1}, v)$ messages on the same value v from $n - t_s$ distinct parties: set $M_2(v) = M_2(v) \cup \{j : P_j$ previously sent $(\mathsf{mv2}, v)\}$.

Figure 6.9: MV-broadcast from the view of party P_i.

has received a $(\mathsf{mv1}, v)$ message. On the other hand, $M_2(v)$ keeps track of whether value v is validated (was input by an honest party) or not. After variable initialisation, P_i multicasts $(\mathsf{mv1}, v_i)$ (step 3). For a given v, on first receipt of $(\mathsf{mv1}, v)$ from $t_s + 1$ parties, P_i multicasts it (step 5). On first receipt of $(\mathsf{mv1}, v)$ from $n - t_s$ parties for *any* v (step 4), P_i sets ready = true and multicasts $(\mathsf{mv2}, v)$, i.e., P_i champions v for output. At this time, $M_2(v)$ is also updated (step (i)), If too many values were received, i.e., $|\cup_{v' \in V} M_1(v')| - |M_1(v_{max})| \geq t_s + 1$ where $v_{max} = \arg\max_{v' \in V} |M_1(v')|$, then P_i signals this by multicasting $(\mathsf{mv1}, \perp_{mv})$ (step 6). At any time, P_i updates the set $M_2(v)$ as $M_2(v) = M_2(v) \cup \{j\}$ after it receives proposal $(\mathsf{mv2}, v)$ by party P_j whenever P_i has received the message $(\mathsf{mv1}, v)$ from $\geq n - t_s$ distinct parties. Finally, when P_i first receives $n - t_s$ $(\mathsf{mv2}, w)$ messages where $n - t_s$ $(\mathsf{mv1}, w)$ messages were also received (on possibly different values w), captured by the predicate $|\cup_{v \in V} M_2(v)| \geq n - t_s$ in step 7, P_i outputs $\{v : |M_2(v)| > 0\}$.

Communication Complexity. We argue in the following that the communication complexity of our MV-broadcast protocol Π_{MV} is bounded by $O(n^3)$. Having a look at Figure 6.9, we see that every honest party multicasts at most one message of the form $(\mathsf{mv2}, -)$ (line 4). Furthermore,

a message of the form $(\mathsf{mv1}, v)$ for some $v \neq \perp_{mv}$ is multicast if it was received from at least $t_s + 1$ distinct parties, ensuring that the message had to come from at least one honest party (line 5). In particular, an honest party only $\mathsf{mv1}$-multicasts at most $n - t_s$ times. Moreover, every honest party multicasts at most one message of the form $(\mathsf{mv1}, \perp_{mv})$ (line 6). As a result, the overall communication complexity of Π_{MV} is bounded by $O(n^3)$.

Theorem 6.6.5. *Let n, t_s, t_a be such that $0 \leq t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ and $t_a + 2 \cdot t_s < n$. Then MV-broadcast protocol $\Pi_{\mathsf{MV}}^{t_a,t_s}$ (cf. Figure 6.9) satisfies t_s-validity 1, t_s-validity 2, t_s-valid with liveness, t_a-inclusion and t_a-liveness.*

We prove this by proving the following lemmas.

Lemma 6.6.6. *Let $t_s < n/2$. Then $\Pi_{\mathsf{MV}}^{t_a,t_s}$ achieves t_s-validity 1.*

Proof. Recall that a party P_i only outputs values v such that $|M_2(v)| > 0$ (line 7, Figure 6.9). In order to prove t_s-validity 1, we show that no honest party P_i adds a value $v \in V$ to $M_2(v)$ such that $(\mathsf{mv1}, v)$ was multicast only by dishonest parties. For this, let there be t_s dishonest parties that multicast $(\mathsf{mv1}, v)$ such that $(\mathsf{mv1}, v)$ is not multicast by any honest party. Since $t_s + 1 > t_s$, no honest party echoes the value v and thus v can be received only from the dishonest parties (line 5). As a consequence, no honest party P_i receives $(\mathsf{mv1}, v)$ from $n - t_s > t_s$ distinct parties and so $M_2(v)$ is never populated by P_i (line 'Updating $M_2(v)$' (i) and (ii)). □

Lemma 6.6.7. *Let $t_s < n/2$. Then $\Pi_{\mathsf{MV}}^{t_a,t_s}$ achieves t_s-validity with liveness.*

Proof. In order to prove t_s-validity with liveness, we show that if every honest party's input is equal to the same value $v \in V$, then every honest party outputs the set $S = \{v\}$. Let every honest party multicast $(\mathsf{mv1}, v)$ on the same value $v \in V$ (line 3, Figure 6.9). As argued in the previous proof, no honest party echoes a value w different from v (line 5) Therefore, at most t_s values different from v are multicast as $(\mathsf{mv1}, -)$ messages. Now we consider an honest party P_i in the worst case execution, in which the t_s dishonest parties multicast $(\mathsf{mv1}, w)$ on the same value $w \neq v$. In that case, $|M_1(v)|$ increases monotonically from 0 to $n - t_s$ and $|M_1(w)|$ increases monotonically from 0 to t_s. There are the following two cases.

(i) Suppose that $|M_1(w)| \geq |M_1(v)|$. In this case, the predicate $|\bigcup_{v' \in V} M_1(v')| - |M_1(v_{max})| \geq t_s + 1$ in line 6 reduces to $|M_1(v)| \geq t_s + 1$ and returns false, since $|M_1(v)| \leq |M_1(w)| \leq t_s$. Hence, P_i never multicasts $(\mathsf{mv1}, \perp_{mv})$ (line 6).

(ii) Suppose that $|M_1(v)| \geq |M_1(w)|$. In this case, the predicate $|\bigcup_{v' \in V} M_1(v')| - |M_1(v_{max})| \geq t_s + 1$ in line 6 reduces to $|M_1(w)| \geq t_s + 1$ and returns false, since $|M_1(w)| \leq t_s$. Hence, P_i never multicasts $(\mathsf{mv1}, \perp_{mv})$ (line 6).

As a result, no honest party multicasts $(\mathsf{mv1}, \perp_{mv})$. Therefore, no honest party P_i receives $(\mathsf{mv1}, \perp_{mv})$ from $n - t_s > t_s$ distinct parties and $M_2(\perp_{mv})$ is never populated by P_i (line 'Updating $M_2(v)$' (i) and (ii)). The same is true for the value w. Consequently, every honest party outputs the same set $S = \{v\}$ (line 7). □

Lemma 6.6.8. *Let $t_s < n/2$. Then $\Pi_{\mathsf{MV}}^{t_a,t_s}$ achieves t_s-validity 2.*

Proof. This follows directly from t_s-validity with liveness. □

Lemma 6.6.9. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{MV}}^{t_a,t_s}$ achieves t_a-inclusion.*

Proof. In order to prove t_a-inclusion, we show that if honest parties P_i and P_j output sets S_i and S_j respectively, then $S_i = \{w\}$ implies $w \in S_j$ (note that $w = \perp_{mv}$ is possible). Let honest party P_i output the set $S_i = \{w\}$ (line 7, Figure 6.9). This is conditioned on $|\cup_{v \in V} M_2(v)| \geq n - t_s$ by line 7 and implies that P_i has received $(\mathsf{mv2}, w)$ from at least $n - t_s$ distinct parties by definition of the set $M_2(-)$ in line 'Updating $M_2(v)$' (i) and (ii). Now consider an honest party $P_j \neq P_i$. Before P_j outputs the set S_j, it has received messages $(\mathsf{mv2}, -)$ from $n - t_s$ distinct parties, that is from at least $n - t_s - t_a > t_s$ honest parties. Since $(n - t_s) + (t_s + 1) > n$, it follows that there is an honest party P_k that sent the same message $(\mathsf{mv2}, v)$ to both P_i and P_j. But since P_i has received only messages of the form $(\mathsf{mv2}, w)$ from $n - t_s$ parties, it follows that $v = w$ and thus $w \in S_j$. □

Lemma 6.6.10. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{MV}}^{t_a,t_s}$ achieves t_a-liveness.*

Proof. In order to prove t_a-liveness, we show that every honest party outputs some set. For this, we first show that every honest party eventually multicasts some $(\mathsf{mv2}, -)$ message (line 4, Figure 6.9). We consider the following predicate P: There is a value $v \in V$ such that after some finite time at least $t_s + 1$ honest parties have multicast $(\mathsf{mv1}, v)$. Consider the following two cases.

(i) The predicate P is satisfied. In this case, every honest party eventually multicasts $(\mathsf{mv1}, v)$ by construction of $\Pi_{\mathsf{MV}}^{t_a,t_s}$ (line 5). Since there are at least $n - t_a \geq n - t_s$ honest parties, ready eventually is set to true by every honest party due to delivering $(\mathsf{mv1}, v)$ from $n - t_s$ distinct parties (line 4).

(ii) The predicate P is not satisfied. We consider an honest party P_i. Let $v_{max} = \max_{v' \in V} |M_1(v')|$, i.e., the most frequent mv1 value received, and let r be the number of honest parties that multicast $(\mathsf{mv1}, v_{max})$. Since P is not satisfied, $r \leq t_s$. Since there are at least $n - t_a$ honest parties, P_i receives at least $n - t_a + k$ messages $(\mathsf{mv1}, -)$ with some $k \in [0, t_a]$. At most $r + k$ of these parties sent message $(\mathsf{mv1}, v_{max})$ to P_i, and therefore at least $(n - t_a + k) - (r + k) = n - t_a - r$ of them sent values different from v_{max} to P_i. But since $n - t_a - r \geq n - t_a - t_s > t_s$, the predicate $|\bigcup_{v' \in V} M_1(v')| - |M_1(v_{max})| \geq t_s + 1$ in line 6 is satisfied and P_i multicasts $(\mathsf{mv1}, \perp_{mv})$ (line 6). Analogously, this applies to all honest parties. And thus, every honest party receives messages $(\mathsf{mv1}, \perp_{mv})$ from at least $n - t_a \geq n - t_s$ distinct parties. As a result, ready becomes eventually true for every honest party (line 4).

This concludes our initial assertion that every honest party eventually multicasts some message $(\mathsf{mv2}, -)$ (by setting ready to true) by line 4. For the final step, we show that every honest party eventually receives *validated* messages $(\mathsf{mv2}, -)$ from at least $n - t_s$ distinct parties and thus outputs a set (namely $\{v : M_2(v) \neq \emptyset\}$) by line 7. Here, by a *validated* message $(\mathsf{mv2}, w)$, we mean one such that the party has also received messages $(\mathsf{mv1}, w)$ on the same value w from at least $n - t_s$ distinct parties (as is declared in line 4). By the previous assertion, each honest party P_i multicasts some message $(\mathsf{mv2}, v_i)$ where v_i is such that $|M_1(v_i)| \geq n - t_s$. Since $(n - t_s) - t_a > t_s$, at least $t_s + 1$ honest parties have multicast $(\mathsf{mv1}, v_i)$. Thus, every honest party eventually multicasts $(\mathsf{mv1}, v_i)$ and for every honest party

P_j we have $|M_1(v_i)| \geq n - t_a \geq n - t_s$. As a consequence, every honest party P_j adds $\{j\}$ to its set $M_2(v_i)$. Since this proof hitherto also applies to every honest party P_j (in place of P_i), eventually every honest party receives validated messages $(\mathsf{mv2}, -)$ from at least $n - t_a \geq n - t_s$ distinct parties and thus outputs the set $\{v : M_2(v) \neq \emptyset\}$ (line 7). □

6.6.4 Validity-Optimized Graded Consensus

Next, we define a graded consensus primitive. In graded consensus, each party inputs a value but outputs both a value $v \in V \cup \{\perp\}$ and a corresponding grade $g \in \{0, 1, 2\}$. *Binary* graded consensus (i.e., where $V = \{0, 1\}$) was previously considered for building network-agnostic binary agreement in [BKL19]. Our primitive will also require an intrusion tolerance property that we define below.

Definition 6.6.2 (Graded Consensus)**.** Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i begins holding input $v_i \in V$.

- ***Graded validity.*** Π achieves *t-graded validity* if the following holds whenever at most t parties are corrupted: if every honest party's input is equal to the same value v, then all honest parties output $(v, 2)$.
- ***Graded consistency.*** Π achieves *t-graded consistency* if the following holds whenever at most t parties are corrupted: (1) If two honest parties output grades g, g', then $|g - g'| \leq 1$. (2) If two honest parties output (v, g) and (v', g') with $g, g' \geq 1$, then $v = v'$.
- ***Liveness.*** Π achieves *t-liveness* if whenever at most t parties are corrupted, every honest party outputs (v, g) with either $v \in V$ and $g \geq 1$, or $v = \perp$ and $g = 0$.
- ***Intrusion tolerance.*** Π achieves *t-intrusion tolerance* if whenever at most t parties are corrupted, if (v, g) is output by an honest party and $g \geq 1$, then v was input by an honest party.

We construct a multivalued graded consensus protocol $\Pi_{\mathsf{GC}}^{t_a,t_s}$ (cf. Figure 6.10) Our protocol requires two (implicitly domain separated) instances of $\Pi_{\mathsf{MV}}^{t_a,t_s}$ which we denote by MV_1 and MV_2 with default values $\perp_{mv1}$ and $\perp_{mv2}$, respectively.

