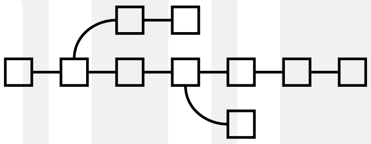
# Cryptography for Bitcoin and Friends



Tim Ruffing

Saarbrücken, 2019

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

|  |  |
|---|---|
| Tag des Kolloquiums: | 2019-11-21 |
| Dekan der Fakultät: | Prof. Dr. Sebastian Hack |

**Prüfungsausschuss**

|  |  |
|---|---|
| Vorsitzender: | Prof. Dr. Christian Rossow |
| Berichterstattende: | Prof. Dr. Aniket Kate |
|  | Prof. Dr. Dominique Schröder |
|  | Prof. Dr.-Ing. Christoph Sorge |
|  | Prof. Dr. Joseph Bonneau |
| Akademischer Mitarbeiter: | Dr. Dominik Schillo |

# Abstract

Numerous cryptographic extensions to Bitcoin have been proposed since Satoshi Nakamoto introduced the revolutionary design in 2008. However, only few proposals have been adopted in Bitcoin and other prevalent cryptocurrencies, whose resistance to fundamental changes has proven to grow with their success. In this dissertation, we introduce four cryptographic techniques that advance the functionality and privacy provided by Bitcoin and similar cryptocurrencies without requiring fundamental changes in their design: First, we realize smart contracts that disincentivize parties in distributed systems from making contradicting statements by penalizing such behavior by the loss of funds in a cryptocurrency. Second, we propose CoinShuffle++, a coin mixing protocol which improves the anonymity of cryptocurrency users by combining their transactions and thereby making it harder for observers to trace those transactions. The core of CoinShuffle++ is DiceMix, a novel and efficient protocol for broadcasting messages anonymously without the help of any trusted third-party anonymity proxies and in the presence of malicious participants. Third, we combine coin mixing with the existing idea to hide payment values in homomorphic commitments to obtain the ValueShuffle protocol, which enables us to overcome major obstacles to the practical deployment of coin mixing protocols. Fourth, we show how to prepare the aforementioned homomorphic commitments for a safe transition to post-quantum cryptography.

# Zusammenfassung

Seit seiner revolutionären Erfindung durch Satoshi Nakamoto im Jahr 2008 wurden zahlreiche kryptographische Erweiterungen für Bitcoin vorgeschlagen. Gleichwohl wurden nur wenige Vorschläge in Bitcoin und andere weit verbreitete Kryptowährungen integriert, deren Resistenz gegen tiefgreifende Veränderungen augenscheinlich mit ihrer Verbreitung wächst. In dieser Dissertation schlagen wir vier kryptographische Verfahren vor, die die Funktionalität und die Datenschutzeigenschaften von Bitcoin und ähnlichen Kryptowährungen verbessern ohne deren Funktionsweise tiefgreifend verändern zu müssen. Erstens realisieren wir Smart Contracts, die es erlauben widersprüchliche Aussagen einer Vertragspartei mit dem Verlust von Kryptogeld zu bestrafen. Zweitens schlagen wir CoinShuffle++ vor, ein Mix-Protokoll, das die Anonymität von Benutzern verbessert, indem es ihre Transaktionen kombiniert und so deren Rückverfolgung erschwert. Sein Herzstück ist DiceMix, ein neues und effizientes Protokoll zur anonymen Veröffentlichung von Nachrichten ohne vertrauenswürdige Dritte und in der Präsenz von bösartigen Teilnehmern. Drittens kombinieren wir dieses Protokoll mit der existierenden Idee, Geldbeträge in Commitments zu verbergen, und erhalten so das ValueShuffle-Protokoll, das uns ermöglicht, große Hindernisse für den praktischen Einsatz von Mix-Protokollen zu überwinden. Viertens zeigen wir, wie die dabei benutzten Commitments für einen sicheren Übergang zu Post-Quanten-Kryptographie vorbereitet werden können.

# Contents

# Preface

*Since cryptography is a tool for shifting power, the people who know this subject well, like it or not, inherit some of that power.*

— Phillip Rogaway [Rog15]

One of the first lessons I learned about cryptography is that I shall not invent my own cryptography. This is good advice, but it was pretty unsatisfactory at the time. Looking back, I feel proud that I have broken that rule, and that my work has an impact on cryptographic systems in the real world.

When Phillip Rogaway reminds us that cryptography rearranges power [Rog15], he reminds us of our responsibility for our research. I think that cryptography not only rearranges power; it is a superpower in itself because it gives us the ability to communicate in secret and make things possible that seem impossible at first glance. I hope I could gain a tiny bit of this superpower and use it to empower others.

## Acknowledgments

This dissertation would not have been possible without the help of countless people. I am deeply grateful to my advisors Aniket Kate and Dominique Schröder. Aniket is an endless source of inspiration, and his advice to work on privacy-enhancing technologies and cryptocurrencies has been the best in my career. Dominique is exactly the right person for deep technical discussions, and always provided excellent guidance. It was fun to work with both of you. Thank you for your selfless support.

Many of my colleagues have become good friends. Special thanks go to Pedro Moreno Sanchez for being the best co-author I can imagine, to Esfandiar Mohammadi for many sleepless nights before deadlines, to Sebastian Meiser for geek debates, to Giulio Malavolta for spontaneous discussions in front of scrawled whiteboards, and to Manuel Reinert for gossip and coffee. No less important, I would like to thank Sandy Heydrich, Jana Rehse, Jonas Schneider, and many others for endless discussions about sense and nonsense in academia. You kept up my motivation.

Last but not least, I thank my family and Lisa and for their unconditional love.

## Previously Published Material

The contents of this dissertation are based on four peer-reviewed publications of which I am the main author.

**Chapter 3** is based on the paper "Liar, Liar, Coins on Fire! – Penalizing Equivocation By Loss of Bitcoins" [RKS15] published at CCS'15. Compared to the results in the published paper, this chapter includes minor corrections and improvements, updated performance numbers, a full security analysis, and it discusses related work which appeared after the publication.

[RKS15] Tim Ruffing, Aniket Kate, and Dominique Schröder. "Liar, Liar, Coins on Fire! – Penalizing Equivocation By Loss of Bitcoins". In: Computer and Communications Security (CCS) 2015. DOI: 10.1145/2810103.2813686.

**Chapter 4** is based on the paper "P2P Mixing and Unlinkable Bitcoin Transactions" [RMK17] published at NDSS'17. Compared to the results in the published paper, this chapter includes minor corrections and improvements and a more detailed security analysis. In particular, it fixes a minor mistake in the construction and discusses related work which appeared after the publication.

[RMK17] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. "P2P Mixing and Unlinkable Bitcoin Transactions". In: Network and Distributed System Security (NDSS) 2017. URL: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/p2p-mixing-and-unlinkable-bitcoin-transactions/.

**Chapter 5** is based on the paper "ValueShuffle: Mixing Confidential Transactions for Comprehensive Transaction Privacy in Bitcoin" [RM17B] published at BITCOIN'17. Compared to the results in the published paper, this chapter includes minor corrections and improvements, and it discusses related work which appeared after the publication.

[RM17B] Tim Ruffing and Pedro Moreno-Sanchez. "ValueShuffle: Mixing Confidential Transactions for Comprehensive Transaction Privacy in Bitcoin". In: Workshop on Bitcoin and Blockchain Research (BITCOIN) 2017. DOI: 10.1007/978-3-319-70278-0_8.

**Chapter 6** is based on the paper "Switch Commitments: A Safety Switch for Confidential Transactions" [RM17A] published at BITCOIN'17. Compared to the results in the published paper, this chapter includes minor corrections and major improvements. In particular, the chapter additionally describes *opt-in switch commitments*, which are hiding even against quantum attackers. The initial construction idea that has led to the design of opt-in switch commitments was suggested by Pieter Wuille in private communication.

[RM17A] Tim Ruffing and Giulio Malavolta. "Switch Commitments: A Safety Switch for Confidential Transactions". In: Workshop on Bitcoin and Blockchain Research (BITCOIN) 2017. DOI: `10.1007/978-3-319-70278-0_10`.

Parts of all other chapters are based on texts from the aforementioned papers.

**Other Work.** During my work as a doctoral candidate, I further contributed to the following papers.

[RSK13] Tim Ruffing, Jonas Schneider, and Aniket Kate. "Identity-based Steganography and its Applications to Censorship Resistance". In: Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs) 2013. URL: `https://petsymposium.org/2013/papers/ruffing-censorship.pdf`.

[BMR14] Michael Backes, Esfandiar Mohammadi, and Tim Ruffing. "Computational Soundness Results for ProVerif – Bridging the Gap from Trace Properties to Uniformity". In: Principles of Security and Trust (POST) 2014. DOI: `10.1007/978-3-642-54792-8_3`.

[RMK14] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. "CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin". In: European Symposium on Research in Computer Security (ESORICS) 2014. DOI: `10.1007/978-3-319-11212-1_20`.

[Bac+15] Michael Backes, Aniket Kate, Sebastian Meiser, and Tim Ruffing. "Secrecy Without Perfect Randomness: Cryptography with (Bounded) Weak Sources". In: Applied Cryptography and Network Security (ACNS) 2015. DOI: `10.1007/978-3-319-28166-7_33`.

[BMR15] Michael Backes, Esfandiar Mohammadi, and Tim Ruffing. "Computational Soundness for Interactive Primitives". In: European Symposium on Research in Computer Security (ESORICS) 2015. DOI: `10.1007/978-3-319-24174-6_7`.

[MRK17] Pedro Moreno-Sanchez, Tim Ruffing, and Aniket Kate. "PathShuffle: Credit Mixing and Anonymous Payments for Ripple". In: Privacy-Enhancing Technologies (PoPETs) 2017.3. DOI: 10.1515/popets-2017-0031.

[Ruf+18] Tim Ruffing, Sri Aravinda Krishnan Thyagarajan, Viktoria Ronge, and Dominique Schröder. "Burning Zerocoins for Fun and for Profit: A Cryptographic Denial-of-Spending Attack on the Zerocoin Protocol". In: Crypto Valley Conference on Blockchain Technology (CVCBT) 2018. Short Paper. DOI: 10.1109/CVCBT.2018.00023.

# 1 Introduction

*I've been working on a new electronic cash system that's fully peer-to-peer, with no trusted third party.*

— Satoshi Nakamoto [Nak08A]

Bitcoin sparked a revolution in the design of electronic cash systems: it was the first *cryptocurrency*, a peer-to-peer electronic cash system that works reliably and securely over the Internet without the help of a trusted third party. Since its invention and implementation by Satoshi Nakamoto in 2008 [Nak08B], Bitcoin has been tremendously successful and led to a profusion of other cryptocurrencies. The market capitalization of all cryptocurrencies combined is more than a hundred billions of euros at the time of writing [Coi]. What makes cryptocurrencies attractive is not only the idea that no single party has control over them and the ability to perform truly irreversible financial transactions across the globe without intermediaries, but also the openness of the technology, which enables developers to build applications without the need to register with a central party or even seek permission.

As the name indicates, cryptography is essential to cryptocurrencies, but maybe surprisingly, the cryptography used in most cryptocurrencies is rather simple. Bitcoin for example relies just on the two basic cryptographic primitives, namely hash functions and digital signatures. This basic use of cryptography in Bitcoin suffices to achieve the most basic security goals that we can expect from a currency and a payment system: money cannot be created out of thin air, received funds can be stored permanently, and they can only be spent by their legitimate owner. Due to the lack of a trusted third party that enforces these properties, they instead need to be publicly verifiable by every peer in the network.

More advanced security goals, which may require the use of sophisticated cryptography, were only secondary to the two aforementioned basic security goals. Arguably, the most important example is privacy. Even though privacy of users was a concern [Nak08B], the focus of Bitcoin's design is not the confidentiality of transactions but in fact exactly the opposite, namely full transparency with the aim to make transactions amenable to public verification. For example, all transactions including a sender identifier, a recipient identifier, and the transferred amount are recorded in plain in a public data structure called the blockchain. The blockchain is written to by miners, i.e., peers willing to invest computational resources necessary for the security of the system in return for freshly generated coins. Miners broadcast updates to the blockchain through the entire peer-to-peer (P2P) network, and most peers in the network maintain a local copy of the entire blockchain. This enables peers in the network to verify that these transactions adhere to the basic rules, e.g., that users do not spend more money than they own.

However, the public availability of all transaction data is clearly a huge threat for the privacy of individual users. Even though virtually all cryptocurrencies use pseudonyms as user (or account) identifiers, the initial perception of pseudonyms alone providing some built-in anonymity has been broadly refuted in the literature [Mei+13; Bar+12; SMZ14; KKM14; RH11; And+13; MO15; Nic15], and several companies [Ell; Cha] offer to trace cryptocurrency payments as a service. To make matters worse, the lack of confidentiality has implications beyond the privacy of users. While fungibility, i.e., the idea that every two funds of the same denomination should be equivalent and interchangeable, is fundamental to the practical success of traditional currencies, this property is at risk in most cryptocurrencies because funds can be traced from owner to owner and build up a history.

A second prominent disadvantage of cryptocurrencies is that transactions are not instantaneous but require large confirmation times, e.g., in the order of an hour in Bitcoin. This is clearly too long for many use cases such as point-of-sale payments at cash desks and fast trading of cryptocurrencies.

In the light of these and other shortcomings, there is a plenitude of research aiming to apply advanced cryptography to improve cryptocurrencies. Remarkably, this research is not only conducted in academia but also in the industry and by interested individuals in the community of users and developers.

The first line of research relevant to this dissertation aims at overcoming the aforementioned privacy issues of cryptocurrencies. A rich set of cryptographic techniques and tools such as homomorphic commitments [Max15], zero-knowledge proofs [Mie+13; Ben+14; Kos+16], and variants of ring signatures [Sab13; NMM16] have been proposed to hide transaction details while still allowing the public to verify that transactions are correct, e.g., they do not generate money out of thin air.

The second line of research relevant to this dissertation aims at improving the functionality provided by cryptocurrencies. For example, a promising solution to enable instantaneous payments are payment channels and off-chain payment networks as a second layer on top of cryptocurrencies [Spi13; DW15; PD; KG17; Mal+17; Dzi+19; Mil+19]. The key insight to these layer-2 systems is that transactions do not need to be processed by the entire network as long as they are not disputed among the involved parties. A related research direction leverages the strong security guarantees of cryptocurrencies and their monetary nature in other cryptographic protocols, e.g., to provide monetary fairness in secure multi-party computation (SMPC) [And+14; BK14]: while SMPC alone suffices to play mental poker, cryptocurrencies can additionally ensure that the honest winner always obtains the prize [KMB15].

What makes cryptocurrencies special within the area of cryptography is that new cryptographic schemes are sometimes adopted within months after their invention. While this quick adoption is risky and unprecedented, it has yielded the first systems that deploy advanced primitives such as zero-knowledge proofs successfully and on a large scale [Mie+13; Ben+14; ZEC; Sab13; NMM16; XMR]. Keeping this in mind, cryptocurrencies are not only fascinating on their own, but they constitute a major opportunity for the deployment of cryptography in general.

However, while a considerable amount of research in the area of cryptocurrencies is undergoing practical deployment, almost all implementations come in the form of new cryptocurrencies [e.g., ZEC; ADA], and not many proposals have been deployed as extensions to existing cryptocurrencies. The reason is that many research proposals aiming at improving privacy, functionality, or other aspects of cryptocurrencies require substantial technical changes, which require broad community consensus to adopt. The Bitcoin community in particular seems very conservative about adopting modifications that influence Bitcoin's security model (including trust assumptions and cryptographic hardness assumptions), break compatibility with existing features and use-cases, or simply make verification of the blockchain less efficient.

To name a specific example, Zerocoin [Mie+13], which improves the anonymity of cryptocurrency users by zero-knowledge proofs, has been explicitly designed as an extension to Bitcoin. Despite being a major improvement for privacy, Zerocoin could not find consensus in the Bitcoin community due to various reasons including large storage requirements in the blockchain [Bac13] and the need for a trusted cryptographic setup [Ler13]. Instead, Zerocoin's successor Zerocash [Ben+14] was later deployed as a new and independent cryptocurrency [ZEC].

In hindsight, it comes as no surprise that making changes to a multi-billion project such as Bitcoin is controversial and difficult with so many different stakeholders involved, namely individual users, corporate users, developers, miners, investors, researchers, and others. Who decides whether the rules of the currency are changed and how? This question of "who is in control" is essential to any currency system. However, in a cryptocurrency without a trusted third party, there is intentionally no clear answer to it. Resistance to fundamental changes may in fact be crucial for the success of cryptocurrency because only then, stakeholders can be ensured that they will not experience unexpected surprises, e.g., affecting their holdings. As a consequence, we must acknowledge that making changes to well-established cryptocurrencies is exceptionally difficult.

Nevertheless, this observation alone clearly does not make the problems of existing cryptocurrencies disappear. If cryptocurrencies survive their infancy, then most probably only few will see widespread adoption, and technological superiority is only one of many aspects that will determine which ones will prevail. Almost a decade after its inception, Bitcoin remains the most used cryptocurrency and is due to network effects certainly a promising candidate to see further adoption in the future. If Bitcoin prevails, we need to improve its technology organically and gradually, and the insight that fundamental technical changes may be too difficult to make is no justification for ignoring Bitcoin's technological shortcomings and their societal implications such as the sacrifice of users' privacy.

## 1.1 Contributions

The goal of this dissertation is to devise tailor-made cryptographic schemes and protocols to advance the functionality and privacy of cryptocurrencies. Our focus is on practical techniques that are either fully compatible with Bitcoin and similar existing cryptocurrencies or require only specific modifications that do not break with fundamental design decisions or existing features. This conservative approach acknowledges the difficulty of making changes to existing cryptocurrencies and consequently provides a pragmatic path to technological progress.

For the sake of concreteness, the results in this dissertation are described with Bitcoin in mind. Our results are in general applicable to other cryptocurrencies as well, though minor modifications may be necessary.

After a brief introduction to cryptocurrencies and Bitcoin (Chapter 2), we provide four main contributions (Chapters 3, 4, 5 and 6) as summarized below. We conclude with a few final remarks (Chapter 7), which are relevant beyond the concrete results in this dissertation.

**Chapter 3: Penalizing Equivocation By Loss of Bitcoins.** In our first contribution, we pick up the idea to leverage the monetary nature of cryptocurrencies to improve other systems. Concretely, we apply cryptocurrencies to incentive honest behavior in distributed systems. We show that equivocation, i.e., asserting conflicting statements to different parties in a distributed system, can be monetarily disincentivized by the use of cryptocurrencies. To this end, we design completely decentralized *non-equivocation contracts*, which make it possible to penalize an equivocating party by the loss of its money. At the core of these contracts, there is a novel cryptographic primitive called *accountable assertions*, which reveals the party's private keys in case of equivocation.

Non-equivocation contracts are particularly useful for distributed systems that employ public append-only logs to protect data integrity. Moreover, as double-spending in Bitcoin is a special case of equivocation, non-equivocation contracts enable us to design *asynchronous payment channels* that enable a payee to receive funds at multiple unsynchronized points of sale while being able to penalize a double-spending payer after the fact.

**Chapter 4: P2P Mixing and Unlinkable Bitcoin Transactions.** Having demonstrated the applicability of cryptocurrencies, we then turn our focus to users' privacy in cryptocurrencies, which is vulnerable to a variety of linkability and deanonymization attacks due to the public nature of the blockchain. In order to improve the anonymity of cryptocurrency users, we propose *CoinShuffle++*, a P2P coin mixing protocol that enables pseudonymous users to perform unlinkable transactions in a manner fully compatible with the current Bitcoin system. CoinShuffle++ is based on the CoinJoin paradigm [Has11; Max13A] and requires at its core a P2P anonymous communication protocol, which enables a group of mutually distrustful peers to publish messages in an anonymous and reliable manner.

Starting with Dining Cryptographers networks (DC-nets) [Cha88], multiple such P2P anonymous communication protocols have been proposed in the literature. However, despite their strong anonymity guarantees, none of them have been deployed in practice so far: most protocols fail to simultaneously address the crucial problems of slot collisions and disruption by malicious peers, and those which address both problems require $O(f^2)$ communication rounds to handle $f$ malicious peers, which is arguably prohibitive to their practical deployment.

We conceptualize these P2P anonymous communication protocols as *P2P mixing*, and then present *DiceMix*, a novel P2P mixing protocol that needs only four communication rounds in the best case, and $4 + 2f$ rounds in the worst case with $f$ malicious peers. As every individual malicious peer can force a restart of a protocol run by simply omitting messages, we find DiceMix with its worst-case complexity of $O(f)$ communication rounds to be optimal to realize the anonymous broadcast required by CoinShuffle++.

We demonstrate the efficiency of DiceMix and CoinShuffle++ with a proof-of-concept implementation. In our evaluation, DiceMix requires less than eight seconds to mix 50 messages of size 160 bits in a setting where all 50 peers have 10 MBit/s connections with a delay of 50 ms to an untrusted bulletin board that interconnects the peers. Since CoinShuffle++ has negligible overhead over its core building block DiceMix, CoinShuffle++ mixes coins in essentially the same time. In contrast, the best protocol in the literature requires almost three minutes in the same setting.

Going beyond the concrete application of anonymous communication to improve the anonymity of cryptocurrency users, we additionally take a more fundamental look at P2P mixing and present a generic deanonymization attack on all P2P mixing protocols that guarantee termination in the presence of disruptive peers; this includes the state-of-the-art P2P mixing protocol Dissent [CF10; Syt+14]. The generic attack implies that it is fundamentally impossible for a P2P mixing protocol to simultaneously support input messages arbitrarily chosen by the user, provide anonymity, and terminate in the presence of disruptive peers.

**Chapter 5: Mixing Confidential Transactions for Comprehensive Privacy.** Given our efficient coin mixing protocol CoinShuffle++, our next goal is to further advance the practicality and privacy of coin mixing by integrating it with other privacy-enhancing techniques addressing other aspects of privacy. As an extension of CoinShuffle++, we design *ValueShuffle*, the first coin mixing protocol compatible with Confidential Transactions [Max15], a proposed enhancement to Bitcoin to hide the amounts in transactions. ValueShuffle ensures the anonymity of peers in the mixing and the confidentiality of their mixed amounts even against malicious peers participating in the same mixing run. In combination with Confidential Transactions and additionally Stealth Addresses [Sab13], a technique to avoid the reuse of pseudonyms when receiving payments, ValueShuffle provides decent privacy (payment value privacy and payer anonymity even against malicious payees) without breaking with fundamental design principles or features of the current Bitcoin system.

Assuming support for Confidential Transactions, ValueShuffle makes it possible to mix funds of different amount as well as to mix and spend funds in the same transaction. These features overcome the two main limitations of all previous coin mixing approaches in practice, namely that users are restricted to mix funds of the same amount, and that they need to do so in a separate transaction before they can actually spend the funds.

**Chapter 6: Switch Commitments for Confidential Transactions.** Finally, we turn our attention to the future of cryptocurrencies and tackle the problem of post-quantum security. We focus on cryptographic agility, which is the ability to switch to larger cryptographic parameters or different algorithms in the case of security doubts. This desirable property of cryptographic systems is inherently difficult to achieve in cryptocurrencies due to their permanent state in the blockchain: for example, if it turns out that the employed signature scheme is insecure, a switch to a different scheme can only protect the outputs of future transactions but cannot fix transaction outputs already recorded in the blockchain, which puts the money at the risk of theft.

This situation is even worse in cryptocurrencies which deploy Confidential Transactions to hide transacted monetary amounts in homomorphic Pedersen commitments. An attacker who manages to break the computational binding property of a commitment can create money out of thin air, jeopardizing the security of the entire currency. The obvious solution is to use statistically or perfectly binding commitment schemes such as ElGamal commitments, but they come with performance drawbacks due to the need for less efficient range proofs, and even worse, they are not hiding against post-quantum attackers.

In order to overcome this dilemma, we introduce *switch commitments*, which constitute a cryptographic middle ground between computationally binding and statistically binding commitments. The key property of this novel primitive is the possibility to switch existing commitments, e.g., recorded in the blockchain, from computational bindingness to statistical bindingness if doubts in the underlying hardness assumption arise. This switch trades off efficiency for security.

We provide two simple and practical constructions of switch commitments. First, we prove that ElGamal commitments with a restricted message space are secure switch commitments. Our technique yields an instantiation of Confidential Transactions that can be switched to be resilient against post-quantum attackers trying to inflate the currency. Second, we construct *opt-in* switch commitments, a variant of switch commitments that guarantee hiding even against post-quantum attackers in almost all cases. Our construction is as compact as Pedersen commitments and as as result, opt-in switch commitments introduce essentially no overhead for the cryptocurrency network.

# 2 Background on Cryptocurrencies

In this chapter, we first give an informal introduction to the fundamental challenge that P2P electronic cash systems need to overcome and to the solution provided by Nakamoto. Then we briefly explain the internals of Bitcoin that are relevant to our work. We give only a brief overview, and since the techniques presented in this dissertation do not depend on the details of the blockchain and the consensus mechanism, our results are applicable to other cryptocurrencies, though minor modifications may be necessary.

**Further Reading.** For a broader introduction to Bitcoin and cryptocurrencies, we recommend the surveys by Bonneau et al. [Bon+15], Tschorsch and Scheuermann [TS16], Conti et al. [Con+18], and Narayanan and Clark [NC17], and the textbook by Narayanan et al. [Nar+16]. For an in-depth explanation of the internal mechanics of Bitcoin, we refer the reader to the textbook by Antonopoulos [Ant17] and the Bitcoin developer guide [BDG].

## 2.1 Nakamoto Consensus

To understand the fundamental challenge in building a P2P electronic cash system without a trusted third party, we need to ask why trusted third parties have been essential for fiat currencies and traditional payment systems. The answer to this question is twofold.

The first essential task of trusted third parties is to exclude double-spending, which is a fundamental problem in any payment system not based on physical cash. The capability of sending a payment, e.g., the capability to produce a handwritten signature on a bank transfer form or the capability to swipe a credit card, is not consumed by itself when the payer uses it once to send a payment. An accounting mechanism is therefore necessary to avoid that malicious payers use their capability to spend more money than they are allowed to. Traditional payment systems rely on trusted third parties, e.g., banks and credit card providers, to perform this accounting, which in its simplest form amounts to recording every payment (or transaction) in a ledger. Remarkably, the vast majority of attempts to build secure electronic cash from sophisticated cryptography that predate Bitcoin, most notably Chaum's proposal [Cha82; CFN88], rely on trusted third parties to exclude double-spending.

The second essential task of trusted third parties in fiat currencies is to oversee and control money supply. For example, in the Eurosystem, only banks have the power to create money, and with respect to physical cash as legal tender, the European Central Bank has the exclusive right to authorize the issuance of euro banknotes [TFEU].

How do cryptocurrencies exclude double-spending and control money supply without a trusted third party? For double-spending, the answer is a transaction ledger, which records the transactions that have been performed in the past, but which is publicly maintained by the entire Bitcoin system instead of relying on a trusted third party. In more detail, the peers in the Bitcoin system aim to reach irreversible consensus over the transactions that have been committed to the ledger. The irreversibility is crucial: if transactions are reversible, a fraudulent user can double-spend by sending a payment, obtaining some good or service from the payee, and later reversing the transaction to be able to spend the money again.

Reaching irreversible consensus turns out to be particularly challenging in an asynchronous P2P network consisting of potentially malicious peers. A common approach in non-P2P networks is to assume that a supermajority of parties in the system is honest and use some form of voting to reach consensus; this is the idea of Byzantine fault tolerance protocols [LSP82; CL02]. In P2P networks, however, since peers can join and leave the network arbitrarily and have no permanent identify, no peer can reliably determine the full set of peers among which consensus must be reached. In particular, a single attacker can easily perform a Sybil attack [Dou02], i.e., spawn a large number of malicious peer nodes in order to obtain a fraudulent majority and thereby take control of the system.

Nakamoto's seminal consensus mechanism solves this problem by determining the majority not in terms of the number of peers but in terms of the amount of computation that the peers perform. As computation inherently requires resources such as hardware and energy, this consensus mechanism withstands Sybil attacks: while an arbitrary number of fake nodes can be spawned at essentially no cost on a given piece of hardware, the overall amount of computational resources available to all those nodes is the same as would be available to a single node.

The data to reach consensus on, namely the transaction ledger, is organized as a blockchain, i.e., a list of blocks, each containing a set of transactions. The blocks are chained together such that every block must contain a cryptographic hash of the previous block. A block (including its transactions) can only be added to the blockchain if it comes with a solution for a proof-of-work puzzle, i.e., an instance of a moderately hard computational problem. The specific problem instance that must be solved is determined by all the data in the block including the transactions and the hash of the previous block, and due to the chaining, it depends not only on the previous block but recursively on the entire transaction history. As a result, each valid block carries a certain expected amount of computational work.

This computational work ensures that the consensus over the transaction history is permanent. Once added to the blockchain, the transactions in a block are difficult to modify: A modification of a block invalidates the proof-of-work solution of this block and all subsequent blocks because different problem instances must be solved after the modification. When peers receive contradicting blocks that constitute a fork in the blockchain, they consider only the chain whose creation required (in expectation) the most computational work to be the correct one. This rule implies that an attacker trying to modify an old block is not only required to redo the work for this and all subsequent blocks but also to catch up, i.e., to find valid proof-of-work solutions at a higher rate than the rest of the network, which continues to work on extending the other chain. As a result, a transaction that has been included in the blockchain and backed by the proof-of-work computations of an increasing number of blocks is thus increasingly difficult to reverse for an attacker with limited computational resources. In particular, reversing such a transaction is considered infeasible if the attacker controls less than 50% of the computing power in the system, which constitutes a fundamental assumption necessary for the security of Bitcoin (the exact value depends for instance on the time it takes to propagate blocks through the P2P network [GKL15; GKL17; PSS17].)

This paradigm adopted in Bitcoin requires computing power as an essential part for the security of the consensus mechanism. But why should peers contribute computing power, given that computation comes with real costs, e.g., energy consumption? The answer to that question is that miners, which are peers willing to take part in the consensus mechanism, are assigned rewards for contributing their computational resources. Transaction fees paid by transacting parties are one part of these rewards, and newly created (or mined) currency units (bitcoins) are the other part. This explains how money creation is organized without a trusted third party: bitcoins are scarce because the computational resources required to generate them are scarce.

The underlying idea to use proof-of-work to ensure some scarcity has been present in previous proposals from the cypherpunk community [Dai98; Sza08; Fin04], starting with Back's Hashcash proof-of-work scheme [Bac02]; however in those proposals the proof-of-work solutions are directly accepted as electronic cash tokens.

What makes Bitcoin fundamentally different from all previous electronic cash systems is that proof-of-work is used within the consensus mechanism to ensure Sybil-resistance, and the consensus rules control how many bitcoins the miners are eligible to claim for finding proof-of-work solutions. Therefore Bitcoin's consensus mechanism is a combined replacement for a trusted third party with respect to both its task of controlling money supply and its task of excluding double-spending. The idea to solve both of these problems interdependently is the key novelty introduced by Nakamoto.

## 2.2 Technical Overview of Bitcoin

The term *Bitcoin* (commonly capitalized) refers to the P2P system and the currency it runs. The currency units are *bitcoins* (commonly not capitalized, symbol ₿). A bitcoin is divisible up to eight decimals such that the smallest unit is $10^{-8}$ bitcoins, which is also called one *satoshi*.

**Peer-to-peer Network.** Peers are connected via the Internet, and typically maintain connections to a few other peers called their *neighbours*. As a result, the P2P network forms a decentralized peer graph. The purpose of the P2P network is to broadcast transactions and blocks (as explained below) by gossiping: if a peer receives a transaction or a block (or if this peer creates a new transaction or a new block), it it is supposed to relay it to every neighbour who does not know it already, and the neighbours are in turn supposed to relay it to their neighbours such that it will eventually reach every peer in the network. This assumes that the peer graph is connected, which is necessary for the network to work properly and secure [Hei+15; MHG18].

**Transactions.** The most fundamental data structure of Bitcoin is a *transaction*, which represents a payment. A transaction moves coins from a set of inputs to a set of outputs. An output is an integer specifying an amount of satoshis together with a *script*, which is a small program that specifies a challenge that must be fulfilled by anyone willing to spend the coins in a further transaction (typically the intended payee). An input is a reference to an *unspent transaction output* (UTXO) created by a previous transaction, together with data that fulfills the challenge specified by the script in this UTXO.

In the simplest and by far most common case, the challenge that must be fulfilled to spend the coins in a UTXO with a transaction is to provide a digital signature on the desired transaction valid under a public key specified through its hash in the script of the UTXO. This implements ownership and authentication: coins can effectively be sent to a public key, and only the user that knows the corresponding secret key (the owner) can access the coins by signing a transaction (which in turn sends the coins to other public keys).

For a transaction to be valid, all scripts of the UTXOs in its inputs must be fulfilled, all inputs must refer to distinct UTXOs, there must be no previous transaction that has already spent those UTXOs, and the transaction must be *balanced*, i.e., the sum of the coins in its outputs does not not exceed the sum of the coins in its inputs.

**Digital Signature Scheme.** At the time of writing, Bitcoin and most other cryptocurrencies use ECDSA signatures [JMV01] on the elliptic curve `secp256k1` [SEC2] for authentication of transactions. The future adoption of Schnorr signatures [Sch91] on the same elliptic curve has been suggested [Core17]. In both ECDSA and Schnorr signatures, a public key is a single group element $pk = g^{sk}$ of the elliptic curve group.

**Scripts and Smart Contracts.** Bitcoin employs a scripting language to specify under which conditions a UTXO can be spent. The language is a simple Forth-like stack-based language. It is intentionally not Turing-complete to avoid complexity and the possibility of creating scripts that are expensive to execute, and could consequently lead to denial-of-service attacks because every node in the Bitcoin network must execute them. The purpose of the scripting language is to enable *smart contracts* [Sza97] that implement more complex access scenarios than simple payments, e.g., escrow of funds for dispute resolution and automated lotteries [And+14; BK14].

To spend the funds protected by some script, the spender must provide an initial execution stack with input values. The script (historically called *ScriptPubKey*) is fulfilled if its execution on the initial stack does not abort early, the final stack at the end of the execution is not empty, and the top stack element is not zero.

**Example Script.** As an example, we consider the most commonly used script type, which is called pay-to-pubkey-hash (P2PKH) and requires a signature on the spending transaction under a public key. The script does not specify a public key directly but instead the hash of a public key under a collision-resistant hash function. When spending the UTXO, the owner is required to show not only a valid signature but also the correct public key, whose hash is the value specified in the script. Concretely, a P2PKH script looks as follows.

```
DUP HASH160
h EQUALVERIFY
CHECKSIG
```

We briefly explain the semantics of the used opcodes. `DUP` duplicates the top item of the stack. `HASH160` replaces the top stack item $s$ by $\mathsf{H}(s)$, where the used hash function is $\mathsf{H}(s) := \mathsf{RIPEMD\text{-}160}(\mathsf{SHA\text{-}256}(s))$. Values that are not opcodes denote push operations, i.e., “$h$” pushes the constant $h$ onto the stack. `EQUALVERIFY` aborts the execution if the two top stack items are not equal and removes the items from the stack. `CHECKSIG` runs the ECDSA verification algorithm with the spending transaction as message and with the two top stack items as signature and verification key, respectively. These stack items are removed, and depending on the verification result, either the constant `0` or the constant `1` is pushed onto the stack.

To spend the coins, the owner has to authenticate the desired spending transaction by providing a signature $\sigma$ of the transaction under the correct public key $pk$ and the public key $pk$ itself, i.e., the owner provides an initial stack with the values $\sigma$ and $pk$ as input data for the script.

Let us walk through the execution of the script. First, the top element $pk$ is duplicated on the stack, and the hash function is applied to the top copy of $pk$. The script fails if the resulting hash $\mathsf{H}(pk)$ does not equal $h$. If the hash equals $h$, the signature $\sigma$ that is supposed to sign the spending transaction is verified under the verification key $pk$. If the verification succeeds, `1` is pushed onto the stack and the script terminates correctly; otherwise `0` is pushed onto the stack and the spending transaction is invalid.

**Addresses.** The hash of a public key in a suitable encoding serves as compact and user-facing recipient identifier and is called *address*. Addresses cannot only be created for pay-to-pubkey-hash scripts but also for more complex scripts by encoding a hash of the entire script; these addresses are called pay-to-script-hash (P2SH).

Even though the same address can be paid to by multiple payers and multiple transactions, it is recommended that payees create fresh addresses for every payment that should be received. This avoids that payments to the same payee can be trivially linked to each other. (Nevertheless, this simple mitigation does not prevent more sophisticated heuristics to link addresses [Mei+13; Bar+12; SMZ14; KKM14; RH11; And+13; MO15; Nic15].)

**Change Outputs.** The transaction must spend each of the coins at each input entirely. In practice, this makes *change* outputs necessary. For example, if the only input of a transaction carries ₿5.2 but the goal is to pay only ₿1.1, then a transaction with two outputs is created: one output that pays ₿1.1 to the desired payee and one change output that pays the remaining ₿4.1 back to a *change output* of the payer (ignoring transaction fees as described below).

**Miners and the Blockchain.** Every peer can choose to be a *miner*. The task of miners is to collect transactions broadcast by other peers and to include them in a data structure called a *block*. Blocks are cryptographically chained together, i.e., a block is required to contain the hash of the previous block, such that blocks form a *blockchain*, which is the central data structure in the consensus mechanism. The first block in the blockchain is called *genesis block* and hardcoded in the client software. The *height* of a block is the number of blocks that precede it on the blockchain, e.g., the height of the genesis block is zero. A block is subject to certain validity rules called *consensus rules*, e.g., transactions in a block may only spend bitcoins that have not been spent already by a second transaction in the same or in a previous block.

The consensus rules require blocks to include a Hashcash [Bac02] proof-of-work solution: the hash value of the block must be below a certain *difficulty target* when interpreted as an integer, and the difficulty target determines an expected amount of work (i.e., number of hash computations) necessary to find a solution. The hash value is taken over the *block header*, which consists of the hash of the previous block, the root hash of a Merkle tree [Mer87] of all transactions in the block, and a nonce available for miners to modify when working on a valid proof-of-work solution.

A miner who finds a valid block is supposed to send it to a few neighboring peers in the network, who verify its validity with respect to the consensus rules and add it to their local copy of the blockchain. In case of forks in the blockchain, e.g., because two miners find a valid block at the same time, peers consider the chain with the most expected work done (typically also the longest chain) as the correct one. To break ties, a peer prefers the chain it has seen first. Miners will typically try to extend the chain that they consider valid, and eventually all but one of the forks will be abandoned if the majority of computing power follows this rule.