$\Pi_{\mathsf{GC}}^{t_a,t_s}$ works as follows. Suppose (honest) P_i inputs v_i. In addition to messages from MV-broadcast, Π_{GC} uses two message types, namely (init, v) and (echo, v) for some $v \in V$. After initialising variables, P_i multicasts (init, v_i) (and doesn't send init messages hereafter). On receipt of messages (msg, v) (where $\mathsf{msg} \in \{\mathsf{init}, \mathsf{echo}\}$ from $t_s + 1$ parties (and thus an honest party input v), P_i multicasts (echo, v) if not yet done. Except for these steps, communication takes place through MV_1 and MV_2. Let $R(v)$ be the set of parties from which P_i has received a message of the form (init, v) or (echo, v) (step 2). Then, whenever P_i receives a message (msg, v), P_i checks some conditions to determine whether it can input a value to MV_1 (after which the conditions are ignored). If $|R(v)| \geq t_s + 1$ and $v \neq v_i$, P_i inputs $\perp_{rd}$ to MV_1, indicating disagreement between two honest parties. Similarly, if $|\cup_{v' \in V} R(v')| - \max_{v' \in V} |R(v')| \geq t_s + 1$, P_i inputs $\perp_{rd}$ to MV_1. If $|R(v)| \geq n - t_s$, indicating enough parties received v, P_i inputs v to MV_1. Then, P_i eventually outputs a set S_1 from MV_1. If $|S| = \{v\}$, P_i runs MV_2 with that input; otherwise P_i runs MV_2 on input $\perp$ (step 6). On outputting S_2 from MV_2 (step 2), if $S_2 = \{v\}$

$\Pi_{\mathsf{GC}}^{t_a,t_s}(v_i)$

1. Set output = false, val = $\perp$, aux = $\perp$.
2. For every $v \in V$, let $R(v)$ be the set of parties from which P_i has received (msg, v) for $\mathsf{msg} \in \{\mathsf{init}, \mathsf{echo}\}$. Initially, set $R(v) = \emptyset$ for all $v \in V$.
3. Multicast the message (init, v_i).
4. Upon receiving (init, v) messages on the same value v from $t_s + 1$ distinct parties: if $v \neq v_i$ and (echo, v) not yet multicast, multicast (echo, v).
5. Upon receiving (msg, v) for some $\mathsf{msg} \in \{\mathsf{init}, \mathsf{echo}\}$ and v when output = false:
   (a) If $v \neq v_i$ and $|R(v)| \geq t_s + 1$, set output = true and val = $\perp_{rd}$.
   (b) If $|R(v)| \geq n - t_s$, set output = true and val = v.
   (c) If $|\cup_{v' \in V} R(v')| - m \geq t_s + 1$ for $m = \max_{v' \in V} |R(v')|$, set output = true and val = $\perp_{rd}$.
   (d) If output = true, run MV_1 using input val.
6. Upon receiving output S_1 from MV_1: if $S_1 = \{v\}$, set aux = v. Run MV_2 using input aux.
7. Upon receiving output S_2 from MV_2:
   (a) If $S_2 = \{v\}$ and $v \notin \{\perp_{rd}, \perp_{mv1}, \perp_{mv2}, \perp\}$, output $(v, 2)$.
   (b) Otherwise, if $\exists\, v \in S_2$ s.t. $v \notin \{\perp_{rd}, \perp_{mv1}, \perp_{mv2}, \perp\}$, output $(v, 1)$.
   (c) Otherwise, output $(\perp, 0)$.

Figure 6.10: Multivalued graded consensus from the view of party P_i.

and not a default value, then it outputs $(v, 2)$. Otherwise, if S_2 contains a non-default value, P_i outputs that value and $g = 1$; else, P_i outputs $(\perp, 0)$.

Communication Complexity. We argue in Lemma 6.6.16 that the communication complexity of our graded consensus protocol $\Pi_{\mathsf{GC}}^{t_a,t_s}$ is bounded by $O(n^3)$. Furthermore, we argue that if the network happens to be asynchronous, then the communication complexity of the protocol improves to $O(n^2)$. On the other hand, if the networks happens to be synchronous and parties input the same values to the graded consensus protocol, then the communication complexity can trivially be bounded by $O(n^2)$. We elaborate more on these facts in Lemma 6.6.16.

Theorem 6.6.11. *Let n, t_s, t_a be such that $0 \leq t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ and $t_a + 2 \cdot t_s < n$. Then graded consensus protocol $\Pi_{\mathsf{GC}}^{t_a,t_s}$ (cf. Figure 6.10) satisfies t_s-graded validity, t_s-intrusion tolerance, t_a-graded consistency and t_a-liveness.*

We prove this by proving the following lemmas.

Lemma 6.6.12. *Let $t_s < n/2$. Then $\Pi_{\mathsf{GC}}^{t_a,t_s}$ achieves t_s-graded validity.*

Proof. In order to prove t_s-graded validity, we show that if every honest party's input is equal to the same value v, then all honest parties output $(v, 2)$. Let every honest party multicast (init, v) (line 3, Figure 6.10). Since there are at most $t_s < t_s + 1$ dishonest parties, no honest party echoes a value different from v (line 4). Therefore, from the perspective of every honest party P_i the set $R(w)$ is of size $\leq t_s$ for every $w \neq v$. In particular, conditions (a) and (c) of line 5 are not satisfied for the value v. On the other hand, since there are at least $n - t_s$ honest parties that multicast (init, v), it is $|R(v)| \geq n - t_s$, and P_i sets $\mathsf{output} = \mathsf{true}$ and $\mathsf{val} = v$ (line 5, condition (b)). As this applies to every honest party, each of them run MV_1 on the same input v (line 5 (d)). Now, t_s-validity with liveness from MV_1 ensures that every honest party outputs the set $S_1 = \{v\}$. As a result, each of them sets $\mathsf{aux} = v$ and runs MV_2 again on input v (line 6). The same argument yields that every honest party outputs the set $S_2 = \{v\}$. Finally, as $v \notin \{\perp_{rd}, \perp_{mv1}, \perp_{mv2}, \perp\}$, every honest party outputs $(v, 2)$ (line 7 (a)). □

Lemma 6.6.13. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{GC}}^{t_a,t_s}$ achieves t_a-graded consistency.*

Proof. In order to prove t_a-graded consistency, we show that (1) if two honest parties output grades g, g', then $|g - g'| \leq 1$, and (2) if two honest parties output (v, g) and (v', g') with $g, g' \geq 1$, then $v = v'$. For the first clause, assume that there are two honest parties P_i and P_j where P_i outputs $(v, 2)$. This means P_i's output S_2 from MV_2 is $S_2 = \{v\}$ with $v \notin \{\perp_{rd}, \perp_{mv1}, \perp_{mv2}, \perp\}$ (line 7 (a), Figure 6.10). By t_a-inclusion of MV_2, every honest party P_j outputs a set S_2 such that $v \in S_2$ and therefore cannot output grade $g' = 0$ for its final decision (line 7 (c)). This proves the first clause. For the second clause, consider two honest parties P_i and P_j that output (v, g) and (v', g') respectively with $g, g' \geq 1$. We write $S_2(P_k)$ for the output set S_2 from MV_2 of party P_k; for the output set S_1 we likewise define $S_1(P_k)$. In particular, $v \in S_2(P_i)$ and $v' \in S_2(P_j)$ with $v, v' \notin \{\perp_{rd}, \perp_{mv1}, \perp_{mv2}, \perp\}$ (line 7 (a) and (b)). By t_a-inclusion of MV_2, it follows that $v \in S_2(P_j)$ and thus $\{v, v'\} \subseteq S_2(P_j)$. By t_s-validity 1 of MV_2, v' was input by some honest party P_k to MV_2. In particular, $\mathsf{aux} = v'$ for party P_k and thus $S_1(P_k) = \{v'\}$ (line 6). Analogously, there is some honest party P_l that input v to MV_2 and hence $S_1(P_l) = \{v\}$. But by t_a-inclusion of MV_1, it follows that $v = v'$, and we conclude. □

Lemma 6.6.14. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{GC}}^{t_a,t_s}$ achieves t_a-liveness.*

Proof. In order to prove t_a-liveness, we show that every honest party outputs (v, g) with either $v \in V$ and $g \geq 1$, or $v = \perp$ and $g = 0$ (line 7, Figure 6.10). We consider the following predicate P: There is a value $v \in V$ such that after some finite time at least $t_s + 1$ honest parties have multicast (init, v). Consider the following two cases.

(i) The predicate P is satisfied. It follows that every honest party eventually multicasts (echo, v) (if not yet multicast (init, v)) (line 4). As there are at least $n - t_a \geq n - t_s$ honest parties, the predicate $|R(v)| \geq n - t_s$ in line 5 (b) becomes eventually true for every honest party and each of them sets $\mathsf{output} = \mathsf{true}$ and runs MV_1 (line 5 (d)). By t_a-liveness of MV_1, every honest party outputs some set S_1 and runs MV_2 on input aux (line 6). And by t_a-liveness of MV_2, every honest party outputs some set S_2 and finally outputs some graded value (v, g) (line 7). The requirement [$v \in V$ and $g \geq 1$, or $v = \perp$ and $g = 0$] is trivially satisfied by construction of $\Pi_{\mathsf{GC}}^{t_a,t_s}$ in its final step (line 7).

(ii) The predicate P is not satisfied. This means there is no value v that is multicast (init, v) by at least $t_s + 1$ honest parties. We consider an honest party P_i. Let v_{max} be its most

often received value from distinct parties as defined in line 5 (c) and let r be the number of honest parties that multicast (init, v_{max}). By assumption, we have $r \leq t_s$. Let $k \in [0, t_a]$ be the number of distinct dishonest parties from which P_i has received the message (init, v_{max}). Clearly, at most $t_s + k \geq r + k$ distinct parties have sent the value v_{max} to P_i. Now assume that the waiting predicates $[|R(v)| \geq n - t_s]$ and [for $w \neq v_i$ and $|R(w)| \geq t_s + 1$] in line 5 (a) and (b) are never satisfied (otherwise P_i sets $\mathsf{output} = \mathsf{true}$ and proceeds with the execution of MV_1). Since P_i does not terminate at these predicates, it receives a message from each honest party, that is from at least $n - t_a$ parties. Hence, P_i receives messages from at least $n - t_a + k$ distinct parties. As a consequence, at least $(n - t_a + k) - (t_s + k) = n - t_a - t_s > t_s$ distinct parties have sent values different from v to P_i. As a result, the predicate $|\cup_{v' \in V} R(v')| - |R(v_{max})| \geq t_s + 1$ in line 5 (c) is eventually satisfied and P_i runs MV_1 on input $\mathsf{val} = \perp_{rd}$. Therefore, every honest party runs MV_1 on some input (line 5 (d)) and the remainder of the proof proceeds as in the previous case.

□

Lemma 6.6.15. *Let $t_s < n/2$. Then $\Pi_{\mathsf{GC}}^{t_a,t_s}$ achieves t_s-intrusion tolerance.*

Proof. In order to prove t_s-intrusion tolerance, we show that no honest party P_i outputs a graded value (v, g) with $g \geq 1$ that was multicast (init, v) by dishonest parties only. For this, let there be t_s dishonest parties that multicast (init, v) such that (init, v) is not multicast by any honest party. Since $t_s + 1 > t_s$, no honest party echoes (echo, v) the value v and thus v can be received only from the t_s dishonest parties (line 4, Figure 6.10). As a consequence, for every honest party P_i it is $|R(v)| \leq t_s < n - t_s$ and thus no honest party sets val equal to v (line 5 (b)). The claim follows from t_s-validity 1 of MV_1 and MV_2. □

Finally, we bound the complexity of the protocol.

Lemma 6.6.16. *Let $t_s < n/2$. The total communication complexity of the graded consensus protocol $\Pi_{\mathsf{GC}}^{t_a,t_s}$ is bounded by $O(n^3)$. Furthermore, if the underlying network is asynchronous, this improves to a communication complexity of $O(n^2)$.*

Proof. First, we show that each honest party may echo at most two messages $(\mathsf{echo}, -)$ with different values. For this, let $n = 2t_s + t_a + 1 < 3(t_s + 1)$ and consider an honest party P_i. Clearly, P_i receives at most one message $(\mathsf{init}, -)$ from any other party, as it otherwise would know that the sender is dishonest. In order to multicast message (echo, v) for some $v \neq v_i$, P_i needs to receive (init, v) from $t_s + 1$ distinct parties (line 4, Figure 6.10). Since $n < 3(t_s + 1)$, it follows that P_i can echo at most two such messages $(\mathsf{echo}, -)$ carrying different values. This gives a communication complexity of $O(n^2)$ up to line 5 (c).

Next, we show that the communication complexity of MV_1 is bounded by $O(n^3)$. Having a look at Figure 6.9, we see that every honest party multicasts at most one message of the form $(\mathsf{mv2}, -)$ (line 4). Furthermore, a message of the form $(\mathsf{mv1}, v)$ for some $v \neq \perp_{mv}$ is multicast if it was received from at least $t_s + 1$ distinct parties, ensuring that the message had to come from at least one honest party (line 5). In particular, an honest party only $\mathsf{mv1}$-multicasts at most $n - t_s$ times. Moreover, every honest party multicasts at most one message of the form $(\mathsf{mv1}, \perp_{mv})$ (line 6). As a result, the overall communication complexity of MV_1 is bounded by $O(n^3)$. Getting back to Figure 6.10, line 5 (d) and line 6 give two instances of the MV-broadcast

protocol with communication complexity $O(n^3)$, so that the overall communication complexity of $\Pi_{\mathsf{GC}}^{t_a,t_s}$ is bounded by $O(n^3)$.

In the following, we do a communication complexity analysis of MV_1 for the case where the underlying network is asynchronous. Specifically, we show that the communication complexity of MV_1 is bounded by $O(n^2)$ in this case, which leads to an overall complexity of $O(n^2)$ for our graded consensus protocol. We say *vote* for $v \neq \perp_{rd}$ a message (init, v) or (echo, v) sent by some party. Observe that MV_1 is run by an honest party P_i on input val conditioned on output being true. In that case, val is set to either $\perp_{rd}$ or to some v such that (msg, v) was received by P_i from at least $n - t_s$ distinct parties. We will first show that there are at most five different values $\neq \perp_{rd}$ that are input by the set of honest parties to MV_1. In order for a value $v \neq \perp_{rd}$ to be input to MV_1 by an honest party, P_i must received the message (msg, v) from at least $n - t_s$ distinct parties. The discussion in the first paragraph tells us that a party can vote for at most three distinct values, while a dishonest party can vote for any number of values. To maximize the total number of values that can be input into MV_1 by the set of honest parties, we consider the worst-case scenario in which there are t_a dishonest parties and if an honest party P_i inputs a value v into MV_1 it has received for this value $n - t_s - t_a$ votes from honest parties and a vote from each of the t_a dishonest parties. Therefore, we have to consider the maximal number of votes that can be sent by the honest parties and the knowledge that only $n - t_s - t_a$ of them are needed to input a value into MV_1. Again by the first paragraph, there are at most $3(n - t_a)$ votes sent by honest parties. It follows that the total number of different values that can be input into MV_1 by the set of honest parties is bounded by the following

$$\frac{3(n - t_a)}{n - t_s - t_a} = \frac{3(2t_s + 1)}{t_s + 1} < \frac{6(t_s + 1)}{t_s + 1} = 6.$$

As a result, the total number of different values that can be input into MV_1 by the honest parties is bounded by 6 (when including the default value $\perp_{rd}$ also). Now, the same argument as in the previous paragraph applies: a message of the form $(\mathsf{mv1}, v)$ for some $v \neq \perp_{mv}$ is multicast if it was received from at least $t_s + 1$ distinct parties, ensuring that the message had to come from at least one honest party and therefore an honest party only $\mathsf{mv1}$-multicasts at most six times (as the previous analysis has shown that there are at most 6 different input values from the set of honest parties). As a consequence, the communication complexity of MV_1 is bounded by $O(n^2)$. Hence, we find that the overall communication complexity of $\Pi_{\mathsf{GC}}^{t_a,t_s}$ is bounded by $O(n^2)$ in case the underlying network is asynchronous. □

Remark 6.6.1. From the above analysis it is clear that even in the synchronous case, conditioned on all honest parties having the same input value, the overall communication complexity of the graded consensus protocol is bounded by $O(n^2)$. Again, the reason for this is that in case all honest parties start with the same input value $v \in C$, then an honest party will never echo any other value and as a consequence they all will input v into the MV_1 protocol. On the other hand, when all honest parties input the same value v into MV_1, then by design any honest party will only $\mathsf{mv2}$-multicast that value v and also eventually terminate with it. Having said that, it is clear that the overall communication complexity of $\Pi_{\mathsf{GC}}^{t_a,t_s}$ is bounded by $O(n^2)$ in case the underlying network is synchronous *and* all honest parties start with the same value. This observation is crucial if the graded consensus protocol is used as a building block in a more complex consensus protocol, such as a network-agnostic DKG protocol. The reason is that many protocols are build

in such a way that the validity property (i.e., all honest parties start with the same value) holds if the underlying network is synchronous.

Appendix 6B: Binary Byzantine Agreement

We present a Byzantine binary agreement protocol $\Pi_{\mathsf{BA}}^{t_a,t_s}$. Let $\Pi_{\mathsf{GC}}^{t_s}$ be a (binary) graded consensus protocol, i.e. takes as input a value in $\{0,1\}$ where each party outputs a value $v \in \{0,1,\perp\}$ and a grade $g \in \{0,1,2\}$. We require that $\Pi_{\mathsf{GC}}^{t_s}$ satisfies t_s-graded validity and is t_a-secure: the protocol from [BKL19] (see Figure 4 there) satisfies these properties with $O(n^2)$ communication complexity. We also require a coin-flip mechanism Coin which allows all parties to generate and know an unbiased binary value $\mathsf{Coin}_r \in \{0,1\}$ for $r \geq 2$. Upon receiving input $r \geq 2$ from $t_s + 1$ parties, the coin flip mechanism generates an unbiased coin $\mathsf{Coin}_r \in \{0,1\}$ and sends (r, Coin_r) to all parties. In particular, if at most t_s parties are corrupted, at least one honest party must send r to Coin before the adversary can learn the coin Coin_r. We will rely on a $\tilde{p}$-weak coin flip with $\tilde{p} = 1/3$, where honest parties agree on the coin only with probability $\tilde{p} < 1$. This comes with an increase in the expected round complexity by a factor of $O(1/\tilde{p}) = O(1)$. Our coin flip mechanism is the one from [Gao+22] (see Algorithm 4 there), which costs only $O(\lambda n^3)$ bits and has expected constant rounds (and ensures that with probability at least $1/3$, all honest parties output an unbiased common coin).

Description. Our asynchronous Byzantine agreement protocol $\Pi_{\mathsf{BA}}^{t_a,t_s}$ (cf. Figure 6.11) works as follows. We describe it from the perspective of a party P_i with input value v_i. In the protocol, we say message $(\mathsf{commit}, b, \sigma)$ from party P_i (as in 'Termination procedure', steps (i) and (ii)) is valid if $b \in \{0,1\}$ and σ is a valid signature from P_i on (commit, b), i.e $\sigma \leftarrow \langle \mathsf{commit}, b \rangle_i$. Also, we say that a set of signatures is a certificate for b (as in step (i) and (ii)) if the set contains valid signatures on (commit, b) from at least $t_s + 1$ distinct parties.

As usual, at the beginning helper variables are set (step 1). Afterwards, the protocol proceeds in round defined by the parameter r. In such a round, P_i first runs the graded consensus protocol $\Pi_{\mathsf{GC}}^{t_s}$ on input $b = v_i$ with (b, g) being the output (step 2). Note that for rounds $r < 3$ we set the coin to some deterministic default value (which does not affect the safety of the algorithm). We do this because we can can then assume our coin flip mechanism only works in asynchrony (and thus use the algorithm from [Gao+22]) so that when we show t_s-validity with termination, it will also hold that the coin is never used. Otherwise, $\mathsf{Coin}(r)$ is invoked.

Then, if $g < 2$, P_i runs $\Pi_{\mathsf{GC}}^{t_s}$ on input Coin_r (step 4), otherwise on input b (steps 4 and 5), with (b', g) being the output. Now if $g > 0$, set $b = b'$ (step 5). In case $g = 2$, the next step is done only once (which is guaranteed by setting helper variable cm to true): compute the signature σ on (commit, b) and multicast the message $(\mathsf{commit}, b, \sigma)$ (step 7). Finally, the next round starts by increasing r by one (step 8) with a small restriction: As soon as P_i computes a commit message, it only executes one additional round of the protocol and then stops (this is ensures by the variable stop in steps 7 and 8). Furthermore, party P_i terminates and ends the protocol as soon as (i) it receives valid commit messages on the same value b from at least $t_s + 1$ distinct parties (before termination, P_i combines the signatures into a certificate Σ, multicasts $(\mathsf{notify}, b, \Sigma)$ and outputs b), or (ii) it receives $(\mathsf{notify}, b, \Sigma)$ with Σ being a certificate on messages (commit, b) for the same value b (before termination, P_i multicasts $(\mathsf{notify}, b, \Sigma)$ and outputs b).

$\Pi_{\mathsf{BA}}^{t_a,t_s}(v_i)$

1. Set cm = output = false, stop = 0, $b = v_i$, and $r = 1$.
   While output = false do:
2. Run $\Pi_{\mathsf{GC}}^{t_s}$ on input b. Let (b, g) be the output.
3. If $r < 3$, set $\mathsf{Coin}_r = 1$. Otherwise, let $\mathsf{Coin}_r \leftarrow \mathsf{Coin}(r)$.
4. If $g < 2$, set $b = \mathsf{Coin}_r$.
5. Run $\Pi_{\mathsf{GC}}^{t_s}$ on input b. Let (b', g) be the output.
6. If $g > 0$, set $b = b'$.
7. If $g = 2$ and cm = false:
   - Set cm = true and stop = $r + 1$.
   - Compute $\sigma \leftarrow \langle \mathsf{commit}, b \rangle_i$.
   - Multicast $(\mathsf{commit}, b, \sigma)$.
8. If stop $> r$, set $r = r + 1$ (and repeat from step 2).

Termination procedure:

(i) Upon receiving (valid) commit messages $(\mathsf{commit}, b, \sigma)$ on the same value b from $t_s + 1$ distinct parties:
   - Combine the signatures into a certificate $\Sigma = (\sigma_1, \ldots, \sigma_{t_s+1})$.
   - Multicast $(\mathsf{notify}, b, \Sigma)$.
   - Output b, set output = true, and terminate.

(ii) Upon receiving $(\mathsf{notify}, b, \Sigma)$ such that Σ contains valid signatures on commit messages (commit, b) on the same value b from $t_s + 1$ distinct parties:
   - Multicast $(\mathsf{notify}, b, \Sigma)$.
   - Output b, set output = true, and terminate.

Figure 6.11: Binary agreement protocol from the view of party P_i.

The purpose of the stop variable is to ensure that the communication complexity of the protocol stays bounded. The protocol is well-defined and terminates, since we have the following: if one honest party set cm = true in round r, then all parties set cm = true in (at most) round $r + 1$. We sketch the argument for this claim in the following. We will say a party *commited* if it sets cm = true. Suppose an honest party commited in round r. By construction, it must have $g = 2$ and by graded consistency of $\Pi_{\mathsf{GC}}^{t_s}$, it is $|g - g'| \leq 1$ for all g' output by honest parties. Therefore, all honest parties output $g = 1$ or $g = 2$. Furthermore, graded consistency implies that if one honest party outputs $(v, g = 2)$, then all honest parties output the same v, and so all honest parties will input v in the next round. Again by graded validity, all honest parties will output $g = 2$ from the first execution of $\Pi_{\mathsf{GC}}^{t_s}$ and by similar arguments every honest party will have commited in that round (we will see this argumentation more thoroughly in the proof for t_s-graded validity below).

We have the following theorem.

Theorem 6.6.17. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then Π_{BA} achieves t_s-validity with termination and is t_a-secure.*

Proof. First, we prove the t_s-validity with termination. For this, assume all honest parties initially hold the same value $v \in \{0, 1\}$. All honest parties use v as input in the first execution of the graded consensus protocol $\Pi_{\mathsf{GC}}^{t_s}$ (step 2). By t_s-graded validity of $\Pi_{\mathsf{GC}}^{t_s}$, all honest parties output $(v, 2)$ from that execution. Thus, all honest parties run a second instance of $\Pi_{\mathsf{GC}}^{t_s}$ using input v (step 5), again receiving $(v, 2)$ as output. Therefore, all honest parties multicast a commit message on v (step 7). Additionally, by the stop variable, all honest parties will only execute one additional round of the protocol. Furthermore, the honest parties will receive at most $t_s < t_s + 1$ commit messages on some $w \neq v$. Therefore, all honest parties eventually receive valid commit messages on v from $n - t_s \geq t_s + 1$ distinct parties, output v, and terminate (step (i) and (ii) of 'Termination procedure'). The proof of the t_a-security follows the same argumentation as the proof of Lemma 11 in [BKL19] and therefore we skip it here. □

7
Final Remarks

7.1 Open Problems for Future Work

We conclude this thesis by presenting several open problems motivated by our results, which may serve as directions for future research.

7.1.1 Open Problems Related to Threshold Signatures – Chapter 3

In Chapter 3, we proved the security of the threshold BLS signature of Boldyreva [Bol03] for up to t adaptive corruptions, assuming the algebraic group model and the hardness of the one-more discrete logarithm (OMDL) problem. We further showed that even in the algebraic group model, a strong interactive assumption, such as one-more discrete logarithm, is necessary in order to prove the scheme adaptively secure. Therefore, a natural question is whether one can design a threshold variant of the BLS signature with adaptive security, based on more standard assumptions. In that regard, Das and Ren [DR24] recently made major progress by designing an adaptively secure threshold BLS signature that relies on the hardness of the decisional Diffie-Hellman (DDH) and co-computational Diffie Hellman (co-CDH) problem in asymmetric pairing groups. The next step would thus be to remove the reliance on the DDH assumption and asymmetric pairing groups.

Open Problem 1. *Construct a non-interactive threshold BLS signature scheme for up to t adaptive corruptions from a well-studied non-interactive search assumption, such as CDH, potentially working over any pairing groups.*

Our security proof in Chapter 3 is tight and thus justifies the choice of parameters of the threshold BLS signature in real-world applications. Tightness of the security proof, however, crucially relies on the algebraic group model, as supported by our impossibility result which rules out a natural class of tight security reductions from $(t+1)$-OMDL. On top of that, existing impossibility results [Cor02; KK18] rule out any tight security reduction for the (single-signer) BLS signature scheme from CDH.[1] While there are single-signer signatures with tight security reductions [KW03; GJ03], our understanding of tight threshold signatures is still lacking. In particular, there is no non-interactive threshold signature with tight security proof from an established assumption, not even for static corruptions. In the distinct (but related) realm of multi-signatures, recent progress towards tight security has been made, and techniques such as a two-key approach [PW23; BW24a] or a signer partitioning approach [BW25b] could potentially be useful to resolve this question in the threshold setting.

Open Problem 2. *Construct a non-interactive threshold signature scheme for up to t adaptive (resp. static) corruptions from a well-studied non-interactive assumption with a tight security proof (without relying on the AGM).*

We note that such a non-interactive threshold signature will most likely be pairing-based, unless one uses expensive tools such as a succinct non-interactive argument of knowledge (SNARK).[2] As such, the best we can hope for in the pairing-free setting (without the use of heavy tools) is a two-round signing protocol. In that setting, Chen [Che25] recently presented

[1] We note that assuming hardness of the CDH problem is necessary to show security of the BLS signature.

[2] Essentially, upon receiving $t+1$ valid individual signatures, one can compute a succinct proof that demonstrates knowledge of $t+1$ valid individual signatures from distinct parties.

an adaptively secure two-round threshold signature from DDH, relying on the adaptively secure three-round scheme Twinkle from Bacho et al. [Bac+24d]. However, the security reductions for these schemes rely on a guessing argument and therefore have a security loss linear in the number of signing queries of the adversary. In particular, these schemes are not tight. While there are two adaptively secure three-round constructions with tight security proof from an established assumption [Che25; BW24b], we still lack a two-round scheme with the same properties, even for static corruptions. Again, the existing techniques in the realm of multi-signatures could potentially be useful to resolve this question in the threshold setting.

Open Problem 3. *Construct a pairing-free two-round threshold signature scheme for up to t adaptive (resp. static) corruptions from a well-studied non-interactive assumption with a tight security proof (without relying on the AGM).*

Recently, a number of follow-up works [CKM23b; Bac+25d; Bac+25c; Bac+24c; Cri+25; KRT24] have focused on the design of adaptively secure threshold versions of the Schnorr signature scheme [Sch90]. The Schnorr signature is a pairing-free scheme with highly efficient verification[3] and it has also been been adopted by Bitcoin and other cryptocurrencies. These features make it particularly attractive, and threshold variants of it are explicitly sought by the NIST in its recent calls for threshold cryptography [BP25; BD22]. The state-of-the-art adaptively secure two-round threshold Schnorr signature is FROST [Cri+25]. However, its security proof relies on the algebraic group model, the hardness of the algebraic one-more discrete logarithm (AOMDL) problem, and the hardness of a newly defined combinatorial search problem called "low-dimensional vector representation (LDVR)".[4] On the other hand, there is an adaptively secure three-round threshold Schnorr signature that relies only on the hardness of DDH [Bac+25c]. This leaves a large gap between our understanding of two-round and three-round threshold Schnorr signatures, raising the following open problems.

Open Problem 4. *Construct a two-round threshold Schnorr signature scheme for up to t adaptive corruptions from a well-studied non-interactive assumption, such as DDH, potentially in the AGM.*

Open Problem 5. *Construct a three-round threshold Schnorr signature scheme for up to t adaptive corruptions from a well-studied non-interactive search assumption, such as DLOG, potentially in the AGM.*

Note that the former allows any well-established assumption, including decisional assumptions such as DDH, whereas the latter specifically requires a search assumption such as DL. As a first step towards this goal, the AGM could potentially be useful.