Since the amount of total computing resources in the system is not constant, the consensus rules prescribe that the difficulty target of the proof-of-work is adjusted after every 2016 blocks (about 2 weeks), based on the time it took the system to produce these 2016 blocks. This mechanism ensures that the average time to mine a block remains roughly constant at 10 min.

Most implementations consider a transaction final if it has been backed by least six blocks, i.e., by the block of the transaction itself and five subsequent blocks. Since at every point of time it takes 10 min on average until the next block is found, it takes 60 min on average before a peer, who is supposed to receive funds in a transaction, should assume that the transaction is irreversible and accept the funds as a payment.

**Transaction Fees.** Transaction fees are one of two incentives for miners to work on blocks. A transaction fee is a (typically small) amount of coins that a miner receives as a reward for confirming a transaction, i.e., for including it into a block. Technically, the fee corresponds to the difference between the sum of outputs and the sum of inputs in a transaction, i.e., it is taken from the inputs of the transaction.

Technically, there are no strict requirements on the amount of the fee, and it may be zero or even the entirety of input coins of the transaction. However miners may choose not to include a transaction in a block if a too low fee is provided.

**Block Reward.** The second incentive for miners is the *block reward*. The consensus rules allow for inclusion of a *coinbase transaction* in every block, which sends a fixed amount of newly generated coins to the miner. Whenever a valid proof-of-work solution is found, the lucky winning miner obtains these newly created coins.

Therefore the consensus mechanism can be seen as a decentralized lottery. A simple analogy is an (imaginary) lottery with free scratch-off tickets [Mil+15], in which the winner is the first to find a valid ticket. While tickets (nonces given to the proof-of-work puzzle as input) can be obtained essentially for free, work is required to scratch them (compute the hash). Scratching a single ticket is very cheap but the winning probability of a single ticket is very low. As a result, many tickets need to be scratched, and scratching can easily be done in parallel. Participants with more scratching power (computing power) are able to scratch more tickets per time, and thus are more likely to be the next winner and claim the reward.

For Bitcoin to be incentive-compatible and to avoid centralization of the mining process, it is desirable that the probability to be selected as the next winner is roughly proportional to a miner's fraction of the total computing power in the system. However, miners can optimize their winning probability by not broadcasting blocks immediately in certain situations even if their fraction of computing power is less than 50% [ES14; Kwo+17].

In Nakamoto's design the block reward is halved every 210,000 blocks (about four years), which implies that there is a maximum number of bitcoins. When block rewards are low, transaction fees are supposed to constitute the main incentive for miners. However, Carlsten et al. [Car+16] raise doubts that fees can fully replace block rewards as an incentive mechanism.

**Changing the Consensus Rules.** The consensus rules are hardcoded in the client software, but they are not written in stone. Software can be modified, and different users can choose to run different software implementations with different consensus rules. A safe change of the consensus rules requires wide agreement and coordination in the Bitcoin ecosystem to avoid permanent chain splits, in which some peers will assume that some particular blockchain is valid whereas the other peers will assume that this blockchain is invalid and will maintain a different blockchain. Nevertheless, there is nothing that prevents a subset of the peers to adopt different consensus rules and risk a permanent chain split, e.g., if a proposed change to the consensus rules supported by this subset does not reach wide agreement. Such chain splits have occurred in multiple cryptocurrencies including Bitcoin [Wir16; Bitm17].

In general, a change of the consensus rules is only meaningful if the new rules will not invalidate blocks already in the blockchain. This can be ensured by making the new rules valid only for blocks added to the blockchain after some point in the future, i.e., the old rules apply to blocks with a height $b < c$ for some constant $c$ greater than the current block height, and the new rules apply to blocks with a height $b \geq c$.

We can distinguish two types of such changes. A change to the consensus rules is a *softfork* if the blocks valid under the new consensus rules are a subset of the blocks valid under the old consensus rules; otherwise it is a *hardfork*. In other words, a softfork makes previously valid blocks invalid, but it does not make previously invalid blocks valid.

Softforks are often considered preferable because they ensure backward compatibility: a peer which has not upgraded to the new consensus rules will consider new blocks as valid and maintain consensus with the rest of the network.

The most prevalent way to deploy a softfork is to let the miners active it. If the majority of miners (measured in computing power) enforce the new stricter rules, they will eventually produce the chain with the most expected work, and eventually any other fork that contains blocks invalid under the new rules will be abandoned.

# 3 Penalizing Equivocation By Loss of Bitcoins

Making conflicting statements to others, or *equivocation*, is a simple yet remarkably powerful tool of malicious participants in distributed systems of all kinds [Cle+12; Bac+14; Lev+09; Chu+07]. In distributed computing protocols, equivocation leads to Byzantine faults and fairness issues. When feasible, equivocation is handled by assuming a supermajority of honest nodes (i.e., larger replication factors [Ho+08]), synchrony assumptions and digital signatures [FM00], or trusted hardware [Bac+14; Chu+07; Cle+12; Lev+09]. Moreover, publicly verifiable append-only logs [MS02; Fel+10; Fah+14; Fel+12] make it possible to detect equivocation after the fact, but they do not suffice to stop or penalize equivocation.

Decentralized cryptocurrency systems such as Bitcoin and its derivatives follow a novel approach to handle equivocation. To protect against equivocation in the form of *double-spending*, i.e., spending the same funds to different parties, Bitcoin employs a special decentralized public append-only log based on proof-of-work called the *blockchain*: In a peer-to-peer cryptocurrency, peers transfer their funds by publishing digitally signed transactions. Transactions are confirmed only when they are included in the blockchain, which is generated by currency miners that solve proof-of-work puzzles. Although a malicious owner can sign over the same funds to multiple receivers through multiple transactions, eventually only one transaction will be approved and added to the publicly verifiable blockchain (Section 2.2).

As a result, a possible approach to stop equivocation in a distributed system is to record all messages in the system that are vulnerable to equivocation in a blockchain. However, due to proof-of-work computations and the decentralized nature of blockchain systems, the process of reaching consensus is not only expensive but also only slowly converging. In Bitcoin, it takes tens of minutes to reach consensus on the set of valid transactions.

To enable faster transactions, a contractual solution in the form of payment channels [Spi13; Tod14A; PD; Mal+17] is emerging in the Bitcoin community. Here, a payer makes a time-locked deposit for a predetermined payee such that double-spending (or equivocation) is excluded even when payments are performed offline and without waiting. However, payment channels are not secure against double-spending when the payee runs several geographically distributed and unsynchronized points of sale, e.g., a bus company selling tickets on buses with only sporadic Internet connectivity.

Our goal in this chapter is to address these equivocation issues by a generic solution that disincentives equivocation and is applicable to various distributed systems and scenarios including the aforementioned payment channels with unsynchronized points of sale. Our solution is fully compatible with the current Bitcoin system without the necessity to make any changes it. We provide four main contributions.

**Accountable Assertions.** As a first step, we establish a cryptographic connection between equivocation and the loss of funds by introducing a cryptographic primitive called *accountable assertions* (Section 3.3). The main idea of this primitive is to bind *statements* to *contexts* in an accountable way: if the attacker equivocates, i.e., asserts two contradicting statements in the same context, then any observer can extract the attacker's Bitcoin secret key and use it to force the loss of the attacker's funds.

We present a construction of accountable assertions based on chameleon hash functions [KR00] and prove it secure in the random oracle model under the discrete logarithm assumption (Section 3.4). A performance evaluation of our construction demonstrates its practicality with respect to computation, communication, and storage costs.

**Non-equivocation Contracts.** To ensure that a secret key obtained through equivocation is indeed associated with funds, parties that should be prevented from equivocating are required to put aside a certain amount of funds in a *deposit* [And+14; BK14; KB14; BtcWikiB]. The deposit is *time-locked*, i.e., the depositor cannot withdraw the funds during a predetermined time period. This prevents an attacker from spending the funds and rendering the secret key useless just before equivocating.

Accountable assertions and deposits together enable us to design *non-equivocation contracts*, a generic method to penalize paltering in distributed systems (Section 3.5). We propose several applications of non-equivocation contracts to ensure the linearity of append-only logs [MS02; Fel+10; Fah+14; Fel+12].

**Asynchronous Payment Channels.** *Payment channels* [Spi13; BIP65] enable peers to perform payments to a predetermined party offline and without waiting for the consensus process. However, if the payee is a unsynchronized entity (e.g., points of sale with only sporadic Internet connectivity) then payment channels do not prevent double-spending. Since double-spending is an instance of equivocation, non-equivocation contracts enable us to design *asynchronous payment channels*, which make it possible to penalize double-spending payers (Section 3.6).

**Double-Authentication-Preventing Signatures.** Of independent interest, we observe that accountable assertions are similar to *double-authentication-preventing signatures* (DAPS) as proposed by Poettering and Stebila [PS14]. While accountable assertions are in general a weaker primitive, certain accountable assertions are DAPS. It was left as an open problem to construct DAPS based on trees or chameleon hash functions [PS14]. We solve these problems, and our accountable assertion scheme based on Merkle trees *and* chameleon hash functions in the random oracle model yields a DAPS scheme, which is the first one in the discrete logarithm setting (Section 3.8) and remains the only known one in this setting that supports a context space of exponential size. For practical parameters, our DAPS scheme is two orders of magnitude faster than the original DAPS construction [PS14; PS17] and uses one order of magnitude less communication.

## 3.1 Overview

We conceptualize decentralized non-equivocation contracts and discuss their potential applications.

**Problem Statement.** Equivocation, i.e., making conflicting statements to different protocol parties, is a universal problem in fault-tolerant security protocols involving three or more parties [Cle+12; Bac+14; Lev+09; Chu+07]. In all bounded or partial synchronous communication settings, equivocation can be detected using digital signatures (together with a public-key infrastructure) and some interaction among the parties [Cle+12]: two recipients who are expected to receive the same message from a sender can exchange the received signed messages to expose and prove equivocation. This principle underlies many append-only logs [MS02; Fel+10; Fah+14; Fel+12].

However, it is often not possible to impose a penalty on a maliciously or carelessly equivocating sender after the fact, as the sender may be anonymous or pseudonymous. Even when the sender is not anonymous and risks reputation once a case of equivocation is detected, the effect of such paltering can be damaging.

**Key Idea.** Our key idea is to let the sender create a time-locked Bitcoin deposit [And+14; BK14; KB14; BtcWikiB] that can be opened by the recipients if the sender equivocates. In that case, the funds will be given either to a predefined beneficiary or, once the expiry time of the deposit is reached, to the miners. If the expected loss is high enough, the attacker has no incentive to make conflicting statements.

**Threat Model.** The attacker is a malicious sender whose goal is to equivocate without losing the deposit. To achieve that goal, the attacker can deviate arbitrarily from the prescribed protocol but does not risk to lose its deposit if the expected loss is higher than the expected gain.

We assume that the attacker cannot break the fundamental security of Bitcoin, e.g., the attacker does not have the majority of computing power in the Bitcoin network.

**Non-equivocation Contracts.** We describe the main idea of non-equivocation contracts, which are a form of smart contracts [BtcWikiB; But13; Kos+16], in more detail. The sender $A$ creates a time-locked deposit to guarantee honest behavior. The deposit is secured by the sender's secret key $sk_A$; the corresponding public key is $pk_A$. Furthermore, the deposit expires at some point $T$ in the future. That is, the sender $A$ cannot access the funds in the deposit until time $T$ (despite knowledge of the secret key $sk_A$). Before time $T$, only $A$ *together* with a predefined beneficiary $P$ can access the funds. This beneficiary will be given the funds if $A$ equivocates. (There is also a variant of deposits for which the beneficiary is a randomly selected miner. We will explain this later.)

Non-equivocation contracts rely on the possibility to learn the key $sk_A$ if the sender $A$ equivocates. To enforce this cryptographically, we introduce *accountable assertions*, which allow party $A$ to produce assertions $\tau$ of statements $st$ in contexts $ct$ (where $st$ and $ct$ can be arbitrary bitstrings) under the public key $pk_A$.

The sender $A$ is held accountable in the following sense: If $A$ behaves honestly, the secret key $sk_A$ will stay secret, and $A$ can use it to withdraw the deposit once time $T$ has been reached. However, if $A$ equivocates to some honest parties $B$ and $C$, i.e., $A$ asserts two different statements $st_0 \neq st_1$ in the same context $ct$, then $B$ and $C$ can use $st_0$, $st_1$, $ct$ and the two corresponding assertions $\tau_0$ and $\tau_1$ to extract the sender's secret key $sk_A$. Due to the way the deposit is created, the recipients $B$ and $C$ alone cannot make use of $sk_A$. However, $B$ and $C$ can send $sk_A$ to the beneficiary $P$, who can use $sk_A$ (and some credentials belonging to $P$) to withdraw the deposit and thereby penalize the malicious sender $A$. Note that $B$, $C$ and $P$ could as well be protocol parties that belong to essentially the same distributed entity but are just not synchronized when receiving statements from $A$.

## 3.2 Deposits and Payment Channels

Throughout this chapter, we rely on advanced features of Bitcoin scripts (Section 2.2) to create *deposits* and *payment channels*, which we explain in this section.

### 3.2.1 Deposits

Using specially-crafted scripts, funds can be locked away in a so-called *deposit*, where they can only be accessed under a set of predetermined conditions. While scripts can express a variety of such conditions [BDG], we focus on *time-locked* deposits with the property that the depositor cannot access the funds in the deposit until a specified *expiry time*.

With non-equivocation contracts in mind, we consider two types of deposits that differ in the *beneficiary*, i.e., the party that receives the funds in case of equivocation. The deposits of the first type do not specify a beneficiary. In this case, the beneficiary will be a randomly selected miner. Deposits of the second type are associated with an *explicitly* beneficiary $P$ identified by his public key $pk_P$.

**Creating Deposits.** To create time-locked deposits, we use the novel Bitcoin script command `CHECKLOCKTIMEVERIFY` [BIP65]. This command takes one argument $T$, the *expiry time*, from the execution stack and compares it to the *nLockTime* data field of the transaction. If $nLockTime < T$, the evaluation fails and the transaction is consequently invalid. Thus, only transactions with $nLockTime \geq T$ can spend the funds covered by such a script. By the semantics of *nLockTime*, those transactions are valid only in blocks found after time $T$, and consequently, the funds protected by `CHECKLOCKTIMEVERIFY` are spendable only after $T$.

We remark that the value of *nLockTime* can be specified either by a UNIX timestamp or a *height of a block*, which is the number of blocks that precede it in the blockchain. Throughout the paper, we use timestamps, and to simplify presentation, we ignore that miners have some flexibility to lie about the current time when writing a timestamp to a block they mine [BtcWikiA]; at least 120 min must be added to $T$ to account for that issue.

**Deposits Without Explicit Beneficiary.** Suppose that some party $A$ wishes to create a deposit with expiration time $T$ without an explicit beneficiary. Then, party $A$ sends the desired amount ₿ $d$ to the following script:

```
(T + T_conf) CHECKLOCKTIMEVERIFY DROP
pk_A CHECKSIG
```

The literals $(T + T_{conf})$ and $pk_A$ in the script denote push operations that push a constant value on the stack. The value $T_{conf}$ is a safety margin; we postpone its discussion to the analysis of non-equivocation contracts (Section 3.5.1).

The first line of the script ensures that the deposit cannot be spent before time $T$ as explained. (`DROP` just drops the constant value from the stack.) In the second line, CHECKSIG takes the key $pk_A$ and a signature $\sigma$ from the stack; $\sigma$ is supposed to be provided by the spender on the initial stack; the command verifies that $\sigma$ is a valid signature of the spending transaction under the key $pk_A$ and pushes the boolean result of the verification on the stack. This boolean value is the output of the script. Thus, if the check succeeds, the transaction is valid; otherwise it is invalid. In sum, the script ensures that the funds can only be spent after $T$ and only by a transaction signed under $pk_A$.

If the corresponding secret key $sk_A$ is revealed, everybody can create transactions that try to spend the funds from time $(T + T_{conf})$ on. Whenever this happens, each miner has a large incentive to include a transaction in a block that sends the money to an own key. Consequently, we can assume that the miner who finds the next block will claim the funds.

**Deposits with Explicit Beneficiary.** Suppose that some party $A$ wishes to create a deposit with an explicit beneficiary $P$. Then, party $A$ sends the desired amount ₿ $d$ to the following script:

```
IF
      $pk_P$ CHECKSIGVERIFY
ELSE
      $(T + \tilde{T}_{conf} + \tilde{T}_{net})$ CHECKLOCKTIMEVERIFY DROP
ENDIF
$pk_A$ CHECKSIG
```

In this script $\tilde{T}_{conf}$ and $\tilde{T}_{net}$ are safety margins, which will be discussed below. IF obtains its condition from the stack, allowing the spender to choose the branch to be executed. CHECKSIGVERIFY is like CHECKSIG but causes the whole script to fail immediately if the signature is not valid (instead of pushing the result of the signature verification to the stack).

The script ensures that before time $T$, the funds can be spent only if the spending transaction is signed under both $pk_A$ and $pk_P$. Thus, if $P$ learns $sk_A$ before time $T$, he can spend the funds. Otherwise, $A$ is refunded after time $(T + \tilde{T}_{conf} + \tilde{T}_{net})$, even if $P$ disappears.

The safety margins are necessary because the closing transaction must have been broadcast to the Bitcoin network and confirmed by it before the deposit can be spent by $A$ alone. For the broadcast, $\tilde{T}_{net} = 10$ min is more than sufficient [DW13]. For the confirmations, we except the network to find 24 blocks in $\tilde{T}_{conf} = 240$ min. Since their arrival is Poisson-distributed, the probability that fewer than six desired blocks have been found is $\Pr[X \leq 5] < 2^{-18}$ for $X \sim \mathrm{Pois}(24)$.

### 3.2.2 Payment Channels

Payment channels [Spi13; BIP65] allow a party $A$ to perform many transactions to a predefined recipient $B$ up to a predefined amount ₿ $d$ of money. Although establishing a channel between $A$ and $B$ involves waiting for a transaction to be confirmed, the advantages of a payment channels are various: First, no matter how many payments are sent, only two transactions need to be included in the blockchain (one to establish the channel and one to close it). This makes payment channels a promising method to scale the Bitcoin network to higher transaction throughputs; advanced methods support payments over multiple intermediate hubs, e.g., the Lightning Network [PD]. Second, $A$ can perform payments to $B$ through the channel even if both parties are offline. Third, fast transactions are possible through the payment channel because $B$ does not have to wait for the transaction to be confirmed.

Many advanced constructions of payment channels have been proposed in the literature [DW15; PD; Hei+17; Mal+17; DFH18; Dzi+19; KG17; Mil+19; GM17], For our purposes it suffices to consider a very simple unidirectional form of payment channels. We stress that in general our techniques are applicable to more advanced payment channels, though minor adjustments may be necessary.

**Creating a Payment Channel.** To create a payment channel from $A$ to $B$ with maximal payment value ₿ $d$ and expiry time $T$, party $A$ follows the procedure for creating a deposit with explicit beneficiary $B$.

$B$ waits until the deposit is confirmed by the Bitcoin network. From now on, the funds can only be spent if both $A$ and $B$ agree because any spending transaction must be signed by both $A$ and $B$ to be valid. Since $B$ will check any transaction before endorsing it, $B$ is protected from attempts by $A$ to send funds to another party (or back to $A$), i.e., $B$ is protected from double-spending attempts.

**Paying through the Channel.** The channel has an associated state $b$ that specifies how much of the ₿ $d$ has been paid so far to $B$. In the beginning, $b = 0$, i.e., all money in the channel belongs to $A$ and none belongs to $B$. To pay through the channel, i.e., to raise $b$ to $b'$, party $A$ creates an ordinary Bitcoin transaction that sends ₿ $b'$ from the deposit to $B$. Party $A$ signs this transaction with the secret key $sk_A$ and sends the transaction to $B$, who validates the transaction and the correctness of the signature. However, the transaction is not yet signed by $B$ or published to the Bitcoin network.

**Closing the Channel.** The channel needs to be closed before time $T$. When the recipient $B$ wants to close the channel at some state $\hat{b}$, then $B$ sends the most recently received transaction, i.e., the one with the value $\hat{b}$, to the Bitcoin network. Once the network confirms the transaction, $B$ has received ₿ $\hat{b}$. If $B$ does not close the channel by time $T$, e.g., as $B$ has disappeared, $A$ can claim the whole channel of value ₿ $d$.

## 3.3 Accountable Assertions

In this section we introduce our main tool *accountable assertions*. Intuitively, this primitive allows parties to assert *statements* in *contexts* such that they can be held accountable for equivocation: On the one hand, if a party holding a secret key *ask* asserts two different statements $st_0 \neq st_1$ in the same context *ct*, then a public algorithm can *extract* the secret key *ask* of the party from the two assertions. On the other hand, secrecy of the secret key *ask* remains intact for a well-behaved non-equivocating party.

Accountable assertions are supposed to be attached to other public-key primitives, i.e., the key pairs are supposed to correspond to key pairs of the other primitive. For example, the key pairs of our scheme will be valid discrete logarithm key pairs like in the ECDSA signature scheme [JMV01] currently used in Bitcoin, and like in the Schnorr signature scheme [Sch91] which has been proposed to be adopted in Bitcoin [Core17]. Attaching accountable assertions to other primitives is crucial in practice because the concrete secret key used in accountable assertions needs to be worth something, e.g., for redeeming funds. Otherwise, the party has no incentive to keep it secret in the first place.

**Definition 3.1** (Accountable Assertions). *An accountable assertion scheme $\mathcal{S}$ is a tuple of ppt algorithms $\mathcal{S} = (\mathsf{Gen}, \mathsf{Assert}, \mathsf{Verify}, \mathsf{Ext})$ as follows:*

$(apk, ask, auxsk) := \mathsf{Gen}(1^\lambda)$: *The* key generation *algorithm outputs a key pair consisting of a public key apk and a secret key ask, and auxiliary secret information auxsk. It is required that for each public key, there is exactly one secret key, i.e., for all $\lambda$ and all outputs $(apk, ask, auxsk)$ and $(apk', ask', auxsk')$ of $\mathsf{Gen}(1^\lambda)$ with $apk = apk'$, we have $ask = ask'$.*

$\tau/\bot := \mathsf{Assert}(ask, auxsk, ct, st)$: *The stateful* assertion algorithm *takes as input a secret key ask, auxiliary secret information auxsk, a context ct, and a statement st. It returns either an assertion $\tau$ or $\bot$ to indicate failure.*

$b := \mathsf{Verify}(apk, ct, st, \tau)$: *The* verification algorithm *outputs* 1 *if and only if $\tau$ is a valid assertion of a statement ct in the context st under the public key apk.*

$ask/\bot := \mathsf{Ext}(apk, ct, st_0, st_1, \tau_0, \tau_1)$: *The* extraction algorithm *takes as input a public key apk, a context ct, two statements $st_0, st_1$, and two assertions $\tau_0, \tau_1$. It outputs either the secret key ask or $\bot$ to indicate failure.*

*The accountable assertion scheme $\mathcal{S}$ is* correct *if for all security parameters $\lambda$, all keys $(apk, ask, auxsk) := \mathsf{Gen}(1^\lambda)$, all statements st, all contexts ct, and all assertions $\tau \neq \bot$ resulting from a successful assertion $\tau := \mathsf{Assert}(ask, auxsk, ct, st)$, we have* $\mathsf{Verify}(apk, ct, st, \tau) = 1$.

Note that the secret information is divided into a secret key *ask* and auxiliary secret information *auxsk*. In case of equivocation, only *ask* will be guaranteed to be extractable but not *auxsk*.

**Completeness.** Our definition of accountable assertions allows the assertion algorithm to fail. We do not consider such failure a problem if it happens only with small (but not necessarily negligible) probability. The reason is that failure hurts only the liveness of the system that makes use of the accountable assertions, but liveness is typically not guaranteed anyway due to unreliable networks. As a consequence, we do not insist generally on accountable assertions fulfilling a completeness criterion. At first glance, this might look a bit contrived, but the purpose of this is to trade off reliability against efficiency. Accountable assertions are, unlike signatures, not required to be unforgeable, and it turns out that setting unforgeability aside will allow for more efficient constructions.

To understand how failing and unforgeability are related, suppose an attacker asks a party to assert a statement $st_0$ in a context $ct_0$, i.e., to output $\tau_0 := \mathsf{Assert}(ask, auxsk, ct_0, st_0)$. Due to the lack of unforgeability, the attacker might use $\tau_0$ to obtain another assertion $\tau_1$ that is valid for some *related* but different context $ct_1 \neq ct_0$ and the same statement $st_0$, i.e., $\mathsf{Verify}(apk, ct_1, st_0, \tau_1) = 1$. So far, this is not a problem: the attacker cannot use the extraction algorithm to obtain the secret key *ask* from $\tau_0$ and $\tau_1$ because the two assertions are valid in different contexts $ct_0 \neq ct_1$. However, the attacker can now ask the party to assert another statement $st_1 \neq st_0$ in the context $ct_1$, i.e., to output $\tau_1' := \mathsf{Assert}(ask, auxsk, ct_1, st_1)$. Observe that this is a valid request: the attacker does not ask the party to equivocate because the party has not asserted any statement in the context $ct_1$ so far. But if the party replied to the request, the attacker could run the extraction algorithm $\mathsf{Ext}(apk, ct_1, st_1, st_1', \tau_1, \tau_1')$ to extract the secret key *ask*, which clearly should not happen.

To avoid this attack, while allowing for a construction that is "forgeable" as just described, the stateful assertion algorithm may fail if it detects that the context $ct_1$, for which an assertion is requested, is related to a previously used context $ct_0$. Nevertheless, the ability of the attacker to force failure may be a problem in certain scenarios, e.g., if it allows the attacker to perform a denial-of-service attack. In those cases, it is possible to consider *complete* accountable assertions, which are guaranteed to succeed on all honestly chosen inputs.

**Definition 3.2** (Completeness). *An accountable assertion scheme* $\mathcal{S} = (\mathsf{Gen}, \mathsf{Assert}, \mathsf{Verify}, \mathsf{Ext})$ *is* complete *if for all security parameters* $\lambda$*, all outputs* $(apk, ask, auxsk)$ *of* $\mathsf{Gen}(1^\lambda)$*, all statements* $st$*, and all contexts* $ct$*, we have*

$$\mathsf{Assert}(ask, auxsk, ct, st) \neq \bot.$$

Note that the definition of accountable assertions additionally demands correctness whenever $\mathsf{Assert}(ask, auxsk, ct, st) \neq \bot$.

### 3.3.1 Security Properties

Accountable assertions need to fulfill two security properties. The first security property is *extractability*, which states that whenever two distinct statements have been asserted in the same context, the secret key can be extracted.

**Definition 3.3** (Extractability). *An accountable assertion scheme* $\mathcal{S} = (\mathsf{Gen}, \mathsf{Assert}, \mathsf{Verify}, \mathsf{Ext})$ *is* extractable *if for all ppt attackers* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr\left[\begin{array}{l} \mathsf{Ext}(apk, ct, st_0, st_1, \tau_0, \tau_1) \neq ask \\ \quad \wedge\ \mathsf{Verify}(apk, ct, st_0, \tau_0) = 1 \\ \quad \wedge\ \mathsf{Verify}(apk, ct, st_1, \tau_1) = 1 \\ \quad \wedge\ st_0 \neq st_1 \end{array} \middle| \ (apk, ct, st_0, st_1, \tau_0, \tau_1) := \mathcal{A}(1^\lambda) \right] \leq \mathsf{negl}(\lambda).$$

*Here,* $ask$ *is the unique secret key corresponding to* $apk$*.*

The second security property *secrecy* is opposed to extractability. Secrecy prevents the extraction of the secret key against an attacker who can ask the challenger to assert chosen statements in chosen contexts. Since accountable assertions are extractable, the attacker's success is excluded after requesting the assertion of two different statements in the same context.

**Definition 3.4** (Secrecy). *An accountable assertion scheme* $\mathcal{S} = (\mathsf{Gen}, \mathsf{Assert}, \mathsf{Verify}, \mathsf{Ext})$ *is* secret *if for all ppt attackers* $\mathcal{A}$*, there is a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr[\mathsf{Sec}^{\mathcal{S}}_{\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda),$$

*where the experiment* $\mathsf{Sec}^{\mathcal{S}}_{\mathcal{A}}(\lambda)$ *is defined in Fig. 3.1.*

**Limitations of the Secrecy Definition.** Recall that a secret key used with accountable assertions must be worth something, e.g., a valid secret key that protects funds

```
**experiment** $\mathrm{Sec}^{\mathcal{S}}_{\mathcal{A}}(\lambda)$
    $(apk, ask, auxsk) := \mathsf{Gen}(1^\lambda)$
    $Q := \emptyset$
    $ask^* := \mathcal{A}^{\mathsf{Assert}'(ask, auxsk, \cdot, \cdot)}(apk)$
    **return** $ask^* = ask$
        $\wedge\ \forall (ct_0, st_0), (ct_1, st_1) \in Q.\ ct_0 \neq ct_1 \vee st_0 \neq st_1$
**oracle** $\mathsf{Assert}'(ask, auxsk, ct, st)$
    $Q := Q \cup \{(ct, st)\}$
    **return** $\mathsf{Assert}(\mathsf{msk}, auxsk, ct, st)$
```

**Figure 3.1:** Experiment $\mathrm{Sec}^{\mathcal{S}}_{\mathcal{A}}(\lambda)$ for Definition 3.4

in Bitcoin. We would like to draw the reader's attention to the fact that the definition of secrecy does not take into account the other usages of the secret key. That is, while our secrecy definition of accountable assertions is meaningful in itself, it is only a heuristic for analyzing their security when combined with other primitives, and it is formally not guaranteed that the use of secret accountable assertions keeps the security of the other primitives intact.[1]

While we are confident that the combined use of our concrete accountable assertions construction (Section 3.4) together with ECDSA or Schnorr signatures does not render these signature schemes insecure in practice, a more formal treatment of the joint use of accountable assertions with other primitives is desirable. We leave this for future work.

**Relation to DAPS.** Double-authentication-preventing signatures (DAPS) [PS14; PS17] have similar properties as accountable assertions but are additionally required to be unforgeable. We have discussed an informal relation between the unforgeability of accountable assertions and their completeness. This intuition can be formalized, and it turns out that a slightly modified variant of our accountable assertions construction (Section 3.4) is an efficient DAPS scheme. We refer the reader to Section 3.8 for a discussion.

[1]We cannot hope that stronger variants of secrecy, e.g., definitions based on indistinguishability or even non-interactive zero-knowledge, help to achieve some form of general composability of accountable assertions with signatures: Given an accountable assertion scheme with a secrecy notion based on non-interactive zero-knowledge, one can construct a pathological unforgeable signature scheme that becomes forgeable when the attacker learns an accountable assertion that has been generated with the same secret key.

## 3.4 Construction

In this section, we propose a construction of accountable assertions based on chameleon hash functions. Our construction builds upon the idea of chameleon authentication trees (CATs), as suggested by Schröder and Schröder [SS12] and improved in follow-up schemes [SS15; Kru+16]. In contrast to these schemes, the novelty of our construction is the extractability.

**Chameleon Hashes.** A chameleon hash function is a randomized hash function that is collision-resistant but provides a trapdoor to efficiently compute collisions [KR00]. Formally, a chameleon hash function $\mathcal{CH}$ is a tuple of ppt algorithms $(\mathsf{GenCh}, \mathsf{Ch}, \mathsf{Col})$. The key generation algorithm $\mathsf{GenCh}(1^\lambda)$ returns a key pair $(cpk, csk)$ consisting of a public key $cpk$ and a trapdoor $csk$. The evaluation function $\mathsf{Ch}(cpk, x, r)$ produces a hash value for a message $x$ and a random value $r$; we may write just $\mathsf{Ch}(x, r)$ whenever $cpk$ is clear from the context. The collision-finding algorithm $\mathsf{Col}(csk, x_0, r_0, x_1)$ takes as input a trapdoor $csk$ and a triple $(x_0, r_0, x_1)$; it outputs some value $r_1$ such that $\mathsf{Ch}(cpk, x_0, r_0) = \mathsf{Ch}(cpk, x_1, r_1)$.

Chameleon hash functions need to fulfill collision-resistance and uniformity as defined by Krawczyk and Rabin [KR00].

**Definition 3.5** (Collision-Resistance). *A chameleon hash function $\mathcal{CH} = (\mathsf{GenCh}, \mathsf{Ch}, \mathsf{Col})$ is* collision-resistant *if for all ppt attackers $\mathcal{A}$,*

$$\Pr\left[\begin{array}{c} \mathsf{Ch}(cpk, x_0, r_0) = \mathsf{Ch}(cpk, x_1, r_1) \\ \wedge\ (x_0, r_0) \neq (x_1, r_1) \end{array} \middle| \begin{array}{l} (cpk, csk) := \mathsf{GenCh}(1^\lambda); \\ (x_0, r_0, x_1, r_1) := \mathcal{A}(cpk) \end{array}\right]$$

*is negligible in $\lambda$.*

**Definition 3.6** (Uniformity). *A chameleon hash function $\mathcal{CH} = (\mathsf{GenCh}, \mathsf{Ch}, \mathsf{Col})$ is* uniform *if for all messages $x_0$, $x_1$, and all trapdoors csk output by* $\mathsf{GenCh}$*, and for a uniformly random value $r_0$, the value $\mathsf{Col}(csk, x_0, r_0, x_1)$ is a uniformly distributed random value as well.*

This definition of uniformity, which is also used by [SS12], is slightly stronger than the one in by Krawczyk and Rabin [KR00], which mandates only that $\mathsf{Ch}(cpk, x, r)$ is distributed independently of $x$.

In addition to these standard security properties, we require the trapdoor to be extractable from a collision. While this key exposure property is typically considered a problem in the literature [AM04; CZK04; SS12], it will be the crucial tool to ensure extractability of our construction.

**Definition 3.7** (Extractability). *A chameleon hash function $\mathcal{CH} = (\mathsf{GenCh}, \mathsf{Ch}, \mathsf{Col})$ is extractable if*

- *there exists a deterministic polynomial-time algorithm* $\mathsf{ValidatePk}$*, which takes as input a bitstring cpk and returns* 1 *if and only if there is a trapdoor csk such that* $(cpk, csk)$ *is a possible output of* $\mathsf{GenCh}(1^\lambda)$*, and*
- *there exists a deterministic polynomial-time algorithm* $\mathsf{ExtractCsk}$ *with the following property: For all key pairs* $(cpk, csk)$ *output by* $\mathsf{GenCh}(1^\lambda)$*, and for all collisions under cpk, i.e., for all input pairs* $(x_0, r_0)$ *and* $(x_1, r_1)$ *with* $x_0 \neq x_1$ *and* $\mathsf{Ch}(cpk, x_0, r_0) = \mathsf{Ch}(cpk, x_1, r_1)$*, we have*

$$\mathsf{ExtractCsk}(cpk, x_0, r_0, x_1, r_1) = csk.$$

## 3.4.1 Intuition

One obvious but flawed approach to construct accountable assertions is to let the assertion algorithm output a value $r$ such that $ct = \mathsf{Ch}(st, r)$. The intuition is that if the attacker does this for two different statements $st_0, st_1$ in the same context $ct$, then this would yield a collision $\mathsf{Ch}(st_0, r_0) = ct = \mathsf{Ch}(st_1, r_1)$ in the chameleon hash function, and one could extract the trapdoor. This simple idea does not work. The reason is that $ct$ would live in the output space of the chameleon hash function but in all known constructions of chameleon hash functions compatible with discrete logarithm key pairs as used in Bitcoin, the trapdoor can only be used to find collisions efficiently, not to invert the function, so assertions cannot be computed efficiently.[2]

**Full Idea.** Observe that the aforementioned approach works, however, as a scheme that supports only one context, for which inverting the chameleon hash is not necessary. If the public key of the accountable assertions scheme includes $\mathsf{Ch}(x^*, r^*)$ for randomly chosen $x^*$ and $r^*$, then one can use the trapdoor to compute $r$ as an assertion for a statement $st$ such that $\mathsf{Ch}(x^*, r^*) = \mathsf{Ch}(st, r)$.

The basic idea of our construction is to generalize this approach to many contexts by applying it recursively, resulting in a Merkle-style tree based on chameleon hash functions. The contexts are associated with the leafs of the tree, and a digest of the root node is part of the public key.

Let $n$ denote the arity and $\ell$ denote the depth of the tree. We explain the main steps with the help of Fig. 3.2 for $n = 3$. In our construction (a digest of) the context defines

[2]To the best of our knowledge, the only chameleon hash function that supports inverting is based on the hardness of factoring [KR00]. Poettering and Stebila; Poettering and Stebila's construction of double-authentication-preventing signatures (DAPS) [PS14], which are similar to accountable

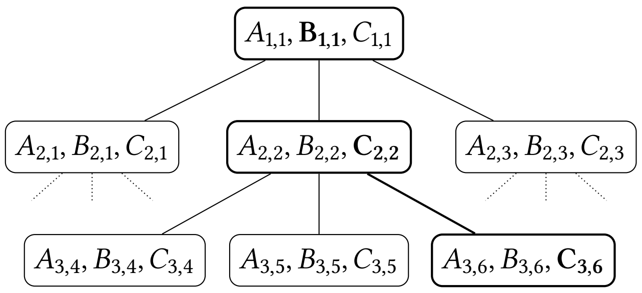

$A_{1,1}, \mathbf{B_{1,1}}, C_{1,1}$
$A_{2,1}, B_{2,1}, C_{2,1}$
$A_{2,2}, B_{2,2}, \mathbf{C_{2,2}}$
$A_{2,3}, B_{2,3}, C_{2,3}$
$A_{3,4}, B_{3,4}, C_{3,4}$
$A_{3,5}, B_{3,5}, C_{3,5}$
$A_{3,6}, B_{3,6}, \mathbf{C_{3,6}}$

**Figure 3.2:** A tree as in our construction

its position in the tree. That is, the context with the lowest digest is stored in the leftmost leaf and the context with the highest digest in the rightmost node. Since the tree is of exponential size, storing or computing the entire tree at once is not possible. Instead, we compute each element $A_{i,j}, B_{i,j}, C_{i,j}$ as a chameleon hash value of its children, i.e., the element $A_{i,j}$ is computed as $A_{i,j} := \mathsf{Ch}((A_{i+1,s}, B_{i+1,s}, C_{i+1,s}), r_{i,j})$ for some integer $s$. So far, we have described an $n$-ary Merkle tree whose nodes are computed via a chameleon hash function.

Now we explain how to handle an exponential number of nodes without computing all of them. The basic idea is to exploit the collision property of the chameleon hash function. Instead of computing the node $A_{i,j}$ as $A_{i,j} := \mathsf{Ch}((A_{i+1,s}, B_{i+1,s}, C_{i+1,s}), r_{i,j})$, we replace all elements with dummy elements, i.e., $A_{i,j} := \mathsf{Ch}(x_{i,j}, r_{i,j})$. These elements are derived via a pseudo-random function PRF with key $k$, i.e., $x_{i,j} := \mathsf{PRF}_k(i, j)$, and can be computed on the fly. That is, to compute $A_{i,j}$, no other tree nodes are necessary. Since all elements are computed deterministically, this modification results in an exponential number of nodes without any connection to each other. We re-establish this connection using the trapdoor of the chameleon hash function whenever we assert a new element.

We illustrate the assertion operation with Fig. 3.2. Assume that we would like to assert a statement in the context (associated with) $C_{3,6}$. To do so, we need to compute the elements $A_{3,6}, B_{3,6}, A_{2,2}, B_{2,2}, A_{1,1}, C_{1,1}$ and the corresponding randomness for each node. This information will suffice for the verifier to reconstruct the *assertion path* from $C_{3,6}$ to the root as in an ordinary Merkle tree. To compute the aforementioned elements, we compute all dummy elements $x^A_{3,6}, x^B_{2,2}, x^C_{1,1}$ and we also derive the randomness for each node via PRF. Now, to assert the statement $st$ in the context $C_{3,6}$, we compute the first collision in $C_{3,6} := \mathsf{Ch}(x^C_{3,6}, r^C_{3,6})$. We use the trapdoor of the chameleon hash to find a matching randomness $r'$ such that $\mathsf{Ch}(x^C_{3,6}, r^C_{3,6}) = C_{3,6} = \mathsf{Ch}(\mathsf{S}(st), r')$, where

assertions (Section 3.8), can be interpreted as an elaboration of the described idea.

S computes a digest of the statement $st$. Now, to assert $(A_{3,6}, B_{3,6}, C_{3,6})$ with respect to the parent $C_{2,2}$, we need to find a second collision in $C_{2,2}$, which is computed as $C_{2,2} := \mathsf{Ch}(x^C_{2,2}, r^C_{2,2})$. Again, we use the trapdoor to compute some randomness $r''$ such that $\mathsf{Ch}(x^C_{2,2}, r^C_{2,2}) = C_{2,2} = \mathsf{Ch}(h, r'')$ where $h = (A_{3,6}, B_{3,6}, C_{3,6})$. We repeat this procedure up to the root. Observe that independent of the statements asserted in the contexts $A_{3,6}$, $B_{3,6}$, and $C_{3,6}$, the value $h$ will always be the same because the first collision is always computed in the leaf. This concludes the description of the underlying asserted data structure.

Now, we will explain how to extract the secret key in the case that the sender asserts two different statements in the same context. Let us assume that the sender asserted two statements $st_0, st_1$ in the context associated with $C_{3,6}$.

In the simplest case, there exist two pairs $(st_0, r_0), (st_1, r_1)$ such that $\mathsf{Ch}(\mathsf{S}(st_0), r_0) = C_{3,6} = \mathsf{Ch}(\mathsf{S}(st_1), r_1)$. (This is like in the “first approach”.) In a more complicated case, we could have $\mathsf{Ch}(\mathsf{S}(st_0), r_0) = C_{3,6} \neq C'_{3,6} = \mathsf{Ch}(\mathsf{S}(st_1), r_1)$ because the attacker could have used a collision in $C_{2,2}$ to associate its rightmost child with a value $C'_{3,6} \neq C_{3,6}$. But then, this collision can be used to extract the trapdoor. Generally speaking, we will find a collision somewhere on the path from the leaf to the root. An algorithm implementing this idea always terminates because a digest of the root is fixed in the public key.

## 3.4.2 Full Construction

We present the full description of our scheme. Let $\ell$ and $n$ be positive integers defining the height and the branching factor of a tree whose number of leafs $n^{\ell-1}$ is polynomial in the security parameter $\lambda$. Let $\mathsf{PRF}_k$ be a pseudorandom function, and let H and S be two collision-resistant hash functions.[3] Furthermore, let G be a hash function modeled as a random oracle. (We note that for extractability, it suffices that G is collision-resistant.)

Furthermore, let L be a (non-cryptographic) hash function that maps bitstrings (contexts) to leafs $\{1, \ldots, n^{\ell-1}\}$. If collisions in L occur only with low probability, then the assertion algorithm fails only with low probability. If the context space is equal to the output space of L, then L can be the identity function. (Note that we do not and cannot require L to be collision-resistant in a cryptographic sense because its output space is only polynomially large in the security parameter $\lambda$.)

Let $\mathcal{CH} = (\mathsf{GenCh}, \mathsf{Ch}, \mathsf{Col}, \mathsf{ExtractCsk})$ be a collision-resistant, uniform, and extractable chameleon hash function. The accountable assertion scheme allows asserting in $\ell^n$ contexts and is defined as follows.

[3]For the sake of readability, we omit the keys of the hash functions and assume they are drawn by the key generation algorithm defined below.

**Key Generation.** The key generation algorithm chooses a key for the pseudorandom function $k := \{0,1\}^\lambda$, and a key pair $(cpk, csk) := \mathsf{GenCh}(1^\lambda)$ for the chameleon hash function. Let $p$ be a unique identifier for the position of the root node. The algorithm computes the entries in the root node as $y_i^0 := \mathsf{Ch}(x_i^1, r_i^1)$ where $x_i^1 := \mathsf{PRF}_k(p, i, 0)$, $r_i^1 := \mathsf{PRF}_k(p, i, 1)$, and $i \in \{1, \dots, n\}$. It sets $z := \mathsf{H}(y_1^1, \dots, y_n^1)$ and finally $apk := (cpk, z)$, $ask := csk$, and $auxsk := k$.

**Assertion.** The stateful assertion algorithm maintains an initially empty set $L$ of used leaf positions. To assert a statement $st$ in a context $ct$, the algorithm verifies that $\mathsf{L}(ct) \notin L$ and fails by outputting $\perp$ otherwise.[4] Then, it adds $\mathsf{L}(ct)$ to $L$ and computes the assertion path $(Y_\ell, a_\ell, Y_{\ell-1}, a_{\ell-1}, \dots, Y_1, a_1)$ from a leaf $Y_\ell$ to the root $Y_1$. Each node $Y_j = (y_1^j, \dots, y_n^j)$ stores $n$ entries at positions $a_j \in \{1, \dots, n\}$ within the node. $Y_\ell$ is the leaf that stores the entry with the number $\mathsf{L}(ct)$, counted across all leaves from left to right, and $a_\ell$ is the position of this entry within $Y_\ell$. In the following, let $x_i^j := \mathsf{PRF}_k(p_j, i, 0)$ and $r_i^j := \mathsf{PRF}_k(p_j, i, 1)$, where $p_j$ is a unique identifier of the position of the node $Y_j$.

Compute $Y_\ell$: Assert the statement $st$ with respect to $Y_\ell$ by computing $r'^\ell_{a_\ell} := \mathsf{Col}(csk, x^\ell_{a_\ell}, r^\ell_{a_\ell}, \mathsf{S}(st))$. Compute the entry at position $a_\ell$ as $y^\ell_{a_\ell} := \mathsf{G}(\mathsf{Ch}(\mathsf{S}(st)), r'^\ell_{a_\ell}), r'^\ell_{a_\ell}) = \mathsf{G}(\mathsf{Ch}(x^\ell_{a_\ell}, r^\ell_{a_\ell}), r'^\ell_{a_\ell})$. Compute the remaining entries in node $Y_\ell$ as $y_i^\ell := \mathsf{Ch}(x_i^\ell, r_i^\ell)$ for $i \in \{1, \dots, n\} \setminus \{a_\ell\}$. The leaf $Y_\ell$ stores the entries $(y_1^\ell, \dots, y_n^\ell)$. Let $z_\ell := \mathsf{H}(y_1^\ell, \dots, y_n^\ell)$ and let further $f_\ell := (y_1^\ell, \dots, y^\ell_{a_\ell - 1}, y^\ell_{a_\ell + 1}, \dots, y_n^\ell)$.

Compute the nodes up to the root for $h := \ell - 1, \dots, 1$: Assert the value $z_{h+1}$ with respect to $Y_h$ by computing $r'^h_{a_\ell} := \mathsf{Col}(csk, x^h_{a_\ell}, r^h_{a_\ell}, z_{h+1})$. Compute the entry $y^h_{a_\ell} := \mathsf{Ch}(z_{h+1}, r'^h_{a_\ell}) = \mathsf{Ch}(x^h_{a_\ell}, r^h_{a_\ell})$. Compute the remaining entries in this node $Y_h$ as $y_i^h = \mathsf{Ch}(x_i^p, r_i^p)$ for $i \in \{1, \dots, n\} \setminus \{a_\ell\}$. The node $Y_h$ stores the entries $(y_1^h, \dots, y_n^h)$. Let $z^h := \mathsf{H}(y_1^h, \dots, y_n^h)$ and $f_h := (y_1^h, \dots, y^h_{a_\ell - 1}, y^h_{a_\ell + 1}, \dots, y_n^h)$.

The assertion is $\tau := ((r'^\ell_{a_\ell}, f_\ell, a_\ell), \dots, (r'^1_1, f_1, a_1))$.

**Verification.** The verification algorithm parses the assertion public key $apk$ as $(cpk, z)$ and outputs 0 if $\mathsf{ValidatePk}(cpk) = 0$.

Otherwise it parses $\tau$ as $((r'^\ell_{a_\ell}, f_\ell, a_\ell), \dots, (r'^1_1, f_1, a_1))$, and checks the validity of a statement $st$ in a context $ct$ by reconstructing the nodes $(Y_\ell, Y_{\ell-1}, \dots, Y_1)$ in a bottom-up order, from the leaf $Y_\ell$ to the root $Y_1$, which contains the entries $y_1^1, , \dots, y_n^1$. The verification algorithm outputs 1 if and only if $\mathsf{H}(y_1^1, \dots, y_n^1) = z$.

[4]The set $L$ can be implemented efficiently by a Bloom filter [Blo70; TRL12], at the cost of a slightly increased probability of failure. A Bloom filter is a space-efficient probabilistic data structure. It may indicate "$x \in L$" incorrectly with small probability, but it never indicates "$x \notin L$" incorrectly.

**Extraction.** The extraction algorithm takes as input $(apk, ct, st_0, st_1, \tau_0, \tau_1)$. It computes like the verification algorithm the assertion paths for both $st_0$ and $st_1$ from the bottom up to the root until a position in the tree is found where the two assertion paths form a collision in the chameleon hash function, i.e., a position in the tree where values $x_0, r_0$ are used in the assertion path of $st_0$ and values $x_1, r_1$ are used in the assertion path of $st_1$ such that $\mathsf{Ch}(x_0, r_0) = \mathsf{Ch}(x_1, r_1)$. Then the extraction algorithm outputs the secret key $ask = csk := \mathsf{ExtractCsk}(x_0, r_0, x_1, r_1)$ computed via the extraction algorithm of the chameleon hash function. If no such position is found, the extraction algorithm fails and outputs $\perp$.

**Stateless and Complete Variant of the Construction.** We can obtain stateless and complete accountable assertions by slightly modifying the construction at the cost of decreased efficiency as follows. We require the size $n^{\ell-1}$ of the output space of $\mathsf{L}$ to be super-polynomial in the security parameter $\lambda$, and additionally we require $\mathsf{L}$ to be a cryptographic hash function modeled as a random oracle. We drop the check "$\mathsf{L}(ct) \notin L$" in the assertion algorithm, which then fails only with negligible probability because $\mathsf{L}$ is collision-resistant. This eliminates the state from the authentication algorithm. Furthermore, this modification makes the scheme complete, i.e., the assertion algorithm always succeeds.

### 3.4.3 Security Analysis

We establish the security of the construction.

**Theorem 3.1.** *If the chameleon hash function $\mathcal{CH}$ is extractable, the hash functions* $\mathsf{H}$, $\mathsf{G}$, *and* $\mathsf{S}$ *are collision-resistant, then both the normal variant and the stateless and complete variant of construction are extractable.*

*Proof.* Let $\mathcal{A}$ a ppt attacker, and assume that $\mathcal{A}$ breaks extractability, i.e., $\mathcal{A}$ outputs a public key $apk$ and two assertions $\tau_0, \tau_1$ that are valid for different statements $st_0 \neq st_1$ in the same context $ct$, but the extraction algorithm fails to extract the secret key $ask$ given these values.

By construction of the verification algorithm, the assertion paths of $\tau_0$ and $\tau_1$ belong to two Merkle trees $T_0$ and $T_1$ such that *i)* the roots of $T_0$ and $T_1$ are identical, and *ii)* the two leaves of $T_0$ and $T_1$ that belong to the context $ct$ have different inputs $st_0 \neq st_1$ for the chameleon hash function; note that these leaves are at the same position in $T_0$ and $T_1$. Thus, there is a node position on the assertion paths output by $\mathcal{A}$ such that the nodes of $T_0$ and $T_1$ at this position form a collision either in the chameleon hash function $\mathsf{Ch}$, or in one of the collision-resistant hash functions $\mathsf{H}$, $\mathsf{G}$, and $\mathsf{S}$. By construction of the extraction algorithm, this algorithm would not fail to output *ask* if the collision was a collision in the chameleon hash function. Thus, it is a collision in one of hash functions, which happens only with negligible probability and consequently, the event that $\mathcal{A}$ breaks extractability happens only with negligible probability. □

**Theorem 3.2.** *If* $\mathsf{PRF}_k$ *is a pseudorandom function, the chameleon hash function* $\mathcal{CH}$ *is uniform, collision-resistant, and extractable, and the hash function* $\mathsf{G}$ *is modeled as a random oracle, then the (normal variant of) construction is secret.*

*If additionally the hash function* $\mathsf{L}$ *is modeled as a random oracle, then the stateless and complete variant of the construction is secret.*

*Proof.* First, we first give a proof for the stateless and complete variant of the construction, in which the size of the output space of $\mathsf{L}$ is super-polynomial in the security parameter $\lambda$ and $\mathsf{L}$ is modeled as a random oracle.

Assume for contradiction that there is a ppt attacker $\mathcal{A}$ that breaks secrecy. That is, with non-negligible probability, $\mathcal{A}$ outputs the secret key *ask* at the end of $\mathrm{Sec}^{\mathcal{S}}_{\mathcal{A}}(\lambda)$ without querying the assertion oracle for assertions of two different statements in the same context. Let $q(\lambda)$ be a polynomial that upper bounds the number of unique assertion and random oracle queries of $\mathcal{A}(1^\lambda)$.

We construct a reduction $\mathcal{B}$ against the collision-resistance of $\mathcal{CH}$ as follows: Given a public key *cpk*, $\mathcal{B}(cpk)$ chooses a family $\{Q^{\mathsf{L}}_i\}_{0\le i<q(\lambda)}$ of $q(\lambda)$ bitstrings in the output space of $\mathsf{L}$ and a family $\{Q^{\mathsf{G}}_i\}_{0\le i<q(\lambda)}$ of $q(\lambda)$ bitstrings in the output space of $\mathsf{G}$ uniformly at random. ($\mathcal{B}$ can compute $q(\lambda)$ because a description of the polynomial $q$ is hardwired in $\mathcal{B}$.) Then $\mathcal{B}(cpk)$ computes the root of the tree from the bottom up, assuming that the leaf entry with the number $Q^{\mathsf{L}}_i$, counted across all leaves from left to right, is $y_i := Q^{\mathsf{G}}_i$. Entries that are roots of subtrees that do not contain any of those $q$ entries are computed as random dummy entries, i.e., as $\mathsf{Ch}(x, r)$ for random $x$ and $r$. This computation of the root involves computing incomplete assertions $\{\tau_i\}_{0\le i<q(\lambda)}$, which are paths from the leaf entry with the number $Q^{\mathsf{L}}_i$ to the root of the tree. These are incomplete in the following sense: since the computation assumed fixed values $y_i$ for the leaf entries, the randomness value for the level $\ell$ is not determined in $\tau_i = ((\bot, f^i_\ell, a^i_\ell), (r^i_{\ell+1}, f^i_{\ell+1}, a^i_{\ell+1}), \dots, (r^i_1, f^i_1, a^i_1))$. (Recall that an honest assertion contains a randomness value $r'^{\ell}_{a_\ell}$ for level $\ell$ such that $y_i = \mathsf{G}(\mathsf{Ch}(\mathsf{S}(st), r'^{\ell}_{a_\ell}), r'^{\ell}_{a_\ell})$.) For a randomness value $r$, let $\tau_i(r)$ be the complete assertion that is obtained by setting the missing randomness value for level $\ell$ in $\tau_i$ to $r$, i.e., $\tau_i(r) := ((r, f^i_\ell, a^i_\ell), (r^i_{\ell+1}, f^i_{\ell+1}, a^i_{\ell+1}), \dots, (r^i_1, f^i_1, a^i_1))$. Furthermore, let $z$ be the obtained hash value of the root, and let $apk := (cpk, z)$.

After computing the root of tree, $\mathcal{B}(cpk)$ calls $ask := \mathcal{A}^{\mathsf{SimAssert}(\cdot,\cdot)}(apk)$. The random oracles $\mathsf{L}$ and $\mathsf{G}$ and the assertion oracle $\mathsf{SimAssert}$ are implemented as follows, where $G$, $I$, and $R$ are initially empty partial functions.

On query "$\mathsf{G}(s, r)$": If $G(s, r)$ has not yet been set, $\mathcal{B}$ chooses a random value $y$ in the output space of G and sets $G(s, r) := y$. Then, $\mathcal{B}$ returns $G(s, r)$.

On query "$\mathsf{L}(ct)$": If $I(ct)$ has not yet been set, $\mathcal{B}$ chooses an index $i$ that is not yet in the image of $I$ and sets $I(ct) := i$. Then, $\mathcal{B}$ returns $Q^{\mathsf{L}}_{I(ct)}$.

On query "$\mathsf{SimAssert}(ct, st)$": If $I(ct)$ has not yet been set, $\mathsf{L}(ct)$ chooses an index $i$ that it is not yet in the image of $I$ and sets $I(ct) := i$. If $R(ct)$ has not been set, $\mathcal{B}$ chooses a random value $r$ and sets $R(ct) := r$. Finally, $\mathcal{B}$ sets $G(\mathsf{Ch}(\mathsf{S}(st), R(ct)), R(ct)) := Q^{\mathsf{G}}_{I(ct)}$ and returns the complete assertion path $\tau^{I(ct)}(R(ct))$.

After having obtained a candidate secret key $ask$ from $\mathcal{A}^{\mathsf{SimAssert}(\cdot,\cdot)}(apk)$, the reduction $\mathcal{B}$ uses $ask = csk$ to compute and output a collision (on an arbitrary message) in the chameleon hash function $\mathsf{Ch}$.

Observe that $\mathcal{B}$ is efficient. In particular, the computation of the root produces a subset of the parts of the tree that are required for $q(\lambda)$ assertions, i.e., only polynomially many nodes are computed.

We show that the simulation towards $\mathcal{A}$ is correct with overwhelming probability. Let Guess be the event that for some $st$ and $r$, the attacker $\mathcal{A}$ queries $\mathsf{G}(\mathsf{Ch}(\mathsf{S}(st), R(ct)), R(ct))$ and later $\mathsf{SimAssert}(ct, st)$, which in turn chooses $R(ct) = r$. Since $R(ct)$ is chosen uniformly at random, Guess occurs only with negligible probability. By construction, $\mathsf{SimAssert}$ overwrites a value of the function $G$ that has been set in query to G if and only if Guess occurs. Observe that as long as this does not happen, the implementations of the oracles are consistent with each other. Furthermore, the outputs of the random oracle are chosen randomly from the correct output spaces, and the outputs of $\mathsf{SimAssert}$ are equally distributed to honestly generated assertions. In particular, the distribution of randomness values in the outputs of $\mathsf{SimAssert}$ and honestly generated assertions is identical because the chameleon hash function is uniform.

Since $\mathcal{A}$ outputs the correct secret key with non-negligible probability by assumption, and the simulation is correct with overwhelming probability, the attacker $\mathcal{B}$ outputs a collision in $\mathsf{Ch}$ with non-negligible probability. This contradicts the collision-resistance of $\mathsf{Ch}$ and concludes the proof for the stateless and complete variant of the construction.

The proof for the normal variant of the construction, in which the size of the output space of $\mathsf{L}$ is only polynomial in the security parameter $\lambda$ and $\mathsf{L}$ is a non-cryptographic hash function, is analogous. We just describe the three main differences: First, $\mathcal{B}$ fixes the leaf entries (in the output space of G) of all leaves and uses them to precompute entire tree, which consists of polynomially many nodes. Second, instead of choosing fresh indices from the output space of $I$, $\mathcal{B}$ chooses uniformly random values. Third, $\mathcal{B}$ implements the set $L$ like the assertion algorithm. □

**Failure Probability of the Assertion Algorithm.** If $\mathsf{L}$ is an adequate hash function, the construction allows a context space of $\{0,1\}^*$. In that case, the probability that the assertion algorithm fails when given $q$ queries is the probability that two contexts $ct_0 \neq ct_1$ appear in the queries with $\mathsf{L}(ct_0) = \mathsf{L}(ct_1)$. Under the assumption that $\mathsf{L} : \{0,1\}^* \rightarrow \{1,\ldots,n^\ell\}$ has uniform outputs, its (birthday) collision probability is below $(q+1)^2/(2 \cdot (n^\ell + 1 - q))$ [PRK98].

### 3.4.4 Instantiation and Implementation

We have implemented the construction given in the previous section. In this section, we describe the details of the implementation, and we evaluate the practicality of the construction, as it will dominate the computation as well as communication costs of non-equivocation contracts. Our implementation is available online [Accas]. It makes use of the `libsecp256k1` library [Wui13], which is used in the Bitcoin Core client, the most widely used implementation of Bitcoin.

**Chameleon Hash Function.** We use a chameleon hash function proposed by Krawczyk and Rabin [KR00], which is secure if the discrete logarithm assumption (Appendix A) holds in the underlying group. In the elliptic curve setting, the chameleon hash function $\mathcal{CH} = (\mathsf{GenCh}, \mathsf{Ch}, \mathsf{Col})$ with extraction algorithm $\mathsf{ExtractCsk}$ is defined as follows.

$\mathsf{GenCh}(1^\lambda)$: The key generation algorithm chooses a secure elliptic curve and a base point $g$ of prime order $q$ where $q$ is at least $2\lambda$ bits long. It chooses a random integer $\alpha \in \mathbb{Z}_q^*$ and returns $(csk, cpk) = (\alpha, X)$ with $X = g^\alpha$.

$\mathsf{Ch}(x; r)$: The input of the hash algorithm is a public key $cpk = X$ and a message $x \in \mathbb{Z}_q^*$. It picks a random value $r \in \mathbb{Z}_q^*$ and outputs $g^x X^r$.

$\mathsf{Col}(csk, x_0, r_0, x_1)$: The collision finding algorithm returns $r_1 = \alpha^{-1}(x_0 - x_1) + r_0$.

$\mathsf{ExtractCsk}(cpk, x_0, r_0, x_1, x_1)$: If the inputs are a collision, we have $g^{x_0 + \alpha r_0} = g^{x_1 + \alpha r_1}$. The extraction algorithm returns $\alpha = (x_0 - x_1)/(r_1 - r_0)$.

A public key can be validated by verifying that it is an elliptic curve point in the correct prime-order group. To be compatible with Bitcoin keys, we work on the prime-order elliptic curve `secp256k1` [SEC2] at a security level of 128 bits.

**Algorithms and Parameters.** We use `HMAC-SHA256` to instantiate the pseudorandom function $\mathsf{PRF}$, `SHA256` to instantiate the collision-resistant hash function $\mathsf{H}$, and `HMAC-SHA256` with fixed keys to instantiate the hash functions $\mathsf{S}$ and $\mathsf{G}$.

The function L is the identity function, and we have chosen $\ell = 65$ as the height and $n = 2$ as the branching factor of the tree. As a result, the statement space is $\{0,1\}^*$, the context space $\{0,1\}^{64}$, and the assertion algorithm never fails. (Alternatively, we can implement L by a uniform hash function, allowing for the context space of $\{0,1\}^*$ at the cost of a rare failure of the assertion algorithm. The failure probability of the assertion algorithm is below $2^{-37}$ for $q = 10000$ queries.)

**Computation Cost.** On a 2.10 GHz (Intel Core i7-4600U) machine with DDR3-1600 RAM, a chameleon hash evaluation takes 66 µs with a secret key, and the computation time increases to 85 µs if only a public key is available.

Let $\ell$ denote the height of the authentication tree. The assertion algorithm of our accountable assertion scheme in Section 3.4.2 requires $n\ell$ chameleon hash evaluations using a secret key, while the verification algorithm of our accountable assertion scheme requires $\ell$ chameleon hash evaluations using a public key.

In our test environment, the assertion algorithm takes approximately 3.9 ms, while the verification algorithm takes approximately 1.8 ms to complete.

**Storage Costs.** A chameleon hash value is a point on the elliptic curve `secp256k1` and thus requires 257 bits < 33 bytes in compressed form. A randomness input of the chameleon hash function is an integer in the underlying field of the curve, and requires 32 bytes. An assertion is a sequence of $\ell = 64$ chameleon hash values and chameleon hash randomness inputs, and thus requires $64 \cdot (33\,\text{bytes} + 32\,\text{bytes}) = 4160\,\text{bytes}$. Storing $q = 10\,000$ assertions requires 42 MB.

## 3.5 Non-equivocation Contracts

Putting everything together, we explain how to realize non-equivocation contracts by combining accountable assertions and deposits. Non-equivocation contracts make it possible to penalize paltering in distributed protocols monetarily.

**Setup.** Let $A$ be a party to be penalized by the loss of ₿ $p$ if it equivocates before time $T$ and let $d$ be a parameter that depends on $p$ (we will discuss the choice of $d$ in Section 3.5.1).

1. Party $A$ creates a Bitcoin key pair $(pk, sk)$. Also, $A$ sets up the accountable assertion scheme given in Section 3.4.2 with the Bitcoin key pair $(pk, sk)$. That is, $A$ predefines the secret key $ask := sk$ of the accountable assertion scheme and creates the corresponding public key $apk = (pk, z)$ and the auxiliary secret information $auxsk$ as specified in the key generation algorithm.

2. Party $A$ creates a deposit of ₿ $d$ with expiry time $T$ (Section 3.2.1) using $pk$. The deposit may or may not specify an explicit beneficiary $P$, who will receive the funds in case of equivocation.

3. Every recipient $B$ expecting to receive asserted statements from $A$ waits until the transaction that creates the deposit has been confirmed by the Bitcoin network.

**Usage.** The distributed protocol is augmented as follows:

1. Whenever $A$ is supposed to send a statement $st$ to different protocol parties in a context $ct$, party $A$ additionally sends an assertion $\tau := \mathsf{Assert}(ask, auxsk, ct, st)$.

2. Each recipient $B$ verifies that $\mathsf{Verify}(apk, ct, st, \tau) = 1$ and that $T \leq t$ for the current time $t$. Recipient $B$ ignores the message if any of the checks fail.

   Otherwise, $B$ sends the record $(apk, ct, st, \tau)$ to the beneficiary $P$, who will store it. (If there is no explicit beneficiary, $B$ publishes the record to the miners, who have an incentive to store it.)

**Penalty.** Equivocation is penalized as follows:

1. If $P$ (or the miners) detect an equivocation in two records $(apk, ct, st_0, \tau_0)$ and $(apk, ct, st_1, \tau_1)$, they use the corresponding assertions to extract $A$'s secret key $sk := \mathsf{Ext}(apk, ct, st_0, st_1, \tau_0, \tau_1)$.

2. Using $sk$, the beneficiary $P$ transfers the funds in the deposit to an own address. (If there is no explicit beneficiary, the miners wait until the expiry time of the deposit is reached. Then each miner will try to create a block that includes a transaction transferring the deposit to an address owned by the miner.)

Observe that party $A$ will re-obtain full control over the deposit after its expiry time $T$ if it chooses not to equivocate.

### 3.5.1 Security Analysis

We analyze the consequences of an equivocation by $A$.

**With Explicit Beneficiary.** If an explicit beneficiary $P$ is specified in the deposit, then the properties of the deposit ensure that only $P$ can spend the deposit in case of an equivocation. In particular, the safety margins as discussed in Section 3.2.1 ensure that the transaction created by $P$ will have been confirmed already and thus the deposit will have been withdrawn already when its expiry will be reached. The size ₿ $d$ of the deposit should be equal to the penalty ₿ $p$.

**Without Explicit Beneficiary.** If no explicit beneficiary is given, the analysis is more complicated because a malicious sender $A$ can participate in the mining process.

The goal of $A$ is to establish the validity of a transaction $tx$ that withdraws the funds in the deposit to an address controlled by party $A$, even though its secret key has been published. Recall that such a transaction cannot be included in a block before the expiry of the deposit (Section 3.2.1). First, we explain how to choose the safety margin $T_{conf}$ to prevent $A$ from *pre-mining* the transaction $tx$. First observe that, if $T_{conf}$ is too small (say $T_{conf} = 0$ for simplicity), $A$ can pursue the following strategy: Before the expiry time $T$, party $A$ tries to mine a block $\mathcal{B}$ that includes $tx$ and builds upon the most current block $\mathcal{B}_{cur}$. If party $A$ manages to find such a block $\mathcal{B}$, it will keep its block $\mathcal{B}$ secret at first. If additionally no other miner finds another block $\mathcal{B}'$ building upon $\mathcal{B}_{cur}$, the malicious sender $A$ will equivocate just before $T$. Then, by publishing $\mathcal{B}$ after time $T$, $A$ will have a very high chance not to lose its deposit because the transaction $tx$ in $\mathcal{B}$ will most likely prevail. However, if party $A$ does not manage to find a block $\mathcal{B}$, it will refrain from the equivocation attack. This strategy is successful because the malicious sender avoids the risk of losing the deposit by performing the equivocation only if success is almost guaranteed; it is a variant of Finney attack [Fin11].

However, assume that $T_{conf}$ is larger, e.g., $T_{conf} = 60$ min. Then 60 min before the expiry time of the deposit, $A$ will need to have secretly pre-mined several sequential blocks (one of them containing $tx$) on top of the current block $\mathcal{B}_{cur}$ to perform the equivocation. Precisely, $A$ will need to have pre-mined more blocks expected to be found by honest miners within the next 60 min. This is considered infeasible if $A$ controls only the minority of the computation power in the network, which is the one of the underlying assumptions for security of the Bitcoin network.

**Size of Deposit (Without Explicit Beneficiary).** While a safety margin $T_{conf}$ excludes pre-mining attacks, $A$ can try to mine the first block $\mathcal{B}$ after time $T$. Even if other miners find a contradicting block $\mathcal{B}'$ (and maybe more sequential blocks), $A$ can try to catch up with the competing blockchain, which may be worthwhile in the case of a sufficiently large deposit.

We counter such attacks by a careful selection of the deposit size ₿ $d$. Assume that the mining power of the whole network and $A$'s fraction $f$ of it stay constant. If $f < 0.5$, the probability that its block $\mathcal{B}$ prevails is $f/(1-f)$ [Ros14]. Thus the expected penalty $E$ for $A$ is $E = d - d \cdot f/(1-f)$. At minimum, we require $E \geq p$, which yields $d \geq p(f-1)/(2f-1)$. For example, a deposit of $d \geq 3p/2$ is required for a malicious fraction of $f = 0.25$.

### 3.5.2 Application Examples

Many systems require users to trust in a service provider for data integrity. However, the service provider may choose to equivocate and show different users different states of the system. For instance, this has indeed been reported in the case of online social networks. A user of the Chinese microblogging service Sina Weibo claims that Sina Weibo censored his posts by not showing them to other users [Son11]. However, the server showed the posts to the user himself to avoid complaints from him.

To detect misbehavior of the service provider, a variety of systems have been proposed for different scenarios, e.g., SUNDR [MS02] for cloud storage, SPORC [Fel+10] for group collaboration, Application Transparency [Fah+14] for software distribution, and Frientegrity [Fel+12] for social networks.

They basically ensure the following property: If the server violates the *linearity* of the system by showing contradicting states to different users, the server cannot merge these states again without being detected. Furthermore, if users have received contradicting states and exchange them via out-of-band messages, they can detect and prove the wrongdoing of the server. (The property is called *fork consistency* [MS02; CSS07]). Observe that a violation of linearity is a case of equivocation. Although clients can verify the append-only property, i.e., that a new system state is a proper extension of an old known system state, a malicious server can still provide different extensions to different clients.

Non-equivocation contracts are applicable in these settings. The context is often a revision number of the state, and the statement is a digest of the state itself at this revision number. Depending on the system, the context may be more complex than a simple increasing revision number. To avoid sacrificing performance, Frientegrity [Fel+12] for instance does not maintain a total order on all operations in the system but only a total order per object. In this case, the context is a pair consisting of an object identifier and a per-object revision number.

As a concrete application, imagine a non-equivocation contract between a cloud storage provider and a client company, which is willing to pay a slightly higher usage fee as an insurance against accidental or malicious equivocation. The client company is specified as the beneficiary of the deposit. Then the resulting contract serves as cryptographically-enforced insurance. If the service provider equivocates to individual employees of the client company, the company receives the deposit as a penalty without the need to rely on a trusted third party.

In another example scenario, consider a market with two main providers of app stores. Both providers put down a global deposit without explicit beneficiary. If one of the providers becomes malicious and sends different binaries of the same app (and version) to different users, then it will lose its deposit. Thus, after the expiry of the deposit, the malicious provider will have to put down a new second deposit to remain in business and competitive with the honest provider, even if the loss of reputation was small. In comparison, the honest service provider can re-use the funds to put down a second deposit after the first deposit has expired. Alternatively, the malicious provider could choose not to put down a second deposit, but then the honest provider can do the same while getting the funds back.

## 3.6 Asynchronous Payments

As explained in Section 3.2.2, payment channels [Spi13; BIP65] allow a party $A$ to perform many payments to a predefined payee $B$ up to a predefined cumulative amount ₿ $d$. Once the channel is established, it is possible for $A$ to send funds to $B$ even when both parties are offline.

However, if the payee $B$ is a distributed system, i.e., $B$ actually consists of many unsynchronized entities $B_1, \dots, B_n$, then offline transactions are not secure. The problem is that $A$ can double-spend the same funds to $B_i$ and $B_j$, who cannot talk to each other because they are offline and thus not synchronized. When $B$ wants to close its channel and settle the payment in the Bitcoin network, it can settle these funds only once. We can secure offline transaction through payment channels in cases where a reasonable finite penalty for double-spending can be found.

**Example: Public Transport.** For an illustrative example, assume $B$ is a company offering public transport on buses. $A$ would like to use $B$'s services as a passenger. Thus, $A$ establishes a payment channel to $B$ by sending a transaction to the Bitcoin network. Once the transaction is confirmed, the payment channel is open and $A$ can use it to pay for several single rides when entering one of $B$'s buses $B_i$ up to the limit ₿ $d$ of the channel. It is reasonable to assume that $A$ and $B$ have at most sporadic Internet connectivity in this mobile setting, so the payment should be performed offline. Still, $B$'s buses are synchronized every night.

This system is flawed: $A$ can double-spend to the buses of $B$. Assume the current state in the channel is $b = 3$. Then $A$ can ride two (or more) buses $B_i$ and $B_j$ on the same day, by presenting them the same proof that the channel state has been update to $b = 4$. The bus company $B$ will notice only at night during the synchronization that it has been defrauded by $A$.

Using accountable assertions, we can secure this protocol. Then $B$ can penalize the double-spending party $A$ when closing the channel. A reasonable penalty is at least the fare for a day ticket (valid for several rides on the same day).

**Basic Idea.** The idea of the modified protocol is as follows: Since the points of sale $B_i$ are offline and not synchronized, we let party $A$ keep the state of the payment channel. The state consists essentially of just the current value of the channel, and a revision number of the state. To ensure that party $A$ cannot modify the state, it is signed by the individual points of sale $B_i$. However, the user can still show an old signed state and re-use it. This is exactly where we can use accountable assertions: Whenever the party $A$ would like to perform a payment through the channel and claims that the latest state has revision number $k$, we require it to assert the statement "I buy a ticket with serial number $r$" in context $ct = k$, where $st = r$ is a fresh nonce created by $B_i$. Thus, if $A$ reuses an old signed state, the key of $A$ will be extractable.

### 3.6.1 Full Protocol

Our full protocol for asynchronous payment channels consists of three phases. It uses an unforgeable signature scheme with algorithms Sign and VrfySig, and assumes that $B$ and its points of sale $B_i$ have corresponding key pairs $(spk_B, ssk_B)$ and $(ssk_{B_i}, spk_{B_i})$, respectively.

**Setup.** To create an asynchronous payment channel from $A$ to $B$ with amount ฿ $d$, penalty ฿ $p$, and expiry time $T$, the parties execute the following steps:

1. $A$ sets up a Bitcoin key pair $(pk, sk)$ and accountable assertions keys $(apk, ask = sk, auxsk)$ as for non-equivocation contracts (Section 3.5).

2. $A$ creates a payment channel with $B$ with amount ฿ $(d + p)$ and expiry time $T$ (Section 3.2.2).

3. After the channel is confirmed by the Bitcoin network, $B$ provides $A$ with a signed statement $\sigma = \mathsf{Sign}(ssk_B, state)$, where $state = (T, d, k = 0, b = 0, B)$.

**Payment.** Whenever $A$ would like to pay ฿ $x$ offline at some point of sale $B_i$, the parties execute the following protocol. Every point of sale $B_i$ keeps an initially empty blacklist $X$.

1. $B_i$ creates a fresh nonce $r$ and sends it to $A$.

2. $A$ sets $b := b + x$ and $\tau := \mathsf{Assert}(ask, auxsk, k, r)$. $A$ creates a transaction $tx$ updating the channel to state $b$, and sends $(tx, \tau, state, \sigma)$ to $B_i$.

3. $B_i$ receives $(tx^*, \tau^*, state^*, \sigma^*)$, parses $state^*$ as $(T^*, d^*, k^*, b^*, B_j)$, and verifies all the following conditions:
   - $\mathsf{VrfySig}(spk_{B_j}, state^*, \sigma^*) = 1$ (valid state)
   - $\mathsf{Verify}(apk, k^*, r, \tau^*) = 1$ (valid assertion)
   - $tx^*$ is a valid transaction that updates the state of the channel to $b^* + x$
   - $b^* + x \leq d^*$ (unexhausted channel)
   - $A \notin X$ ($A$ is not blacklisted)
   - $t < T^*$ for the current time $t$ (unexpired deposit)

   If any of the checks fail, $B_i$ aborts the payment. Otherwise, $B_i$ computes a new state $state' = (T^*, d^*, k^* + 1, b^* + x, B_i)$, signs it via $\sigma' := \mathsf{Sign}(ssk_{B_i}, state')$, and sends $(state', \sigma')$ to $A$. $B_i$ records $tx$ and $\tau$ and provides service to $A$.