7.1.2 Open Problems Related to PVSS and Distributed Randomness Beacons – Chapter 4

In Chapter 4, we proved the security of several existing (aggregatable) PVSS schemes [Bol03] for up to t adaptive corruptions, assuming the algebraic group model and the hardness of the

[3] Its signature verification is more than 10× faster than that of pairing-based schemes such as BLS.

[4] The problem is defined over a finite prime field $\mathbb{Z}_p$ and without reference to any cryptographic group $\mathbb{G}$. Our trust in the hardness of the LDVR problem therefore remains primarily conjectural.

one-more discrete logarithm (OMDL) problem. We further showed that this implies the adaptive security of state-of-the-art distributed randomness beacons [Bha+21; Das+22a; Bha+23], which build on top of these (aggregatable) PVSS schemes. A natural question is whether one can construct an adaptively secure (aggregatable) PVSS scheme from standard assumptions. This question remains open, and we therefore pose it as an open problem for future work.

Open Problem 6. *Construct a PVSS scheme for up to t adaptive corruptions from a well-studied non-interactive assumption, such as DDH or CDH, potentially without the use of pairings.*

We believe that an aggregatable PVSS scheme[5] will most likely rely on pairings, unless one resorts to expensive tools such as SNARKs. Therefore, for aggregatable PVSS schemes, we do not set any restriction on the use of pairings.

Open Problem 7. *Construct an aggregatable PVSS scheme for up to t adaptive corruptions from a well-studied non-interactive assumption, such as DDH or CDH.*

Moreover, all existing aggregatable PVSS schemes are restricted to sharing secrets that are elements in a cryptographic group $\mathbb{G}$, whereas many applications require the secret to lie in a prime field $\mathbb{Z}_p$. This raises the question of whether one can design an aggregatable PVSS scheme that shares field-element secrets, ideally with adaptive security. While homomorphic non-interactive zero-knowledge (NIZK) proofs could potentially address this challenge, they generally increase the size of the PVSS transcripts by roughly an order of magnitude, which makes them undesirable for practical applications. We therefore aim to design a practically efficient instantiation of such a scheme.

Open Problem 8. *Construct a practically efficient aggregatable PVSS scheme that shares secrets over a finite field $\mathbb{Z}_p$, potentially with adaptive security from a well-studied non-interactive assumption.*

A particularly valuable feature of the distributed randomness beacons in [Bha+21; Das+22a; Bha+23] is their reconfiguration-friendliness, which allows to add and remove parties to the system while maintaining quadratic communication complexity per epoch (where each epoch takes about 10 rounds of all-to-all communication). This is in contrast to DKG-based distributed randomness beacon protocols [Dra20; CKS00; Cam+21], which require the generation of new threshold keys whenever a party joins or leaves the system.[6] On the other hand, once the setup phase is completed, DKG-based protocols operate much more efficiently: each epoch takes only a single round of communication in which each party sends a single group element to every other party. This naturally leads to the question of whether the efficiency gap between these two types of distributed randomness beacon protocols can be closed. More concretely:

Open Problem 9. *Construct a reconfiguration-friendly distributed randomness beacon protocol with quadratic communication cost per epoch and each epoch takes at most 4 rounds, potentially with adaptive security.*

[5] Recall that we require transcript aggregation to be non-interactive. Otherwise, the parties could engage in an interactive protocol to generate a *distributed* proof of correct aggregation, which would essentially resemble a multi-party signing protocol.

[6] Essentially, these protocols execute a distributed key generation (DKG) protocol each time the set of participating parties changes, which makes them rather unsuitable for highly dynamic systems.

7.1.3 Open Problems Related to Distributed Key Generation – Chapter 5

In Chapter 5, we presented a synchronous distributed key generation (DKG) protocol with a communication complexity of $O(\lambda n^2 \log n)$ bits, where n is the number of parties executing the protocol. In fact, this is the first DKG protocol to break the previous $O(\lambda n^3)$ communication barrier. This naturally raises the question of how much further the communication complexity of DKG protocols can be reduced. While we strongly believe that any (even statically) t-secure[7] DKG protocol must incur at least $O(\lambda n^2)$ bits of communication, we currently lack a formal proof of this lower bound.

Open Problem 10. *Provide a formal lower bound on the communication complexity of t-secure DKG protocols, potentially separating between static and adaptive security.*

Another metric of significant interest is the round complexity. Our DKG protocol terminates in $O(n)$ rounds, and an appealing goal is to reduce this to (expected) constant rounds. Given that our DKG protocol construction has a binary tree structure of depth $\log(n)$, it seems unlikely to improve beyond $O(\log(n))$ rounds and fundamentally new ideas are required.

Open Problem 11. *Construct a synchronous DKG protocol with $O(\lambda n^2 \log n)$ communication and (expected) $O(1)$ rounds, potentially with adaptive security.*

Our DKG protocol relies on aggregatable PVSS, where the aggregation property is crucial for enabling recursion. As noted earlier, all existing efficient aggregatable PVSS schemes share secrets over a cryptographic group $\mathbb{G}$. Consequently, the public–secret keys generated by our DKG protocol are pairs of group elements, whereas applications such as BLS or Schnorr signatures require the secret key to lie in a prime field $\mathbb{Z}_p$. Efficient PVSS schemes that share field elements, however, rely on non-aggregatable NIZK proofs. We note that generic transformations can be applied to our DKG protocol to generate secret keys as field elements at the cost of $O(\lambda^2 n^2)$ added communication and reliance on secure erasures.[8] We therefore aim to avoid these undesirable features.

As a first step, one could explore designing an efficient *interactive* protocol to aggregate the NIZK proofs with at most linear communication (in the number of parties n). This would allow the parties to invoke it at each level of the recursion to aggregate the transcripts. In this context, it would also be desirable to avoid pairings completely, making the scheme suitable for applications such as Schnorr signatures or ElGamal encryption.

Open Problem 12. *Construct a (recursive) synchronous DKG protocol with $O(\lambda n^2 \log n)$ communication that shares a secret key over a finite field $\mathbb{Z}_p$, potentially with adaptive security and without the use of pairings.*

While our DKG protocol achieves sub-cubic communication complexity (in the number of parties n), its recursive design fundamentally relies on the synchronous network assumption, where parties share a global clock, making it possible to detect silent malicious parties. In

[7] We always assume an adversary that can corrupt up to $t \in \Theta(n)$ parties.

[8] At a high level, each party generates a PVSS transcript for a field element, erases the underlying secrets, and disperses the transcript using linear erasure codes. A common coin (provided by our DKG) then elects a random committee of $\Theta(\lambda)$ parties, whose transcripts are reconstructed jointly. Finally, the parties run $\Theta(\lambda)$ instances of a quadratic consensus protocol to agree on these transcripts, from which the secret key shares are obtained.

contrast, in asynchrony, time is unbounded, and there is, in general, no way to distinguish between a silent malicious party and a slow honest party. In particular, any attempt at a round-based, recursive protocol can get stuck waiting forever. Interestingly, in a recent breakthrough, Abraham et al. [Abr+25] presented a recursive asynchronous DKG protocol. Specifically, for any $k \in [\log(n)]$, they give an adaptively secure asynchronous DKG protocol with $O(\lambda^2 n^{2+1/k})$ communication and $O(k)$ rounds. An interesting open question is whether their techniques can be transferred to the synchronous setting.

Open Problem 13. *Transfer the techniques from the asynchronous DKG protocol by Abraham et al. [Abr+25] to the synchronous setting to obtain a synchronous DKG protocol with $O(\lambda^2 n^{2+1/k})$ communication and $O(k)$ rounds (with adaptive security), where $k \in [\log(n)]$.*

The factor λ^2 in the communication complexity yields a significant overhead for practical applications. For example, to achieve an 128-bit security level, the necessary choices of $\lambda \geq 128$, incurs a factor into the communication complexity as heavy as $\sqrt{n}$, even for relatively large systems of size $n \approx 16000$ (and much heavier for smaller ones). Thus, it is highly desirable to avoid the λ^2 overhead. But this appears highly challenging and fundamentally new ideas are required.

Open Problem 14. *Construct an asynchronous DKG protocol with $O(\lambda n^{2+1/k})$ communication and $O(k)$ rounds where $k \in [\log(n)]$, potentially with adaptive security.*

The λ^2-overhead arises for the following reason. Each party is associated with a uniformly sampled random committee of size $\Theta(\lambda)$, from which it obtains some secrets. Probabilistic concentration bounds then guarantee an honest majority,[9] ensuring that the combined secret remains hidden from the adversary. Such techniques are common in distributed computing, but they critically rely on secure erasures of internal state in the presence of an adaptive adversary. Without secure erasures, an adaptive adversary could simply observe the committee members and corrupt them all, thereby learning their secrets and the combined secret.

Open Problem 15. *Construct an adaptively secure asynchronous DKG protocol with near-quadratic communication complexity without the use of secure erasures, potentially with (expected) $O(1)$ rounds.*

7.1.4 Open Problems Related to Network-Agnostic Security – Chapter 6

In Chapter 6, we presented a network-agnostic DKG protocol whose communication complexity matches the state-of-the-art DKG protocols in the purely synchronous and asynchronous settings. Along the way, we also designed an honest-majority synchronous broadcast protocol with (potentially) optimal communication complexity $O(n\ell + \lambda n^2)$ from a plain public key infrastructure (PKI), where ℓ denotes the length of the input message.[10] Our protocol, however, assumes static adversaries and fails in the adaptive setting. On the other hand, Momose and Ren presented a synchronous Byzantine agreement protocol with the same communication

[9] Note that in asynchrony, there are at most $t < n/3$ Byzantine parties. The concentration bounds ensure that the fraction of honest parties in each uniformly sampled committee remains close to $2/3$.

[10] The Dolev–Reischuk lower bound for deterministic broadcast [DR85] states that $\Omega(n^2)$ messages are necessary when $t \in \Theta(n)$. However, in the cryptographic setting, we believe that most messages must be signed, suggesting a lower bound of $\Omega(\lambda n^2)$ bits.

complexity from plain PKI, secure against adaptive adversaries corrupting up to $t \leq (\frac{1}{2} - \varepsilon)n$ parties for any fixed constant $\varepsilon > 0$. This naturally raises the question of whether it is possible to close the gap and achieve the best of all worlds: (near-)quadratic communication, adaptive security, and optimal resilience. In a recent breakthrough by Gelles et al. [Gel+25] this open problem was resolved. Building on the work of Momose and Ren, the authors present a Byzantine agreement protocol with communication complexity $O(\lambda n^2 \log(n))$ bits, optimal resilience $t < n/2$, and adaptive security. Their protocol, however, relies on secure erasures of internal state to achieve adaptive security. This leaves open the following question.

Open Problem 16. *Construct an adaptively secure synchronous multivalued Byzantine agreement protocol with near-quadratic communication complexity and optimal resilience threshold $t < n/2$ from plain PKI without the use of secure erasures.*

Our network-agnostic DKG protocol has a communication complexity of $O(\lambda n^3)$ bits and terminates in $O(n)$ rounds. Given the recent advances in both synchronous and asynchronous DKG that achieve (near-)quadratic protocols with optimal resilience and adaptive security, it is natural to ask whether similar results can be achieved in the network-agnostic case.

Open Problem 17. *Construct a network-agnostic DKG protocol with near-quadratic communication complexity and optimal resilience threshold $t < n/2$, potentially with adaptive security and (expected) $O(1)$ rounds.*

Finally, our DKG protocol uses NIZK proofs to show correct computation of encrypted secrets. However, such proofs for verifiable encryption are significantly less efficient than Σ-protocol based proofs. It would therefore be desirable to have a network-agnostic DKG built solely from plain PKI primitives, avoiding the use of heavy cryptographic tools.

Open Problem 18. *Construct a network-agnostic DKG protocol with cubic communication complexity and optimal resilience $t < n/2$ without the use of heavy cryptographic tools.*

7.2 Conclusion

In this thesis, we have made substantial progress toward strengthening the security guarantees and improving the efficiency of distributed key generation (DKG) and its applications, such as threshold signatures and distributed randomness generation. Our contributions can be viewed along two main dimensions.

First, we showed that some existing security definitions are formulated too strongly to support advanced guarantee - and, in certain cases, even to admit any meaningful security guarantees at all. By refining these definitions to better reflect their intended use cases, we were able to derive stronger and more practically relevant security results. We demonstrated this both for publicly verifiable secret sharing (PVSS), in the context of distributed randomness beacons, and for DKG protocols, in the context of the threshold BLS signature. Second, we designed distributed cryptographic protocols that achieve improved efficiency and stronger security guarantees than previously known constructions. Our protocols are well-suited to serve as building blocks for practical decentralized systems. We achieve these results by carefully adapting techniques from related domains, such as recursive methods from consensus, to the context of DKG.

We hope that our work will serve as a foundation for future advances in distributed cryptography. With this, we conclude our dissertation.

Bibliography

Other references

[Abr+25] Abraham, I., Bacho, R., Loss, J., and Stern, G. *Nearly Quadratic Asynchronous Distributed Key Generation*. Cryptology ePrint Archive, Paper 2025/006. 2025.

[Abr+19a] Abraham, I., Chan, T.-H. H., Dolev, D., Nayak, K., Pass, R., Ren, L., and Shi, E. Communication complexity of byzantine agreement, revisited. In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. PODC '19. Association for Computing Machinery, Toronto ON, Canada, 2019, 317–326.

[Abr+19b] Abraham, I., Devadas, S., Dolev, D., Nayak, K., and Ren, L. Synchronous byzantine agreement with expected $O(1)$ rounds, expected $O(n^2)$ communication, and optimal resilience. In: *FC 2019: 23rd International Conference on Financial Cryptography and Data Security*. Ed. by Goldberg, I. and Moore, T. Vol. 11598. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Frigate Bay, St. Kitts and Nevis, 2019, 320–334.

[Abr+22a] Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., and Stern, G. *Bingo: Adaptively Secure Packed Asynchronous Verifiable Secret Sharing and Asynchronous Distributed Key Generation*. Cryptology ePrint Archive, Report 2022/1759. 2022.

[Abr+22b] Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., and Stern, G. *Bingo: Adaptively Secure Packed Asynchronous Verifiable Secret Sharing and Asynchronous Distributed Key Generation*. Cryptology ePrint Archive, Paper 2022/1759. https://eprint.iacr.org/2022/1759. 2022.

[Abr+23] Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., and Stern, G. Bingo: adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In: *Advances in Cryptology – CRYPTO 2023, Part I*. Ed. by Handschuh, H. and Lysyanskaya, A. Vol. 14081. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Santa Barbara, CA, USA, 2023, 39–70.

[Abr+21a] Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., and Tomescu, A. Reaching consensus for asynchronous distributed key generation. In: *40th ACM Symposium Annual on Principles of Distributed Computing*. Ed. by Miller, A., Censor-Hillel, K., and Korhonen, J. H. Association for Computing Machinery, Virtual Event, Italy, 2021, 363–373.

[Abr+18] Abraham, I., Malkhi, D., Nayak, K., and Ren, L. *Dfinity Consensus, Explored*. Cryptology ePrint Archive, Report 2018/1153. 2018.

[AMS19] Abraham, I., Malkhi, D., and Spiegelman, A. Asymptotically optimal validated asynchronous byzantine agreement. In: *38th ACM Symposium Annual on Principles of Distributed Computing*. Ed. by Robinson, P. and Ellen, F. Association for Computing Machinery, Toronto, ON, Canada, 2019, 337–346.

[Abr+21b] Abram, D., Nof, A., Orlandi, C., Scholl, P., and Shlomovits, O. *Low-Bandwidth Threshold ECDSA via Pseudorandom Correlation Generators*. Cryptology ePrint Archive, Report 2021/1587. 2021.

[Ale+22a] Alexandru, A. B., Blum, E., Katz, J., and Loss, J. *State Machine Replication under Changing Network Conditions*. Cryptology ePrint Archive, Paper 2022/698. https://eprint.iacr.org/2022/698. 2022.

[Ale+22b] Alexandru, A. B., Blum, E., Katz, J., and Loss, J. State machine replication under changing network conditions. In: *Advances in Cryptology – ASIACRYPT 2022, Part I*. Ed. by Agrawal, S. and Lin, D. Vol. 13791. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Taipei, Taiwan, 2022, 681–710.

[AVZ21] Alhaddad, N., Varia, M., and Zhang, H. High-threshold AVSS with optimal communication complexity. In: *FC 2021: 25th International Conference on Financial Cryptography and Data Security, Part II*. Ed. by Borisov, N. and Díaz, C. Vol. 12675. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Virtual Event, 2021, 479–498.

[AB21] Alper, H. K. and Burdges, J. Two-round trip Schnorr multi-signatures via delinearized witnesses. In: *Advances in Cryptology – CRYPTO 2021, Part I*. Ed. by Malkin, T. and Peikert, C. Vol. 12825. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Virtual Event, 2021, 157–188.

[ACC22] Appan, A., Chandramouli, A., and Choudhury, A. Perfectly-secure synchronous mpc with asynchronous fallback guarantees. In: *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*. PODC'22. Association for Computing Machinery, Salerno, Italy, 2022, 92–102.

[Ara+21] Aranha, D. F., Dalskov, A. P. K., Escudero, D., and Orlandi, C. Improved threshold signatures, proactive secret sharing, and input certification from LSS isomorphisms. In: *Progress in Cryptology - LATINCRYPT 2021: 7th International Conference on Cryptology and Information Security in Latin America*. Ed. by Longa, P. and Ràfols, C. Vol. 12912. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Bogotá, Colombia, 2021, 382–404.

[ACS22] Asharov, G., Cohen, R., and Shochat, O. Static vs. adaptive security in perfect MPC: A separation and the adaptive security of BGW. In: *ITC 2022: 3rd Conference on Information-Theoretic Cryptography*. Ed. by Dachman-Soled, D. Vol. 230. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Cambridge, MA, USA, 2022, 15:1–15:16.

[ACM05] Ateniese, G., Camenisch, J., and Medeiros, B. de. Untraceable rfid tags via insubvertible encryption. In: *Proceedings of the 12th ACM Conference on Computer and Communications Security*. CCS '05. Association for Computing Machinery, Alexandria, VA, USA, 2005, 92–101.

[ACR21] Attema, T., Cramer, R., and Rambaud, M. Compressed Σ-protocols for bilinear group arithmetic circuits and application to logarithmic transparent threshold signatures. In: *Advances in Cryptology – ASIACRYPT 2021, Part IV*. Ed. by Tibouchi, M. and Wang, H. Vol. 13093. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Singapore, 2021, 526–556.

[AMM18] Azouvi, S., McCorry, P., and Meiklejohn, S. *Winning the Caucus Race: Continuous Leader Election via Public Randomness*. 2018. arXiv: 1801.07965 [cs.CR].

[BC26] Bacho, R. and Chen, Y. *Earpicks: Tightly Secure Two-Round Multi- and Threshold Signatures*. Cryptology ePrint Archive, Paper 2026/572. 2026.

[Bac+25a] Bacho, R., Chen, Y., Loss, J., Tessaro, S., and Zhu, C. *Adaptively Secure Partially Non-Interactive Threshold Schnorr Signatures in the AGM*. Cryptology ePrint Archive, Paper 2025/1953. 2025.

[Bac+22] Bacho, R., Collins, D., Liu-Zhang, C.-D., and Loss, J. *Network-Agnostic Security Comes (Almost) for Free in DKG and MPC*. Cryptology ePrint Archive, Paper 2022/1369. 2022.

[Bac+23a] Bacho, R., Collins, D., Liu-Zhang, C.-D., and Loss, J. Network-agnostic security comes (almost) for free in DKG and MPC. In: *Advances in Cryptology – CRYPTO 2023, Part I*. Ed. by Handschuh, H. and Lysyanskaya, A. Vol. 14081. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Santa Barbara, CA, USA, 2023, 71–106.

[Bac+25b] Bacho, R., Das, S., Loss, J., and Ren, L. *Adaptively Secure Three-Round Threshold Schnorr Signatures from DDH*. Cryptology ePrint Archive, Paper 2025/1009. 2025.

[Bac+25c] Bacho, R., Das, S., Loss, J., and Ren, L. Adaptively secure three-round threshold schnorr signatures from ddh. In: *Advances in Cryptology – CRYPTO 2025*. Ed. by Tauman Kalai, Y. and Kamara, S. F. Springer Nature Switzerland, Cham, 2025, 390–422.

[Bac+25d] Bacho, R., Das, S., Loss, J., and Ren, L. Glacius: threshold schnorr signatures from ddh with full adaptive security. In: *Advances in Cryptology – EUROCRYPT 2025: 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4–8, 2025, Proceedings, Part II*. Springer-Verlag, Madrid, Spain, 2025, 304–334.

[Bac+25e] Bacho, R., Das, S., Loss, J., and Ren, L. Glacius: threshold schnorr signatures from DDH with full adaptive security. In: *Advances in Cryptology – EUROCRYPT 2025, Part II*. Ed. by Fehr, S. and Fouque, P.-A. Vol. 15602. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Madrid, Spain, 2025, 304–334.

[BK25] Bacho, R. and Kavousi, A. Sok: dlog-based distributed key generation. In: *2025 IEEE Symposium on Security and Privacy (SP)*. 2025, 614–632.

[Bac+23b] Bacho, R., Lenzen, C., Loss, J., Ochsenreither, S., and Papachristoudis, D. *GRand-Line: Adaptively Secure DKG and Randomness Beacon with (Log-)Quadratic Communication Complexity*. Cryptology ePrint Archive, Paper 2023/1887. 2023.

[Bac+24a] Bacho, R., Lenzen, C., Loss, J., Ochsenreither, S., and Papachristoudis, D. GRand-Line: adaptively secure DKG and randomness beacon with (log-)quadratic communication complexity. In: *ACM CCS 2024: 31st Conference on Computer and Communications Security*. Ed. by Luo, B., Liao, X., Xu, J., Kirda, E., and Lie, D. ACM Press, Salt Lake City, UT, USA, 2024, 941–955.

[BL22a] Bacho, R. and Loss, J. On the adaptive security of the threshold BLS signature scheme. In: *ACM CCS 2022: 29th Conference on Computer and Communications Security*. Ed. by Yin, H., Stavrou, A., Cremers, C., and Shi, E. ACM Press, Los Angeles, CA, USA, 2022, 193–207.

[BL22b] Bacho, R. and Loss, J. *On the Adaptive Security of the Threshold BLS Signature Scheme*. Cryptology ePrint Archive, Paper 2022/534. 2022.

[BL22c] Bacho, R. and Loss, J. On the adaptive security of the threshold bls signature scheme. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. CCS '22. Association for Computing Machinery, Los Angeles, CA, USA, 2022, 193–207.

[BL23a] Bacho, R. and Loss, J. Adaptively secure (aggregatable) PVSS and application to distributed randomness beacons. In: *ACM CCS 2023: 30th Conference on Computer and Communications Security*. Ed. by Meng, W., Jensen, C. D., Cremers, C., and Kirda, E. ACM Press, Copenhagen, Denmark, 2023, 1791–1804.

[BL23b] Bacho, R. and Loss, J. *Adaptively Secure (Aggregatable) PVSS and Application to Distributed Randomness Beacons*. Cryptology ePrint Archive, Paper 2023/1348. 2023.

[BL23c] Bacho, R. and Loss, J. *Adaptively Secure (Aggregatable) PVSS and Application to Distributed Randomness Beacons*. Cryptology ePrint Archive, Report 2023/1348. 2023.

[Bac+24b] Bacho, R., Loss, J., Stern, G., and Wagner, B. *HARTS: High-Threshold, Adaptively Secure, and Robust Threshold Schnorr Signatures*. Cryptology ePrint Archive, Paper 2024/280. https://eprint.iacr.org/2024/280. 2024.

[Bac+24c] Bacho, R., Loss, J., Stern, G., and Wagner, B. HARTS: high-threshold, adaptively secure, and robust threshold Schnorr signatures. In: *Advances in Cryptology – ASIACRYPT 2024, Part III*. Ed. by Chung, K.-M. and Sasaki, Y. Vol. 15486. Lecture Notes in Computer Science. Springer, Singapore, Singapore, Kolkata, India, 2024, 104–140.

[Bac+24d] Bacho, R., Loss, J., Tessaro, S., Wagner, B., and Zhu, C. Twinkle: threshold signatures from DDH with full adaptive security. In: *Advances in Cryptology – EUROCRYPT 2024, Part I*. Ed. by Joye, M. and Leander, G. Vol. 14651. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Zurich, Switzerland, 2024, 429–459.

[BW24a] Bacho, R. and Wagner, B. Tightly secure non-interactive BLS multi-signatures. In: *Advances in Cryptology – ASIACRYPT 2024, Part II*. Ed. by Chung, K.-M. and Sasaki, Y. Vol. 15485. Lecture Notes in Computer Science. Springer, Singapore, Singapore, Kolkata, India, 2024, 397–422.

[BW24b] Bacho, R. and Wagner, B. *Tightly Secure Threshold Signatures over Pairing-Free Groups*. Cryptology ePrint Archive, Paper 2024/1557. 2024.

[BW25a] Bacho, R. and Wagner, B. *T-Spoon: Tightly Secure Two-Round Multi-Signatures with Key Aggregation*. Cryptology ePrint Archive, Paper 2025/840. 2025.

[BW25b] Bacho, R. and Wagner, B. T-spoon: tightly secure two-round multi-signatures with key aggregation. In: *Advances in Cryptology – CRYPTO 2025*. Ed. by Tauman Kalai, Y. and Kamara, S. F. Springer Nature Switzerland, Cham, 2025, 256–290.

[BW26] Bacho, R. and Wagner, B. Tightly secure threshold signatures over pairing-free groups. *IACR Communications in Cryptology* 2, 4 (Jan. 8, 2026).

[Ban+24] Bandarupalli, A., Bhat, A., Bagchi, S., Kate, A., and Reiter, M. K. Random beacons in Monte Carlo: efficient asynchronous random beacon *without* threshold cryptography. In: *ACM CCS 2024: 31st Conference on Computer and Communications Security*. Ed. by Luo, B., Liao, X., Xu, J., Kirda, E., and Lie, D. ACM Press, Salt Lake City, UT, USA, 2024, 2621–2635.

[BP97] Bari, N. and Pfitzmann, B. Collision-free accumulators and fail-stop signature schemes without trees. In: *Advances in Cryptology – EUROCRYPT'97*. Ed. by Fumy, W. Vol. 1233. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Konstanz, Germany, 1997, 480–494.

[BFL20] Bauer, B., Fuchsbauer, G., and Loss, J. A classification of computational assumptions in the algebraic group model. In: *Advances in Cryptology – CRYPTO 2020, Part II*. Ed. by Micciancio, D. and Ristenpart, T. Vol. 12171. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Santa Barbara, CA, USA, 2020, 121–151.

[BFP21] Bauer, B., Fuchsbauer, G., and Plouviez, A. The one-more discrete logarithm assumption in the generic group model. In: *Advances in Cryptology – ASIACRYPT 2021, Part IV*. Ed. by Tibouchi, M. and Wang, H. Vol. 13093. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Singapore, 2021, 587–617.

[Bau+20] Baum, C., David, B., Dowsley, R., Nielsen, J. B., and Oechsner, S. *CRAFT: Composable Randomness and Almost Fairness from Time*. Cryptology ePrint Archive, Report 2020/784. 2020.

[Bel+03] Bellare, M., Namprempre, C., Pointcheval, D., and Semanko, M. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology* 16, 3 (June 2003), 185–215.

[BN06] Bellare, M. and Neven, G. Multi-signatures in the plain public-key model and a general forking lemma. In: *ACM CCS 2006: 13th Conference on Computer and Communications Security*. Ed. by Juels, A., Wright, R. N., and De Capitani di Vimercati, S. ACM Press, Alexandria, Virginia, USA, 2006, 390–399.

[BR93] Bellare, M. and Rogaway, P. Random oracles are practical: A paradigm for designing efficient protocols. In: *ACM CCS 93: 1st Conference on Computer and Communications Security*. Ed. by Denning, D. E., Pyle, R., Ganesan, R., Sandhu, R. S., and Ashby, V. ACM Press, Fairfax, Virginia, USA, 1993, 62–73.

[BGP89] Berman, P., Garay, J. A., and Perry, K. J. Towards optimal distributed consensus (extended abstract). In: *30th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Research Triangle Park, NC, USA, 1989, 410–415.