4. $A$ updates $state := state'$ and $\sigma := \sigma'$.

**Synchronization.** At the end of each time period, $B$ synchronizes with each point of sale $B_i$:

1. $B$ collects all transactions recorded by point of sale $B_i$, which can delete the transactions afterwards.

2. $B$ verifies that there are no double-spends among all transactions collected so far. If $B$ detects that $A$ has double-spent, $B$ extracts $A$'s secret key $sk$ and uses it to sign a transaction that spends the whole payment channel worth ₿ $(d + p)$ to an address under the control of $B$. $B$ adds $A$ to the blacklist $X$, and sends updates of the blacklist $X$ to each point of sale $B_i$.

3. Before time $T$, party $B$ closes the channel (Section 3.2.2), adds $A$ to the blacklist $X$, and sends updates of the blacklist $X$ to each point of sale $B_i$.

## 3.6.2 Security Analysis

Observe that party $A$ can double-spend on at most one day because it will be blacklisted afterwards.

Assume $A$ has successfully double-spent. Since the signed state contains the value $b$ of the payment channel, $A$ must have shown the same signed state with some revision number $k$ twice successfully. But then, $A$ has sent two assertions $\tau_0$ and $\tau_1$

that are valid in the same context $ct = k$. Since the corresponding statements $st_0$ and $st_1$ are fresh nonces, they differ with overwhelming probability. Thus $B$ can extract $A$'s secret key successfully, and close the payment channel at the maximum value ฿ $(d + p)$. Since the points of sale $B_i$ accept payments only up to ฿ $b$, the penalty for $A$ in case of double-spending is at least ฿ $p$.

## 3.7 Related Work

In this section, we discuss related work to non-equivocation. Related work for DAPS is discussed in Section 3.8.

**Trusted Hardware.** One way to prevent equivocation is to rely on trusted hardware assumptions [Cle+12; Bac+14; Lev+09; Chu+07]. In particular, the resilience of tasks such as reliable broadcast, Byzantine agreement, and multiparty computation can be improved using a non-equivocation functionality based on trusted increment-only counters in combination with digital signatures. Unlike our approach, which disincentives parties from equivocation, these systems fully prevent it but require a strong assumption about the hardware.

**Smart Contracts.** Cryptocurrencies with more expressive (e.g., Turing-complete) script languages such as Ethereum [But13] offer a simpler way to achieve non-equivocation contracts. In such systems, it is possible to create a deposit that can be opened when presented with cryptographic evidence of equivocation. As digital signatures suffice to provide such evidence and extractability is not required, they can be used instead of accountable assertions. The monetary penalty is enforced by the consensus rules of the currency. In contrast, the distinguishing advantage of non-equivocation contracts based on our construction of accountable assertions is its full compatibility with the current Bitcoin system.

**Traditional E-Cash.** Similar to accountable assertions, Chaumian e-cash systems and one-show anonymous credential systems [CFN88; BL13; CHL05; CL01] allow a secret to be revealed in case of double-spending. In these settings, the revealed secret is not used as a key but as the identity of the double-spender, only i.e., the anonymity is revoked upon double-spending.

However, these protocols are not applicable to our scenario because they work in a fundamentally different setting: they rely on the property that a central authority (a bank), which holds a secret, issues coins by generating cryptographic tokens. In the P2P setting of Bitcoin, no central coin issuer exists, and the cryptographic secrets are generated by the peers individually and independently.

## 3.8 Comparison to DAPS

Similar to accountable assertions, the idea of double-authentication-preventing signatures (DAPS) [PS14; PS17] is prevent the authentication of different statements in the same context by providing an algorithm that extracts the secret key in case of such double-authentication.[5] DAPS are a stronger primitive than accountable assertions, with two main differences. First, DAPS do not allow for "auxiliary secret information" in the strongest security notion, i.e., the full secret key must be extractable in case of double-authentication. Second, DAPS are unforgeable like ordinary signatures. Note that despite realizing a stronger primitive, the DAPS construction by Poettering and Stebila [PS14; PS17] is based on the hardness of factoring and thus not suitable for our concrete application to Bitcoin, which uses discrete logarithm key pairs.

Despite DAPS being a stronger primitve in general, Theorem 3 captures that certain accountable assertions are DAPS.

**Theorem 3.3.** *A secret and extractable accountable assertion scheme that is additionally complete, has a stateless assertion algorithm, and has no auxiliary secret information is a* double-signature extractable *DAPS scheme.*

*Proof.* An accountable assertion scheme with a stateless assertion algorithm and without auxiliary information is syntactically a DAPS scheme. (This allows us to stick to the terminology of accountable assertions in the following, even though we are relating accountable assertions and DAPS).

Since there is no auxiliary information by assumption, it is immediate that extractability of accountable assertions implies *double-signature extractability* [PS14; PS17] of DAPS.

For unforgeability, assume towards contradiction that a ppt attacker $\mathcal{A}(1^\lambda)$ breaks *existential unforgeability under chosen message attacks* [PS14; PS17]. In other words, the attacker outputs a valid assertion $\tau$ on a pair $(ct, st)$ such that *(i)* the pair $(ct, st)$ has not been used as a query for the signing (or assertion) oracle, and *(ii)* the attacker has not queried the assertion oracle to assert two different statements in some context because unforgeability can be broken trivially in this case.

We distinguish two cases: In the first case, the attacker has not queried the oracle to assert any statement in the context $ct$. Then the reduction queries its assertion oracle

[5]The terminology used by Poettering and Stebila [PS14; PS17] is different. While we speak of "*asserting* a statement $st$ in a context $ct$", they speak of "*signing* a message $st$ for a subject $ct$."

to assert some other statement $st' \neq st$ in $ct$. The oracle replies with an assertion $\tau' \neq \bot$ because the accountable assertion scheme is complete. Then the reduction uses the extraction algorithm to extract *ask* from $\tau$ and $\tau'$. This violates the secrecy of the accountable assertion scheme.

In the second case, the attacker has queried the oracle to assert some statement $st^*$ in the context $ct$. (Observe that $st^* \neq st$: otherwise $\tau$ would not be a valid forgery on $(ct, st)$ because the attacker has queried the oracle for $(ct, st) = (ct, st^*)$.) The reduction has relayed the answer $\tau^*$ of the oracle $(ct, st^*)$ query to the attacker. The reduction uses the extraction algorithm to extract *ask* from $\tau$ and $\tau^*$. This violates the secrecy of the accountable assertion scheme. □

**Our Construction Yields Efficient DAPS.** It was left as an open problem to construct DAPS based on Merkle trees or chameleon hash functions [PS14; PS17]. We can solve these problems in the random oracle model. We modify the stateless and complete variant of the construction (Section 3.4.2) as follows. Instead of choosing a key $k$ for the pseudorandom function $F$ at random, we set $k := \mathsf{K}(csk)$ for a hash function $\mathsf{K}$ modeled as random oracle, where *csk* is the trapdoor of the chameleon hash function. This eliminates the auxiliary secret information. However, this modified construction achieves only extractability with trusted setup, i.e., if the key is generated honestly. (We share this limitation with the basic construction proposed by Poettering and Stebila [PS14; PS17].) Indeed, only the extractability of *csk* can be guaranteed. Suppose the attacker can generate the keys. If the attacker just choose $k$ uniformly at random, knowing *csk* does not help to obtain $k$. Consequently, signing messages is not possible with *csk* alone.

Nevertheless, our modified construction is extractable with trusted setup, and it is more efficient than the initial construction by Poettering and Stebila [PS14; PS17]. On a 2.10 GHz (Intel Core i7-4600U) machine with DDR3-1600 RAM, their construction takes about 6700 ms for signing and 1500 ms for verification with asymmetric key size 2048 bits and hash size 160 bits. Our construction with corresponding parameters (in particular $\ell = 160$) takes about 9.9 ms for signing and 4.6 ms for verification.Signatures in their construction need about 40 kB, while signatures in our construction need about 4 kB.

**Other DAPS Constructions.** After the initial publication of our results [RKS15], Bellare, Poettering, and Stebila [BPS17] proposed new DAPS constructions based on RSA and factoring. These constructions beat our construction by an order of magnitude in running time and by two orders of magnitude in signature size. Boneh, Kim, and Nikolaenko [BKN17] propose post-quantum DAPS based on lattice assumptions, and Li et al. [Li+17] propose DAPS based on pairings. Furthermore, Derler, Ramacher, and Slamanig [DRS] and Poettering [Poe18] propose DAPS in the discrete logarithm setting with very efficient running time and signature size, but these constructions have public keys of size polynomial in the size of the context space, and consequently only support only a small number of contexts and are not suitable when the context is a large bitstring such as the output of a hash function (e.g., when the goal is to assert statements in contexts such as Bitcoin transactions identified by their hash). We refer to Poettering [Poe18] for an excellent historic overview of DAPS schemes.

Our construction remains the one with the smallest public keys and the only known scheme in the discrete logarithm setting with an exponentially large context space. If a small context space is acceptable, then the DAPS scheme by Derler, Ramacher, and Slamanig [DRS] can in fact be used as an accountable assertions scheme to implement non-equivocation contracts in Bitcoin more efficiently. (However, the scheme by Poettering [Poe18] cannot be used because it does not support the extraction of a single secret key. Instead, the secret key consists of many secret discrete logarithms among which only one can be extracted, and a trusted setup assumption is necessary to ensure that the other discrete logarithms can be recovered as well.)

In terms of security, all known construction in their basic variant are only extractable with trusted setup. The construction by Poettering and Stebila [PS14; PS17] can be made secure against malicious key generation at the cost of adding rather expensive zero-knowledge proofs to show that the public key is a well-formed Blum integer (a product of two primes $p$, $q$ with $p \equiv q \equiv 3 \mod 4$). In contrast, we are not aware of any (practical) approach to make our construction or the constructions by Bellare, Poettering, and Stebila [BPS17] secure without trusted setup. While zero-knowledge proofs can be applied in principle, it seems necessary to prove relations involving a preimage of a hash function.

# 4 Peer-to-peer Mixing and Unlinkable Bitcoin Transactions

Bitcoin addresses are not directly associated with the real names and identities of their users: the addresses are pseudonyms. As a result, Bitcoin is often perceived to provide a decent level of anonymity. However, this perception of privacy is wrong. Due to the lack of a trusted party verifying transactions, Bitcoin relies on a public transaction ledger to enforce public verifiability of transactions between addresses, and every transaction is effectively carried out in public. By simply observing the transaction ledger on the blockchain, pseudonyms belonging to the same user can be linked using heuristics [Mei+13; Bar+12; SMZ14; KKM14; RH11; And+13; MO15; Nic15], and consequently Bitcoin payments sent or received by a particular user can be identified. Due to this linkability, cryptocurrencies may ultimately provide *less* anonymity than traditional banking.

Coin mixing has emerged as a technique to overcome linkability while maintaining full compatibility with the current Bitcoin protocol. In coin mixing, a set of peers sends their coins to freshly created addresses in order to ensure that these fresh addresses are unlinkable to their previously used addresses. A promising solution in this direction is CoinShuffle [RMK14], a P2P mixing protocol based on a mixnet run by the peers to ensure the unlinkability of inputs and output addresses in a CoinJoin [Has11; Max13A], i.e., a jointly created mixing transaction with multiple inputs (from multiple users) and multiple outputs.

However, a run with a decent anonymity set of $n = 50$ peers takes about three minutes to complete [RMK14] in an Internet-like setting, *assuming that every peer is honest.* In the presence of $f$ disruptive peers trying to impede the protocol, $O(fn)$ communication rounds are required, most of them inevitably taking longer due to the disruptive peers delaying their messages intentionally, which arguably hinders a practical deployment of CoinShuffle.

The large number of rounds in CoinShuffle stems from the fact that it relies on a cascade of mixes in which *each participating peer implements a mix.* All peers need to be actively involved in creating anonymity because they do not trust each other, and following the P2P trust model of Bitcoin, there should be no trusted third party. Therefore the main tool to build a more efficient coin mixing protocol is a more efficient anonymous communication protocol that is compatible with a P2P trust model. We call such a protocol a *P2P mixing protocol*.

In this chapter, we improve the efficiency of P2P mixing protocols and bring them from the realm of feasibility to the realm of practicality. This enables us to provide a practical P2P coin mixing solution for Bitcoin which improves anonymity without requiring changes to the Bitcoin system. We provide four main contributions.

**Conceptualizing P2P Mixing.** As our first contribution, we conceptualize P2P mixing as a particular form of anonymous communication. A P2P mixing protocol enables a set of mutually distrusting peers to publish their messages simultaneously and anonymously without any trusted or untrusted third-party anonymity proxies.

We design an interface and execution model for P2P mixing that allows an easy integration in different application scenarios. Our generic treatment has turned out to be useful beyond the scope of this dissertation, e.g., for online credit networks [MRK17], which suffer from similar privacy problems as cryptocurrencies [MZK16].

**The DiceMix Protocol.** Although some existing anonymous communication systems [CF10; Syt+14; RMK14] satisfy the P2P mixing requirements, their efficiency is not optimal. As our second contribution, we present the new P2P mixing protocol DiceMix, which builds on the Dining Cryptographers network (DC-net) protocol by Chaum [Cha88]. DiceMix handles collisions by redundancy, and disruption by revealing session secrets to expose malicious peers. It requires only $4 + 2f$ rounds in the presence of $f$ malicious peers, i.e., only four rounds if every peer behaves honestly. The resulting communication round complexity is by a linear factor better than in state-of-the-art approaches.

**The CoinShuffle++ Protocol.** As our third contribution, we apply DiceMix to Bitcoin. In particular, building on the CoinJoin paradigm [Has11; Max13A] and DiceMix, we present CoinShuffle++, a practical decentralized mixing protocol for Bitcoin users. CoinShuffle++ not only is considerably simpler and thus easier to implement than its predecessor CoinShuffle [RMK14] but also inherits the efficiency of DiceMix and thus outperforms CoinShuffle significantly.

We provide a proof-of-concept implementation of the DiceMix protocol, (with parameters as necessary for CoinShuffle++) and evaluate it in Emulab [Whi+02]. Our results show that in an Internet-like setting, 50 peers can anonymously broadcast their messages (and create a successful coin mixing transaction in CoinShuffle++) in about eight seconds, instead of the almost three minutes required by the state-of-the-art protocol CoinShuffle.

In order to make CoinShuffle++ accessible and to ease implementation, we provide precise pseudocode for CoinShuffle++ as well as the underlying DiceMix protocol.

**A Generic Attack on P2P Mixing Protocols.** As our fourth contribution, we present a deanonymization attack on existing P2P mixing protocols that guarantee termination in the presence of disruptive peers. We exemplify the attack on the Dissent shuffle protocol [CF10; Syt+14] and then generalize the attack to demonstrate that no P2P mixing protocol simultaneously supports user-chosen fixed input messages, provides anonymity, and terminates in the presence of disruptive peers.

The proposed attack is similar to statistical disclosure attacks across several protocol runs (e.g., [Bor+07; WSF13]) but works with certainty because a protocol which is supposed to terminate successfully can be forced to start a new run to ensure termination. Finally, we discuss how DiceMix resists this attack by requiring fresh and discardable input messages (e.g., cryptographic keys never used before), and we discuss why this is not a problem for applications such as coin mixing.

## 4.1 Background on P2P Mixing

Chaum [Cha81] introduced the concept of anonymous digital communication in the form of mixing networks (or *mixnets*). In the mixnet protocol, a batch of encrypted messages from users is decrypted, randomly permuted, and relayed by a sequence of routers to avoid individual messages getting traced through the network. The original mixnet protocol, as well as its successors such as onion routing [GRS96], AN.ON [AN.ON], and Tor [DMS04], inherently require access to a set of geographically distributed third-party routers such that at least some of them are trusted to not break peers' anonymity.

Starting with the Dining Cryptographers network (DC-net) protocol [Cha88], another line of research on anonymous communication networks emerged, in which peers do not depend on any third-party routers and instead communicate with each other to send their messages anonymously. While the DC-net protocol can guarantee anonymity against an adversary controlling a subset of peers, it is prone to disruption by a single malicious peer who sends invalid protocol messages (active disruption), or simply omits protocol messages entirely (passive disruption). Moreover, a DC-net protects the anonymity of the involved malicious peers, making it impossible for honest peers to detect and exclude the malicious peer.

To address this termination issue, recent successors of the DC-net protocol [BB89; GJ04; CF10; Syt+14; FG14; Fra14] incorporate cryptographic accountability mechanisms against active disruptions. The employed techniques are either proactive, e.g., zero-knowledge proofs proving the validity of sent messages [GJ04], or reactive, e.g, the revelation of session secrets to expose and exclude malicious disruptors after a failed protocol run [CF10]. These protocols have demonstrated that, for a set of mutually distrusting peers, sending their messages anonymously is *feasible* purely by communicating with each other in a P2P manner. Moreover, given the lack of anonymity in P2P cryptocurrencies such as Bitcoin [Mei+13; Bar+12; SMZ14; KKM14; RH11; And+13; MO15], these protocols have led to real-world P2P coin mixing systems [RMK14; NxtSh; BCHSh].

Nevertheless, these solutions are still not ideal: with communication rounds quadratic in the worst case with many malicious peers, these current pure P2P solutions [CF10; Syt+14; RMK14] do not scale as the number of participating peers grows. For instance, the mixnet used in the state-of-the-art Bitcoin P2P mixing protocol CoinShuffle [RMK14] requires a few minutes to anonymize the communication of 50 peers if every peer is honest, and much longer in the presence of malicious peers.

## 4.2 Conceptualizing P2P Mixing

A P2P mixing protocol [CF10; RMK14; Zie+15] allows a group of mutually distrusting peers, each having an input message, to simultaneously broadcast their messages in an anonymous manner *without the help of a third-party anonymity proxy* such as an onion router or a mix server. An attacker controlling the network and some peers should not be able to tell which of the messages belongs to which honest peer. In more detail, the anonymity set of an individual honest peer should be the set of all honest participating peers, and we expect the size of this set to be at least two.

The requirement to achieve sender anonymity without the help of any third-party anonymity proxy makes P2P mixing fundamentally different from most well-known anonymous communication techniques in the literature. Unlike standard techniques such as onion routing or mix cascades, P2P mixing relies on a much weaker trust assumption and is expected to terminate successfully and provide a meaningful anonymity guarantee in the presence of an attacker controlling all but two peers. As a consequence, each peer must be actively involved in the anonymous communication process which comes with inherent restrictions and expense.

### 4.2.1 Setup and Communication Model

We assume that peers are connected via a bulletin board, e.g., a server receiving messages from each peer and broadcasting them to all other peers. We stress that sender anonymity will be required to hold even against a malicious bulletin board; the bulletin board is purely a means of communication.

We assume the bounded synchronous communication setting, where time is divided into fixed communication rounds such that all messages broadcast by a peer in a round are available to the peers by the end of the same round, and absence of a message on the bulletin board indicates that the peer in question failed to send a message during the round.

Such a bulletin board can be seamlessly deployed in practice, and in fact already-deployed Internet Relay Chat (IRC) servers suffice.[1] The bulletin board can alternatively be substituted by an (early stopping) reliable broadcast protocol [ST87; DRS90] if one is willing to accept the increased communication cost.

We assume that all peers participating in a P2P mixing protocol are identified by verification keys of a digital signature scheme, and that the peers know each other's verification keys at the beginning of a protocol execution. To find other peers willing to mix messages, a suitable bootstrapping mechanism is necessary. Note that a malicious bootstrapping mechanism may hinder sender anonymity by preventing honest peers from participating in the protocol and thereby forcing a victim peer to run the P2P mixing protocol with no or only a few honest peers, decreasing the size of the victim's effective anonymity set. This is a realistic threat against any anonymous communication protocol in general, and we consider protection against a malicious bootstrapping mechanism orthogonal to our work.

### 4.2.2 Inputs and Outputs

Our treatment of a P2P mixing protocol is special with respect to inputs and outputs. Regarding inputs (the messages to mix), allowing the adversary to control all but two peers introduces an unexpected requirement, namely, that input messages must be randomized and freshly sampled from a large enough space. Regarding outputs, a P2P mixing protocol according to our definitions provides the feature that the peers explicitly agree on the protocol output, i.e., the set of mixed messages.

**Freshness and Discardability of Input Messages.** In contrast to state-of-the-art anonymous and terminating P2P mixing protocols such as the Dissent shuffling protocol [CF10] and the protocol by Golle and Juels [GJ04], we require that input messages to be mixed are freshly drawn from a distribution with sufficient entropy, e.g., input messages can be random bitstrings or public keys never used before. Furthermore, if the honest peers exclude a peer from the protocol, e.g., because the peer appears offline or is deemed malicious, all input messages used so far will be

[1]Servers supporting IRC version 3.2 are capable of adding a server timestamp to every message [Koc+12]; this can ensure that peers agree whether a certain message arrived in time.

discarded. Then, all remaining peers again generate fresh input messages and are required to continue the protocol and retry mixing with these fresh messages.

While this seems to be a severe restriction of functionality and privacy compared to the aforementioned protocols, a restriction of this kind is in fact necessary to guarantee anonymity. If instead peers can arbitrarily choose their messages in a P2P mixing protocol guaranteeing termination, the protocol is inherently vulnerable to an attack breaking sender anonymity. We will explain this attack in detail in Section 4.7; it works against state-of-the-art P2P mixing protocols and has been overlooked in this form in the literature so far.

**Explicit Confirmation of the Output.** Anonymity-seeking P2P applications such as coin mixing [Has11; Max13A; RMK14; Zie+15] or identity mixing [FWB15] require that the peers agree explicitly on the outcome of the mixing before it comes into effect, e.g., by collectively signing the set $M$ of anonymized messages.

We call this additional functionality *confirmation* and incorporate it in our model. The form of the confirmation depends on the application and is left to be defined by the application which calls the protocol. For example in coin mixing, the confirmation consists of signatures on the CoinJoin transaction provided by all peers; we will discuss this in detail in Section 4.6.

While the protocol cannot force malicious peers to confirm $M$, those malicious peers should be excluded and the protocol should finally terminate successfully with a proper confirmation by all unexcluded peers.

### 4.2.3 Interface and Execution Model

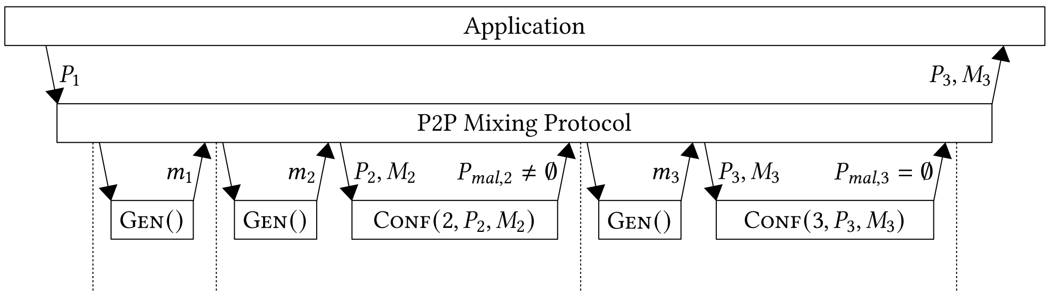
To deploy a P2P mixing protocol in various anonymity-seeking applications, our generic definition leaves it up to the application to specify exactly how fresh input messages are obtained and how the confirmation on the result is performed. We restrict our discussion here to terminology and a syntactic description of the interface between the anonymity-seeking application and an employed P2P mixing protocol, and leave the semantic requirements to the protocol construction later.

A protocol instance consists of one or several *runs*, each started by calling the application-defined algorithm $m := \textsc{Gen}()$ to sample a fresh input message to be mixed. If a run is disrupted, the protocol can exclude peers that turned out to be unreachable or malicious, and the protocol instance fails if no one other peer is left in the protocol instance after exclusion.

Otherwise, if the run is not disrupted, the protocol will obtain a candidate result, i.e., a candidate *output set* $M$ of anonymized messages. Then it calls the application-defined *confirmation subprotocol* $\textsc{Conf}(i, P, M)$, whose task is to obtain confirmation for $M$ from the *final peer set* $P$ of all unexcluded peers. (We require that $m \in M$ for a message $m$ obtained by $\textsc{Gen}()$. The first argument $i$ is an identifier of the run.) Possible confirmations range from signatures on $M$ from all peers to a complex task requiring interaction among the peers, e.g., the creation of a multi-signature in a distributed fashion.

If confirmation can be obtained from every other peer, then the run and the P2P mixing protocol *terminate successfully*. Otherwise, $\textsc{Conf}(i, P, M)$ by convention fails and reports the malicious peers deviating from the confirmation steps back to the P2P mixing protocol. In this case, the protocol can start a new run by obtaining a fresh message via $\textsc{Gen}()$; the malicious peers are excluded in this new run.

An example execution is depicted in Fig. 4.1. Note that while in this example execution all runs are sequential, this is not a requirement. For improved efficiency, a P2P mixing protocol can perform several runs concurrently, e.g., to have an already-started second run in reserve in case the first fails. Then the protocol can terminate with the first run that confirms successfully, and abort all other runs. Nevertheless, we disallow concurrent calls to $\textsc{Conf}()$ to keep the model simple.



The figure shows the calls during the execution; time proceeds from left to right. The execution starts with the application calling the P2P mixing protocol with an initial set $P_1$ of peers. The P2P mixing protocol then starts Run 1 by generating a new message $m_1$ (via calling $\textsc{Gen}()$). Run 1 fails early (e.g., due to active disruption by a peer $p$) and $m_1$ is discarded. The P2P mixing protocol then starts Run 2 with peer set $P_2 = P_1 \setminus \{p\}$ by generating a new message $m_2$. Run 2 is initially not disrupted, and the P2P mixing protocol calls the confirmation subprotocol to confirm the set $M_2$ of mixed messages with the peers in $P_2$. The confirmation subprotocol fails, because a set $P_{mal,2}$ of peers refuse to confirm. The confirmation subprotocol reports those malicious peers back to the P2P mixing protocol, which in turn discards $m_2$. The P2P mixing protocol then starts Run 3 with peer set $P_3 = P_2 \setminus P_{mal,2}$. This time, the confirmation subprotocol succeeds and indicates this by returning an empty set (of malicious peers) to the P2P mixing protocol. That is, all peers in $P_3$ have confirmed that the set $M_3$ of anonymized messages is the final output. The P2P mixing protocol returns $P_3$ and $M_3$ to the application and terminates.

**Figure 4.1:** Example of an Execution of a P2P Mixing Protocol

### 4.2.4 Security Goals and Threat Model

In general, we assume that the attacker controls some number $f$ of $n$ peers; the other peers are honest. In this setting, a P2P mixing protocol must fulfill the two security properties *sender anonymity* and *termination* as follows.

**Sender Anonymity** A P2P mixing protocol provides *sender anonymity* if the following holds, even if the attacker controls the bulletin board: If the protocol succeeds for honest peer $p$ in a run (as described in Section 4.2.3) with message $m_p$ and final peer set $P$, and $p' \in P$ is another honest peer, then the attacker cannot distinguish whether message $m_p$ belongs to $p$ or to $p'$.

**Termination** A P2P mixing protocol provides *termination* if the following holds, under the assumption that the bulletin board is honest: If there are at least two honest peers, the protocol eventually terminates successfully for every honest peer.

**Threat Model.** For the sender anonymity property, we assume that the attacker additionally controls the bulletin board, i.e., the network. In particular, the attacker can partition the network and block messages from honest peers. In the case of successful termination, the anonymity set of each honest peer will be the set of unexcluded honest peers.[2] This means that we need $f < n - 1$ at the end of the protocol, where $n$ is the number of unexcluded peers, to ensure that at least two honest peers are present and the anonymity guarantee is meaningful.

For the termination property, we trust the bulletin board to relay messages reliably and honestly, because termination (or any liveness property) is impossible to achieve against a malicious bulletin board, which can just block all communication.

**No Sender Anonymity in Failed Runs.** Our definition of sender anonymity is only concerned with the messages *in a successful run*, i.e., no anonymity is guaranteed for messages discarded in failed runs (Section 4.2.3). This demands explanation, because giving up anonymity in the case of failed confirmation seems to put privacy at risk at first glance. However, the discarded messages have never been and will never be used outside the P2P mixing protocol; in particular the messages have been not returned back to the application. So it is safe to give up sender anonymity for discarded messages. It turns out that this permissive definition is sufficient for a variety of applications and allows for very efficient constructions.

[2] A honest peer might appear offline due to the attacker blocking network messages. Such a peer can be excluded to allow the remaining peers to proceed.

## 4.3 Solution Overview

Our core tool to design an efficient P2P mixing protocol is a Dining Cryptographers network (DC-net) [Cha88]: Suppose that there are peers $p_1$, $p_2$ and $p_3$, each pair of peers $(i, j)$ shares a symmetric key $k_{i,j}$ and one of the peers (e.g., $p_1$) wishes to anonymously publish a message $m$ such that $|m| = |k_{i,j}|$. The protocol works as follows. $p_1$ publishes $M_1 := m \oplus k_{1,2} \oplus k_{1,3}$, $p_2$ publishes $M_2 := k_{1,2} \oplus k_{2,3}$ and finally $p_3$ publishes $M_3 := k_{1,3} \oplus k_{2,3}$. Now, the peers (and observers) can compute $M_1 \oplus M_2 \oplus M_3$, effectively recovering $m$. However, the origin of the message $m$ is hidden: without knowing the secrets $k_{i,j}$, no observer can determine which peer published $m$. Additionally, the origin is also hidden for peers themselves (e.g., as $p_2$ does not know $k_{1,3}$, $p_2$ cannot distinguish whether $p_1$ or $p_3$ is the origin of the message). It is easy to extend this basic protocol to more peers [Cha88; GJ04].

Besides the need for pairwise symmetric keys, which can be overcome by a key exchange mechanism, there are two challenges in a DC-net: first, it should be possible for all peers to publish their messages simultaneously (and not just for single peers), and second, the protocol should terminate even in the presence of malicious disruptors, while preserving anonymity.

### 4.3.1 Handling Collisions

Each peer $p \in P$ in the mixing seeks to anonymously publish a message $m_p$. Naively, they could run $|P|$ instances (called *slots*) of a DC-net in parallel, where each peer randomly selects one slot to publish the message. However, even if all peers are honest, two peers can choose the same slot with high probability, and their messages are then unrecoverable and a further protocol run becomes necessary [GJ04; Fra14].

One proposed solution is to perform an anonymous reservation mechanism so that peers agree in advance on a slot assignment for publishing [Goe+03; KNS16]. However, this mechanism adds communication rounds among the peers and it must also provide anonymity, which typically makes it prone to the same issues (e.g., slot collisions) that we would like to overcome in the first place. Alternatively, it is possible to establish many more slots so that the probability of a collision decreases [CBM15]. However, this becomes inefficient quickly, and two honest peers could still collide with some probability.

Instead, we follow the paradigm of handling collisions by redundancy [DK13; BB89; CBM15]. Let $n = |P|$ be the number of peers, and assume that messages to be mixed are encoded as elements of a finite field $\mathbb{F}_q$ of prime size $q > n$. Given $n$ slots, each peer $p$, with message $m_p$, publishes $m_p^s$ (i.e., $m_p$ raised to power of $s$) in slot $s$, where $s = 1, \ldots, n$. This yields an intentional collision involving all peers in each of the slots. Using addition in $\mathbb{F}_q$ instead of bitwise XOR to combine DC-net messages, slot $s$ contains the power sum $S_s = \sum_p m_p^s$.

Now, we require a mechanism to extract the messages $m_p$ from the power sums $S_s$. Let $g(x) = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_1 x + c_0$ be a polynomial with roots $m_1, m_2, \ldots, m_n$. Newton's identities [Gou99] state

$$\begin{aligned}
c_n &= 1, \\
c_{n-1} &= S_1, \\
c_{n-2} &= (c_{n-1} S_1 - S_2)/2, \\
c_{n-3} &= (c_{n-2} S_1 - c_{n-1} S_2 + S_3)/3, \\
&\vdots
\end{aligned}$$

By knowledge of all coefficients $c_o$ of the polynomial $g$, we can find its $n$ roots, which are the $n$ input messages.[3]

### 4.3.2 Handling Disruption and Ensuring Termination

Recovering the messages only works when all peers honestly follow the protocol. If a malicious peer disrupts the DC-net by simply sending inconsistent DC-net messages, we must ensure that the protocol still terminates eventually.

When a candidate set $M$ is determined, every honest peer checks whether its input message is indeed in $M$. Depending on the outcome of this check, the peer either starts the confirmation subprotocol to confirm a good $M$, or reveals the secret key used in the key exchange to determine who is responsible for an incorrect $M$. We face two challenges on the way to successful termination.

**Consistent Detection of Disruption.** The first challenge is to ensure that indeed $M$ does not contain *any* honest message if some peer finds that it does not contain its message. Only then will all honest peers agree on whether disruption has occurred and are able to take the same control flow decision at this stage of the protocol, which is crucial for termination.

To overcome this challenge, every peer must provide a non-malleable commitment (using a hash function modeled as a random oracle) to the DC-net vector before it sees

[3] We discuss only this simple power sum encoding in detail, but we note that other encodings are possible. For example, if the DC-net is indeed based on bitwise XOR (instead of addition in $\mathbb{F}_q$), it is possible to use a decoding method [Dod+08] which effectively constitutes one step of the decoding of BCH error-correcting codes [PJ72, Section 6.4]. This method has recently been implemented primarily with a different application in the context of Bitcoin in mind, namely bandwidth-efficient transaction relay in the P2P network [Nau+19; WMN18].

the vectors of other peers. In this manner, malicious peers are forced to create their DC-net vectors independently of the input messages of honest peers. The redundant encoding of messages using powers ensures that malicious peers is not able to create a malformed DC-net vector that results in a distortion of only a subset of the messages of the honest peers. Intuitively, to distort some messages but keep some other message $m$ of a honest peer intact, the malicious peers would need to influence all power sums consistently. This, however, would require a DC-net vector that depends on $m$ (as we show in Section 4.4.5), which is prevented by the non-malleability of the commitments. This ensures that all honest peers agree on whether $M$ is correct, and take the same control flow decision.

**Exposing a Disruptor.** The second challenge is that the misbehaving peer is not trivially detected given the sender anonymity property of DC-nets. To overcome this, every peer is required to reveal the ephemeral secret key used in the initial key exchange. Then every peer can replay the steps done by every other peer and eventually detect and expel the misbehaving peer from further runs.

Note that the revelation of the secret keys clearly breaks sender anonymity for the current run of the protocol. However, the failed run will be discarded and a new run with fresh cryptographic keys and fresh messages will be started without the misbehaving peer. This is in line with our definition of sender anonymity, which does not impose a requirement on failed runs.

An important guarantee provided by DiceMix is that if a protocol run fails, the honest peers agree on the set of malicious peers to be excluded. Although this is critical for termination, this aspect has not been properly formalized or addressed in some previous P2P mixing protocols [CF10; Syt+14; RMK14].

## 4.4 The DiceMix Protocol

In this section we present DiceMix, an efficient P2P mixing protocol, which terminates in only $4 + 2f$ rounds in the presence of $f$ malicious peers.

### 4.4.1 Building Blocks

We rely on the following cryptographic primitives.

**Digital Signatures.** We require a digital signature scheme (KeyGen, Sign, Verify) unforgeable under chosen-message attacks (UF-CMA).

The algorithm KeyGen returns a private signing key $sk$ and the corresponding public verification key $vk$. On input message $m$, $\mathsf{Sign}(sk, m)$ returns $\sigma$, a signature on message $m$ using signing key $sk$. The verification algorithm $\mathsf{Verify}(pk, \sigma, m)$ outputs *true* iff $\sigma$ is a valid signature for $m$ under the verification key $vk$.

**Non-interactive Key Exchange.** We require a non-interactive key exchange (NIKE) mechanism (NIKE.KeyGen, NIKE.SharedKey) which is secure in the model by Cash, Kiltz, and Shoup (CKS model) [Fre+13; CKS09].

The algorithm $\mathsf{NIKE.KeyGen}(id)$ outputs a public key $npk$ and a secret key $nsk$ for a given party identifier $id$. $\mathsf{NIKE.SharedKey}(id_1, id_2, nsk_1, npk_2, sid)$ outputs a shared key for the two parties $id_1$ and $id_2$ and session identifier $sid$. NIKE.SharedKey must fulfill the standard correctness requirement that for all session identifiers $sid$, all parties $id_1$, $id_2$, and all corresponding key pairs $(npk_1, nsk_1)$ and $(npk_2, nsk_2)$, it holds that $\mathsf{NIKE.SharedKey}(id_1, id_2, nsk_1, npk_2, sid) = \mathsf{NIKE.SharedKey}(id_2, id_1, nsk_2, npk_1, sid)$. Additionally, we require an algorithm $\mathsf{NIKE.ValidatePK}(npk)$ which outputs *true* iff $npk$ is a public key in the output space of NIKE.KeyGen, and we require an algorithm $\mathsf{NIKE.ValidateKeyPair}(npk, nsk)$ which outputs *true* iff $nsk$ is a valid secret key for the public key $npk$.

Static Diffie-Hellman key exchange satisfies these requirements [CKS09], assuming a standard key derivation algorithm such as $\mathsf{NIKE.SharedKey}(id_1, id_2, x, g^y) := \mathsf{K}(g^{xy}, \{id_1, id_2\}, sid)$ for a hash function K modeled as a random oracle.

**Hash Functions.** We require hash functions H and G modeled as a random oracles.

## 4.4.2 Contract with the Application

In the following, we specify the contract between DiceMix and the application calling it. We start with two guarantees provided by DiceMix to the application and then we describe features required of the application by DiceMix.

**Guarantees Provided to the Application.** The confirmation subprotocol is provided with two guarantees. First, DiceMix ensures that all honest peers call the confirmation subprotocol in the same communication round with the same parameters; we call this property *agreement*.

Second, to ensure that no peer refuses confirmation for a legitimate reason, e.g., an incorrect set final set $M$ not containing the peer's message, our protocol ensures that all honest peers deliver the same and correct message set $M$. Then, the confirmation subprotocol $\textsc{Conf}(i, P, M)$ can safely assume that peers refusing to confirm are malicious. We call this property *validity*.

The purpose of both of these guarantees is to ensure correct functionality of the confirmation subprotocol, and the guarantees are only provided if the bulletin board is honest. As a consequence, it is up to the confirmation subprotocol to fail safely if they do not hold. The guarantees are detailed below.

**Agreement** Assume that the bulletin board is honest. Let $p$ and $p'$ be two honest peers in a protocol execution. If $p$ calls $\textsc{Conf}(i, P, M)$[4] in some communication round $r$, then $p'$ calls $\textsc{Conf}(i, P, M)$ with the same message set $M$ and final peer set $P$ in the same communication round $r$.