[Bha+25] Bhangale, A., Liu-Zhang, C.-D., Loss, J., Nayak, K., and Yandamuri, S. Leader election with poly-logarithmic communication per party. In: *Advances in Cryptology – CRYPTO 2025*. Ed. by Tauman Kalai, Y. and Kamara, S. F. Springer Nature Switzerland, Cham, 2025, 37–68.

[Bha+23] Bhat, A., Shrestha, N., Kate, A., and Nayak, K. OptRand: optimistically responsive reconfigurable distributed randomness. In: *ISOC Network and Distributed System Security Symposium – NDSS 2023*. The Internet Society, San Diego, CA, USA, 2023.

[Bha+21] Bhat, A., Shrestha, N., Luo, Z., Kate, A., and Nayak, K. RandPiper - reconfiguration-friendly random beacons with quadratic communication. In: *ACM CCS 2021: 28th Conference on Computer and Communications Security*. Ed. by Vigna, G. and Shi, E. ACM Press, Virtual Event, Republic of Korea, 2021, 3502–3524.

[Blu+20] Blum, E., Katz, J., Liu-Zhang, C.-D., and Loss, J. Asynchronous byzantine agreement with subquadratic communication. In: *TCC 2020: 18th Theory of Cryptography Conference, Part I*. Ed. by Pass, R. and Pietrzak, K. Vol. 12550. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Durham, NC, USA, 2020, 353–380.

[BKL19] Blum, E., Katz, J., and Loss, J. Synchronous consensus with optimal asynchronous fallback guarantees. In: *TCC 2019: 17th Theory of Cryptography Conference, Part I*. Ed. by Hofheinz, D. and Rosen, A. Vol. 11891. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Nuremberg, Germany, 2019, 131–150.

[BKL21] Blum, E., Katz, J., and Loss, J. Tardigrade: an atomic broadcast protocol for arbitrary network conditions. In: *Advances in Cryptology – ASIACRYPT 2021, Part II*. Ed. by Tibouchi, M. and Wang, H. Vol. 13091. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Singapore, 2021, 547–572.

[BLZL20] Blum, E., Liu-Zhang, C.-D., and Loss, J. Always have a backup plan: fully secure synchronous mpc with asynchronous fallback. In: *Advances in Cryptology – CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II*. Springer-Verlag, Santa Barbara, CA, USA, 2020, 707–731.

[Bol03] Boldyreva, A. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In: *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*. Ed. by Desmedt, Y. Vol. 2567. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Miami, FL, USA, 2003, 31–46.

[Bon+18] Boneh, D., Bonneau, J., Bünz, B., and Fisch, B. Verifiable delay functions. In: *Advances in Cryptology – CRYPTO 2018, Part I*. Ed. by Shacham, H. and Boldyreva, A. Vol. 10991. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Santa Barbara, CA, USA, 2018, 757–788.

[BBF19] Boneh, D., Bünz, B., and Fisch, B. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In: *Advances in Cryptology – CRYPTO 2019, Part I*. Ed. by Boldyreva, A. and Micciancio, D. Vol. 11692. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Santa Barbara, CA, USA, 2019, 561–586.

[BDN18] Boneh, D., Drijvers, M., and Neven, G. Compact multi-signatures for smaller blockchains. In: *Advances in Cryptology – ASIACRYPT 2018, Part II*. Ed. by Peyrin, T. and Galbraith, S. Vol. 11273. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Brisbane, Queensland, Australia, 2018, 435–464.

[BGG19] Boneh, D., Gennaro, R., and Goldfeder, S. Using level-1 homomorphic encryption to improve threshold DSA signatures for bitcoin wallet security. In: *Progress in Cryptology - LATINCRYPT 2017: 5th International Conference on Cryptology and Information Security in Latin America*. Ed. by Lange, T. and Dunkelman, O. Vol. 11368. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Havana, Cuba, 2019, 352–377.

[Bon+03] Boneh, D., Gentry, C., Lynn, B., and Shacham, H. Aggregate and verifiably encrypted signatures from bilinear maps. In: *Advances in Cryptology – EUROCRYPT 2003*. Ed. by Biham, E. Vol. 2656. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Warsaw, Poland, 2003, 416–432.

[BLS01] Boneh, D., Lynn, B., and Shacham, H. Short signatures from the Weil pairing. In: *Advances in Cryptology – ASIACRYPT 2001*. Ed. by Boyd, C. Vol. 2248. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Gold Coast, Australia, 2001, 514–532.

[BD22] Brandão, L. and Davidson, M. *Notes on threshold EdDSA/Schnorr signatures.* Tech. rep. 2022.

[BP25] Brandão, L. and Peralta, R. *NIST first call for multi-party threshold schemes.* Tech. rep. 2025.

[BGB17] Bünz, B., Goldfeder, S., and Bonneau, J. Proofs-of-delay and randomness beacons in ethereum. In: 2017.

[Cac+02] Cachin, C., Kursawe, K., Lysyanskaya, A., and Strobl, R. Asynchronous verifiable secret sharing and proactive cryptosystems. In: *ACM CCS 2002: 9th Conference on Computer and Communications Security*. Ed. by Atluri, V. ACM Press, Washington, DC, USA, 2002, 88–97.

[Cac+01] Cachin, C., Kursawe, K., Petzold, F., and Shoup, V. Secure and efficient asynchronous broadcast protocols. In: *Annual International Cryptology Conference*. Springer. 2001, 524–541.

[CKS00] Cachin, C., Kursawe, K., and Shoup, V. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In: *19th ACM Symposium Annual on Principles of Distributed Computing*. Ed. by Neiger, G. Association for Computing Machinery, Portland, OR, USA, 2000, 123–132.

[CKS05] Cachin, C., Kursawe, K., and Shoup, V. Random oracles in Constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology* 18, 3 (July 2005), 219–246.

[CM07] Cafure, A. and Matera, G. An effective bertini theorem and the number of rational points of a normal complete intersection over a finite field. *Acta Arithmetica* 130 (2007), 19–35.

[Cam+21] Camenisch, J., Drijvers, M., Hanke, T., Pignolet, Y.-A., Shoup, V., and Williams, D. *Internet Computer Consensus*. Cryptology ePrint Archive, Report 2021/632. 2021.

[CHP07] Camenisch, J., Hohenberger, S., and Pedersen, M. Ø. Batch verification of short signatures. In: *Advances in Cryptology – EUROCRYPT 2007*. Ed. by Naor, M. Vol. 4515. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Barcelona, Spain, 2007, 246–263.

[Can+96] Canetti, R., Feige, U., Goldreich, O., and Naor, M. Adaptively secure multi-party computation. In: *28th Annual ACM Symposium on Theory of Computing*. ACM Press, Philadephia, PA, USA, 1996, 639–648.

[Can+20] Canetti, R., Gennaro, R., Goldfeder, S., Makriyannis, N., and Peled, U. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In: *ACM CCS 2020: 27th Conference on Computer and Communications Security*. Ed. by Ligatti, J., Ou, X., Katz, J., and Vigna, G. ACM Press, Virtual Event, USA, 2020, 1769–1787.

[Can+21] Canetti, R., Gennaro, R., Goldfeder, S., Makriyannis, N., and Peled, U. *UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts*. Cryptology ePrint Archive, Report 2021/060. 2021.

[Can+99] Canetti, R., Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T. Adaptive security for threshold cryptosystems. In: *Advances in Cryptology – CRYPTO'99*. Ed. by Wiener, M. J. Vol. 1666. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA, 1999, 98–115.

[CR93] Canetti, R. and Rabin, T. Fast asynchronous byzantine agreement with optimal resilience. In: *25th Annual ACM Symposium on Theory of Computing*. ACM Press, San Diego, CA, USA, 1993, 42–51.

[CD17] Cascudo, I. and David, B. SCRAPE: scalable randomness attested by public entities. In: *ACNS 2017: 15th International Conference on Applied Cryptography and Network Security*. Ed. by Gollmann, D., Miyaji, A., and Kikuchi, H. Vol. 10355. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Kanazawa, Japan, 2017, 537–556.

[CD20] Cascudo, I. and David, B. ALBATROSS: publicly AttestabLe BATched Randomness based On Secret Sharing. In: *Advances in Cryptology – ASIACRYPT 2020, Part III*. Ed. by Moriai, S. and Wang, H. Vol. 12493. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Daejeon, South Korea, 2020, 311–341.

[CD24] Cascudo, I. and David, B. Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. In: *Advances in Cryptology – EUROCRYPT 2024, Part V*. Ed. by Joye, M. and Leander, G. Vol. 14655. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Zurich, Switzerland, 2024, 216–248.

[CF13] Catalano, D. and Fiore, D. Vector commitments and their applications. In: *International Workshop on Public Key Cryptography*. Springer. 2013, 55–72.

[Cat92] Catanese, F. Chow varieties, hilbert schemes, and moduli spaces of surfaces of general type. *Journal of Algebraic Geometry* 1 (Jan. 1992).

[CPS20] Chan, T.-H. H., Pass, R., and Shi, E. Sublinear-round byzantine agreement under corrupt majority. In: *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II*. Ed. by Kiayias, A., Kohlweiss, M., Wallden, P., and Zikas, V. Vol. 12111. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Edinburgh, UK, 2020, 246–265.

[CP93] Chaum, D. and Pedersen, T. P. Wallet databases with observers. In: *Advances in Cryptology – CRYPTO'92*. Ed. by Brickell, E. F. Vol. 740. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA, 1993, 89–105.

[Che25] Chen, Y. Dazzle: improved adaptive threshold signatures from DDH. In: *PKC 2025: 28th International Conference on Theory and Practice of Public Key Cryptography, Part III*. Ed. by Jager, T. and Pan, J. Vol. 15676. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Røros, Norway, 2025, 233–261.

[CSS19] Cherniaeva, A., Shirobokov, I., and Shlomovits, O. *Homomorphic Encryption Random Beacon*. Cryptology ePrint Archive, Report 2019/1320. 2019.

[Chi22] Chia. *Chia Network FAQ*. Website. https://www.chia.net/faq/. 2022.

[Cho+23] Choi, K., Arun, A., Tyagi, N., and Bonneau, J. *Bicorn: An optimistically efficient distributed randomness beacon*. Cryptology ePrint Archive, Report 2023/221. 2023.

[CMB23] Choi, K., Manoj, A., and Bonneau, J. Sok: distributed randomness beacons. In: *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 2023, 75–92.

[Cho+85a] Chor, B., Goldwasser, S., Micali, S., and Awerbuch, B. Verifiable secret sharing and achieving simultaneity in the presence of faults. *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)* (1985), 383–395.

[Cho+85b] Chor, B., Goldwasser, S., Micali, S., and Awerbuch, B. Verifiable secret sharing and achieving simultaneity in the presence of faults. In: *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*. IEEE. 1985, 383–395.

[Cho+85c] Chor, B., Goldwasser, S., Micali, S., and Awerbuch, B. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In: *26th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Portland, Oregon, 1985, 383–395.

[CKS20] Cohen, S., Keidar, I., and Spiegelman, A. Not a COINcidence: Sub-Quadratic Asynchronous Byzantine Agreement WHP. In: *34th International Symposium on Distributed Computing (DISC 2020)*. Ed. by Attiya, H. Vol. 179. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2020, 25:1–25:17.

[con22] contributors, arkworks. `arkworks` *zkSNARK ecosystem*. 2022.

[con23] contributors, T. *Tokio library for networking in Rust*. 2023.

[Cor02] Coron, J.-S. Optimal security proofs for PSS and other signature schemes. In: *Advances in Cryptology – EUROCRYPT 2002*. Ed. by Knudsen, L. R. Vol. 2332. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Amsterdam, The Netherlands, 2002, 272–287.

[Cri+25] Crites, E., Katz, J., Komlo, C., Tessaro, S., and Zhu, C. *On the Adaptive Security of FROST*. Cryptology ePrint Archive, Paper 2025/1061. 2025.

[CKM21] Crites, E., Komlo, C., and Maller, M. *How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures*. Cryptology ePrint Archive, Report 2021/1375. 2021.

[CKM23a] Crites, E., Komlo, C., and Maller, M. *Fully Adaptive Schnorr Threshold Signatures*. Cryptology ePrint Archive, Report 2023/445. 2023.

[CKM23b] Crites, E. C., Komlo, C., and Maller, M. Fully adaptive Schnorr threshold signatures. In: *Advances in Cryptology – CRYPTO 2023, Part I*. Ed. by Handschuh, H. and Lysyanskaya, A. Vol. 14081. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Santa Barbara, CA, USA, 2023, 678–709.

[Dal+20] Dalskov, A. P. K., Orlandi, C., Keller, M., Shrishak, K., and Shulman, H. Securing DNSSEC keys via threshold ECDSA from generic MPC. In: *ESORICS 2020: 25th European Symposium on Research in Computer Security, Part II*. Ed. by Chen, L., Li, N., Liang, K., and Schneider, S. A. Vol. 12309. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Guildford, UK, 2020, 654–673.

[Das+22a] Das, S., Krishnan, V., Isaac, I. M., and Ren, L. Spurt: scalable distributed randomness beacon with transparent setup. In: *2022 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 2022, 2502–2517.

[Das+24] Das, S., Pinkas, B., Tomescu, A., and Xiang, Z. *Distributed Randomness using Weighted VRFs*. Cryptology ePrint Archive, Paper 2024/198. https://eprint.iacr.org/2024/198. 2024.

[DR24] Das, S. and Ren, L. Adaptively secure BLS threshold signatures from DDH and co-CDH. In: *Advances in Cryptology – CRYPTO 2024, Part VII*. Ed. by Reyzin, L. and Stebila, D. Vol. 14926. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Santa Barbara, CA, USA, 2024, 251–284.

[Das+23] Das, S., Xiang, Z., Kokoris-Kogias, L., and Ren, L. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. In: *USENIX Security 2023: 32nd USENIX Security Symposium*. Ed. by Calandrino, J. A. and Troncoso, C. USENIX Association, Anaheim, CA, USA, 2023, 5359–5376.

[DXR22] Das, S., Xiang, Z., and Ren, L. *Powers of Tau in Asynchrony*. Cryptology ePrint Archive, Paper 2022/1683. https://eprint.iacr.org/2022/1683. 2022.

[Das+22b] Das, S., Yurek, T., Xiang, Z., Miller, A., Kokoris-Kogias, L., and Ren, L. Practical asynchronous distributed key generation. In: *2022 IEEE Symposium on Security and Privacy (SP)*. 2022, 2518–2534.

[Das+22c] Das, S., Yurek, T., Xiang, Z., Miller, A. K., Kokoris-Kogias, L., and Ren, L. Practical asynchronous distributed key generation. In: *2022 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 2022, 2518–2534.

[Dav+18] David, B., Gazi, P., Kiayias, A., and Russell, A. Ouroboros praos: an adaptively-secure, semi-synchronous proof-of-stake blockchain. In: *Advances in Cryptology – EUROCRYPT 2018, Part II*. Ed. by Nielsen, J. B. and Rijmen, V. Vol. 10821. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Tel Aviv, Israel, 2018, 66–98.

[DHLZ21] Deligios, G., Hirt, M., and Liu-Zhang, C.-D. Round-efficient byzantine agreement and multi-party computation with asynchronous fallback. In: *TCC 2021: 19th Theory of Cryptography Conference, Part I*. Ed. by Nissim, K. and Waters, B. Vol. 13042. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Raleigh, NC, USA, 2021, 623–653.

[Del74] Deligne, P. La conjecture de weil : i. fre. *Publications Mathématiques de l'IHÉS* 43 (1974), 273–307.

[Des88] Desmedt, Y. Society and group oriented cryptography: A new concept. In: *Advances in Cryptology – CRYPTO'87*. Ed. by Pomerance, C. Vol. 293. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA, 1988, 120–127.

[DF90] Desmedt, Y. and Frankel, Y. Threshold cryptosystems. In: *Advances in Cryptology – CRYPTO'89*. Ed. by Brassard, G. Vol. 435. Lecture Notes in Computer Science. Springer, New York, USA, Santa Barbara, CA, USA, 1990, 307–315.

[DMS04] Dingledine, R., Mathewson, N., and Syverson, P. F. Tor: the second-generation onion router. In: *USENIX Security 2004: 13th USENIX Security Symposium*. Ed. by Blaze, M. USENIX Association, San Diego, CA, USA, 2004, 303–320.

[DS83a] Dolev, D. and Strong, H. R. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing* 12, 4 (1983), 656–666. eprint: https://doi.org/10.1137/0212045.

[DY83] Dolev, D. and Yao, A. On the security of public key protocols. *IEEE Transactions on Information Theory* 29, 2 (1983), 198–208.

[DR85] Dolev, D. and Reischuk, R. Bounds on information exchange for byzantine agreement. *J. ACM* 32, 1 (1985), 191–204.

[DS83b] Dolev, D. and Strong, H. R. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing* 12, 4 (1983), 656–666.

[Dra18] Drake, J. Minimal vdf randomness beacon (2018).

[Dra20] Drand. *Drand - A Distributed Randomness Beacon Daemon*. GitHub repository. 2020.

[Dri+19] Drijvers, M., Edalatnejad, K., Ford, B., Kiltz, E., Loss, J., Neven, G., and Stepanovs, I. On the security of two-round multi-signatures. In: May 2019, 1084–1101.

[Fel87a] Feldman, P. A practical scheme for non-interactive verifiable secret sharing. In: *28th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Los Angeles, CA, USA, 1987, 427–437.

[Fel87b] Feldman, P. A practical scheme for non-interactive verifiable secret sharing. In: *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*. IEEE. 1987, 427–438.

[FLT24] Feng, H., Lu, Z., and Tang, Q. *Breaking the Cubic Barrier: Distributed Key and Randomness Generation through Deterministic Sharding*. Cryptology ePrint Archive, Paper 2024/168. https://eprint.iacr.org/2024/168. 2024.

[FLP85] Fischer, M. J., Lynch, N. A., and Paterson, M. S. Impossibility of distributed consensus with one faulty process. *J. ACM* 32, 2 (Apr. 1985), 374–382.

[FKL18] Fuchsbauer, G., Kiltz, E., and Loss, J. The algebraic group model and its applications. In: *Advances in Cryptology – CRYPTO 2018, Part II*. Ed. by Shacham, H. and Boldyreva, A. Vol. 10992. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Santa Barbara, CA, USA, 2018, 33–62.

[FPS20] Fuchsbauer, G., Plouviez, A., and Seurin, Y. Blind Schnorr signatures and signed ElGamal encryption in the algebraic group model. In: *Advances in Cryptology – EUROCRYPT 2020, Part II*. Ed. by Canteaut, A. and Ishai, Y. Vol. 12106. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Zagreb, Croatia, 2020, 63–95.

[Ful98] Fulton, W. Intersection theory. *Springer New York, NY* XIII (June 1998), 470.

[Gao+21] Gao, Y., Lu, Y., Lu, Z., Tang, Q., Xu, J., and Zhang, Z. *Efficient Asynchronous Byzantine Agreement without Private Setups*. Cryptology ePrint Archive, Report 2021/810. 2021.

[Gao+22] Gao, Y., Lu, Y., Lu, Z., Tang, Q., Xu, J., and Zhang, Z. Efficient asynchronous byzantine agreement without private setups. In: *42nd IEEE International Conference on Distributed Computing Systems, ICDCS 2022, Bologna, Italy, July 10-13, 2022*. IEEE, 2022, 246–257.

[Gat21] Gathmann, A. Algebraic geometry (class notes) (2021/2022).

[Gel+25] Gelles, R., Lenzen, C., Loss, J., and Yandamuri, S. Nearly optimal parallel broadcast in the plain public key model. In: *Advances in Cryptology – CRYPTO 2025*. Ed. by Tauman Kalai, Y. and Kamara, S. F. Springer Nature Switzerland, Cham, 2025, 101–134.

[GK24] Gelles, Y. and Komargodski, I. Scalable agreement protocols with optimal optimistic efficiency. In: *Security and Cryptography for Networks*. Ed. by Galdi, C. and Phan, D. H. Springer Nature Switzerland, Cham, 2024, 297–319.

[GG18] Gennaro, R. and Goldfeder, S. Fast multiparty threshold ECDSA with fast trustless setup. In: *ACM CCS 2018: 25th Conference on Computer and Communications Security*. Ed. by Lie, D., Mannan, M., Backes, M., and Wang, X. ACM Press, Toronto, ON, Canada, 2018, 1179–1194.

[GGN16] Gennaro, R., Goldfeder, S., and Narayanan, A. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In: *ACNS 2016: 14th International Conference on Applied Cryptography and Network Security*. Ed. by Manulis, M., Sadeghi, A.-R., and Schneider, S. Vol. 9696. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Guildford, UK, 2016, 156–174.

[Gen+99] Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T. Secure distributed key generation for discrete-log based cryptosystems. In: *Advances in Cryptology – EUROCRYPT'99*. Ed. by Stern, J. Vol. 1592. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Prague, Czech Republic, 1999, 295–310.

[Gen+07] Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology* 20, 1 (Jan. 2007), 51–83.

[Gen+21a] Gentry, C., Halevi, S., Krawczyk, H., Magri, B., Nielsen, J. B., Rabin, T., and Yakoubov, S. Yoso: you only speak once: secure mpc with stateless ephemeral roles. In: *Advances in Cryptology – CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II*. Springer-Verlag, Berlin, Heidelberg, 2021, 64–93.

[Gen+21b] Gentry, C., Halevi, S., Magri, B., Nielsen, J. B., and Yakoubov, S. Random-index PIR and applications. In: *TCC 2021: 19th Theory of Cryptography Conference, Part III*. Ed. by Nissim, K. and Waters, B. Vol. 13044. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Raleigh, NC, USA, 2021, 32–61.

[Gil+17a] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., and Zeldovich, N. Algorand: scaling byzantine agreements for cryptocurrencies. In: *Proceedings of the 26th Symposium on Operating Systems Principles*. SOSP '17. Association for Computing Machinery, Shanghai, China, 2017, 51–68.

[Gil+17b] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., and Zeldovich, N. *Algorand: Scaling Byzantine Agreements for Cryptocurrencies*. Cryptology ePrint Archive, Report 2017/454. 2017.

[GJ03] Goh, E.-J. and Jarecki, S. A signature scheme as secure as the Diffie-Hellman problem. In: *Advances in Cryptology – EUROCRYPT 2003*. Ed. by Biham, E. Vol. 2656. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Warsaw, Poland, 2003, 401–415.

[Gro06] Groth, J. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In: *Advances in Cryptology – ASIACRYPT 2006*. Ed. by Lai, X. and Chen, K. Vol. 4284. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Shanghai, China, 2006, 444–459.

[Gro21] Groth, J. *Non-interactive distributed key generation and key resharing*. Cryptology ePrint Archive, Report 2021/339. 2021.

[Guo+20] Guo, B., Lu, Z., Tang, Q., Xu, J., and Zhang, Z. Dumbo: faster asynchronous BFT protocols. In: *ACM CCS 2020: 27th Conference on Computer and Communications Security*. Ed. by Ligatti, J., Ou, X., Katz, J., and Vigna, G. ACM Press, Virtual Event, USA, 2020, 803–818.

[Gur+21a] Gurkan, K., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., and Tomescu, A. Aggregatable distributed key generation. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2021, 147–176.

[Gur+21b] Gurkan, K., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., and Tomescu, A. Aggregatable distributed key generation. In: *Advances in Cryptology – EUROCRYPT 2021, Part I*. Ed. by Canteaut, A. and Standaert, F.-X. Vol. 12696. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Zagreb, Croatia, 2021, 147–176.

[HYL20] Han, R., Yu, J., and Lin, H. *RandChain: Decentralised Randomness Beacon from Sequential Proof-of-Work*. Cryptology ePrint Archive, Report 2020/1033. 2020.

[HMW18] Hanke, T., Movahedi, M., and Williams, D. *DFINITY Technology Overview Series, Consensus System*. 2018. arXiv: 1805.04548 [cs.DC].

[Har77] Hartshorne, R. *Algebraic Geometry*. Vol. 52. Graduate Texts in Mathematics. Springer, 1977.

[HV09] Heidarvand, S. and Villar, J. L. Public verifiability from pairings in secret sharing schemes. In: *SAC 2008: 15th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Avanzi, R. M., Keliher, L., and Sica, F. Vol. 5381. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Sackville, New Brunswick, Canada, 2009, 294–308.

[JL00] Jarecki, S. and Lysyanskaya, A. Adaptively secure threshold cryptography: introducing concurrency, removing erasures. In: *Advances in Cryptology – EUROCRYPT 2000*. Ed. by Preneel, B. Vol. 1807. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Bruges, Belgium, 2000, 221–242.

[Jha11] Jhanwar, M. P. A practical (non-interactive) publicly verifiable secret sharing scheme. In: *Information Security Practice and Experience (ISPEC)*. Vol. 6672. Springer, 2011, 273–287.

[JVSN14] Jhanwar, M. P., Venkateswarlu, A., and Safavi-Naini, R. Paillier-based publicly verifiable (non-interactive) secret sharing. *Designs, Codes and Cryptography, Volume* 73, 2 (2014), 529–546.

[KK18] Kakvi, S. A. and Kiltz, E. Optimal security proofs for full domain hash, revisited. *Journal of Cryptology* 31, 1 (Jan. 2018), 276–306.

[KLX22] Kastner, J., Loss, J., and Xu, J. On pairing-free blind signature schemes in the algebraic group model. In: *PKC 2022: 25th International Conference on Theory and Practice of Public Key Cryptography, Part II*. Ed. by Hanaoka, G., Shikata, J., and Watanabe, Y. Vol. 13178. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Virtual Event, 2022, 468–497.

[KG09] Kate, A. and Goldberg, I. Distributed key generation for the internet. In: *2009 29th IEEE International Conference on Distributed Computing Systems*. 2009, 119–128.

[Kat+23] Kate, A., Mangipudi, E. V., Mukherjee, P., Saleem, H., and Thyagarajan, S. A. K. *Non-interactive VSS using Class Groups and Application to DKG*. Cryptology ePrint Archive, Report 2023/451. 2023.

[KRT24] Katsumata, S., Reichle, M., and Takemure, K. Adaptively secure 5 round threshold signatures from MLWE/MSIS and DL with rewinding. In: *Advances in Cryptology – CRYPTO 2024, Part VII*. Ed. by Reyzin, L. and Stebila, D. Vol. 14926. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Santa Barbara, CA, USA, 2024, 459–491.

[KW03] Katz, J. and Wang, N. Efficiency improvements for signature schemes with tight security reductions. In: *ACM CCS 2003: 10th Conference on Computer and Communications Security*. Ed. by Jajodia, S., Atluri, V., and Jaeger, T. ACM Press, Washington, DC, USA, 2003, 155–164.

[KWJ23] Kavousi, A., Wang, Z., and Jovanovic, P. *SoK: Public Randomness*. Cryptology ePrint Archive, Paper 2023/1121. https://eprint.iacr.org/2023/1121. 2023.

[Kia+17] Kiayias, A., Russell, A., David, B., and Oliynykov, R. Ouroboros: a provably secure proof-of-stake blockchain protocol. In: *Advances in Cryptology – CRYPTO 2017, Part I*. Ed. by Katz, J. and Shacham, H. Vol. 10401. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Santa Barbara, CA, USA, 2017, 357–388.

[KMS20] Kokoris-Kogias, E., Malkhi, D., and Spiegelman, A. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In: *ACM CCS 2020: 27th Conference on Computer and Communications Security*. Ed. by Ligatti, J., Ou, X., Katz, J., and Vigna, G. ACM Press, Virtual Event, USA, 2020, 1751–1767.

[KG20] Komlo, C. and Goldberg, I. FROST: flexible round-optimized Schnorr threshold signatures. In: *SAC 2020: 27th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Dunkelman, O., Jacobson Jr., M. J., and O'Flynn, C. Vol. 12804. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Halifax, NS, Canada (Virtual Event), 2020, 34–65.

[Kon+21] Kondi, Y., Magri, B., Orlandi, C., and Shlomovits, O. Refresh when you wake up: proactive threshold wallets with offline devices. In: *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 2021, 608–625.

[Lab21] Lab, S. *C++ library for Finite Fields and Elliptic Curves*. GitHub repository. 2021.

[LSP82a] Lamport, L., Shostak, R., and Pease, M. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.* 4, 3 (July 1982), 382–401.

[LSP82b] Lamport, L., Shostak, R. E., and Pease, M. C. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.* 4, 3 (1982), 382–401.

[LS22] Lenzen, C. and Sheikholeslami, S. A recursive early-stopping phase king protocol. In: *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*. PODC'22. Association for Computing Machinery, Salerno, Italy, 2022, 60–69.