**Validity** Assume an honest bulletin board. If honest peer $p$ calls $\textsc{Conf}(i, P, M)$ with message set $M$ and final peer set $P$, then *(i)* for all honest peers $p'$ and their messages $m_{p'}$, we have $m_{p'} \in M$, and *(ii)* we have $|M| = |P|$.

**Guarantees Provided by the Application.** Next, we specify the guarantees that the application must provide to DiceMix to ensure proper function.

We require that input messages generated by $\textsc{Gen}()$ have sufficient entropy such that they can be predicted only with negligible probability; this implies that $\log q$ is at least proportional to the security parameter. We assume that input messages generated by $\textsc{Gen}()$ are elements of a prime field $\mathbb{F}_q$. We need $q > |P|$ (which is not a restriction in practice due to the previous requirement).

We require two natural properties from the confirmation subprotocol. The first property (*correct confirmation*) states that a successful call to the subprotocol indeed confirms that the honest peers in $P$ agree on $M$. The second property (*correct exclusion*) states that in an unsuccessful call, the confirmation subprotocol identifies at least one malicious peer, and no honest peer is falsely identified as a malicious peer.

**Correct Confirmation** Even if the bulletin board is malicious,[5] we require the following: If a call to $\textsc{Conf}(i, P, M)$ succeeds for peer $p$ (i.e., if the call returns an empty set $P_{mal} = \emptyset$ of malicious peers refusing confirmation), then all honest peers in $P$ have called $\textsc{Conf}(i, P, M)$.

**Correct Exclusion** Assume that the bulletin is honest. If $\textsc{Conf}(i, P, M)$ returns a set $P_{mal} \neq \emptyset$ for honest peer $p$, then $\textsc{Conf}(i, P, M)$ returns the same set $P_{mal}$ for every honest peer $p'$. Furthermore, the returned set $P_{mal}$ does not contain honest peers.

[4] $\textsc{Conf}()$ will actually take more arguments, but they are not relevant for this subsection.

[5] This property puts forth a requirement on a successful call of the confirmation subprotocol. Such a successful call will result in a successful run and ultimately in a successful termination of the whole P2P mixing protocol, which implies that the messages are not discarded and sender anonymity is required for this run. As a result, this property is crucial for sender anonymity and thus we must assume that it holds even if the bulletin board is malicious.

### 4.4.3 Protocol Description

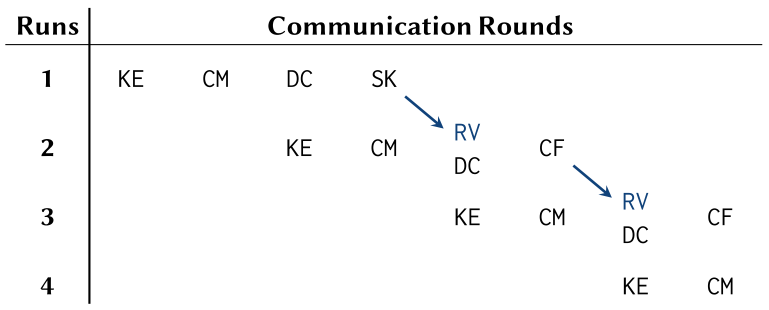
In this section, we describe the DiceMix protocol. The full pseudocode is presented in Section 4.4.4.

**Single Run of the Protocol (Black Pseudocode).** The protocol starts in the procedure DiceMix(), which takes as input a set of other peers $P$, the peer's own identity $my$, an array VK[ ] of verification keys of all peers, the peer's own signing key $sk$, and a predetermined unique session identifier $sid$. A single protocol run, implemented in Run(), consists of four rounds.

In the first round (KE), the NIKE is used to establish pairwise symmetric keys between all peers (DC-Keys()). Then each peer can derive the DC-net pads from these symmetric keys (DC-Slot-Pad()) and use them to create the vector of messages for the DC-net (DC-Mix()). In the second round (CM), each peer commits to its DC-net vector using hash function H; adding randomness is not necessary because we assume that the input messages contained in the DC-net vector have sufficient entropy. In the third round (DC), the peers open their commitments. They are non-malleable and their purpose is to prevent a rushing attacker from letting its DC-net vector depend on messages by honest peers, which will be crucial for the agreement property. After opening the commitments, every peer has enough information to decode the DC-net and extract the list of messages from the power sums (DC-Mix-Res()).

Finally, every peer checks whether its input message is in the result of the DC-net, determining how to proceed in the fourth round. Agreement will ensure that either every peer finds its message or no honest peer finds it. If a peer finds its message, the peer proceeds to the confirmation subprotocol (CF). Otherwise, the peer reveals its secret key. In this case, every other peer publishes its secret key as well, and the peers can replay each other's protocol messages for the current run. This will expose the misbehaving peer, and honest peers will exclude it from the next run (SK).

**Concurrent Runs of the Protocol (Blue Pseudocode).** A simple but inefficient way of having several runs is to start a single run of the protocol and only after misbehavior is detected, start a new run without the misbehaving peer. This approach requires $4 + 4f$ rounds, where $f$ is the number of disruptive peers (assuming that Conf() takes one round). To reduce the number of communication rounds to $4 + 2f$, we use concurrent runs and pipeline them as depicted in Fig. 4.2. We need to address two main challenges. First, when a peer disrupts the DC-net phase of run $i$, it must be possible to patch the already-started run $i + 1$ to discard messages from misbehaving peers in run $i$. For that, run $i$ must reach the last round (SK or CF) before run $i + 1$ reaches the DC round.



Run 1 fails due to DC-net disruption. Run 2 fails to confirm. Run 3 succeeds, and run 4 is then aborted. Rows represent protocol runs and columns represent communication rounds. Blue parts are for concurrency; the arrows depict the dependency between runs, i.e., when a run notifies the next run about the peers to exclude. `KE`: Key exchange; `CM`: Commitment; `DC`: DC-net; `RV`: Reveal pads; `SK`: Reveal secret key; `CF`: Confirmation.

**Figure 4.2:** Example of an Execution of DiceMix

Before its `DC` round, run $i+1$ can be patched as follows. In the `DC` round of run $i+1$, honest peers broadcast not only their DC-net messages but also in parallel they reveal (`RV`) the symmetric keys shared in run $i+1$ with malicious peers detected in run $i$. In this manner, DC-net messages can be partially unpadded, effectively excluding malicious peers from run $i+1$. We note that a peer could reveal wrong symmetric keys in this step. This, however, leads to wrong output from the DC-net, which is then handled by revealing secret keys in round $i+1$. Publishing partial symmetric keys does not compromise sender anonymity for unexcluded peers because messages remain partially padded with symmetric keys shared between the honest peers.

**Handling Offline Peers (Blue Pseudocode).** So far we have only discussed how to ensure termination against actively disruptive peers who send wrong messages. However, a malicious peer can also just send no message at all. This case is easy to handle in our protocol. If a peer $p$ has not provided a (valid) broadcast message to the bulletin board in time, all honest peers will agree on that fact, and exclude the unresponsive peer. In particular, it is easy to see that all criteria specifying whether a message is valid will evaluate the same for all honest peers (if the bulletin board is reliable, which we assume for termination).

To be able to achieve termination $4+2f$ in communication rounds, it is crucial that missing messages in the first two broadcasts (`KE` and `CM`) do not require aborting the run. Luckily, the current run can simply be continued in those cases. Peers not sending `KE` are just ignored in the rest of the run; peers not sending `CM` are handled by revealing symmetric keys exactly as done with concurrent runs (see the code blocks starting with the "**missing**" instruction).

### 4.4.4 Full Pseudocode

In this section, we state the full pseudocode of DiceMix.

**Conventions and Notation for the Pseudocode.** We use arrays written as $\textsc{Arr}[i]$, where $i$ is the index. We denote the full array (all its elements) as $\textsc{Arr}[\,]$.

Message $x$ is broadcast using "**broadcast** $x$". The statement "**receive** $\mathrm{X}[p]$ **from all** $p \in P$ **where** $X(\mathrm{X}[p])$ **missing** $C(P_{\mathit{off}})$" attempts to receive a message from all peers $p \in P$. The first message $\mathrm{X}[p]$ from peer $p$ that fulfills predicate $X(\mathrm{X}[p])$ is accepted and stored as $\mathrm{X}[p]$; all further messages from $p$ are ignored. When a timeout is reached, the statement $C$ is executed, which has access to a set $P_{\mathit{off}} \subseteq P$ of peers that did not send a (valid) message.

Regarding concurrency, a thread $t$ that runs a procedure $\mathrm{P}\,(\mathit{args})$ is started using a statement "$t :=$ **fork** $\mathrm{P}\,(\mathit{args})$", where $t$ is a handle for the thread. A thread with handle $t$ can either be joined using "$r :=$ **join** $t$", where $r$ is its return value, or it can be aborted using "**abort** $t$". A thread can wait for a notification and receive a value from another thread using "**wait**". The notifying thread uses "**notify** $t$ **of** $v$" to notify thread $t$ of some value $v$.

**Full Pseudocode.** DiceMix is specified as follows. The black code is the basic part of the basic protocol for a single run; the blue code handles concurrent runs and offline peers as explained in Section 4.4.3.

```
proc DiceMix ($P, my, \mathrm{VK}[\,], sk, sid$)
    $sid := (sid, P, \mathrm{VK}[\,])$
    if $my \notin P$ then
        fail "not in the set of peers"
    end if
    return Run($P, my, \mathrm{VK}[\,], sk, sid, 0$)
end proc

proc Run ($P, my, \mathrm{VK}[\,], sk, sid, run$)
    $P^* := P \setminus \{my\}$
    if $P^* = \emptyset$ then
        fail "no other honest peers"
    end if
```

```
▹ Exchange pairwise keys
$(\mathrm{NPK}[my], \mathrm{NSK}[my]) := \mathsf{NIKE.KeyGen}(my)$
$sidPre := \mathsf{H}((\mathtt{sidPre}, sid, run))$
broadcast $(\mathtt{KE}, \mathrm{NPK}[my], \mathsf{Sign}(sk, (\mathrm{NPK}[my], sidPre)))$
receive $(\mathtt{KE}, \mathrm{NPK}[p], \sigma[p])$ from all $p \in P^*$
    where $\mathsf{NIKE.ValidatePK}(\mathrm{NPK}[p])$
            $\wedge\ \mathsf{Verify}(\mathrm{VK}[p], \sigma[p], (\mathrm{NPK}[p], sidPre))$
missing $P_{off}$ do
    $P := P \setminus P_{off}$                                ▹ Exclude offline peers
end missing
$sid' := \mathsf{H}((\mathtt{sid'}, sid, P, \mathrm{NPK}[\,], run))$
$\mathrm{K}[\,] := \textsc{DC-Keys}(P^*, \mathrm{NPK}[\,], my, \mathrm{NSK}[my], sid')$
▹ Generate fresh message to mix
$m := \textsc{Gen}()$
$\mathrm{DC}[my][\,] := \textsc{DC-Mix}(P^*, my, \mathrm{K}[\,], m)$
$P_{ex} := \emptyset$                        ▹ Malicious (or offline) peers for later exclusion
▹ Commit to DC-net vector
$\textsc{Com}[my] := \mathsf{H}((\mathtt{CM}, \mathrm{DC}[my][\,]))$
broadcast $(\mathtt{CM}, \textsc{Com}[my], \mathsf{Sign}(sk, (\textsc{Com}[my], sid')))$
receive $(\mathtt{CM}, \textsc{Com}[p], \sigma[p])$ from all $p \in P^*$
    where $\mathsf{Verify}(\mathrm{VK}[p], \sigma[p], (\textsc{Com}[p], sid'))$
missing $P_{off}$ do                           ▹ Store offline peers for exclusion
    $P_{ex} := P_{ex} \cup P_{off}$
end missing
if $run > 0$ then
    ▹ Wait for previous run to notify us of malicious peers
    $P_{exPrev} :=$ wait
    $P_{ex} := P_{ex} \cup P_{exPrev}$
end if
▹ Collect shared keys with excluded peers
for all $p \in P_{ex}$ do
    $\mathrm{K}_{ex}[my][p] := \mathrm{K}[p]$
end for
▹ Start next run (in case this one fails)
$P := P \setminus P_{ex}$
$next :=$ fork $\textsc{Run}(P, my, \mathrm{VK}[\,], sk, sid, run + 1)$
▹ Open commitments and keys with excluded peers
broadcast $(\mathtt{DC}, \mathrm{DC}[my][\,], \mathrm{K}_{ex}[my][\,], \mathsf{Sign}(sk, \mathrm{K}_{ex}[my][\,]))$
```

```
    receive $(\mathtt{DC}, \mathrm{DC}[p][\,], \mathrm{K}_{ex}[p][\,], \sigma[p])$ from all $p \in P^*$
        where $\mathsf{H}((\mathtt{CM}, \mathrm{DC}[p][\,])) = \textsc{Com}[p]$
              $\wedge \{p' : \mathrm{K}_{ex}[p][p'] \neq \bot\} = P_{ex}$
              $\wedge \mathsf{Verify}(\mathrm{VK}[p], \mathrm{K}_{ex}[p][\,], \sigma[p])$
    missing $P_{off}$ do                                   ▷ Abort and rely on next run
        return $\textsc{Result-Of-Next-Run}\,(P_{off}, next)$
    end missing
    $M := \textsc{DC-Mix-Res}(P, \mathrm{DC}[\,][\,], P_{ex}, \mathrm{K}_{ex}[\,][\,])$
    ▷ Check if our output is contained in the result
    if $m \in M$ then
        $P_{mal} := \textsc{Conf}(i, P, M, my, \mathrm{VK}[\,], sk, sid)$
        if $P_{mal} = \emptyset$ then                                        ▷ Success?
            abort $next$
            return $m$
        end if
    else
        broadcast $(\mathtt{SK}, \mathrm{NSK}[my])$                          ▷ Reveal secret key
        receive $(\mathtt{SK}, \mathrm{NSK}[p])$ from all $p \in P^*$
            where $\mathsf{NIKE.ValidateKeyPair}(\mathrm{NPK}[p], \mathrm{NSK}[p])$
        missing $P_{off}$ do                               ▷ Abort and rely on next run
            return $\textsc{Result-Of-Next-Run}\,(P_{off}, next)$
        end missing
        ▷ Blame malicious peers using the secret keys
        $P_{mal} := \textsc{Blm}(P^*, \mathrm{NPK}[\,], my, \mathrm{NSK}[\,], \mathrm{DC}[\,][\,], sid', P_{ex}, \mathrm{K}_{ex}[\,][\,])$
    end if
    return $\textsc{Result-Of-Next-Run}\,(P_{mal}, next)$
end proc

proc $\textsc{DC-Mix}\,(P^*, my, \mathrm{K}[\,], m)$
    ▷ Create power sums in individual slots
    for $s := 1, \ldots, |P^*| + 1$ do[6]
        $\textsc{DCMy}[s] := m^s + \textsc{DC-Slot-Pad}(P^*, my, \mathrm{K}[\,], s)$
    end for
    return $\textsc{DCMy}[\,]$
end proc

proc $\textsc{DC-Mix-Res}\,(P, \textsc{DCMix}[\,][\,], P_{ex}, \mathrm{K}_{ex}[\,][\,])$
    for $s := 1, \ldots, |P|$ do
```

[6]If $run > 0$, it suffices to loop up to $|P^*|$ because at least one peer will have been excluded when the DC-net is opened.

```
        $\mathrm{M}^*[s] := \textsc{DC-Slot-Open}(P, \mathrm{DCMix}[\,][\,], s, P_{ex}, \mathrm{K}_{ex}[\,][\,])$
    end for
    ▷ Solve equation system for array $\mathrm{M}[\,]$ of messages
    $\mathrm{M}[\,] := \mathsf{Solve}(\forall s \in \{1, \dots, |P|\}.\ \mathrm{M}^*[s] = \sum_{i=1}^{|P|} \mathrm{M}[i]^s)$
    return $\mathsf{Set}(\mathrm{M}[\,])$                ▷ Convert $\mathrm{M}[\,]$ to an (unordered) multiset
end proc
proc $\textsc{DC-Slot-Pad}\,(P^*, my, \mathrm{K}[\,], s)$
    return $\sum_{p \in P^*} \mathsf{sgn}(my - p) \cdot \mathrm{G}((\mathrm{K}[p], s))$                                  ▷ in $\mathbb{F}_q$
end proc
proc $\textsc{DC-Slot-Open}\,(P, \mathrm{DC}[\,][\,], s, P_{ex}, \mathrm{K}_{ex}[\,][\,])$
    ▷ Pads cancel out for honest peers
    $m^* := \sum_{p \in P} \mathrm{DC}[p][s]$                                                  ▷ in $\mathbb{F}_q$
    ▷ Remove pads for excluded peers
    $m^* := m^* - \sum_{p \in P} \textsc{DC-Slot-Pad}(P_{ex}, p, \mathrm{K}_{ex}[p][\,], s)$
    return $m^*$
end proc
proc $\textsc{DC-Keys}\,(P^*, \mathrm{NPK}[\,], my, nsk, sid')$
    for all $p \in P^*$ do
        $\mathrm{K}[p] := \mathsf{NIKE.SharedKey}(my, p, nsk, \mathrm{NPK}[p], sid')$
    end for
    return $\mathrm{K}[\,]$
end proc
proc $\textsc{Blm}\,(P^*, \mathrm{NPK}[\,], my, \mathrm{NSK}[\,], \mathrm{DC}[\,][\,], sid', P_{ex}, \mathrm{K}_{ex}[\,][\,])$
    $P_{mal} := \emptyset$
    for all $p \in P^*$ do
        $P' := (P^* \cup \{my\} \cup P_{ex}) \setminus \{p\}$
        $\mathrm{K}'[\,] := \textsc{DC-Keys}(P', \mathrm{NPK}[\,], p, \mathrm{NSK}[p], sid')$
        ▷ Reconstruct purported message $m'$ of $p$
        $m' := \mathrm{DC}[p][1] - \textsc{DC-Slot-Pad}(P', p, \mathrm{K}'[\,], 1)$
        ▷ Replay DC-net messages of $p$
        $\mathrm{DC}'[\,] := \textsc{DC-Mix}(P', p, \mathrm{K}'[\,], m')$
        if $\mathrm{DC}'[\,] \neq \mathrm{DC}[p][\,]$ then                        ▷ Exclude inconsistent $p$
            $P_{mal} := P_{mal} \cup \{p\}$
        end if
        ▷ Verify that $p$ has published correct symmetric keys
        for all $p_{ex} \in P_{ex}$ do
            if $\mathrm{K}_{ex}[p][p_{ex}] \neq \mathrm{K}'[p_{ex}]$ then
```

```
                $P_{mal} := P_{mal} \cup \{p\}$
            **end if**
        **end for**
    **end for**
    **return** $P_{mal}$
**end proc**
**proc** Result-Of-Next-Run ($P_{exNext}$, $next$)
    ▷ Hand over to next run and notify of peers to exclude
    **notify** $next$ **of** $P_{exNext}$
    ▷ Return result of next run
    $result$ := **join** $next$
    **return** $result$
**end proc**
```

### 4.4.5 Security and Correctness Analysis

In this section, we argue why DiceMix achieves all properties required by a secure and correct P2P mixing protocol if the attacker is ppt. First we discuss the two security properties sender anonymity and termination, and then we discuss the guarantees of validity and agreement that the application may rely on.

**Sender Anonymity.** Consider a protocol execution in which an honest peer $p$ succeeds with message $m_p$ and final peer set $P$, and let $p' \in P$ be another honest peer. We have to argue that the attacker cannot distinguish whether the messages $m_p$ belongs to the peer $p$ or to the peer $p'$.

Since both $p$ and $p'$ choose fresh messages $m_p$, $m_{p'}$, and fresh NIKE key pairs in each run, it suffices to consider only the successful run $i$. Since $p$ succeeds in run $i$, the call to $\textsc{Conf}(i, P, M)$ has succeeded. By the "correct confirmation" property of $\textsc{Conf}()$, peer $p'$ has started $\textsc{Conf}(i, P, M)$ in the same communication round as peer $p$. By construction of the protocol, this implies two properties about peer $p'$: *(i)* $p'$ will not reveal its secret key in round SK, and *(ii)* $p'$ assumes that $p$ is not excluded in run $i$, and thus has not revealed the symmetric key shared with $p$ in round RV.

As the key exchange scheme is secure in the CKS model and the exchanged public keys are authenticated using unforgeable signatures, the attacker cannot distinguish the pads derived from the symmetric key between $p$ and $p'$ from random pads. Thus, after opening the commitments on the pads, peer $p$ has formed a proper DC-net with at least peer $p'$. The security guarantee of Chaum's original DC-nets [Cha88] implies that the attacker cannot distinguish $m_p$ from $m_{p'}$ before the call to $\textsc{Conf}(i, P, M)$.

Now, observe that the execution of subprotocol $\textsc{Conf}(i, P, M)$ does not help in distinguishing because all honest peers call it with the same arguments, which follows by the "correct confirmation" property as we have already argued. This shows sender anonymity.

**Validity.** To show validity, we have to show that if honest peer $p$ calls $\textsc{Conf}(i, P, M)$ with message set $M$ and final peer set $P$, then *(i)* for all honest peers $p'$ and their messages $m_{p'}$, we have $m_{p'} \in M$, and *(ii)* we have $|M| = |P|$.

For part *(ii)* observe that in the beginning of an execution and whenever $P$ changes, a new run with $|P|$ peers is started, each of which submits exactly one message. This implies $|M| = |P|$.

For part *(i)* of validity, recall that we assume the bulletin board to be honest for validity, so every peer receives the same broadcast messages. Under this assumption and the assumption that the signature scheme is unforgeable, a code inspection shows that after receiving the `DC` message, the entire state of a protocol run $i$ is the same for every honest peer, except for the signing keys, the own identity $my$, and the message $m$ generated by $\textsc{Gen}()$. Additionally the code inspection shows that among these three items, only the message $m$ influences the further state and control flow, and it does so only in the check of the condition "$m \in M$" at the end of $\textsc{Run}()$ at line 57 in the pseudocode (page 68). Observe that the multiset $M$ in this condition is entirely determined by broadcast messages and thus the same in the execution all honest peers. Thus we may just write $M$ globally for all honest peers in the following.

Observe further that the condition "$m \in M$" determines whether $\textsc{Conf}()$ is called. That is, whenever $\textsc{Conf}(i, P, M)$ is called by some honest peer $p$, then $m_p \in M$. It suffices to show that $m_p \in M$ implies $m_{p'} \in M$ for every honest peer $p'$ (or in other words, the condition "$m \in M$" either evaluates to true for all honest peers, or it evaluates to false for all honest peers equally).

We show this by contradiction. Assume for contraction that there is a ppt attacker $\mathcal{A}$ (which thus performs only a polynomial number of random oracle queries) such that the following holds with non-negligible probability in a protocol execution with attacker $\mathcal{A}$: there is a run $i$ and two honest peers $p$ and $p'$ with their input messages $m_p$ and $m_{p'}$ in run $i$ and $\mathcal{A}$ sends DC-net vectors in round `DC` of run $i$ such that $m_p \in M$ but $m_{p'} \notin M$.

Let $n = |M| = |P|$, and we can write the multiset $M$ as $M = \{m_p, t_1, \ldots, t_{n-1}\} \subseteq \mathbb{F}_q$ for some values $t_i$. Let further $\hat{P} \subseteq P$ be the set of honest peers. The DC-net vectors committed to in round `DC` have $n$ components (or slots). For the slots $s = 1, \ldots, n$, let $a_s$ be sum of all values committed to by the malicious peers in slot $s$ of their DC-net vectors. We write $\mu = m_{p'}$ to simplify the notation.

If $m_p \in M$ and $\mu \notin M$, we know that

$$m_p^s + \sum_{i=1}^{n-1} t_i^s = \sum_{p \in \hat{P}} m_p^s + \sum_{p \in P \setminus P^*} a_s, \qquad k = 1, \dots, n, \tag{4.1a}$$

$$\mu \neq t_j, \qquad j = 1, \dots, n-1, \tag{4.1b}$$

where (4.1a) is equivalent to

$$-\mu^s + \sum_{i=1}^{n-1} t_i^s = \sum_{p \in P^* \setminus \{p, p'\}} m_p^s + \sum_{p \in P \setminus \hat{P}} a_s, \qquad k = 1, \dots, n. \tag{4.2}$$

Interpret (4.1) as an equation system with unknowns $\mu, t_1, \dots, t_n$. We prove that the system determines $\mu$ uniquely. Let $(\mu, t_1, \dots, t_n)$ and $(\tilde{\mu}, \tilde{t}_1, \dots, \tilde{t}_n)$ be two solutions of (4.1). From (4.2), we have

$$-\mu^s + \sum_{i=1}^{n-1} t_i^s = -\tilde{\mu} + \sum_{i=1}^{n-1} \tilde{t}_i^s \qquad s = 1, \dots, n,$$

or equivalently

$$\tilde{\mu}^s + \sum_{i=1}^{n-1} t_i^s = \mu^s + \sum_{i=1}^{n-1} \tilde{t}_i^s, \qquad s = 1, \dots, n. \tag{4.3}$$

Both sides of (4.3) are power sums with $n$ summands. Thus the set $\{\tilde{\mu}, t_1, \dots, t_n\}$ of values on the left-hand side is determined uniquely by the right-hand side through Newton's identities (Section 4.3.1), and symmetrically, the set $\{\mu, \tilde{t}_1, \dots, \tilde{t}_n\}$ of values on the right-hand side is uniquely determined by the left-hand side. This implies

$$\{\tilde{\mu}, t_1, \dots, t_n\} = \{\mu, \tilde{t}_1, \dots, \tilde{t}_n\},$$

and in particular, we have $\mu \in \{\tilde{\mu}, t_1, \dots, t_n\}$. Taking (4.1b) into account, we obtain $\mu = \tilde{\mu}$ as desired. This shows that $\mu$ is determined uniquely $a_s$ by the equation system (4.1). In other words, $\mu = m_{p'}$ is determined uniquely by the messages $m_p$ (where $p \neq p'$) of the honest peers and the DC-net vectors submitted by the malicious peers.

Consider an algorithm $\mathcal{B}$ which plays against a challenger that simulates a single honest peer. The algorithm $\mathcal{B}$ controls all other peers in the execution. It is computationally unbounded but performs only a polynomial number of random oracle queries. In any run of the protocol before the honest peer in this run sends the round DC message, the algorithm $\mathcal{B}$ can halt the execution by outputting a message. We say that $\mathcal{B}$ succeeds if it outputs the input message of the honest peer in this round. Since the input message of the honest peer simulated by the challenger is generated by $\textsc{Gen}()$ and thus unpredictable, and since $\mathcal{B}$ learns only a random-oracle based commitment to this input message at this point of the run, every algorithm $\mathcal{B}$ (which performs only a polynomial number of random oracle queries) succeeds only with negligible probability.

As a contradiction, we now construct an algorithm $\mathcal{B}$ that succeeds with non-negligible probability. The algorithm $\mathcal{B}$ guesses a random run $i \in \{0, \dots, |P| - 2\}$ and a random peer $p' \in \hat{P}$. Then $\mathcal{B}$ internally runs the attacker $\mathcal{A}$ while simulating all honest peers except for peer $p'$. The algorithm $\mathcal{B}$ relays all messages from the attacker $\mathcal{A}$ for peer $p'$ to the challenger and relays all messages from peer $p'$ simulated by challenger back to the attacker. Moreover, $\mathcal{B}$ relays all random oracle queries by the attacker $\mathcal{A}$ to the challenger and relays their results back from the challenger to the attacker $\mathcal{A}$, while storing the queries.

By construction of the protocol messages in the rounds `CM` and `DC`, the attacker $\mathcal{A}$ has queried the random oracle $\mathsf{H}$ with the DC-net vectors of its malicious peers with non-negligible probability already when these malicious peers send their messages in round `CM`. Thus, as soon as $\mathcal{B}$ receives all commitments in round `CM` of run $i$, $\mathcal{B}$ knows all padded DC-net vectors of the peers controlled by $\mathcal{A}$ from the intercepted random oracle queries because the DC-net vectors are the bitstrings committed to.

If the protocol finishes before run $i$, the algorithm $\mathcal{B}$ fails. Otherwise the (computationally unbounded) algorithm $\mathcal{B}$ breaks the key exchanges between all peers (messages in round `KE`) in run $i$, which are necessarily not unconditionally secure [Mau97]. Then it removes all pads from the DC-net vectors of the peers controlled by $\mathcal{A}$ and computes $a_s$ for $s = 1, \dots, n$. The algorithm $\mathcal{B}$ additionally knows $m_p$ for all $p \in \hat{P} \setminus \{p'\}$. If there is no peer $p \in \hat{P} \setminus \{p'\}$ with $m_p \in M$, the algorithm $\mathcal{B}$ fails. Otherwise we are in the situation of the polynomial equation system (4.1). The (computationally unbounded) algorithm $\mathcal{B}$ solves equation system (4.1) and outputs the unique solution $\mu = m_{p'}$.

We analyze the algorithm $\mathcal{B}$. The algorithm is clearly successful if it guesses $i$ and $p'$ correctly, i.e., if it guesses $i$ and $p'$ such that there is some peer $p$ with $m_p \in M$ and $m_{p'} \notin M$ in round $i$, which we have assumed holds for some $i$ and $p'$ with non-negligible probability in a protocol execution with $\mathcal{A}$. Since the number of peers and thus the number of runs is constant (see the discussion about the termination property below), $\mathcal{B}$ is successful with non-negligible probability. Moreover $\mathcal{B}$ performs only a polynomial number of random oracle queries because $\mathcal{A}$ is ppt and thus performs only a polynomial number of random oracle queries.

This is a contradiction which shows the remaining part *(i)* of validity, and we have shown all parts of validity.

**Agreement.** To show agreement, we have to show that for each run $i$, if one honest peer $p$ calls $\textsc{Conf}(i, P, M)$ in some round, then every honest peer $p'$ calls $\textsc{Conf}(i, P, M)$ in the same round. This follows from validity: By part *(i)* of validity, we know that if some honest peer calls $\textsc{Conf}(i, P, M)$, then $m_{p'} \in M$ for every peer $p'$ in run $i$. By construction of the protocol (line 57 in page 68), the condition $m_{p'} \in M$ is exactly what determines whether $p'$ calls $\textsc{Conf}(i, P, M)$. Thus every honest peer $p'$ calls $\textsc{Conf}(i, P, M)$ in the same round, which shows agreement.

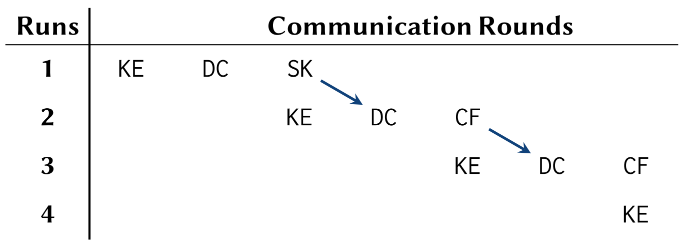
**Termination.** Now, we show why the protocol terminates for every honest peer. We first show that at least one malicious peer is excluded in each failed run. We have already argued above (for validity) that in the presence of an honest bulletin board, all honest peers take the same control flow decision (whether to call $\textsc{Conf}()$ or not at the end of each run). We can thus distinguish cases on this control flow decision.

If in a failed run, $\textsc{Conf}()$ is called, then it returns the same non-empty set of malicious peers (by the "correct exclusion" property), and those peers will be excluded by every honest peer. If $\textsc{Conf}()$ is not called in a run, then there must have been disruption by at least one malicious peer. Replaying all protocol messages of this run (with the help of then-revealed secret keys) clearly identifies at least one malicious peer, and since all honest peers run the same deterministic code (the blame procedure $\textsc{Blm}()$) on the same inputs to identify malicious peers, they will all exclude the same set of malicious peers.

We have shown that in each failed run, all honest peers exclude the same non-empty set of malicious peers. Eventually, we reach one of two cases. In the first case, the number of unexcluded peers will drop below two; in that case the protocol is allowed to fail and thus there is nothing to show. In the second case, we reach a run in which all peers behave honestly (independently of whether they are controlled by the attacker). This run will successfully terminate, which shows termination.

### 4.4.6 Variants of the Protocol

The design of DiceMix follows the P2P paradigm, and consequently, we do not expect the bulletin board to implement any real functionality or perform any computation. The bulletin board is a simple broadcast mechanism and may be replaced by a suitable reliable broadcast protocol [ST87]. However, if one is willing to depend on a more sophisticated bulletin board with dedicated functionality, the efficiency of DiceMix can be improved. It is important to note that even a dedicated bulletin board is still only trusted for termination and not for anonymity.



| Runs | Communication Rounds | | | | | |
|---|---|---|---|---|---|---|
| 1 | KE | DC | SK | | | |
| 2 | | | KE | DC | CF | |
| 3 | | | | | KE | DC | CF |
| 4 | | | | | | KE |

Run 1 fails due to DC-net disruption. Run 2 fails to confirm. Run 3 succeeds and run 4 is then aborted. Rows represent protocol runs and columns represent communication rounds. The blue arrows depict dependencies between runs, i.e., some run informs the next run about the peers to exclude. KE: Key exchange; CM: Commitment; DC: DC-net; SK: Reveal secret key; CF: Confirmation.

**Figure 4.3:** Example of a DiceMix Execution with a Dedicated Bulletin Board

**Dropping the Commitment Phase.** The purpose of the non-malleable commitments is to prevent malicious peers from choosing their DC-net vectors depending on the DC-net vectors of the honest peers. Assume that the bulletin board supports secure channels, and broadcasts the messages in the DC round only after all peers have submitted their messages. Then independence is ensured with an honest bulletin board, and we can drop the CM (commitment) round. This is secure because the independence of the DC-net vectors is necessary for termination but not for anonymity, and we trust the bulletin board for termination already. A serial protocol execution (without concurrency) will then follow the pattern "KE (DC CF/SK)+", where the plus indicates that these phases are performed once or several times. With the help of concurrency, we can run the key exchange (KE) concurrently to the confirmation phase (CF/SK), and reduce the number of rounds to $3 + 2f$ (assuming that the confirmation phase takes one round). An example execution is depicted in Fig. 4.3.

This simplifies the protocol considerably. A revelation of symmetric keys (RV in the original protocol) is not necessary anymore because the malicious peers to exclude are determined before the DC round of the second run (see Section 4.4.3 for an explanation of RV).

**Bulletin Board Performs Expensive Computation.** Moreover, a dedicated bulletin board can perform the expensive computation of solving the equation system involving the power sums, and broadcast the result instead of the DC-net vectors. The bulletin board would then also be responsible for handling inconsistent messages in the SK run; it would then announce the malicious peers after having received all secret keys. This saves communication in the rounds DC and SK. Again, security is preserved because we trust the bulletin board for termination.

## 4.5 Performance Analysis

In this section, we analyze the performance of DiceMix. We first analyze the communication costs, and then evaluate the running time with the help of a prototype implementation. Our results show that DiceMix is practical and outperforms existing solutions for P2P mixing.

### 4.5.1 Communication

Using concurrent runs, DiceMix needs $(c + 3) + (c + 1)f$ communication rounds, where $f$ is the number of peers actually disrupting the protocol execution, and $c$ is the number of rounds of the confirmation subprotocol. In the case $c = 1$, such as in our Bitcoin mixing protocol (Section 4.6), DiceMix needs just $4 + 2f$ rounds.

The communication costs per run and per peer are dominated by the broadcast of the DC-net array $\mathrm{DC}[my][\,]$ of size $n \cdot |m|$ bits, where $n$ is the number of peers and $|m|$ is the length of a mixed message. All three other broadcasts have constant size at any given security level.

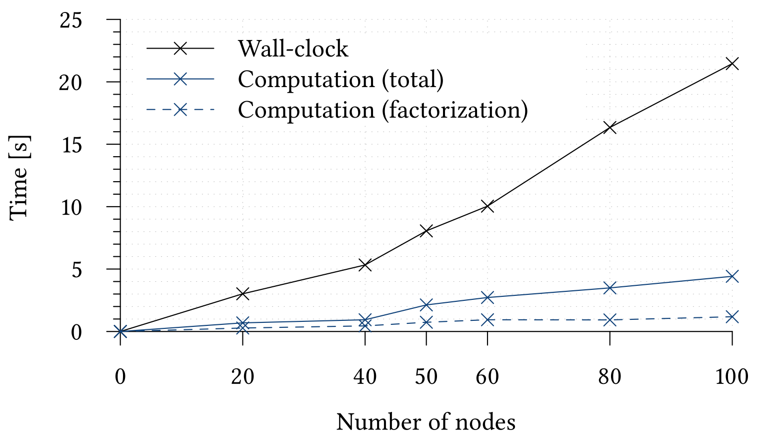
### 4.5.2 Prototype Implementation

We developed a proof-of-concept implementation of the DiceMix protocol. Our unoptimized implementation encompasses the complete functionality to enable testing a successful run of DiceMix without disruptions.

The implementation is written in Python and uses OpenSSL for ECDSA signatures on the `secp256k1` elliptic curve (as used in Bitcoin) at a security level of 128 bits. We use a Python wrapper for the PARI/GP library [PARI-Py; PARI] to find the roots of the power sum polynomial by the Kaltofen-Shoup algorithm for polynomial factorization [KS97].

**Testbed.** We tested our DiceMix implementation in Emulab [Whi+02]. Emulab is a testbed for distributed systems that enables a controlled environment with easily configurable parameters such as network topology or bandwidth of the communication links. We simulated a network setting in which all peers (10 Mbit/s) have pre-established TCP connections to a bulletin board (1 Gbit/s); all links had a delay of 50 ms. We used different Emulab machines (2.2–3.0 GHz) to simulate the peers; note that the slowest machine is the bottleneck due to the synchronization enforced by the broadcasts.

We ran the protocol with a varying number of peers, ranging from 20 to 100. Each peer had as input for the mixing a 160-bit message (e.g., a Bitcoin address).



All peers have a bandwidth of 10 Mbit/s, the bulletin board has a total of 1 Gbit/s, and all links have 50 ms latency.

**Figure 4.4:** Wall-clock Time and Computation Times

**Results.** First, we measured wall-clock time averaged over all peers. As shown in Fig. 4.4, DiceMix runs in about 8 seconds with a moderate size of 50 participants. Second, we measured computation time. We considered the average total computation time spent by a peer and average computation time only for polynomial factorization, i.e., solving the equation system involving the power sums.

**Optimization.** We observe that solving the equation system is quite expensive, e.g., about one second for 100 peers. To demonstrate that this is mostly due to lack of optimization, we developed an optimized stand-alone application for this step in C++ using the FLINT number theory library [HJP15], which provides a highly optimized implementation of the Kaltofen-Shoup algorithm for polynomial factorization over finite fields [KS97]. Our optimized application solves the equation system involving the power sums in about 0.32 seconds for 100 peers on a 2.70 GHz (Intel Core i7-4800MQ) machine, using 6 MB of DDR3-1600 RAM. This shows that optimizations can reduce the running time of the protocol further.

**Discussion.** The experimental results show that even our unoptimized implementation of DiceMix scales to a large number of peers and outperforms state-of-the-art P2P mixing solutions such as CoinShuffle [RMK14] and Dissent [CF10] considerably. In comparison, CoinShuffle (as an tailored variant of the Dissent shuffle protocol) needs slightly less than three minutes to complete a successful run of the P2P mixing protocol in a very similar test environment with 50 peers.

## 4.6 Efficient Coin Mixing for Bitcoin

In this section, we apply DiceMix to Bitcoin, resulting in CoinShuffle++, a highly efficient coin mixing protocol that does not require any changes to the Bitcoin system.

### 4.6.1 Security Goals

Apart from the security goals for a P2P mixing protocol (Section 4.2.4), a coin mixing protocol must guarantee *correct balance*. It ensures that no funds can be stolen from honest peers.

**Correct Balance.** For every honest peer $p$, the total balance of peer $p$ is not reduced by running the coin mixing protocol (ignoring transaction fees).

### 4.6.2 The CoinShuffle++ Protocol

CoinShuffle++ leverages DiceMix to perform a Bitcoin transaction where the input and output addresses for any given honest peer cannot be linked. In particular, CoinShuffle++ creates a fresh pair of signing-verification Bitcoin keys and returns the verification key to implement $\textsc{Gen}()$.

CoinShuffle++ uses CoinJoin [Has11; Max13A; MO15] to perform the actual mixing in the confirmation subprotocol $\textsc{Conf}()$. A CoinJoin transaction allows a set of peers to mix their coins without the help of a third party. In such a jointly created multi-input multi-output transaction, peers set their current coins as input and a mixed list of fresh Bitcoin addresses as output. Crucially, every peer can verify whether the transaction thereby constructed transfers the correct amount of funds to the correct fresh output address. Only if all peers agree and sign the transaction, it becomes valid. So in the case of CoinShuffle++, the explicit confirmation provided by DiceMix is a list of valid signatures, one from each peer, on the CoinJoin transaction.

Note that DiceMix guarantees that everybody receives the correct list of output addresses in the confirmation subprotocol. So a peer refusing to sign the CoinJoin transaction can safely be considered malicious and removed. This is a crucial because otherwise, a single malicious peer can refuse to sign the transaction, while all other peers cannot exclude this malicious peer if not convinced of its guilt.

We define CoinShuffle++ as follows. We denote by $\mathsf{CoinJoinTx}(VK_{in}[\,], VK_{out}, \beta)$ a CoinJoin transaction that transfers $\beta$ bitcoins from every input address in $VK_{in}[\,]$ to the output addresses, where $\beta$ is a pre-arranged parameter; note that if there are $|P|$ unexcluded peers, then the P2P mixing protocol guarantees that there will be

$|M| = |P|$ output addresses. Moreover, we denote by Submit$(tx, \sigma[\,])$ the submission of *tx* including all signatures to the Bitcoin network.

```
proc GEN ( )
    $(vk, sk) :=$ AccountGen()                          ▷ Stores $sk$ in the wallet
    return $vk$
end proc
proc CONF ($i, P, VK_{out}, my, \mathrm{VK}_{in}[\,], sk_{in}, sid$)
    $tx :=$ CoinJoinTx$(VK_{in}[\,], VK_{out}, \beta)$
    $\sigma[my] :=$ Sign$(sk_{in}, tx)$
    broadcast $\sigma[my]$
    receive $\sigma[p]$ from all $p \in P$
        where Verify$(\mathrm{VK}_{in}[p], \sigma[p], tx)$
    missing $P_{off}$ do                     ▷ Peers refusing to sign are malicious
        return $P_{off}$
    end missing
    Submit$(tx, \sigma[\,])$
    return $\emptyset$                                          ▷ Success!
end proc
```

**Security Analysis.** CoinShuffle++ adheres to the requirements specified in Section 4.4.2. Thus, sender anonymity and termination in CoinShuffle++ are immediate. Correct balance is enforced by the CoinJoin paradigm: a peer signs only transactions that will transfer its funds from its input address to its output address.

**Performance Analysis.** In our performance analysis of DiceMix (Section 4.5), GEN() creates a new ECDSA key pair and CONF() obtains ECDSA signatures from all peers (using their initial ECDSA key pairs) on a bitstring of 160 bits. This is almost exactly CoinShuffle++, so the performance analysis of DiceMix carries over.

**Compatibility.** Since CoinJoin transactions work in the current Bitcoin network, CoinShuffle++ is immediately deployable without any change to the system.

### 4.6.3 Practical Considerations

We discuss considerations relevant for a practical deployment of CoinShuffle++.

**Fees.** Bitcoin charges a small fee to prevent transaction flooding attacks. This fee can be split among the peers in the mixing and added to the CoinJoin transaction.

**Change Outputs.** The mixing amount $\beta$ must be the same for all peers, but peers typically do not hold the exact mixing amount in their input Bitcoin address and thus may need an additional *change output*, which is used to receive back extra amount of bitcoins not used in the mixing, this is compatible with CoinJoin. (We refer the reader to [MO15] for a detailed taint-based analysis on the privacy implications of CoinJoin-based coin mixing protocols.)

**Double-Spending Attempts.** After honestly performing the CoinShuffle++ protocol, a peer could try to spend the bitcoins in the input of the CoinJoin transaction in another transaction, before the CoinJoin transaction is confirmed.[7] If this other transaction wins in the consensus process and is confirmed, then the CoinJoin transaction will be invalid. To avoid this disruption, peers should wait in the confirmation subprotocol after completing the CoinJoin transaction, watch the network for any double-spending attempts for a few seconds [DW13; NAH16; BtcStats], and relay them to their peers. If a double-spending attempt is detected, the honest peers can verify the accusation by checking the signature, consider the DiceMix run failed, and exclude the malicious peer. (There will be two valid CoinJoin transactions in the network, one of which will be confirmed. This is not a problem: the honest peers do not care which of them will be confirmed.) In the worst case that the double-spending attempt is not detected, the only consequence is that the mixing fails.

**Extensibility and Multi-Signatures.** The fact that DiceMix is generic in the confirmation subprotocol $\textsc{Conf}()$ makes it possible to define variants of CoinShuffle++ to support a wide range of cryptocurrencies and signature algorithms including multi-signatures [BN06; Max+19] that allow a single combined signature (jointly created in an interactive protocol) for the entire transaction.

For example, the use of Schnorr signatures in Bitcoin has been suggested [Core17]. This modification will enable multi-signatures (with key aggregation), e.g., using the interactive three-round protocol MuSig [Max+19] among the peers in a CoinJoin transaction. Since the partial signatures sent by the peers in the third round of MuSig can be verified individually for each peer, the $\textsc{Conf}()$ subprotocol can still determine which of the peers have sent wrong partial signatures, which is necessary to ensure correct exclusion. Furthermore, the first two rounds of MuSig can be run in parallel to the second (`CM`) and third round (`DC`) of CoinShuffle++, which keeps the number of required communication rounds at $4f + 2$. This is possible because the first two

[7]Since the goal of the attacker is not to use the same money twice, this is not a normal double-spending attack, but we stick to the terminology because the attacker creates two transactions spending the same money.

rounds of MuSig do not depend on the set of signers or the message to be cosigned (the CoinJoin transaction).[8]

Given that signatures are often the largest individual part of the transactions, multi-signatures greatly reduce the size of transactions and consequently the transaction fee, thereby making mixing using CoinJoin cheaper.

**Resistance against DoS Attacks by Sybils.** CoinShuffle++ makes sure that disruptive peers in a mixing will be excluded in due course. To avoid that the same peers disrupt further protocol runs, the bootstrapping mechanism (if executed on the bulletin board) can block the unspent transaction outputs in the blockchain used by the disruptive peers for a predefined period of time, e.g., an hour. (They should not be blocked forever because peers could be unresponsive for legitimate reasons, e.g., unreliable connectivity.)

This ensures that the number of unspent transactions outputs belonging to the attacker limits its ability to disrupt CoinShuffle++ on a particular bulletin board. The attacker can try to overcome the blocking by spending the corresponding funds to create new unspent transaction outputs (*child outputs* of the blocked outputs); however, this is expensive because he needs to pay transactions fees. Moreover, the bootstrapping mechanism can block not only the used transaction outputs but also their child outputs.

**Network-Level Anonymity.** DiceMix does not rely on any external anonymous channel (e.g., the Tor network [DMS04; Tor]) for mixing coins. Nevertheless, to ensure unlinkability of inputs of the CoinJoin transaction with network-level details such as IP addresses, using an external anonymous channel is highly recommended both for running CoinShuffle++ and actually spending the mixed funds later.

### 4.6.4 Related Work

We give an overview of related mixing proposals for Bitcoin and similar cryptocurrencies. Related work for P2P mixing protocols is discussed throughout the chapter, and solutions that aim for more comprehensive privacy but break with the fundamental design of Bitcoin are discussed in Section 5.5

[8]The security of MuSig when used in concurrent sessions relies on the message being fixed before its second round [Nic19], which is ensured by the `CM` round of CoinShuffle++. If the signing sessions needs to be aborted after its second round (due to a disruption in the `DC` phase of DiceMix), no attacker can use the information obtained from the first two rounds to forge signatures on the CoinJoin transaction (which was not used in the signature protocol so far and in fact not even finally determined by DiceMix).

**Naive Tumblers.** A *tumbler* provides a backwards-compatible centralized mixing service [BtcWikiC] to unlink users from their funds: several users transfer their funds to the tumbler, which returns them to the users at fresh addresses. The main advantage of a centralized approach is that it scales well to large anonymity sets because the anonymity set is the set of all users using the service in some predefined time window. However, by using these services naively, a user must fully trust the tumbler: First, anonymity is restricted towards external observers, i.e., the mixing service itself can still determine the owner of the funds. Second and more important, the users have to transfer their funds to the tumbler, which could just steal them by refusing to return them.

**Improved Tumblers.** Mixcoin [Bon+14] mitigates the second problem by holding the tumbler accountable if it steals the funds, but theft is still possible. Blindcoin [VR15] improves upon Mixcoin in that the tumbler additionally cannot break anonymity. CoinSwap [Max13B] is a tumbler protocol that ensures that the tumbler cannot steal the funds, but the tumbler is still trusted for anonymity.

**TumbleBit.** TumbleBit [Hei+17] is a proposal for an untrusted tumbler based on the combination of blind signatures and smart contracts with the aim to solve both aforementioned challenges, i.e., theft and anonymity. To perform ordinary mixing this approach requires at least two transactions to be confirmed sequentially (in two different blocks), whereas CoinShuffle++ requires just one transaction.

TumbleBit supports using the second transaction to send a payment to a payee directly, which is then on par with CoinShuffle++, which also requires one transaction for mixing and one transaction for sending the payment to the real payee. However, this mode of TumbleBit comes with limitations. First, it requires coordination between the tumbler and the payee. Second, it requires more fees than CoinShuffle++, because the CoinJoin transaction used in CoinShuffle++ is cheap, in particular if using aggregate signatures. Third, it requires the payment amount to be exactly the mixing amount, which hinders availability severely because it is very difficult to find enough users that are willing to send the exact same amount of funds at a similar time. With CoinShuffle++, instead, the second transaction, i.e., the actual spending transaction is a normal transaction and supports change addresses, at which peers get their remaining funds back.

On the other hand, TumbleBit offers the following advantages over CoinShuffle++: First, mixing with a tumbler in general scales better to larger anonymity sets, and TumbleBit has been tested with an anonymity set of 800 users. Second, since the design of TumbleBit relies on payment channels (Section 3.2.2), TumbleBit offers very efficient mixing if users are willing to establish long-term payment channels with the tumbler. Third, no interaction between the different senders in the mixing is required.

**CoinJoin with a Server and Blind Signatures.** The CoinJoin proposal [Max13A] describes the idea to use a server to shuffle the list of output addresses. In a first step, every peer sends a transaction input to the server, which returns a blind signature [Cha82] on the input. In a second step, the peer reconnects through an anonymous channel such as Tor [DMS04; Tor] and presents the unblinded input together with a fresh output addresses.

While this approach, which has been refined in ZeroLink [FT17] and implemented in Wasabi Wallet [Was18], is very efficient, it merely moves the trust assumption to the anonymous channel, e.g., to the nodes in the Tor circuits of the peer and to Tor's resistance against traffic analysis. In contrast, CoinShuffle++ does not require an external anonymous channel to achieve unlinkability of inputs and outputs in the CoinJoin transaction. (Nevertheless, to ensure unlinkability of inputs of the CoinJoin transaction with network-level details such as IP addresses, using an external anonymous channel is highly recommended both for running CoinShuffle and actually spending the mixed funds later.)

**CoinParty.** In CoinParty [Zie+15], a set of mixing peers is used to mix funds of users. It is assumed that 1/3 of the mixing parties are honest. This trust assumption is not in line with the philosophy of Bitcoin, which works in a P2P setting without strong identities, where Sybil attacks are easily possible.

CoinShuffle++, instead, does not make any trust assumption on the mixing participants, except that there must be two honest peers, which is a fundamental requirement for any protocol providing anonymity.

**Xim.** Xim [Bis+14] improves on its related previous work [Bar+12] in that it uses a fee-based advertisement mechanism to pair partners for mixing, and provides evidence of the agreement that can be leveraged if a party aborts. Even in the simple case of a mixing between two peers, Xim requires publishing several Bitcoin transactions in the Bitcoin blockchain, which takes on average at least ten minutes for each transaction.

In contrast, CoinShuffle++ requires to submit a single transaction to the Bitcoin blockchain independently on the number of peers.

**Möbius.** Möbius [MM17] is a trustless tumbler in a smart contract, which can be deployed in Ethereum [But13] and similar cryptocurrencies with Turing-complete script languages. Since the entire mixing is performed on-chain, mixing runs need to be stored on the blockchain, including relatively large ring signatures which are necessary for anonymity. Möbius reduces the off-chain latency of the mixing and requires no direct interaction between the peers in the mixing. The goal of CoinShuffle++ goal is the opposite, namely reducing on-chain cost by requiring interaction between the peers involved in the mixing.

**CoinJoin with Different Input Amounts.** Byzantine Cycle Mode [Sun14] and the proposals by Maurer, Neudecker, and Florian [MNF17] and Ficsór [Fic18] are techniques to perform CoinJoin-based mixing even if the peers would like to mix different input amounts at the same time. This is achieved by dividing the set of peers willing to perform mixing in suitable groups [Sun14] or by splitting the transaction outputs suitably [MNF17; Fic18]. These approaches can improve the practicality of CoinJoin considerably but do not tackle the fundamental problem that amounts are publicly visible in the transaction and can be exploited to link inputs and outputs.

In Chapter 5, we will extend CoinShuffle++ to support mixing different input amounts natively. As opposed to the aforementioned proposals, this requires changes to Bitcoin because it relies on homomorphic commitments to hide the monetary amounts in transactions.

## 4.7 A Generic Attack on P2P Mixing Protocols

In this section, we introduce a deanonymization attack on state-of-the-art P2P mixing protocols. We then generalize the attack to demonstrate that no P2P mixing protocol simultaneously supports user-chosen fixed input messages, provides anonymity, and terminates in the presence of disruptive peers.

### 4.7.1 Strong Sender Anonymity

We first observe that user-chosen fixed input messages demand a stronger anonymity requirement: if input messages are chosen by the user, they are not necessarily meaningless at the beginning the protocol execution as opposed to discardable messages that would obtain meaning only later in case of successful mixing, e.g., Bitcoin addresses that receive money only during a CoinJoin transaction that is created in the confirmation subprotocol. Fixed input messages are rather bitstrings that may have meaning already at the beginning of the protocol execution, e.g., leaked documents.

As a consequence, fixed input messages naturally demand a stronger anonymity requirement that is not restricted to successful protocols runs: if the attacker manages to learn that a certain peer $p$ used $m_p$ as input message, then this should arguably already considered a break of the anonymity property, no matter whether the protocol succeeds or not for peer $p$ (whereas we required anonymity only for successful protocol runs if the messages are discardable, see Section 4.2.4).

Formally, a P2P mixing protocol provides *strong sender anonymity* if the following holds, even against an attacker that controls the bulletin board: if some honest peer $p$ has input message $m_p$ in some run that $p$ started with peer set $P$, and $p' \in P$ is another honest peer, then the attacker cannot distinguish whether the message $m_p$ belongs to peer $p$ or to peer $p'$.

### 4.7.2 Example: A Deanonymization Attack on Dissent

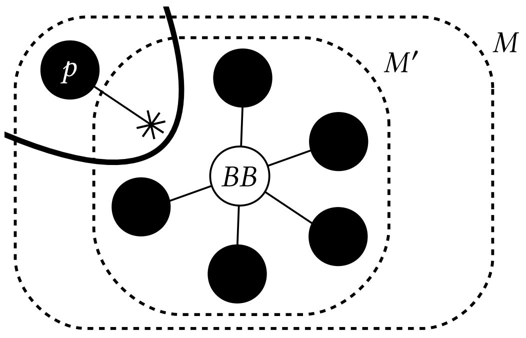
We describe an attack on the Dissent shuffle protocol [CF10; Syt+14].[9] At the core of our attack, which breaks strong sender anonymity, is the problem is the handling of peers that appear to be offline. They cannot be treated like active disruptors: while sacrificing the anonymity of some peer $p$ is not at all a problem if peer $p$ is proven to be an active disruptor and thus malicious, sacrificing the anonymity of $p$ is a serious issue and can renders a protocol insecure if $p$ goes offline. Peer $p$ could in fact be honest because there is no “smoking gun” that allows the other peers to conclude that $p$ is malicious.

Our attack is based on the well-known and very basic observation that an offline peer cannot possibly have sent a message, which comes in many shapes in basically every anonymous communication protocol with reasonable latency [Bor+07; WSF13]. While the attack relies on this very basic observation, it has been overlooked in the literature that a hard requirement to terminate successfully in the presence of offline peers makes existing P2P mixing protocols vulnerable.

In the last communication round of Dissent, every peer publishes a decryption key. All decryption keys taken together enable the peers to decrypt anonymized ciphertexts, resulting in the final set $M$ of anonymized messages. (The rest of the protocol is not relevant for our attack.) The attack on the shuffle protocol now proceeds as follows (Fig. 4.5):

1. The network attacker does not interfere with the protocol until the last communication round. In the last round, the attacker partitions the network into a

[9]There are several protocols named Dissent. First, there is a P2P mixing protocol proposed by Corrigan-Gibbs and Ford [CF10] and formally proven secure by Syta et. al. [Syt+14]. Second, there is protocol [Wol+12] in a client/server setting, which requires trust in one of several servers and is consequently not relevant in our context. The former (P2P) protocol by Corrigan-Gibbs and Ford [CF10] has two variants, a shuffle protocol and a bulk protocol. The shuffle protocol is supposed to provide strong sender anonymity but is restricted to all peers having a message of the same size, whereas the bulk protocol does not share this restriction. When we say Dissent, we always mean the shuffle protocol [CF10, Section 3].



Peer $p$ is partitioned from the bulletin board *BB*. The dashed rectangles indicate the message sets $M$ and $M'$ of the peers in the respective rectangle.

**Figure 4.5:** A P2P Mixing Protocol under Attack

part with only one honest peer $p$ and a part with the remaining peers. Consequently, the last protocol message by peer $p$ (containing its decryption key) does not reach the other peers. As the attacker has learned all decryption keys (including that of $p$), it can decrypt the final set of messages $M$, but nobody else can. However, anonymity is not broken so far.

2. The remaining peers must eventually conclude that peer $p$ is offline and exclude it; otherwise they will not be able to continue the protocol, because they cannot assume that $p$ will ever be reachable again. The strategy by which Dissent provides termination in such a situation is through a wrapper protocol that instructs the remaining peers to attempt a second run of Dissent without peer $p$. In this second run, the remaining peers resubmit their input messages used in the first run [CF10, Section 5.4]. The attacker does not interfere with this second run, and so the run will succeed with a final set $M'$ of mixed messages.

3. Observe that $M' \setminus M = \{m_p\}$, since $p$ is the only peer present in the first run but not in the second. This breaks anonymity of $p$.

The issue on the formal side is an arguably too weak security definition. The core of the Dissent protocol [CF10; Syt+14] does not provide termination on its own but just a form of accountability, which states that at least one active disruptor can be exposed in every failed run of the protocol. The underlying idea is to use the wrapper protocol to ensure termination by starting a new run of Dissent without the exposed disruptor whenever a run has failed.

The formal analysis of the Dissent, however, does not cover the wrapper protocol. It considers only a single run of Dissent, and correctly establishes anonymity and accountability for a single run. It has been overlooked that anonymity is lost under sequential composition of several runs of Dissent using the same input messages, as prescribed in the wrapper protocol.

While Corrigan-Gibbs and Ford [CF10] acknowledge and mention the problem that the last protocol message may be withheld and thus some peer (or the network attacker) may learn the result of the protocol while denying it to others [CF10, Section 5.5], their discussion is restricted to reliability and fails to identify the consequences for anonymity.

### 4.7.3 Generalizing the Attack

The underlying reason for this intersection-like attack is a fairness issue: the attacker, possibly controlling some malicious peers, can learn (parts of) the final message set $M$ of a protocol run while denying $M$ to the other peers. If now some peer $p$ appears to be offline, e.g., because the attacker blocks network messages, the remaining peers must finish the protocol without $p$ with a message set $M'$, which unlike $M$ does not contain $m_p$. Thus the attacker has learned that $m_p$ belongs to $p$.

Since fairness is a general problem in cryptography without an honest majority, it is not surprising that the attack can be generalized, and we show an attack that breaks strong sender anonymity for every P2P mixing protocol that provides termination and supports input messages that are given to the protocol as fixed inputs.

**Fixed Input Messages.** Concretely, we say that a P2P mixing protocol is *executed with fixed input messages* if, in terms of this chapter, every peer calls the algorithm $\textsc{Gen}()$ at most once per execution.

We note that this in this model, input messages are still randomized and have sufficient entropy to be unpredictable, and every peer draws messages from the same distribution. This models that the attacker has no prior knowledge about the assignment of input messages to peers. We could alternatively model fixed input messages as real inputs, e.g., chosen by the attacker, which are given to P2P mixing protocol as arguments. In such a model the input messages would then be assigned to peers randomly. However, our model is simpler and demands a weaker security property, which makes our attack only stronger.

**Attack Description.** Now we are ready to provide a description of a generic attack on any P2P mixing protocol when executed with fixed input messages. Our attack violates strong sender anonymity under the assumption that the P2P mixing protocol guarantees termination when executed with fixed input messages.

We assume an execution of a P2P mixing protocol with peer set $P = \{p_1, \ldots, p_n\}$ and their set of fixed input messages $M = \{m_1, \ldots, m_n\}$ (which are pairwise different with overwhelming probability because they are generated by $\textsc{Gen}()$). We further assume that the attacker controls the network and a majority $A \subset P$ of peers in the execution such that $|P|/2 < |A| \leq |P| - 3$.

Without loss of generality, we assume that no two peers send a protocol message in the same communication round. (This models that the network attacker can determine the order of simultaneous messages arbitrarily.)

For some $i$, let $r$ be the first communication round after which input message $m_i$ of peer $p_i$ is known to a collusion of a minority $S$ of peers with $p_i \notin S$. More formally, this is the first round $r$ for which an efficient extraction algorithm $E$ exists such that $E$ outputs $m_i$ with non-negligible probability, given the full state of all peers in $S$ after round $r$. Such a round exists because every peer outputs $M$ at the end of a successful protocol execution, and $M$ contains $m_i$. Note that knowledge of $m_i$ does not imply that the collusion $S$ of peers collectively knows that $m_i$ belongs to peer $p_i$; it just means that the collusion knows that $m_i$ is *one* of the peers' input messages.

Assume that $S \subset A$, i.e., $S$ is entirely controlled by the attacker. The attacker lets the first $r - 1$ protocol rounds run normally. In round $r$, it collects the protocol message and learns $m_i$ (by control of $S$). Then the attacker selects an index $i^*$ from the set of honest peers. Starting with round $r$, the attacker only delivers protocol messages not from $p_{i^*}$ and not from his own peers in $A$; all these peers appear offline for the remaining peers in $R := P \setminus (\{p_{i^*}\} \cup A)$. By assumption, $|R| \geq 2$, and hence by the termination property, those remaining peers in $R$ will finish the protocol with a public result set $M' \subsetneq M$.

We distinguish cases. If $i^* = i$, then $p_i \notin R$. Since additionally $R$ is a minority, which has not seen any protocol messages from $p_i$ after round $r - 1$, the peers in $R$ do not know $m_i$, and thus $m_i \notin M'$. If instead $i^* \neq i$, then $p_i \in R$, and the correctness of the protocol implies $m_i \in M'$.

In other words, the attacker learns whether $m_i$ belongs to peer $p_{i^*}$ or not by checking whether $m_i \notin M'$. This breaks the anonymity of $p_{i^*}$ and violates strong sender anonymity.

**P2P Mixing Trilemma.** As an intermediate consequence of this attack, no P2P mixing protocol provides strong sender anonymity, termination, and support for fixed input messages simultaneously. DiceMix forgoes fixed input messages to be able to provide (weak) sender anonymity and termination for discardable input messages.

# 5 Mixing Confidential Transactions for Comprehensive Privacy

In Bitcoin's initial design, privacy plays only a minor role. The initial perception of pseudonyms alone providing some built-in anonymity has been broadly refuted [Mei+13; Bar+12; SMZ14; KKM14; RH11; And+13; MO15; Nic15]. This state of affairs has led to a plethora of privacy-enhancing technologies [Mie+13; Ben+14; Sab13; Bon+14; VR15; Hei+17; Zie+15; Bis+14; Bar+12] aiming at overcoming these shortcomings without breaking with the fundamental design of Bitcoin.

However, all of these approaches offer only partial solutions, focusing typically on just one aspect of privacy (payer anonymity or payment amount privacy). For instance, Confidential Transactions (CT) [Max15], a proposed enhancement to the Bitcoin protocol defines a transaction format that ensures payment amount privacy in the blockchain. A second example is Stealth Addresses (SA) [Byt11; Sab13; Tod14B; CM17], which improves payer anonymity because it avoids that the payee needs to communicate with the payer explicitly.

For payer anonymity, the most prevalent approach retaining compatibility with Bitcoin is coin mixing as discussed in Chapter 4. In a coin mixing protocol, a group of peers exchange their coins with each other, effectively hiding the relations between funds and owners. This can be achieved in practice for example by jointly generating a multi-input multi-output CoinJoin [Has11; Max13A] transaction, which enables the peers to atomically transfer their funds from potentially tainted inputs to fresh untainted output addresses. Since such a transaction must be signed by each involved peer to be valid, theft of funds can easily be avoided. Additionally, if peers exchange their output accounts by means of an anonymous broadcast protocol as done in CoinShuffle++ (Chapter 4), inputs cannot be linked to outputs even by malicious peers in the mixing, and such malicious peers cannot prevent the honest peers from successfully completing the protocol.

To achieve comprehensive privacy, it is necessary to combine all the three aforementioned partial privacy solutions (CT, SA, and mixing) into one solution, but this poses a challenge. SA or other means to generate one-time addresses can be easily combined with coin mixing, but while CT has in fact been designed with CoinJoin mixing in mind, it is not clear that the trust models of CT and P2P coin mixing can be made compatible. The design of CT assumes that a transaction is created by just one peer, whereas in P2P coin mixing it is a group of mutually distrusting peers who jointly must create a CoinJoin transaction. This leads to the following question:

> Can we design a P2P coin mixing protocol that enables a group of mutually distrusting peers to create a CoinJoin confidential transaction, without revealing the relation between inputs and outputs or their payment amounts to each other?

## 5.1 ValueShuffle: Mixing Confidential Transactions

In this chapter, we answer this question affirmatively. We design ValueShuffle, the first coin mixing protocol compatible with CT. ValueShuffle is an extension of CoinShuffle++. Since ValueShuffle successfully combines coin mixing, SA and the CT proposal, the resulting currency provides comprehensive privacy, i.e., payer anonymity even against malicious payees and amount privacy. Since it builds upon CoinJoin, ValueShuffle inherits a variety of features crucial to its practical deployment in the Bitcoin ecosystem, e.g., compatibility with Bitcoin scripts and compatibility with blockchain pruning.

**Exploiting Synergies.** By combining coin mixing with SA and CT, we exploit important synergies which make P2P coin mixing both more efficient and more practical, thereby releasing the full potential of coin mixing. We achieve that goal by overcoming the two main limitations of current coin mixing approaches.

First, all forms of coin mixing have been heavily restricted to mixing funds of the same amount because otherwise it is trivial for an observer to link inputs and outputs together just based on their monetary amount, independently of how the mixing is organized. While a peer can mix an input only partly to match the commonly agreed mixing amount, this will create a change output, which is trivially linkable to the input and leads to severe practical issues: the peer must handle the change output very carefully to avoid that it is linked to the actual output in the mixing because this linkage can lead to retroactive deanonymization attacks, and moreover, the change output must be mixed before it can be spent safely, which probably creates another unmixed change output, and so forth. Adding amount privacy to coin mixing removes the necessity to mix funds of the same amount entirely (and consequently all the aforementioned issues with the change output) but comes with the challenge of proving to the network that no money is created in the mixing, since payment amounts are no longer in clear.

Second, P2P coin mixing protocols such as CoinShuffle++ suffer from the problem that peers are required to mix their funds (in a CoinJoin transaction) by sending them to a fresh address of their own first, which removes the trace to the owner. Only afterwards, peers can spend the mixed funds to a payee in a second transaction.[1]

This two-step process renders mixing expensive for users, who pay additional fees and need to wait longer, and for the entire Bitcoin network, which has to process essentially twice the amount of transaction data. As a result, privacy comes at a large expense. This is undesirable and creates a conflict between privacy and efficiency.

In ValueShuffle,we rely instead on SA and CT to enable users to send their funds directly to the expected payees in the CoinJoin transaction, which is arguably the most desirable mode of use of CoinJoin.

### 5.1.1 Features

ValueShuffle achieves the following main features as a combination of the three privacy-enhancing technologies CT, SA, and coin mixing.

**Privacy** ValueShuffle provides decent privacy guarantees in Bitcoin: It ensures that no attacker observing the blockchain or the network, or even participating in the protocol, can link inputs and outputs of the CoinJoin transaction created in an execution of ValueShuffle (input-output unlinkability). That ensures that given an output of this transaction, the payer's input address cannot be identified among the honest input addresses in the mixing (payer anonymity). Payer anonymity holds even against malicious payees (which for example is necessary for anonymous donations) because the use of SA ensures that a payer can derive an one-time payment address of a payee non-interactively and thereby avoids that the payee learns the identity of the payer when setting up the payment. Additionally, CT provides amount privacy.

**Single Transaction** ValueShuffle can be used to transfer funds to payees directly without any form of premixing as required by P2P coin mixing solutions such as CoinShuffle++, and without requiring interaction with the payee. As a result, private payments can be performed with just one single transaction on the blockchain.

**DoS Resistance** ValueShuffle succeeds in the presence of denial-of-service attacks by disruptive peers aiming to prevent honest peers from completing the mixing.

[1]This is due to a fundamental restriction of P2P mixing protocols; they can only handle freshly generated messages which can be discarded if the protocol is disrupted, e.g., Bitcoin addresses generated in the beginning of the protocol (Section 4.7). As a result, paying to a payee directly is not possible because that would require using a fixed amount or a fixed address as a message.

While disruptive peers can delay the protocol, they cannot stop it. Since ValueShuffle is based on CoinShuffle++, it terminates in only $4+2f$ communication rounds in the presence of $f$ disruptive peers.

**Anonymous Channel Not Strictly Required** Like CoinShuffle++, ValueShuffle does not rely on any external anonymous channel such as Tor [DMS04; Tor] for providing unlinkability of inputs and outputs in a CoinJoin transaction. (However, to avoid an observer being able to link inputs of the CoinJoin transaction with network-level identifiers such as IP addresses, using an external means of anonymous communication is highly recommended.)

Since ValueShuffle is based on the CoinJoin paradigm, it additionally inherits all of its practical advantages:

**Theft Resistance** Since honest peers will check the final CoinJoin transaction before signing it, no money can be stolen from them.

**Script Compatibility** While ValueShuffle does not keep the scripts confidential, it is compatible with transaction outputs that use complex scripts, e.g., advanced smart contracts, and provides meaningful privacy guarantees for them.

**No Overhead for the Network** Unlike ring signatures, as for example deployed in Monero [XMR], which require a signature of size proportional to the anonymity set, our approach—while requiring interaction between peers—provides anonymity without putting an additional burden in terms of blockchain space or verification time on the Bitcoin network.

**Reduced Fees and Space Requirements** Taking this one step further, CoinJoin makes Bitcoin in fact more efficient, assuming the availability of Schnorr signatures [Sch91], which have been proposed to be deployed in Bitcoin in the future [Core17]. The introduction of Schnorr signatures will enable multi-signatures using an interactive three-round protocol among the peers in a CoinJoin [Max+19], reducing the number of signatures from $n$ to 1, where $n$ is the number of peers. This protocol can easily be integrated in CoinShuffle++ (Section 4.6.3) and consequently also in ValueShuffle while keeping the number of rounds at $4 + 2f$.

Moreover, a promising future direction is to combine ValueShuffle with an interactive protocol to produce a single very short multi range proof for all newly created commitments in the outputs of the CoinJoin transaction, reducing the size of the transaction significantly [Bün+18]. We conjecture that this combination is possible but leave its details for future work.

The aforementioned enhancements greatly reduce the size of transactions, thereby providing large savings in terms of blockchain space, verification time, and transaction fees as compared to $n$ individual confidential transactions.

**Incentive for Privacy** Due to the reduced fees, users save money by performing privacy-preserving transactions. This provides an unprecedented incentive for deployment and use of privacy-enhancing technologies in Bitcoin.

**Compatibility with Pruning** Unlike in Zerocash [Ben+14] or Monero [XMR], CoinJoin makes it possible to observe which transaction outputs are unspent. While this releases some information to the public, it allows pruning spent outputs from the set of (potentially) unspent transaction outputs. Pruning helps to mitigate the scaling issues of Bitcoin.

**Overlay Design** The unlinkability provided by ValueShuffle through the use of CoinJoin is built as a separate layer on top of Bitcoin, which avoids additional complexity and risk in the underlying Bitcoin protocol.

## 5.2 Building Blocks

Besides efficient P2P mixing as discussed in Chapter 4, ValueShuffle needs Confidential Transactions and Stealth Addresses as building blocks.

### 5.2.1 Confidential Transactions

Confidential Transactions (CT) [Max15; Gib16] is a cryptographic extension to Bitcoin that allows a single peer to perform a transaction such that none of the monetary amounts in the inputs or outputs are revealed, thereby guaranteeing *amount privacy*. Nevertheless, the balance property, i.e., no new coins are generated in the transaction, remains publicly verifiable.

This is mainly achieved by hiding the amounts using additively homomorphic commitments, i.e., $\mathsf{Com}(x, r) \oplus \mathsf{Com}(x', r') = \mathsf{Com}(x + x', r + r')$. As an example, assume a peer has an input amount $x_1$ and two output amounts $x_2$ and $x_3$. The peer can commit to $x_1, x_2$, and $x_3$, as $c_i := \mathsf{Com}(x_i, r_i)$, where $r_i$ is chosen uniformly at random. Then, the peer computes $r_\Delta = r_1 + r_2 - r_3$ and adds this value in clear to the transaction. Ignoring fees, a verifier can then verify the balance property by checking whether $c_1 \oplus c_2 = c_3 \oplus \mathsf{Com}(0, -r_\Delta)$.

In fact, the CT proposal avoids adding $r_\Delta$ explicitly by choosing the randomness values such that always $r_\Delta = 0$, saving a few bytes in the transaction. ValueShuffle is not compatible with this optimization because it is not clear how to enable the optimization without adding communication rounds to the protocol.

If the commitments on the output side of the transaction could contain negative amounts, then a malicious peer could create money out of thin air. For example, a malicious peer could first create a (seemingly balanced) transaction with an input with amount 1, and two outputs with amounts −1000 and +1001 (and then just never use the use the output with −1000). To ensure that commitments do not contain negative or too-large amounts that could overflow, a non-interactive zero-knowledge *range proof* is added to every newly created commitment, proving that the amount is positive and in a certain range. Also a few other components, e.g., an ephemeral public key, are added to each commitment. To simplify presentation, we assume throughout the chapter that these other components are part of the range proof.

### 5.2.2 One-time Addresses

Peers performing transactions via ValueShuffle require a sufficient supply of fresh unlinkable addresses of the payee. This will make it possible to discard a recipient address used in a failed run of DiceMix. In this case, a fresh address can be used for the following run, satisfying the freshness requirement of messages mixed using DiceMix. (If there are $n$ peers in the mixing, DiceMix will require at most $n - 1$ addresses.) Several methods are possible. First, the payee can post a stealth address [Byt11; Sab13; Tod14B; CM17], which enables any payer to derive fresh addresses. Second, the payee can send a BIP32 public key [BIP32] to the payer, which enables the payer to derive fresh addresses. The necessary derivation index can be derived from public information, e.g., a hash of the amount commitment. Third, the payee can simply send enough fresh addresses to the payer.

The method based on stealth addresses provides the strongest privacy guarantees. A stealth address is a public long-term address of a payee, which enables a payer to derive an arbitrary number of unlinkable addresses owned by the payee. A payment using a stealth address does not require any direct communication between payer and payee, and thus provides strong payer anonymity when used together with coin mixing: not even the payee can identify the payer, which is a useful property for anonymous donations for example.

Nevertheless, ValueShuffle is oblivious of the method to generate fresh addresses; we only require that the payee has access to some method, and we refer the reader to the respective descriptions of the individual methods for details.

## 5.3 Solution Overview

In this section, we give an overview on ValueShuffle, the first P2P coin mixing protocol compatible with CT. We detail the protocol and the security analysis in Section 5.4.

Since ValueShuffle is an extension of CoinShuffle++, which uses DiceMix to mix Bitcoin addresses of the peers, we rely on the same communication model and bootstrapping assumptions as CoinShuffle++ and DiceMix (Section 4.2). In particular, the peers are connected through a bulletin board, which is not trusted (except for termination, which is a liveness property).

### 5.3.1 Security and Privacy Goals

ValueShuffle provides the following security and privacy guarantees, where the first three are as in CoinShuffle++.