[LJY14] Libert, B., Joye, M., and Yung, M. Born and raised distributively: fully distributed non-interactive adaptively-secure threshold signatures with short shares. In: *33rd ACM Symposium Annual on Principles of Distributed Computing*. Ed. by Halldórsson, M. M. and Dolev, S. Association for Computing Machinery, Paris, France, 2014, 303–312.

[Ark] *Library implementation for the BN254 pairing-friendly elliptic curve*. docs.rs. 2023.

[LN96] Lidl, R. and Niederreiter, H. Finite fields and their applications. In: 1996.

[Lin17] Lindell, Y. Fast secure two-party ECDSA signing. In: *Advances in Cryptology – CRYPTO 2017, Part II*. Ed. by Katz, J. and Shacham, H. Vol. 10402. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Santa Barbara, CA, USA, 2017, 613–644.

[Lin22] Lindell, Y. *Simple Three-Round Multiparty Schnorr Signing with Full Simulatability*. Cryptology ePrint Archive, Report 2022/374. 2022.

[LN18] Lindell, Y. and Nof, A. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In: *ACM CCS 2018: 25th Conference on Computer and Communications Security*. Ed. by Lie, D., Mannan, M., Backes, M., and Wang, X. ACM Press, Toronto, ON, Canada, 2018, 1837–1854.

[Lip12] Lipmaa, H. Secure accumulators from euclidean rings without trusted setup. In: *ACNS 2012: 10th International Conference on Applied Cryptography and Network Security*. Ed. by Bao, F., Samarati, P., and Zhou, J. Vol. 7341. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Singapore, 2012, 224–240.

[Los19] Loss, J. New techniques for the modular analysis of digital signature schemes. PhD thesis. Ruhr University Bochum, Germany, 2019.

[LM18] Loss, J. and Moran, T. *Combining Asynchronous and Synchronous Byzantine Agreement: The Best of Both Worlds*. Cryptology ePrint Archive, Report 2018/235. 2018.

[Lu+20] Lu, Y., Lu, Z., Tang, Q., and Wang, G. Dumbo-MVBA: optimal multi-valued validated asynchronous byzantine agreement, revisited. In: *39th ACM Symposium Annual on Principles of Distributed Computing*. Ed. by Emek, Y. and Cachin, C. Association for Computing Machinery, Virtual Event, Italy, 2020, 129–138.

[Luo22] Luo, Z. *Implementation for RandPiper*. Github. 2022.

[Mic17] Micali, S. Very simple and efficient byzantine agreement. In: *ITCS 2017: 8th Innovations in Theoretical Computer Science Conference*. Ed. by Papadimitriou, C. H. Vol. 4266. Leibniz International Proceedings in Informatics (LIPIcs), Berkeley, CA, USA, 2017, 6:1–6:1.

[MRV99] Micali, S., Rabin, M. O., and Vadhan, S. P. Verifiable random functions. In: *40th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, New York, NY, USA, 1999, 120–130.

[MR21a] Momose, A. and Ren, L. Multi-threshold byzantine fault tolerance. In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, 1686–1699.

[MR21b] Momose, A. and Ren, L. Optimal Communication Complexity of Authenticated Byzantine Agreement. In: *35th International Symposium on Distributed Computing (DISC 2021)*. Ed. by Gilbert, S. Vol. 209. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2021, 32:1–32:16.

[MR21c] Momose, A. and Ren, L. Optimal Communication Complexity of Authenticated Byzantine Agreement. In: *35th International Symposium on Distributed Computing (DISC 2021)*. Ed. by Gilbert, S. Vol. 209. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2021, 32:1–32:16.

[MR10] Mostéfaoui, A. and Raynal, M. Signature-free broadcast-based intrusion tolerance: never decide a byzantine value. In: *International Conference On Principles Of Distributed Systems*. Springer. 2010, 143–158.

[MR17] Mostéfaoui, A. and Raynal, M. Signature-free asynchronous byzantine systems: from multivalued to binary consensus with $t < n/3$, $O(n^2)$ messages, and constant time. *Acta Informatica* 54, 5 (2017), 501–520.

[Mou18] Mouhartem, F. *Implementation of Libert et al.'s Threshold BLS Signature*. Gitlab. https://gitlab.inria.fr/fmouhart/threshold-signature. 2018.

[Mö+18] Möser, M., Soska, K., Heilman, E., Lee, K., Heffan, H., Srivastava, S., Hogan, K., Hennessey, J., Miller, A., Narayanan, A., and Christin, N. *An Empirical Analysis of Traceability in the Monero Blockchain*. 2018. arXiv: 1704.04299 [cs.CR].

[Nak08] Nakamoto, S. Bitcoin: a peer-to-peer electronic cash system (2008).

[Nay23a] Nayak, K. Https://decentralizedthoughts.github.io/2023-01-05-player-replaceability-ii/ (2023).

[Nay23b] Nayak, K. Player replaceability - towards adaptive security and sub-quadratic communication simultaneously (part i) (2023).

[Nay+20a] Nayak, K., Ren, L., Shi, E., Vaidya, N. H., and Xiang, Z. Improved Extension Protocols for Byzantine Broadcast and Agreement. In: *34th International Symposium on Distributed Computing (DISC 2020)*. Ed. by Attiya, H. Vol. 179. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2020, 28:1–28:17.

[Nay+20b] Nayak, K., Ren, L., Shi, E., Vaidya, N. H., and Xiang, Z. Improved Extension Protocols for Byzantine Broadcast and Agreement. In: *34th International Symposium on Distributed Computing (DISC 2020)*. Ed. by Attiya, H. Vol. 179. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2020, 28:1–28:17.

[Ngu05] Nguyen, L. Accumulators from bilinear pairings and applications. In: *Topics in Cryptology – CT-RSA 2005*. Ed. by Menezes, A. Vol. 3376. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, San Francisco, CA, USA, 2005, 275–292.

[NRS21] Nick, J., Ruffing, T., and Seurin, Y. MuSig2: simple two-round Schnorr multi-signatures. In: *Advances in Cryptology – CRYPTO 2021, Part I*. Ed. by Malkin, T. and Peikert, C. Vol. 12825. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Virtual Event, 2021, 189–221.

[Nic+20] Nick, J., Ruffing, T., Seurin, Y., and Wuille, P. MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. In: *ACM CCS 2020: 27th Conference on Computer and Communications Security*. Ed. by Ligatti, J., Ou, X., Katz, J., and Vigna, G. ACM Press, Virtual Event, USA, 2020, 1717–1731.

[Nie02] Nielsen, J. B. Separating random oracle proofs from complexity theoretic proofs: the non-committing encryption case. In: *Advances in Cryptology — CRYPTO 2002*. Ed. by Yung, M. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, 111–126.

[Nik+22] Nikolaenko, V., Ragsdale, S., Bonneau, J., and Boneh, D. *Powers-of-Tau to the People: Decentralizing Setup Ceremonies*. Cryptology ePrint Archive, Report 2022/1592. 2022.

[Pai99] Paillier, P. Public-key cryptosystems based on composite degree residuosity classes. In: *Advances in Cryptology – EUROCRYPT'99*. Ed. by Stern, J. Vol. 1592. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Prague, Czech Republic, 1999, 223–238.

[PW23] Pan, J. and Wagner, B. Chopsticks: fork-free two-round multi-signatures from non-interactive assumptions. In: *Advances in Cryptology – EUROCRYPT 2023, Part V*. Ed. by Hazay, C. and Stam, M. Vol. 14008. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Lyon, France, 2023, 597–627.

[Pap23a] Papachristoudis, D. *Cryptography for GRandLine*. GitHub repository. 2023.

[Pap23b] Papachristoudis, D. *Implementation of GRandLine*. GitHub repository. 2023.

[Par+13] Parno, B., Howell, J., Gentry, C., and Raykova, M. Pinocchio: nearly practical verifiable computation. In: *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, Berkeley, CA, USA, 2013, 238–252.

[Ped91] Pedersen, T. P. A threshold cryptosystem without a trusted party (extended abstract) (rump session). In: *Advances in Cryptology – EUROCRYPT'91*. Ed. by Davies, D. W. Vol. 547. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Brighton, UK, 1991, 522–526.

[Ped92] Pedersen, T. P. Non-interactive and information-theoretic secure verifiable secret sharing. In: *Advances in Cryptology – CRYPTO'91*. Ed. by Feigenbaum, J. Vol. 576. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA, 1992, 129–140.

[RS60] Reed, I. S. and Solomon, G. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics* 8, 2 (1960), 300–304.

[RV05] Ruiz, A. and Villar, J. L. Publicly verifiable secret sharing from paillier's cryptosystem. In: *WEWoRC 2005 – Western European Workshop on Research in Cryptology*. Ed. by Wulf, C., Lucks, S., and Yau, P.-W. Gesellschaft für Informatik e.V., Bonn, 2005, 98–108.

[Sak+22] Sakemi, Y., Kobayashi, T., Saito, T., and Wahby, R. S. *Internet Research Task Force (IRTF) Draft for Pairing-Friendly Curves*. Nov. 2022.

[Sch+21] Schindler, P., Judmayer, A., Hittmeir, M., Stifter, N., and Weippl, E. R. RandRunner: distributed randomness from trapdoor VDFs with strong uniqueness. In: *ISOC Network and Distributed System Security Symposium – NDSS 2021*. The Internet Society, Virtual, 2021.

[Sch+19] Schindler, P., Judmayer, A., Stifter, N., and Weippl, E. *ETHDKG: Distributed Key Generation with Ethereum Smart Contracts*. Cryptology ePrint Archive, Report 2019/985. 2019.

[Sch+20] Schindler, P., Judmayer, A., Stifter, N., and Weippl, E. R. HydRand: efficient continuous distributed randomness. In: *2020 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 2020, 73–89.

[Sch90] Schnorr, C.-P. Efficient identification and signatures for smart cards. In: *Advances in Cryptology – CRYPTO'89*. Ed. by Brassard, G. Vol. 435. Lecture Notes in Computer Science. Springer, New York, USA, Santa Barbara, CA, USA, 1990, 239–252.

[Sch99] Schoenmakers, B. A simple publicly verifiable secret sharing scheme and its application to electronic. In: *Advances in Cryptology – CRYPTO'99*. Ed. by Wiener, M. J. Vol. 1666. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA, 1999, 148–164.

[Sho97] Shoup, V. Lower bounds for discrete logarithms and related problems. In: *Advances in Cryptology – EUROCRYPT'97*. Ed. by Fumy, W. Vol. 1233. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Konstanz, Germany, 1997, 256–266.

[Sho00] Shoup, V. Practical threshold signatures. In: *Advances in Cryptology – EUROCRYPT 2000*. Ed. by Preneel, B. Vol. 1807. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Bruges, Belgium, 2000, 207–220.

[SG98] Shoup, V. and Gennaro, R. Securing threshold cryptosystems against chosen ciphertext attack. In: *Advances in Cryptology – EUROCRYPT'98*. Ed. by Nyberg, K. Vol. 1403. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Espoo, Finland, 1998, 1–16.

[Shr22] Shrestha, N. *Implementation for OptRand*. Github. 2022.

[Shr+21a] Shrestha, N., Bhat, A., Kate, A., and Nayak, K. *Synchronous Distributed Key Generation without Broadcasts*. Cryptology ePrint Archive, Report 2021/1635. 2021.

[Shr+21b] Shrestha, N., Bhat, A., Kate, A., and Nayak, K. *Synchronous Distributed Key Generation without Broadcasts*. Cryptology ePrint Archive, Paper 2021/1635. https://eprint.iacr.org/2021/1635. 2021.

[Sta96] Stadler, M. Publicly verifiable secret sharing. In: *Advances in Cryptology – EUROCRYPT'96*. Ed. by Maurer, U. M. Vol. 1070. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Saragossa, Spain, 1996, 190–199.

[SS01] Stinson, D. R. and Strobl, R. Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates. In: *ACISP 01: 6th Australasian Conference on Information Security and Privacy*. Ed. by Varadharajan, V. and Mu, Y. Vol. 2119. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, Sydney, NSW, Australia, 2001, 417–434.

[Syt+17] Syta, E., Jovanovic, P., Kokoris-Kogias, E., Gailly, N., Gasser, L., Khoffi, I., Fischer, M. J., and Ford, B. Scalable bias-resistant distributed randomness. In: *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Jose, CA, USA, 2017, 444–460.

[Syt+16] Syta, E., Tamas, I., Visher, D., Wolinsky, D. I., Jovanovic, P., Gasser, L., Gailly, N., Khoffi, I., and Ford, B. Keeping authorities "honest or bust" with decentralized witness cosigning. In: *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Jose, CA, USA, 2016, 526–545.

[TZ22] Tessaro, S. and Zhu, C. Short pairing-free blind signatures with exponential security. In: *Advances in Cryptology – EUROCRYPT 2022, Part II*. Ed. by Dunkelman, O. and Dziembowski, S. Vol. 13276. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Trondheim, Norway, 2022, 782–811.

[Tom+20] Tomescu, A., Chen, R., Zheng, Y., Abraham, I., Pinkas, B., Gueta, G. G., and Devadas, S. Towards scalable threshold cryptosystems. In: *2020 IEEE Symposium on Security and Privacy (SP)*. 2020, 877–893.

[TLP22a] Tsimos, G., Loss, J., and Papamanthou, C. Gossiping for communication-efficient broadcast. In: *Advances in Cryptology – CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part III*. Springer-Verlag, Santa Barbara, CA, USA, 2022, 439–469.

[TLP22b] Tsimos, G., Loss, J., and Papamanthou, C. Gossiping for communication-efficient broadcast. In: *Advances in Cryptology – CRYPTO 2022, Part III*. Ed. by Dodis, Y. and Shrimpton, T. Vol. 13509. Lecture Notes in Computer Science. Springer, Cham, Switzerland, Santa Barbara, CA, USA, 2022, 439–469.

[Yin+19] Yin, M., Malkhi, D., Reiter, M. K., Golan-Gueta, G., and Abraham, I. HotStuff: BFT consensus with linearity and responsiveness. In: *38th ACM Symposium Annual on Principles of Distributed Computing*. Ed. by Robinson, P. and Ellen, F. Association for Computing Machinery, Toronto, ON, Canada, 2019, 347–356.

[Zha+23] Zhang, H., Duan, S., Liu, C., Zhao, B., Meng, X., Liu, S., Yu, Y., Zhang, F., and Zhu, L. Practical asynchronous distributed key generation: improved efficiency, weaker assumption, and standard model. In: *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE Computer Society, Los Alamitos, CA, USA, June 2023, 568–581.