**Unlinkability** Given an output and two inputs belonging to honest peers in the CoinJoin transaction created by the protocol, the attacker is not able to tell which of the two inputs pays to the output.

**Correct Balance** For every honest peer $p$, the total balance of peer $p$ is not reduced by running the coin mixing protocol (ignoring transaction fees).

**Termination** If the bulletin board is honest and there are at least two honest peers, the protocol eventually terminates successfully for every honest peer.

**CT Compatibility** The protocol generates a CoinJoin transaction without compromising the amount privacy of honest peers as provided by CT.

### 5.3.2 Challenges and Overview on the Solutions

To combine coin mixing with CT and one-time addresses, we need to overcome the following challenges. For the sake of explanation, we assume that each peer has only one input and one output in the transaction, and that there is no transaction fee. The full protocol does not have these limitations.

**Basic Design.** From a high-level point of view, the peers in an execution of ValueShuffle run DiceMix to mix not only their output addresses (as done in CoinShuffle++) but their *output triples*, i.e., triples consisting of output address (or script), CT amount commitment, and corresponding range proof. If DiceMix runs successfully, then it will pass a set of anonymized triples to an application-defined *confirmation* mechanism, which confirms the result of the mixing. As in CoinShuffle++, the confirmation mechanism in ValueShuffle is the collective signing of the CoinJoin transaction, either by collecting a plain list of signatures or by performing an interactive protocol to create an multi-signature [Max+19].

**Handling Disruption.** If a run of DiceMix fails, it must be possible to identify at least one disruptive peer to be excluded in a subsequent run of the protocol. This will eventually guarantee termination. Crucially, DiceMix requires the confirmation mechanism to output at least one such peer if confirmation itself is disrupted. The confirmation mechanism can assume that the result of the mixing is correct, i.e., it contains the messages of all honest peers. Given that assumption, identifying a disruptive peer is straightforward: a peer that refuses to sign the final CoinJoin transaction, or provides wrong signatures (or wrong partial signatures in the case of multi-signatures) is obviously disruptive.

**Freshness of Mixed Output Triples.** Recall that DiceMix requires mixed messages (i.e., the output triples in our case) to be fresh and have sufficient entropy to ensure anonymity (Section 4.4.2). This is exactly where we are able to exploit one-time addresses and CT. In particular, the payer is able to create fresh unlinkable output triples: We assume that the payer has a method to create fresh unlinkable output addresses all belonging to same payee, so the address component of the output triple is fresh. Moreover, the payer uses CT and since the commitment scheme and the range proof are randomized, the payer is able to generate many fresh unlinkable amount commitments and range proofs. So all three components of the output triple can be freshly generated.

Only by combining one-time addresses and CT, we are able to guarantee anonymity if peers are mixing and performing actual payments in the same transaction. Other P2P coin mixing protocols require peers to mix funds to a fresh output address of their own because using the fixed address of the payee or even using the plain monetary amount in the mixing is not possible if anonymity and termination are desired (Section 4.7).

**Multi-Payer Confidential Transactions.** In the original design of CT, the single payer can easily craft the randomness for the commitments to input and output amounts in the transaction such that anyone can verify its correctness (Section 5.2.1). However, in a mixing transaction with several payers, a naive construction of such a verifiable transaction would require that peers reveal to each other the randomnesses used in the commitments, thereby forgoing the hiding property of the commitments and effectively revealing the amounts.

To overcome this issue, the peers can run a secure sum protocol to jointly compute the sum $r_\Delta$ of their random values, i.e., $r_\Delta = \sum_p r_p - r'_p$, where $r_p$ denotes the randomness in the commitment to the input amount of peer $p$, and $r'_p$ denotes the randomness in the commitment to the output amount of the same peer $p$. As a sum, $r_\Delta$ does not reveal which peer contributed which summand to $r_\Delta$. Now all peers can add $r_\Delta$ as an explicit public randomness value to the transaction, and the overall transaction is valid again, which can be publicly verified by checking whether $\bigoplus c_p = \bigoplus c'_p \oplus \mathsf{Com}(0, -r_\Delta)$.

The value $r_\Delta$ can be obtained with a standard secure sum protocol based on additive secret sharing, where every peer $p$ broadcasts its value $r_p - r'_p$ blinded by multiple pads, each one shared with one other peer. The messages from all peers are then combined so that shared keys cancel out and the sum $r_\Delta$ is obtained. This mechanism is in essence equivalent to a DC-net as already used in DiceMix.

**Handling Disruption of the Secure Sum Protocol.** Malicious peers can disrupt not only the mixing of output triples but also the secure sum protocol by creating an output amount commitment that does not match the amount of the input commitment, which can be detected when creating the CoinJoin transaction.

Similar to sacrificing anonymity in the DC-net used for mixing output triples, we can sacrifice anonymity in the DC-net that we use as a secure sum protocol. This reveals for every peer $p$ the value $r_p - r'_p$. Using the verification equation of CT, all honest peers can easily check the balance property of every peer $p$ individually, i.e., check whether peer $p$'s output commitments are consistent with its input commitments. Note that this approach does not reveal the random values used for the input commitments or the output commitments of peer $p$, which would also reveal its intended payment amount.

**Combining P2P Mixing and Secure Sum.** Since both DiceMix and the secure sum protocol are similar in structure (they both rely on DC-nets after all), we can optimize their combination. First, we can rely on a single key exchange and derive independent subkeys for the P2P mixing and the secure sum protocol. This means that if one of the two protocols is aborted, then the other must be aborted as well because the same ephemeral secret is used for the key exchange and must be revealed. However, this is not a problem because the proper result of one of the two protocols does not yield a valid mixing transaction, and the peers have to restart from scratch by generating a fresh output triple anyway.

**Mixing Long Messages in DiceMix.** While DiceMix in its current form is practical for small messages $m$ (e.g., $|m| = 160$ bits as used by CoinShuffle++), it is prohibitively slow for messages of the size we require; we need $|m| \approx 20\,000$ bits to mix the quite large range proofs necessary in CT. The most expensive computation step is a polynomial factorization and requires each message to be an element of a finite field and consequently the finite field must have a size of about $2^{|m|}$.

To overcome this issue, we split $m$ into several chunks, i.e., $m = m_1 \| \cdots \| m_\ell$ and mix those chunks in different parallel runs of the essential mixing step in DiceMix. The challenge that arises is to recombine the messages again because the mixing ensures that it is not possible to know which chunks belong together (i.e., to the same peer). Our solution is to prefix every $m_i$ for $1 < i \leq \ell$ with $\mathsf{F}(m_1)$, where $\mathsf{F}$ is a collision-resistant hash function, so that every peer mixes: $m'_1 = m_1$ and $m'_i = \mathsf{F}(m_1) \| m_i$ for $1 < i \leq \ell$. This arrangement allows for a trade-off between computation and communication required for mixing: bigger chunks reduce the number of required parallel mixing instances but demand higher computation costs for the polynomial factorization.

**Supporting Arbitrary Scripts.** So far we have discussed only output addresses, which are essentially hashes, but not the type of these addresses. While mixing works fine with ordinary pay-to-pubkey-hash (P2PKH) addresses, we require pay-to-script-hash (P2SH) hashes [BIP16][2] to support arbitrary scripts. However, it is not possible to mix P2PKH and P2SH hashes in the same mixing, because this would require adding the address type explicitly to the mixing message, which breaks anonymity: in case of a disruption, it becomes clear which inputs go a P2PKH address and which inputs go to a P2SH address.

To support P2PKH and P2SH together, we can instead perform P2PKH transactions nested in P2SH;[3] then all peers use P2SH addresses in the mixing. A more advanced alternative is rely on Taproot [Max18], a proposal to hide a script $S$ in an ordinary public key $g^x$ by tweaking the public to $g^{x+\mathsf{H}(g^x,S)}$ for a hash function $\mathsf{H}$ [GH12]; then all peers use ordinary public keys in the mixing.

For the sake of simplicity, we will ignore the issue of different address types and assume that the peers use some fixed address type in the remainder of the chapter.

[2]In P2PKH, funds are sent to a public key specified by its hash, and the peer who wants to spend the resulting output is responsible for showing the public key. P2SH is a generalization: in P2SH, funds are sent to a script specified by its hash, and the peer who wants to spend the resulting output is responsible for providing the script.

[3]Such nesting has also been proposed in the context of Segregated Witness [BIP141].

### 5.3.3 Overview of ValueShuffle

We assume that every peer $p$ is represented by a triple $in_p = (c_p = \mathrm{Com}(x_p, r_p), \pi_p, vk_p)$, where $c_p$ denotes the commitment to the input amount $x_p$ using randomness $r_p$, $\pi_p$ denotes a range proof for $c_p$, and $vk_p$ denotes a Bitcoin address owned by the peer $p$. For ease of explanation, we assume here that every peer has only one input triple and that there are no fees in place.

From a high-level perspective, an execution of ValueShuffle consists of *runs*, and each run of ValueShuffle consists of four phases as follows.

**1. Output Generation.** Every peer $p$ locally generates its output triple $out_p = (c'_p = \mathrm{Com}(x'_p, r'_p), \pi'_p, addr'_p)$, where $c'_p$ is a CT-style commitment, $\pi'_p$ is the corresponding range proof, and $addr'_i$ is a fresh one-time address of the payee. Note that peers can have several output triples (including change outputs) but for simplicity, we restrict our attention to only one output here.

**2. Mixing and Secure Sum.** Peers run in parallel a P2P mixing protocol to mix their output triples $out_p$ and a secure sum protocol to privately compute the sum $r_\Delta = \sum_p r'_p - r_p$. Finally, input and output messages can be combined to deterministically form a (still unsigned) CoinJoin transaction by adding the explicit random value $r_\Delta$.

**3. Check.** Peers check validity of the resulting CoinJoin transaction, i.e., they check whether all range proofs $\pi'_p$ verify with respect to commitments $c'_p$, and check whether the overall balance of the intended transaction is correct, i.e., whether $\bigoplus_p c_p = \bigoplus_p c'_p + \mathrm{Com}(0, -r_\Delta)$. Also, every peer verifies that its own output triple is part of the mixing result, i.e., no coins are stolen by the transaction.

**4a. Confirm.** If all checks pass, the transaction is valid and peers are required to sign it. While every peer checked only that its output is present, DiceMix guarantees that this suffices to ensure that the outputs of all peers are present. Thus if some honest peer reached this point, the peer can be sure that peers refusing to sign the transaction are disruptive. If this happens, they will be excluded and a new run of the protocol is started.

**4b. Blame.** If any of the aforementioned checks fail, a blame phase is performed to detect at least one malicious peer. Every peer $p$ broadcasts the secrets that it used for the mixing and secure sum protocols, thereby revealing the value $r_p - r'_p$, which suffices to check that peer $p$ committed the same amount in the input and output addresses (and therefore no coins were created). Now every other peer $p'$ can recompute the mixing and secure sum steps of peer $p$ and detect whether it faithfully followed the protocol specification. The thereby exposed malicious peer is then excluded from the protocol and a new run is started.

## 5.4 Full Protocol Description

In this section we specify ValueShuffle fully. We start by describing the building blocks that the protocol relies on. For the sake of completeness, we repeat the description of digital signatures and non-interactive key exchange, which are already necessary for DiceMix (Section 4.4.1).

**Digital Signatures.** We require a digital signature scheme (KeyGen, Sign, Verify) unforgeable under chosen-message attacks (UF-CMA).

The algorithm KeyGen returns a private signing key $sk$ and the corresponding public verification key $vk$. On input message $m$, $\mathsf{Sign}(sk, m)$ returns $\sigma$, a signature on message $m$ using signing key $sk$. The verification algorithm $\mathsf{Verify}(pk, \sigma, m)$ outputs *true* iff $\sigma$ is a valid signature for $m$ under the verification key $vk$.

**Non-interactive Key Exchange.** We require a non-interactive key exchange (NIKE) mechanism ($\mathsf{NIKE.KeyGen}$, $\mathsf{NIKE.SharedKey}$) which is secure in the model by Cash, Kiltz, and Shoup (CKS model) [Fre+13; CKS09].

The algorithm $\mathsf{NIKE.KeyGen}(id)$ outputs a public key $npk$ and a secret key $nsk$ for a given party identifier $id$. $\mathsf{NIKE.SharedKey}(id_1, id_2, nsk_1, npk_2, sid)$ outputs a shared key for the two parties $id_1$ and $id_2$ and session identifier $sid$. NIKE.SharedKey must fulfill the standard correctness requirement that for all session identifiers $sid$, all parties $id_1$, $id_2$, and all corresponding key pairs $(npk_1, nsk_1)$ and $(npk_2, nsk_2)$, it holds that $\mathsf{NIKE.SharedKey}(id_1, id_2, nsk_1, npk_2, sid) = \mathsf{NIKE.SharedKey}(id_2, id_1, nsk_2, npk_1, sid)$. Additionally, we require an algorithm $\mathsf{NIKE.ValidatePK}(npk)$ which outputs *true* iff $npk$ is a public key in the output space of NIKE.KeyGen, and we require an algorithm $\mathsf{NIKE.ValidateKeyPair}(npk, nsk)$ which outputs *true* iff $nsk$ is a valid secret key for the public key $npk$.

Static Diffie-Hellman key exchange satisfies these requirements [CKS09], assuming a standard key derivation algorithm such as $\mathsf{NIKE.SharedKey}(id_1, id_2, x, g^y) := \mathsf{K}(g^{xy}, \{id_1, id_2\}, sid)$ for a hash function K modeled as a random oracle.

**Hash Functions.** We require hash functions H, G, and F modeled as random oracles.

**Confidential Transactions.** Confidential Transactions (CT) relies on a non-interactive commitment scheme (Com, Open), which uses public parameters we keep implicit, and a range proof (RPCreate, RPVerify). The commitment algorithm $c := \mathsf{Com}(m, r)$ uses some randomness $r$ to output a commitment $c$ of a message $m$. The opening algorithm $b := \mathsf{Open}(param, c, m, r)$ returns *true* iff $c$ is a valid commitment of message $m$ with randomness $r$. Informally, a commitment scheme is *hiding*, i.e., the commitment $c$ reveals nothing about $m$; and *binding*, i.e., no attacker can produce a commitment that it can open to two different messages $m' \neq m$. CT requires an additively homomorphic commitment scheme. A commitment scheme is additively homomorphic if there is an efficient operation $\oplus$ on commitments such that $\mathsf{Com}(m_1, r_1) \oplus \mathsf{Com}(m_2, r_2) = \mathsf{Com}(m_1 + m_2, r_1 + r_2)$. In practice, CT uses Pedersen commitments [Ped91] $g^m h^r$ on the prime-order elliptic curve `secp256k1`, where $g$ and $h$ are generators of the curve group such that the discrete logarithm of $h$ with respect to $g$ is not known.

In a range proof scheme, the algorithm $\pi := \mathsf{RPCreate}(m, r)$ creates a zero-knowledge proof $\pi$ that $c = \mathsf{Com}(m, r)$ is a commitment of an amount in a valid range. The algorithm $b := \mathsf{RPVerify}(\pi, c)$ returns *true* iff $\pi$ is a valid range proof for $c$. We refer the reader to the CT proposal [Max15; Gib16] for details.

**Confirmation.** The confirmation subprotocol $\textsc{Conf}()$ uses CoinJoin to perform the actual mixing. The algorithm $\mathsf{CoinJoinTx}()$ creates a CoinJoin transaction, and the algorithm $\mathsf{Submit}(tx, \sigma[\,])$ submits transaction $tx$ including the corresponding signatures $\sigma[\,]$ to the Bitcoin network.

A typical implementation of $\textsc{Conf}()$ produces a CoinJoin transaction with one signature from each peer. More sophisticated forms of confirmation, e.g., Schnorr multi-signatures with key aggregation [Max+19], are possible as in the case of CoinShuffle++ (Section 4.6.3).

**Conventions and Notation for the Pseudocode.** We use arrays written as $\textsc{Arr}[i]$, where $i$ is the index. We denote the full array (all its elements) as $\textsc{Arr}[\,]$.

Message $x$ is broadcast using "**broadcast** $x$". The statement "**receive** $\mathrm{X}[p]$ **from all** $p \in P$ **where** $X(\mathrm{X}[p])$ **missing** $C(P_{off})$" attempts to receive a message from all peers $p \in P$. The first message $\mathrm{X}[p]$ from peer $p$ that fulfills predicate $X(\mathrm{X}[p])$ is accepted and stored as $\mathrm{X}[p]$; all further messages from $p$ are ignored. When a timeout is reached, the statement $C$ is executed, which has access to a set $P_{off} \subseteq P$ of peers that did not send a (valid) message.

Regarding concurrency, a thread $t$ that runs a procedure $\mathrm{P}\,(args)$ is started using a statement "$t :=$ **fork** $\mathrm{P}\,(args)$", where $t$ is a handle for the thread. A thread with handle $t$ can either be joined using "$r :=$ **join** $t$", where $r$ is its return value, or it can be aborted using "**abort** $t$". A thread can wait for a notification and receive a value from another thread using "**wait**". The notifying thread uses "**notify** $t$ **of** $v$" to notify thread $t$ of some value $v$.

**Setup.** We assume that funds that should be used as input in ValueShuffle can only be spent by providing signatures, i.e., they are associated with a verification key that can also be used in ValueShuffle. Furthermore, for ease of explanation we assume here that every peer has only one input. However, ValueShuffle can easily be adapted to overcome this restriction: if a peer has more than one input, this peer can simply sign its messages using all signing keys corresponding to all verification keys, and the code for checking the balance can be adapted to consider the homomorphic combination of several input commitments.

As a result of these assumptions, every peer in the beginning knows an unspent transaction output $\mathrm{UTXO}[p]$, its corresponding CT commitment $\mathrm{C}[p]$ and verification key $\mathrm{VK}[p]$ for every other peer $p$.

Furthermore, every peer has its corresponding secrets, i.e., the amount $x$ and randomness $r$ such that $c = \mathrm{Com}(x, r)$, the secret key $sk$ corresponding to $vk$, and every peer has a set *Payments* with payees and corresponding amounts (including a change address if necessary), describing the payments it wants to perform. We assume that every peer wants to perform the same number of payments and that the transaction fee *fee* is evenly split among the peers.

**Full Pseudocode.** Here we describe the full protocol in pseudocode. We assume that the reader is familiar with the details of DiceMix and CoinShuffle++ (Chapter 4) to understand the code. For the sake of better readability, our essential changes to CoinShuffle++, which result in ValueShuffle, are printed in blue.

```
proc ValueShuffle ($P, my, \mathrm{VK}[\,], sk, \mathrm{UTXO}[\,], \mathrm{C}[\,], r, Payments, sid$)
    $sid := (sid, P, \mathrm{VK}[\,])$
    if $my \notin P$ then
        fail "not in the set of peers"
    end if
    return Run($P, my, \mathrm{VK}[\,], sk, sid, 0$)
end proc
proc Run ($P, my, \mathrm{VK}[\,], sk, sid, run$)
    $P^* := P \setminus \{my\}$
    if $P^* = \emptyset$ then
        fail "no other honest peers"
```

```
12      end if
13      ▹ Exchange pairwise keys
14      $(\mathrm{NPK}[my], \mathrm{NSK}[my]) := \mathsf{NIKE.KeyGen}(my)$
15      $sidPre := \mathsf{H}((\mathtt{sidPre}, sid, run))$
16      broadcast $(\mathtt{KE}, \mathrm{NPK}[my], \mathsf{Sign}(sk, (\mathrm{NPK}[my], sidPre)))$
17      receive $(\mathtt{KE}, \mathrm{NPK}[p], \sigma[p])$ from all $p \in P^*$
18          where $\mathsf{NIKE.ValidatePK}(\mathrm{NPK}[p])$
                    $\wedge\ \mathsf{Verify}(\mathrm{VK}[p], \sigma[p], (\mathrm{NPK}[p], sidPre))$
19      missing $P_{off}$ do
20          $P := P \setminus P_{off}$                                ▹ Exclude offline peers
21      end missing
22      $sid' := \mathsf{H}((\mathtt{sid'}, sid, P, \mathrm{NPK}[\,], run))$
23      $\mathrm{K}[\,] := \textsc{DC-Keys}(P^*, \mathrm{NPK}[\,], my, \mathrm{NSK}[my], sid')$
24      ▹ Generate fresh outputs to mix
25      $(myOut, myr) := \textsc{Gen}(Payments)$
26      $\mathrm{DC}[my][\,][\,][\,] := \textsc{DC-Mix}(P^*, my, \mathrm{K}[\,], myOut)$
27      $\textsc{SumDC}[my] := myr + \textsc{DC-Slot-Pad}(P, my, \mathrm{K}[\,], \mathtt{sum})$
28      $P_{ex} := \emptyset$                  ▹ Malicious (or offline) peers for later exclusion
29      ▹ Commit to DC-net vector
30      $\textsc{Com}[my] := \mathsf{H}((\mathtt{CM}, \mathrm{DC}[my][\,][\,][\,], \textsc{SumDC}[my]))$
31      broadcast $(\mathtt{CM}, \textsc{Com}[my], \mathsf{Sign}(sk, (\textsc{Com}[my], sid')))$
32      receive $(\mathtt{CM}, \textsc{Com}[p], \sigma[p])$ from all $p \in P^*$
33          where $\mathsf{Verify}(\mathrm{VK}[p], \sigma[p], (\textsc{Com}[p], sid'))$
34      missing $P_{off}$ do                           ▹ Store offline peers for exclusion
35          $P_{ex} := P_{ex} \cup P_{off}$
36      end missing
37      if $run > 0$ then
38          ▹ Wait for previous run to notify us of malicious peers
39          $P_{exPrev} :=$ wait
40          $P_{ex} := P_{ex} \cup P_{exPrev}$
41      end if
42      ▹ Collect shared keys with excluded peers
43      for all $p \in P_{ex}$ do
44          $\mathrm{K}_{ex}[my][p] := \mathrm{K}[p]$
45      end for
46      ▹ Start next run (in case this one fails)
47      $P := P \setminus P_{ex}$
48      $next :=$ fork $\textsc{Run}(P, my, \mathrm{VK}[\,], sk, sid, run + 1)$
```

```
    ▷ Open commitments and keys with excluded peers
    **broadcast** $(\texttt{DC}, \mathrm{DC}[my][\,][\,][\,], \textsc{SumDC}[my], \mathrm{K}_{ex}[my][\,], \mathsf{Sign}(sk, \mathrm{K}_{ex}[my][\,]))$
    **receive** $(\texttt{DC}, \mathrm{DC}[p][\,][\,][\,], \textsc{SumDC}[p], \mathrm{K}_{ex}[p][\,], \sigma[p])$ **from all** $p \in P^*$
        **where** $\mathsf{H}((\texttt{CM}, \mathrm{DC}[p][\,][\,][\,], \textsc{SumDC}[p])) = \textsc{Com}[p]$
                $\land \{p' : \mathrm{K}_{ex}[p][p'] \neq \bot\} = P_{ex}$
                $\land \mathsf{Verify}(\mathrm{VK}[p], \mathrm{K}_{ex}[p][\,], \sigma[p])$
    **missing** $P_{off}$ **do**                                  ▷ Abort and rely on next run
        **return** $\textsc{Result-Of-Next-Run}\,(P_{off}, next)$
    **end missing**
    $Out := \textsc{DC-Mix-Res}(P, \mathrm{DC}[\,][\,][\,][\,], P_{ex}, \mathrm{K}_{ex}[\,][\,])$
    $r_\Delta := \textsc{DC-Slot-Open}(P \cup \{my\}, \textsc{SumDC}[\,], \mathsf{sum}, P_{ex}, K_{ex}[\,][\,])$

    ▷ Check if our output is contained in the result
    $(balanced, Out_{mal}) := \textsc{VerifyResult}(i, P^*, Out, \mathrm{C}[\,], r_\Delta)$
    **if** $myOut \subseteq Out \land balanced \land Out_{mal} = \emptyset$ **then**
        $P_{mal} := \textsc{Conf}(i, P, Out, my, \mathrm{VK}[\,], sk, \mathrm{UTXO}[\,], sid)$
        **if** $P_{mal} = \emptyset$ **then**                                        ▷ Success?
            **abort** $next$
            **return** $m$
        **end if**
    **else**
        **broadcast** $(\texttt{SK}, \mathrm{NSK}[my])$                            ▷ Reveal secret key
        **receive** $(\texttt{SK}, \mathrm{NSK}[p])$ **from all** $p \in P^*$
            **where** $\mathsf{NIKE.ValidateKeyPair}(\mathrm{NPK}[p], \mathrm{NSK}[p])$
        **missing** $P_{off}$ **do**                              ▷ Abort and rely on next run
            **return** $\textsc{Result-Of-Next-Run}\,(P_{off}, next)$
        **end missing**
        ▷ Blame malicious peers using the secret keys
        $P_{mal} := \textsc{Blm}(P^*, \mathrm{NPK}[\,], my, \mathrm{NSK}[\,], \mathrm{DC}[\,][\,][\,][\,], sid', P_{ex}, \mathrm{K}_{ex}[\,][\,], Out_{mal}, \mathrm{C}[\,])$
    **end if**
    **return** $\textsc{Result-Of-Next-Run}\,(P_{mal}, next)$
**end proc**

**proc** $\textsc{DC-Mix}\,(P^*, my, \mathrm{K}[\,], myM)$
    $o := 1$
    **for all** $m \in myM$ **do**
        ▷ Split message into chunks and prefix those
        $\mathrm{C}[1] \,\|\, rem := m$               ▷ $\mathrm{C}[1]$ must be long enough to be unpredictable
        $\mathrm{C}[2] \,\|\, \ldots \,\|\, \mathrm{C}[n] := rem$       ▷ $\forall j \in \{2, \ldots, n\}. |\mathrm{C}[j]| = |\mathrm{C}[1]| - |\mathsf{F}(\mathrm{C}[1])|$
        **for** $j := 2, \ldots, n$ **do**
```

```
            $C[j] := F(C[1]) \| C[j]$
        end for
        ▷ Create power sums in individual slots
        for $j := 1, \ldots, n$ do
            for $s := 1, \ldots, |P^*| + 1$ do[4]
                $\text{DCMy}[o][j][s] := C[j]^s + \text{DC-Slot-Pad}(P^*, my, K[\,], (o, j, s))$
            end for
        end for
        $o := o + 1$
    end for
    return $\text{DCMy}[\,][\,][\,]$
end proc
proc DC-Mix-Res $(P, \text{DCMix}[\,][\,][\,], P_{ex}, K_{ex}[\,][\,])$
    $n := |\text{DCMix}[1]|$
    for $j := 1, \ldots, n$ do
        for $s := 1, \ldots, |P|$ do
            $M^*[s] := \text{DC-Slot-Open}(P, \text{DCMix}[\,][j][\,], s, P_{ex}, K_{ex}[\,][\,])$
        end for
        ▷ Solve equation system for array M[ ] of messages
        $M[j][\,] := \text{Solve}(\forall s \in \{1, \ldots, |P|\}.\ M^*[s] = \sum_{i=1}^{|P|} M[i]^s)$
    end for
    ▷ Recombine messages
    $M := \emptyset$
    for $i := 1, \ldots, |P|$ do
        $m := M[1][i]$
        for $j := 2, \ldots, n$ do
            $S := \{(i, m^*) : h \| m^* = M[j][i] \wedge h = F(M[1][i])\}$
            ▷ Unique match?
            if $\exists i, m^*.\ S = \{(i, m^*)\}$ then
                $m := m \| m^*$
            else
                continue (outer loop)            ▷ Invalid encoding, ignore message
            end if
        end for
        $M := M \cup \{m\}$
    end for
```

[4]If *run* $> 0$, it suffices to loop up to $|P^*|$ because at least one peer will have been excluded when the DC-net is opened.

```
        return $M$
    end proc
    proc DC-SLOT-PAD ($P^*, my, \mathrm{K}[\,], s$)
        return $\sum_{p \in P^*} \mathrm{sgn}(my - p) \cdot \mathrm{G}((\mathrm{K}[p], s))$                ▷ in $\mathbb{F}_q$
    end proc
    proc DC-SLOT-OPEN ($P, \mathrm{DC}[\,][\,], s, P_{ex}, \mathrm{K}_{ex}[\,][\,]$)
        ▷ Pads cancel out for honest peers
        $m^* := \sum_{p \in P} \mathrm{DC}[p][s]$                ▷ in $\mathbb{F}_q$
        ▷ Remove pads for excluded peers
        $m^* := m^* - \sum_{p \in P} \text{DC-SLOT-PAD}(P_{ex}, p, \mathrm{K}_{ex}[p][\,], s)$
        return $m^*$
    end proc
    proc DC-KEYS ($P^*, \mathrm{NPK}[\,], my, nsk, sid'$)
        for all $p \in P^*$ do
            $\mathrm{K}[p] := \mathsf{NIKE.SharedKey}(my, p, nsk, \mathrm{NPK}[p], sid')$
        end for
        return $\mathrm{K}[\,]$
    end proc
    proc BLM ($P^*, \mathrm{NPK}[\,], my, \mathrm{NSK}[\,], \mathrm{DC}[\,][\,][\,][\,], sid', P_{ex}, \mathrm{K}_{ex}[\,][\,], Out_{mal}, \mathrm{C}[\,]$)
        $P_{mal} := \emptyset$
        for all $p \in P^*$ do
            $P' := (P^* \cup \{my\} \cup P_{ex}) \setminus \{p\}$
            $\mathrm{K}'[\,] := \text{DC-KEYS}(P', \mathrm{NPK}[\,], p, \mathrm{NSK}[p], sid')$
            ▷ Reconstruct purported message $m'$ of $p$
            for $o := 1, \dots, |\mathrm{DC}[my]|$ do
                $m' := \mathrm{DC}[p][o][1][1] - \text{DC-SLOT-PAD}(P', p, \mathrm{K}'[\,], (o, 1, 1))$
                for $j := 2, \dots, |(\mathrm{DC}[p][o]|$ do
                    $m^* \parallel h := \mathrm{DC}[p][o][j][1] - \text{DC-SLOT-PAD}(P', p, \mathrm{K}'[\,], (o, j, 1))$
                    $m' := m' \parallel m^*$
                end for
                $Out' := Out' \cup \{m'\}$
            end for
            ▷ Replay DC-net messages of $p$
            $\mathrm{DC}'[\,][\,][\,] := \text{DC-MIX}(P', p, \mathrm{K}'[\,], Out')$
            if $\mathrm{DC}'[\,][\,][\,] \neq \mathrm{DC}[p][\,][\,][\,]$ then                ▷ Exclude inconsistent $p$
                $P_{mal} := P_{mal} \cup \{p\}$
            end if
        ▷ Verify that $p$ has sent valid range proofs
        if $Out' \cap Out_{mal} \neq \emptyset$ then
            $P_{mal} := P_{mal} \cup \{p\}$
        end if
        ▷ Reconstruct randomness $r'$ of $p$
        $r' := \textsc{SumDC}[p] - \textsc{DC-Slot-Pad}(P', p, \mathrm{K}'[\,], \mathsf{sum})$
        ▷ Verify that the balance of $p$ is correct
        if $\mathrm{C}[p] = \left(\bigoplus_{(c,\pi,addr)\in Out'} c\right) \oplus \mathsf{Com}(fee/|P'|, -r')$ then
            $P_{mal} := P_{mal} \cup \{p\}$
        end if
        ▷ Verify that $p$ has published correct symmetric keys
        for all $p_{ex} \in P_{ex}$ do
            if $\mathrm{K}_{ex}[p][p_{ex}] \neq \mathrm{K}'[p_{ex}]$ then
                $P_{mal} := P_{mal} \cup \{p\}$
            end if
        end for
    end for
    return $P_{mal}$
end proc
proc $\textsc{Result-Of-Next-Run}$ ($P_{exNext}$, $next$)
    ▷ Hand over to next run and notify of peers to exclude
    notify $next$ of $P_{exNext}$
    ▷ Return result of next run
    $result$ := join $next$
    return $result$
end proc
proc $\textsc{VerifyResult}$ ($i, P^*, Out, \mathrm{C}[\,], r_\Delta$)
    $Out_{mal} := \emptyset$
    ▷ Verify range proofs
    for all $out \in Out$ do
        $(c, \pi, addr) := out$
        if $\mathsf{RPVerify}(\pi, c)$ then
            $Out_{mal} := Out_{mal} \cup \{out\}$
        end if
    end for
    ▷ Check balance
    $balanced := \left(\bigoplus_{p\in P^*} \mathrm{C}[p] = \left(\bigoplus_{(c,\pi,addr)\in Out} c\right) \oplus \mathsf{Com}(fee, -r_\Delta)\right)$
    return $(balanced, Out_{mal})$
end proc
proc Gen $(Payments)$
    $myr := 0$
    $myOut := \emptyset$
    for all $(payee, amount) \in Payments$ do
        $r \xleftarrow{\$} \mathcal{R}$                    ▹ Fresh random value; implicitly stored in wallet
        $c := \mathsf{Com}(amount, r)$
        $\pi := \mathsf{RPCreate}(amount, r)$
        $myr := myr + r$
        $addr := \mathsf{FreshPayeeAddress}(payee)$
        $myOut := myOut \cup \{(c, \pi, addr)\}$
    end for
    return $(myr, myOut)$
end proc
proc Conf $(i, P, my, \mathrm{VK}_{in}[\,], sk_{in}, \mathrm{UTXO}[\,], Out, sid)$
    $tx := \mathsf{CoinJoinTx}(\mathrm{UTXO}_{in}[\,], Out)$
    $\sigma[my] := \mathsf{Sign}(sk_{in}, tx)$
    broadcast $\sigma[my]$
    receive $\sigma[\,]$ from all $p \in P \setminus \{my\}$
        where $\mathsf{Verify}(\mathrm{VK}_{in}[p], \sigma[p], tx)$
    missing $P_{off}$ do                              ▹ Peers refusing to sign are malicious
        return $P_{off}$
    end missing
    $\mathsf{Submit}(tx, \sigma[\,])$
    return $\emptyset$                                          ▹ Success!
end proc
```

### 5.4.1 Security Analysis

We argue why ValueShuffle achieves the desired security and privacy properties.

**Unlinkability.** Unlinkability follows from sender anonymity in DiceMix (Chapter 4): Whenever some honest peer $p$ signs the CoinJoin transaction, the confirmation phase has been reached. In this case, since output triples are freshly generated for each run, DiceMix guarantees that the honest peers form a proper DC-net. This in turn ensures that the attacker cannot distinguish whether an output triple of peer $p$ belongs to $p$ or some other honest peer $p'$. Note that the relation between peer and output triple can be revealed in the blame phase, but then a CoinJoin transaction with the current output triple will never be signed, so it is safe to reveal the relationship. Instead, the output triple will be discarded and a further run will be started, using a fresh output triple, which is unlinkable to the discarded output triples. We refer the reader to Section 4.4.5 for a detailed discussion.

**Termination.** DiceMix itself provides termination, and we have to argue that our extensions do not affect this property. This mainly boils down to ensuring that a malicious peer can be detected in each protocol run.

If one of the DC-nets for mixing the output triples is disrupted, then a malicious peer will be identified. This follows from the termination of DiceMix and the observation that each message chunk is unpredictable. If the DC-net for computing $r_\Delta$ is disrupted, then the blame phase will sacrifice anonymity for this run (discarding the output triples), and the malicious peer will be identified by checking its individual balance property, i.e., whether its set of inputs and outputs is balanced, as done in the blame phase. If a wrong range proof is provided, then the blame phase will sacrifice anonymity for this run (discarding the output triples), and the malicious peer can be identified by checking who provided the wrong range proof. In all other cases, the transaction will be valid by construction, so peers refusing to sign it are malicious (or offline) and thus can be excluded from further runs.

By construction, and if the bulletin board is honest (which we assume for termination, as otherwise it is impossible to achieve), all honest peers agree on the set of peers to exclude and thus on the set of remaining peers in the subsequential run of the protocol. We refer the reader to Section 4.4.5 for the details of termination.

**Correct Balance.** The protocol must ensure that no honest peer incurs money loss (ignoring transaction fees). ValueShuffle ensures correct balance because the mixing of output triples and randomness does not involve the transfer of funds. Before the CoinJoin transaction is formed, every honest peer checks that its output address and the corresponding committed amount is included and only then signs the transaction. As a CoinJoin transaction becomes valid only when every peer has signed the transaction (and thus confirmed that its funds are not stolen), ValueShuffle provides correct balance.

**CT Compatibility.** ValueShuffle does not impair the privacy guarantees provided by CT. The only secrets of CT belonging to some peer that ValueShuffle uses (and actually reveals in the blame phase) is its value $r - \sum_k r'_k$, where $k$ ranges over its output triples. However, since $r'_k$ are random and ephemeral, this sum does not reveal anything about the individual $r$ and $r'_k$ values and thus does not affect the hiding property of the input commitment or any of the individual output commitments.

## 5.5 Related Work

A variety of privacy solutions have been proposed so far in the literature, based on different paradigms. We discuss these in the following (except for related work in the area of coin mixing, which we discuss in Section 4.6.4).

**Monero.** The design of the cryptocurrency Monero is is the closest to our work in terms of provided privacy guarantees. Monero relies on Ring Confidential Transactions (RingCT) [NMM16], a combination of ring signatures (to provide anonymity for the payer in a transaction [Sab13]) and CT (to hide payment amounts). In contrast to ValueShuffle, an online mixing protocol is not required, and a sufficient anonymity set can be created using funds of users currently offline.

However, Monero's use of RingCT comes with two important drawbacks for scalability. First, Monero essentially performs mixing on the blockchain and requires each transaction to contain a ring signature of size $O(n)$, where $n$ is the size of the anonymity set. Storing the ring signatures requires a lot of precious space in the blockchain, and verifying them puts a large burden on all nodes in the currency network. In contrast, ValueShuffle performs the actual mixing off-chain and stores only the result on the blockchain.

Second, Monero is not compatible with pruning, a feature supported, e.g., by the Bitcoin Core client [Core15]. Pruning can reduce the storage requirements of nodes drastically by deleting old blocks and spent transactions once verified. This is impossible in Monero because its use of ring signatures prevents clients from determining whether an transaction output has been spent and thus can be pruned. A CoinJoin-based approach such as ValueShuffle does not have this problem and is fully compatible with pruning.

From a high-level point of view, ValueShuffle moves the overhead of providing payer anonymity from the blockchain and thus the whole Bitcoin network to only the peers actively involved in a single mixing.

**Mimblewimble.** Mimblewimble [Jed16; Poe16] is a design for a cryptocurrency with CT, which has been implemented for example in Grin [Grin]. Multiple Mimblewimble transactions that can be aggregated non-interactively and even across blocks into one transaction. This has tremendous benefits for the scalability of the underlying blockchain, and the resulting aggregate transaction hides the relationship between inputs and outputs like in a CoinJoin transaction. However, such aggregation alone does not ensure input-output unlinkability against parties who perform the aggregation, e.g., the miners, because those parties still get to see the individual transactions that they aggregate. ValueShuffle can be used in Mimblewimble-based currencies to provide this unlinkability.

**Zerocoin and Zerocash.** Zerocoin [Mie+13] and its successor Zerocash [Ben+14], whose implementation Zcash has been deployed in practice, are cryptocurrency protocols that provide anonymity by design. Although these solutions provide strong privacy guarantees, they rely on a trusted setup and non-falsifiable cryptographic assumptions [GW11] due to the use of zkSNARKS. Moreover, since it is not possible to observe which outputs have been spent already, blockchain pruning is not possible in Zerocoin and Zerocash.

# 6 Preparing Commitments for a Post-Quantum World

The security of Bitcoin relies on cryptographic hardness assumptions, e.g., the discrete logarithm assumption on the `secp256k1` [SEC1] elliptic curve, which is necessary for the unforgeability of ECDSA signatures. Advances in solving the discrete logarithm problem can lead to uncertainty about whether currently deployed key sizes or algorithms are still safe. In particular, the possible availability of large-scale quantum computers could render the discrete logarithm assumption obsolete in the future.

As soon as quantum computers are considered a realistic and imminent concern, the obvious step in every cryptographic system is to retire the current algorithms and switch to new cryptographic algorithms considered secure against quantum attackers. However, cryptographic agility is much more difficult to achieve in cryptocurrencies than in many other cryptographic systems because the blockchain keeps permanent state. For example, introducing a post-quantum signature scheme is easily possible, but this measure just ensures the security of future transactions and will not retroactively fix transactions performed in the past: the funds stored at previously created unspent transaction outputs on the blockchain will still be only protected by the unforgeability of the obsolete signature scheme.

When a new signature scheme is available, it is the responsibility of users to spend these old outputs to fresh addresses of their own, thereby creating new unspent outputs protected by the unforgeability of the new signature scheme. While moving the funds does not prevent the attacker from breaking the old signing keys, the consensus mechanism will ensure that the old outputs will already have been spent when the attacker obtains the signing keys. Individual users may lose their money if they fail to spend their vulnerable outputs, but the security of Bitcoin as a whole remains largely unaffected.

While the need to transition to a post-quantum signature scheme creates an unfortunate situation because users' funds are at risk of theft, the situation will be much worse in a cryptocurrency with Confidential Transactions (CT) [Max15; Gib16]. CT is a privacy-enhancing technology that has been proposed as an extension to Bitcoin (Section 5.2.1). The purpose of CT is to hide the monetary amounts in transactions by replacing plain amounts by commitments to the amounts. Since the commitment scheme used is additively homomorphic, the creator of a transaction can easily prove to the network that a transaction is *balanced*, i.e., the sum of its outputs is not more money than the sum of its inputs. The proof essentially opens the commitment to the homomorphic sum of the inputs minus the outputs to zero, which does not reveal the individual monetary amounts of the inputs and outputs in the transaction. A non-interactive zero-knowledge proof is added to each commitment to show that the committed amount is an integer in the range $[0, d-1]$ for some $d$. These *range proofs* ensure that the computation of the summands or not negative or that the sum does not overflow (Section 5.2.1).[1]

CT has been tested and evaluated in the Elements platform [Elements], and successfully deployed in multiple cryptocurrencies such as Grin [Grin] and Monero [XMR] (as an integral component of Ring Confidential Transactions (RingCT) [NMM16], a scheme that additionally aims to provide a certain level of payer anonymity).

The original CT proposal relies on Pedersen commitments computed as $c = g^m h^r$, where $m$ is the message, $r$ is a random value, and $g$ and $h$ are public generators of the `secp256k1` elliptic curve. Pedersen commitments are only *computationally* binding under the assumption that computing discrete logarithms is hard. Thus, if an attacker manages to break one discrete logarithm with current parameters, the balance property of the currency breaks down with catastrophic consequences: Knowledge of $\log_g h$ enables the attacker to open each of its commitments, no matter what amount it is supposed to commit to, to an arbitrary amount of money. That is, the attacker can effectively create an arbitrary amount of money and inflate the currency. Even worse, this attack will go unnoticed due to the hiding property of the commitments. As a consequence, if the attacker manages to compute a single discrete logarithm, not only the individual security of funds is threatened, but the entire currency is doomed.

Thus the situation is much worse than without CT when there is doubt about the hardness of breaking the deployed cryptography. With CT, the only safe way out is to introduce new parameters or algorithms and force users to spend unspent transaction outputs protected by the obsolete parameters *before some hard deadline $T$*. After time $T$, such obsolete outputs will not be spendable anymore, i.e., the corresponding funds will expire, effectively destroying money. This is highly undesirable and it is not clear at all if such a change will find consensus in the community.

The obvious way to overcome all of the aforementioned issues is to use a commitment scheme that is statistically binding, i.e., it is binding even against a computationally unrestricted attacker. For instance, just adding $g^r$ turns a computationally

[1]For example, $d = 2^{64} > 2\,099\,999\,997\,690\,000$, where the number on the right-hand side is the maximum number of satoshis that can be in existence in Bitcoin.

binding Pedersen commitment into a statistically binding ElGamal commitment. (It is actually even perfectly binding, but we stick to the more general statistical property in this chapter.)

However, this modification comes with two drawbacks: First, it requires efficient range proofs particularly suitable for the new commitment scheme, e.g., it precludes the use of efficient range proofs developed for Pedersen commitments [Bün+18; Poe+18], which are more efficient than those for ElGamal commitments. Second, statistically binding commitments are necessarily at most computationally hiding. ElGamal commitments are computationally hiding under the decisional Diffie-Hellman assumption, and hence not hiding against quantum attackers.[2] Since the commitments are recorded on the public blockchain, this constitutes a serious threat for the privacy of users. If large-scale quantum computers will be available in the future, they will be able to break the hiding property of all commitments in retrospect. This does not only break the confidentiality of the payment values but enables attackers to trivially and retrospectively link inputs and outputs in a mixing transaction, e.g., created using ValueShuffle (Chapter 5).

## 6.1 Solution Overview

The goal of this chapter is to provide a solution that prepares CT for a post-quantum world but for now remains compatible with efficient range proofs and additionally preserves privacy against post-quantum attackers whenever possible. Our tool to achieve this goal is a novel security notion between computational and statistical bindingness. We introduce *switch commitments*, which are commitments with a *partial* and a *full* verification algorithm and two binding properties: First, the commitment is computationally binding when partially verified. Second, the commitment is *everlastingly binding* in the following sense: if the commitment is created by a computationally bounded attacker and can be opened to some message when partially verified, then later even a computationally unbounded attacker cannot open the commitment to a different message when fully verified. This novel property captures the essence of switch commitments.

[2]One could try to rely on a different commitment scheme instead. However, there are no promising candidates to the best of our knowledge. The only known post-quantum secure additively homomorphic commitment schemes are based on lattice assumptions [e.g., XXW13; Bau+18; Pin+17] and thus less efficient than ElGamal commitments. More importantly, they allow only for a small bounded number of homomorphic additions because the error term grows in the homomorphic addition, and this bound on the number of additions must be fixed in advance. But since outputs of transactions are supposed to be used as inputs of new transactions, thereby forming a chain of transactions, there is no reasonable bound in the application to CT, and those commitments cannot be used directly.

These properties enable verifiers to use the commitment scheme in a computationally binding or a statistically binding way, depending on the verification algorithm used. In particular, everlasting bindingness ensures that it is possible to start with partial verification and then *switch* to full verification, even for already existing commitments, e.g., commitments stored on the blockchain.

**ElGamal Commitments are Switch Commitments.** We prove that ElGamal commitments $(g^m h^r, g^r)$ with a message space of polynomial size (in the security parameter) are homomorphic switch commitments where the partial verification algorithm ignores the element $g^r$ and verifies only the Pedersen commitment $g^m h^r$. Since the message space of commitments used in CT is restricted to integers in a fixed range to avoid overflow anyway, ElGamal commitments are a natural choice for a switch commitment scheme to be used with CT.

**Opt-In Switch Commitments for Post-Quantum Privacy.** However, since ElGamal commitments are not post-quantum hiding, we additionally introduce *opt-in switch commitments*. By using a hash-based commitment to $g^r$ instead of $g^r$ directly, we give the user control about whether $g^r$ will be public or not. In our application to CT, this has the desirable property that the commitments are binding and hiding in almost all cases. Only in the exceptional case that the user fails to transfer before a certain deadline (discussed below), the user needs to forgo one of the properties, and even in this case, the user can choose between forgoing the binding property (i.e., the money) or the hiding property (i.e., privacy). Since the hash-based commitment to $g^r$ can be added to an exponent of the Pedersen commitment, the size of the resulting commitment is just one group element, i.e., our construction of opt-in switch commitments is as efficient as a Pedersen commitment in terms of space.

### 6.1.1 Usage in Confidential Transactions

An (opt-in) switch commitment scheme can be used in CT as follows: When performing a transaction now, the network relies only on the partial verification to ensure that the transaction is balanced, i.e., the transaction does not generate money out of thin air. In particular, creators of transactions are forced to prove that they can open the switch commitments to messages such that no money will be created and just the partial verification algorithm accepts the openings. While this means that the balance property holds only computationally, it is sufficient to use range proofs that cover only partial verification, i.e., the creator of the switch commitment must only demonstrate the ability to open the commitment to a value in range when the opening is partially verified. Applied to ElGamal commitments, this effectively means that it suffices for the range proof to cover only the first component of the ElGamal commitment, which is a Pedersen commitment. This is efficient because the most efficient known range proofs systems operate on Pedersen commitments.

Ideally, a fully post-quantum secure CT scheme (consisting of a post-quantum binding and hiding commitment scheme and a post-quantum sound and zero-knowledge range proof system) will have been integrated in the cryptocurrency already long before large-scale quantum computers will be in reach, assuming a reasonable scheme will be found in the meantime. Then users will be encouraged (or even forced by the consensus rules) to convert their funds from the switch commitment scheme to the post-quantum commitment scheme, e.g., using zero-knowledge proofs. However, some old transaction outputs using the switch commitment scheme will remain unspent in the blockchain simply because some users will fail to perform the conversion in time.

Assume that at some point in the future, there will be serious doubt about the cryptographic strength of the used switch commitment scheme or its parameters. Then a softfork can require confidential transactions that are performed after some *switch deadline* $T$ and that spend funds still secured by the switch commitment scheme to be fully verified. That means that a user who has failed to perform the conversion in time will still be able to spend an old output (containing a switch commitment) to a new output (containing a post-quantum secure commitment) by providing a statistically sound proof of the balance property. This proof of the balance property will include a statistically sound range proof demonstrating the user's ability to open the old switch commitment to a value in range such that the full verification algorithm will accept this opening. Since the switch commitment scheme is everlastingly binding, the switch to full verification ensures that no attacker will be able to spend an unspent output with a switch commitment with more money than it is supposed to contain even if this commitment was created by the attacker before time $T$ (when the attacker was computationally bounded by assumption).

Therefore switch commitments provide a safety switch for CT: by switching to full verification, we can ensure that even old commitments created years or decades ago never need to expire.

**Usage of Opt-In Switch Commitments.** If opt-in switch commitments are used, all old switch commitments in the blockchain (including those already spent) additionally preserve confidentiality against post-quantum attackers. A problem appears only for the few users who have failed to convert their money to outputs secured by quantum-secure commitment schemes by the switch deadline $T$. Those users can opt-in to the switch: By default they keep their privacy but cannot access their money. However, if they would like to opt-in, they can claim their money but risk their privacy by revealing additional information (i.e., $g^r$ as described above), which effectively is a fallback to an ordinary ElGamal commitment. Conveniently, users do not need to guess already today whether they will prefer privacy or money in the future because the decision to opt-in can be taken after the switch deadline $T$, and in fact can be postponed essentially forever.

Furthermore, there is always a way out of the dilemma using zero-knowledge proofs: An affected user can, instead of revealing $g^r$, give a post-quantum secure zero-knowledge proof that the amount in the switch commitment equals the amount in a post-quantum secure commitment to which the money should be converted.

Such a zero-knowledge proof will demonstrate a statement involving a hash computation. Even though we do not know if efficient enough zero-knowledge proof systems will be available in the future, the efficiency of state-of-the-art post-quantum secure proofs systems such ZK-STARKs [Ben+18] gives reasonable hope. We stress that an expensive zero-knowledge proof will be necessary only once per switch commitment and only for the few old switch commitments that have not been converted in time.

**Efficiency Comparison of Known Range Proof Systems.** Our results imply that it is safe that the range proof covers only the first component of the ElGamal commitment (i.e., the Pedersen commitment) until the switch deadline $T$. Here, we explain why this leads indeed to more efficient range proofs. In the following, we assume that we would like to prove that a committed value $m$ is in the range $[0, b^n - 1]$. Due to the fact that blockchain storage is very expensive, our main concern is size, and we further assume that we work in elliptic curves, and hence group and field elements are of roughly the same size.

The currently most efficient range proof for Pedersen commitments uses the Bulletproofs framework [Bün+18] and has size $2\lceil \log_2 n \rceil + 9 = O(\log n)$ where $b = 2$. An optimized range proof for ElGamal commitments using Bulletproofs has not been proposed yet, but it is conceivable that such a range proof needs to be larger because it necessarily needs to prove a statement that involves two group elements instead of just one.[3]

If we ignore Bulletproofs, then for Pedersen commitments, the smallest known range proof has been proposed by Back and Maxwell [Poe+18] and needs $bn + 1$ elements. For ElGamal commitments, the smallest known range proof has been

[3]Since Bulletproofs are only computationally sound, this will be a computationally sound range proof for a statistically binding commitment. This combination may look unnatural, but it fits the attacker model of switch commitments, which assumes that the attacker is initially computational bounded when it creates commitments but is unbounded when it spends those commitments later.

proposed by Andreev [And17] and needs $(b+1)n+1$ elements; it is very similar to the one by Back and Maxwell, and it is statistically sound.

## 6.2 Commitments

We formally introduce (non-interactive) commitment schemes, which we will use throughout this chapter. A commitment scheme [Ped91] is a two-phase protocol between a sender and a receiver. In the first phase, the sender commits to a message $m$ by sending a string com to the receiver. In the second phase, the sender reveals the opening information op and the message $m$ to the receiver, who can check whether com was indeed a valid commitment on $m$. All algorithms have access to a public string crs generated by a trusted setup party. (We will relax this assumption in the random oracle model.)

A commitment scheme is *computationally hiding* if the commitment does not reveal anything about the message to a computationally bounded attacker.

**Definition 6.1** (Computationally Hiding). *A commitment commitment algorithm* Commit *is* computationally hiding *if there exists a function* $\mathsf{negl}(\lambda)$ *negligible in the security parameter* $\lambda$ *such that for all ppt attackers* $\mathcal{A}$, *for all pairs of messages* $(m_0, m_1)$, *and for a random public string* $\mathsf{crs} := \mathsf{Setup}(1^\lambda)$,

$$\Pr\left[\,\mathcal{A}(\mathsf{crs}, \mathsf{com}) = b \mid b \leftarrow \{0,1\}; \mathsf{com} := \mathsf{Commit}(\mathsf{crs}, m_b)\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

A commitment scheme is *binding* if no sender is able to output two openings $(\mathsf{op}, \mathsf{op}')$ for the same commitment com such that the two openings open the commitment to two different values. We consider binding against computationally bounded and against unbounded attackers.

**Definition 6.2** (Computationally and Statistically Binding). *A verification algorithm* Verify *is* computationally binding *if there exists a function* $\mathsf{negl}(\lambda)$ *negligible in the security parameter* $\lambda$ *such that for all ppt attackers* $\mathcal{A}$ *and for a random* $\mathsf{crs} := \mathsf{Setup}(1^\lambda)$, *we have that*

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{crs}, \mathsf{com}, \mathsf{op}, m) = 1 \\ \quad \wedge\ \mathsf{Verify}(\mathsf{crs}, \mathsf{com}, \mathsf{op}', m') = 1 \\ \quad \wedge\ m \neq m' \end{array} \;\middle|\; (\mathsf{com}, \mathsf{op}, m, \mathsf{op}', m') := \mathcal{A}(\mathsf{crs}) \right] \leq \mathsf{negl}(\lambda).$$

Statistical bindingness *is defined identically except that* $\mathcal{A}$ *is computationally unbounded.*

**Pedersen and ElGamal Commitments.** Let $\mathsf{crs} = h$ be a random element of a prime-order group with with generator $g$; CT uses the prime-order elliptic curve `secp256k1` [SEC2]. Given a message $m$ and a random integer $r$, the Pedersen commitment is the group element $g^m h^r$, and the ElGamal commitment is the pair $(g^m h^r, g^r)$ of group elements. For both schemes, the verification algorithm recomputes the commitment from the opening $(m, r)$ and accepts if it matches the original commitment.

Pedersen commitments are computationally binding under the discrete logarithm assumption and they are perfectly hiding [Ped91]. ElGamal commitments are statistically (and even perfectly) binding and they are computationally hiding under the decisional Diffie-Hellman assumption [Elg84].

## 6.3 Switch Commitments

Now we extend the notion of a commitment scheme to support a switch from partial to full verification, and we formally introduce the security definitions for our primitive.

**Definition 6.3** (Switch Commitment Scheme). *A switch commitment scheme* $\mathcal{SC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify}^{\mathsf{part}}, \mathsf{Verify}^{\mathsf{full}})$ *consists of four ppt algorithms as follows:*

$\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$: *Given the security parameter $\lambda$, the setup algorithm* Setup *outputs a public string* crs.

$(\mathsf{com}, \mathsf{op}) \leftarrow \mathsf{Commit}(\mathsf{crs}, m)$: *Given the public string* crs*, and a message $m$, the commitment algorithm* Commit *outputs a commitment* com *and opening information* op.

$b \leftarrow \mathsf{Verify}^{\mathsf{part}}(\mathsf{crs}, \mathsf{com}, \mathsf{op}, m)$: *Given the public string* crs*, a message $m$, a commitment* com *and opening information* op*, the partial verification algorithm* $\mathsf{Verify}^{\mathsf{part}}$ *outputs* 1 *iff* op *is a valid partial opening for commitment* com *on message $m$.*

$b \leftarrow \mathsf{Verify}^{\mathsf{full}}(\mathsf{crs}, \mathsf{com}, \mathsf{op}, m)$: *Given the public string* crs*, a message $m$, a commitment* com *and opening information* op*, the full verification algorithm* $\mathsf{Verify}^{\mathsf{full}}$ *outputs* 1 *iff* op *is a valid full opening for commitment* com *on message $m$.*

### 6.3.1 Security Properties

Since switch commitments with the partial verification algorithm are used like ordinary commitments before the switch, they need to fulfill the same security properties as ordinary commitments.

**Standard Security Properties.** For every secure switch commitment scheme, we require that the commitment algorithm is computationally hiding and that the partial verification algorithm is computationally binding.

**Everlastingly Binding.** To model the security of the switch, we consider an attacker that is computationally bounded *when creating a commitment but not when opening a commitment.* In the following we formally define this notion of *everlasting bindingness* for a switch commitment scheme.

**Definition 6.4** (Everlastingly Binding). *A switch commitment scheme* $\mathcal{SC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify}^{\mathsf{part}}, \mathsf{Verify}^{\mathsf{full}})$ *is* everlastingly binding *if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all attackers* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, *where* $\mathcal{A}_0$ *is ppt (and* $\mathcal{A}_1$ *is not computationally bounded), and for a randomly sampled* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$, *we have that*

$$\Pr\left[\begin{array}{l} \mathsf{Verify}^{\mathsf{part}}(\mathsf{crs}, \mathsf{com}, \mathsf{op}, m) = 1 \\ \quad \wedge \mathsf{Verify}^{\mathsf{full}}(\mathsf{crs}, \mathsf{com}, \mathsf{op}', m') = 1 \\ \quad \wedge m \neq m' \\ \hline (\mathsf{com}, m, \mathsf{op}, \mathit{state}) \leftarrow \mathcal{A}_0(\mathsf{crs}); \\ (m', \mathsf{op}') \leftarrow \mathcal{A}_1(\mathsf{crs}, \mathit{state}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

We note that similar models in which the attacker is bounded only up to a certain point in time have already been considered for privacy properties in the context of electronic voting [MN06], multi-party computation [Unr13], and encryption in the bounded storage model [HN06].

## 6.4 Construction

In the following we describe our construction for an ordinary switch commitment scheme. This construction will also form the basis for our construction of opt-in switch commitments presented in Section 6.5. Our scheme is essentially a combination of a Pedersen and ElGamal commitment scheme with restricted message space. The commitment algorithm outputs an ElGamal commitment $(g^x h^r, g^r)$ and the full verification algorithm recomputes the commitment to verify it. However, the partial verification algorithm verifies only the Pedersen commitment $g^x h^r$. This makes it possible to use efficient range proofs optimized for Pedersen commitments.

It is crucial for the security of our construction that the message space is restricted to a size polynomial in the security parameter and the verification algorithm rejects messages not in the space. In the proof of everlasting bindingness, the reduction guesses a message in a commitment, and thus the reduction incurs a loss proportional to the size of the message space. Slightly increased parameters are necessary to compensate for this loss of security.

Note that the message space of the commitments used in CT is already is restricted to integers in the range $[0, d-1]$ for a fixed non-negative integer $d$ that is a parameter of the system (implicitly known to all algorithms) and determines the maximum amount of a transaction.

With the application in CT in mind, we describe the scheme for concreteness with this message space and we assume that $d$ is constant in the security parameter. We however note that any other restriction of the message space is possible, as long as the message space has polynomial size in the security parameter, membership can be checked efficiently, and elements from the message space can be sampled efficiently.

**Construction.** We work in a group of prime order $p$ with generator $g$, for some prime $p$ of bitlength proportional to the security parameter $\lambda$. Our switch commitment scheme is defined as follows.

$\mathsf{Setup}(1^\lambda)$: Sample a random $x \in \mathbb{Z}_p^*$, compute $h = g^x$ and output $\mathsf{crs} = h$.

$\mathsf{Commit}(\mathsf{crs}, m)$ Parse crs as $h$ and sample $r \in \mathbb{Z}_p$. Return $\mathsf{com} = (g^m h^r, g^r)$ and $\mathsf{op} = r$.

$\mathsf{Verify}^{\mathsf{part}}(\mathsf{crs}, \mathsf{com}, \mathsf{op}, m)$: Parse crs as $h$, com as $(c_1, c_2)$, and op as $r$. If $c_1 = g^m h^r$ and $0 \leq m < d$, then return 1. Return 0 otherwise.

$\mathsf{Verify}^{\mathsf{full}}(\mathsf{crs}, \mathsf{com}, \mathsf{op}, m)$: Parse crs as $h$, com as $(c_1, c_2)$, and op as $r$. If $c_1 = g^m h^r$, $c_2 = g^r$, and $0 \leq m < d$, then return 1. Return 0 otherwise.

**Avoiding Trusted Setup.** We have chosen a description in the standard model to stress that the construction does not require random oracles. However, it is possible to avoid a trusted setup in the random oracle model by setting $h = H(g)$, for a hash function $H$. This is essentially what has been proposed in the draft of CT.

### 6.4.1 Security Analysis

We formally analyze the security of our construction. To prove that the construction is everlastingly binding, we need the inverse Diffie-Hellman assumption, which is known to be equivalent to the computational Diffie-Hellman assumption; see Appendix A for all computational hardness assumptions.

**Claim 6.1** (Standard Security Properties). *The construction fulfills the standard security properties under the decisional Diffie-Hellman assumption. In particular, the commitment* *algorithm of the construction is computationally hiding under the decisional Diffie-Hellman assumption, and the partial verification algorithm is computationally binding under the discrete logarithm assumption.*

*Proof.* Since the commitment algorithm is identical to the one for ElGamal commitments, the construction is computationally hiding under the decisional Diffie-Hellman assumption. Since the partial verification algorithm is identical to the one for Pedersen commitments, the construction is computationally binding under the Discrete Logarithm assumption. We refer the reader to ElGamal [Elg84] and Pedersen [Ped91] for detailed arguments. □

**Theorem 6.2** (Everlastingly Binding)**.** *The construction is everlastingly binding under the inverse Diffie-Hellman assumption.*

*Proof.* We prove that the construction is everlastingly binding under the inverse Diffie-Hellman assumption. Assume towards contradiction that there exists an attacker $(\mathcal{A}_0, \mathcal{A}_1)$ such that $\mathcal{A}_0$ is ppt and

$$\Pr\left[\begin{array}{c} \mathsf{Verify}^{\mathsf{part}}(\mathsf{crs}, \mathsf{com}, \mathsf{op}, m) = 1 \\ \wedge\ \mathsf{Verify}^{\mathsf{full}}(\mathsf{crs}, \mathsf{com}, \mathsf{op}', m') = 1 \\ \wedge\ m \neq m' \\ \hline (\mathsf{com}, m, \mathsf{op}, \mathit{state}) \leftarrow \mathcal{A}_0(\mathsf{crs}); \\ (m', \mathsf{op}') \leftarrow \mathcal{A}_1(\mathsf{crs}, \mathit{state}) \end{array}\right] \geq \epsilon(\lambda).$$

for some non-negligible function $\epsilon(\lambda)$. We construct the following reduction that solves the inverse Diffie-Hellman problem.

On input a random group element $h = g^x$, the reduction sets $\mathsf{crs} = h$. Then it runs $\mathcal{A}_0$ on input crs, which outputs $(\mathsf{com} = (c_1, c_2), m, \mathsf{op} = r, \mathit{state})$ with non-negligible probability. Finally, the reduction samples a random $m^*$ with $0 \leq m^* < d$ and outputs

$$\left(\frac{g^r}{c_2}\right)^{(m'-m)^{-1}}.$$

The reduction is efficient since it only executes $\mathcal{A}_0$, which is ppt; note that the reduction never executes $\mathcal{A}_1$.

By assumption, $\mathcal{A}_1$ will be able to open the commitment to some value $m'$ such that $m' \neq m$ and $m' \leq d$ with probability at least $\epsilon(\lambda)$. Assume that the reduction guesses the value $m^*$ such that $m^* = m'$; this happens with probability $1/d$. Let $r' = \log_g c_2$. By the verification equations of both verification algorithms, we have $c_1 = g^m h^r = g^{m'} h^{r'}$ with $h = g^x$. By comparing exponents, we obtain $m + xr = m' + xr'$, and due to

$m \neq m'$, we have further

$$(r - r')(m' - m)^{-1} = x^{-1}.$$

This implies for the output of the reduction that

$$\left(\frac{g^r}{c_2}\right)^{(m'-m)^{-1}} = g^{(r-r')(m'-m)^{-1}} = g^{x^{-1}}$$

with probability at least $\epsilon(\lambda)/d$, which is non-negligible. This is a contradiction to the inverse Diffie-Hellman assumption and concludes the proof. □

## 6.5 Opt-In Switch Commitments

Our construction in Section 6.4 comes with a crucial weakness: the commitment is essentially an ElGamal commitment and as such it is only computationally hiding, and the required decisional Diffie-Hellman assumption does not withstand quantum attacks. However, we are able to overcome that weakness by a variant of the construction in the random oracle model. This variant additionally reduces the size of the commitment to one group element, and as a result, it introduces essentially no overhead for the cryptocurrency network as compared to Pedersen commitments.

Instead of committing with $(c_1, c_2) = (g^m h^r, g^r)$, a user can commit with $\tilde{c} = g^m h^{r+\mathsf{H}(g^m h^r, g^r)}$ for a hash function $\mathsf{H}$ modeled as a random oracle. In other words, instead of outputting the second component of the ElGamal commitment in clear, the user commits to it by tweaking the Pedersen commitment. This gives the user the possibility to reveal the full ElGamal commitment only if desired, and consequently we call this variant *opt-in switch commitments*. As soon as the user reveals the full ElGamal commitment $(c_1, c_2) = (g^m h^r, g^r)$, verifiers can check that $c_1 \cdot h^{\mathsf{H}(c_1, c_2)} = \tilde{c}$, and proceed as with ElGamal commitments.

More generally, we formalize opt-in switch commitments as follows. Since a formal introduction of quantum algorithms is not necessary to understand our results, we refer the interested reader to Nielsen and Chuang [NC10].

**Definition 6.5.** *A switch commitment scheme is* opt-in *if the commitment algorithm is computationally hiding against quantum-polynomial-time attackers.*

### 6.5.1 Construction

As before, we work in a group of prime order $p$ with generator $g$, for some prime $p$ of bitlength proportional to the security parameter $\lambda$. Our (opt-in) switch commitment scheme is defined as follows. We denote by an underscore (_) a component that is ignored when parsing a tuple.

$\mathsf{Setup}(1^\lambda)$: Sample a random $x \in \mathbb{Z}_p^*$, compute $h = g^x$ and output $\mathsf{crs} = h$.

$\mathsf{Commit}\,(\mathsf{crs}, m)$: Parse crs as $h$, sample $r \in \mathbb{Z}_p$, and compute $\tilde{r} = r + \mathsf{H}(g^m h^r, g^r)$. Return $\mathsf{com} = g^m h^{\tilde{r}}$ and $\mathsf{op} = (\tilde{r}, (g^m h^r, g^r), r)$.

$\mathsf{Verify}^{\mathsf{part}}\,(\mathsf{crs}, \mathsf{com}, \mathsf{op}, m)$: Parse crs as $h$, com as $\tilde{c}$, and op as $(\tilde{r}, \_, \_)$. If $\tilde{c} = g^m h^{\tilde{r}}$ and $0 \leq m < d$, then return 1. Return 0 otherwise.

$\mathsf{Verify}^{\mathsf{full}}\,(\mathsf{crs}, \mathsf{com}, \mathsf{op}, m)$: Parse crs as $h$, com as $\tilde{c}$, and op as $(\_, (c_1, c_2), r)$. If $\tilde{c} = c_1 \cdot h^{\mathsf{H}(c_1, c_2)}$, $(g^m h^r, g^r) = (c_1, c_2)$, and $0 \leq m < d$, then return 1. Return 0 otherwise.

**Remarks.** An alternative partial verification algorithm could just recompute $\tilde{r}$ from $m$ and $r$. However, this is not what we want to capture. Recall that the commitment will not actually be opened before the switch but that a creator of a confidential transaction will instead compute the homomorphic sum of the commitments in a transaction and open the resulting sum commitment to zero to prove that the transaction is balanced. Only if $\tilde{r}$ itself can be used as opening information we retain the homomorphic property with respect to partial verification and hence the ability to use the commitments as intended. (Note that the partial verification algorithms uses only $\tilde{r}$ and ignores the other components of the opening information).

Analogously, creators of confidential transactions use zero-knowledge proofs to prove that the committed value (with is only defined with respect to a verification algorithm) is in range. Using our formulation, it is clear that it suffices for the creator to prove knowledge of $m$ and $\tilde{r}$ such that $\tilde{c} = g^m h^{\tilde{r}}$ and $0 \leq m < d$. In particular, a zero-knowledge proof with a statement involving the hash function is not necessary.

Similar remarks apply to the full verification algorithm: An alternative full verification algorithm could just recompute the entire commitment $\tilde{c}$ from $m$ and $r$. However, we state the full verification algorithm intentionally in the form above to capture that it suffices that the creator of the transaction reveals the ElGamal commitment $(c_1, c_2)$ and then proceeds with this ElGamal commitment, e.g., by adding other ElGamal commitments and opening the resulting sum commitment.

## 6.5.2 Security Analysis

We prove the security of our construction. For computationally hiding property, we assume a quantum-polynomial-time attacker and the quantum random oracle model

(QROM) [Bon+11], i.e., the attacker can query the random oracle in superposition of quantum states.

**Claim 6.3** (Computationally Hiding against Quantum Attackers). *The commitment algorithm of the construction of opt-in switch commitments is computationally hiding against quantum-polynomial-time attackers in the quantum random oracle model (without further computational assumptions).*

*Proof.* Since $g^r$ is unpredictable, the algorithmic one-way to hiding lemma [HHK17] implies that no quantum-polynomial-time attacker can distinguish

$$g^m h^{r+\mathsf{H}(g^m h^r, g^r)} \quad \text{and} \quad g^{m'} h^{r'+\mathsf{H}(g^{m'} h^{r'}, g^{r'})}$$

with non-negligible probability in the QROM.

The lemma (adopted from a lemma by Unruh [Unr14]) is the QROM analogue to the fact that an attacker that has not queried the classical random oracle $\mathsf{H}$ on $x$ has no information about $\mathsf{H}(x)$; we refer the reader to Hofheinz, Hövelmanns, and Kiltz [HHK17] for details. □

**Claim 6.4** (Partially Computationally Binding). *The partial verification algorithm is computationally binding (against classical attackers) under the discrete logarithm assumption in the random oracle model.*

*Proof.* The partial verification algorithm implements a Pedersen commitment $\tilde{c} = g^m h^{\tilde{r}}$ with $\tilde{r} = r + \mathsf{H}(g^m h^r, g^r)$ and thus is computationally binding [Ped91]. □

Having discussed quantum attackers and the QROM, we stress that we do not need to bother with quantum algorithms or the QROM when we prove that our scheme is everlastingly binding because we assume that the first part $\mathcal{A}_0$ of the attacker is classical and computationally bounded (before the switch), and the second part $\mathcal{A}_1$ of the attacker is anyway unbounded.

**Theorem 6.5** (Everlastingly Binding). *The construction of opt-in switch commitments is everlastingly binding under the inverse Diffie-Hellman assumption.*

*Proof.* Assume a computationally bounded attacker $\mathcal{A}_0$ outputs a commitment $g^m h^{\tilde{r}}$ with message $m$ and valid partial opening $\tilde{r}$, and an unbounded attacker $\mathcal{A}_1$ is able to output a message $m'$ and a valid full opening $r'$. Then by construction of the verification algorithms, $\tilde{c} = g^m h^{\tilde{r}} = c_1 \cdot h^{\mathsf{H}(c_1, c_2)}$ and $(c_1, c_2) = g^{m'} h^{r'}$ and thus

$$g^m h^{\tilde{r}} = g^{m'} h^{r'+\mathsf{H}(g^{m'} h^{r'}, g^{r'})}. \tag{6.1}$$

If $\mathcal{A}_0$ has queried $\mathsf{H}(g^{m'}h^{r'}, g^{r'})$, we can build a reduction against the inverse Diffie-Hellman assumption: The reduction runs $\mathcal{A}_0$ to obtain $m$, $\tilde{r}$, and $\mathsf{H}(g^{m'}h^{r'}, g^{r'})$ (from the random oracle queries), it guesses $m'$ and proceeds to compute $g^{x^{-1}}$ for $h = g^x$ as in the proof of Theorem 6.2.

If $\mathcal{A}_0$ has not queried $\mathsf{H}(g^{m'}h^{r'}, g^{r'})$, then a pair $(m', r')$ that fulfills (6.1) only exists with negligible probability: since the partial opening $(m, \tilde{r})$ is fixed by the output of $\mathcal{A}_0$ already, and the choice of $m'$ determines $r'$ fully, the only variable that $\mathcal{A}_1$ can choose is $m'$.

Due to the restriction of the message space, there is only a constant number of possibilities for $m'$. Furthermore, for every $m'$ there is only one possible random oracle output $\mathsf{H}(g^{m'}h^{r'}, g^{r'})$ that makes (6.1) true. The probability that the random oracle outputs this single value is negligible. □

### 6.5.3 Post-Quantum Hiding in Practice

A practical concern for the use of CT is that the payer needs to transmit the opening of the amount commitment to the payee; otherwise the payee who is the new owner of the funds cannot determine the received amount and cannot spend them in a further transaction. To solve this problem, implementations of CT typically rely on a non-interactive key Diffie-Hellman key exchange to establish a secure channel between the payer and the payee [Max15; Gib16]: the ephemeral public key of the payee is included in the address provided to the payer, and the ephemeral public key of the payer is stored together with the transaction on the blockchain. The payer uses the shared secret from the key exchange then to derive the randomness used for the commitment algorithm, and to store the amount in encrypted form in the transaction (or piggy-packed on the range proof [Max15; Gib16] to save space; this is not relevant for our discussion). The payee can recover the randomness and the amount from the key exchange analogously.

Clearly this scheme renders the post-quantum hiding property of opt-in switch commitments useless because a post-quantum attacker that can compute discrete logarithms can simply break the Diffie-Hellman key exchange and decrypt the transmitted amount, even if the commitment itself remains hiding. Breaking the key exchange requires the knowledge of the ephemeral public keys of both the payer and the payee of the transaction, but only the public key of the payer is stored on the blockchain, i.e., the attack is not possible for a weak attacker that only observes the blockchain and does not get to see the address of the payee. If this weak guarantee is insufficient in practice, the obvious defense is to rely on a post-quantum secure key exchange at the cost of possibly decreased efficiency.

# 7 Conclusions

*Bitcoin is great, but "if it's not private, it's not safe."*

— Edward Snowden [Sno17]

We have seen that advanced cryptography has great potential to improve the functionality and privacy of cryptocurrencies, and equally important, a careful design enables us to achieve these improvements in Bitcoin and other cryptocurrencies without the necessity for fundamental changes. Taking a step back, there are a few observations, which are relevant to the design of cryptocurrencies beyond the concrete results in this dissertation.

From a more general point of view, Chapter 6 teaches us an important lesson: when introducing new cryptographic features in cryptocurrencies today, we need to keep cryptographic agility and in particular post-quantum security in mind. Blockchains keep permanent state, and a transaction performed today creates state, e.g., an unspent transaction output, which may be still relevant in a few decades. While we do not need to know what specific cryptographic schemes we will deploy in the future, we need to make sure that such schemes can be deployed in a backward-compatible way. Relying on the expiration of money that has not been transacted for years should only be the last resort, simply because such expiration violates the principle that money should serve as a store of value.

With respect to the privacy, it is worth to emphasize that with techniques such as Confidential Transactions [Max15] enhanced with Bulletproofs [Bün+18] range proofs and Switch Commitments (Chapter 6), Stealth Addresses [Sab13], Taproot [Max18], and CoinJoin-based mixing with ValueShuffle (Chapter 5), we have a large arsenal of privacy-enhancing technologies at hand, whose combination constitutes a competitive alternative to other cryptographic approaches for privacy. I personally am confident that these proposals, which are designed in particular with Bitcoin in mind and address many different aspects of privacy, are ready to be evaluated and refined in practice, and that they provide a realistic path to improving the privacy of users.

# A Hardness Assumptions

For reference, we overview the computational hardness assumptions needed throughout this dissertation. In all of the following computational problems, $g$ is fixed and generates a multiplicatively written cyclic group (implicit in $g$) of known order $p$, where $p$ is $\lambda$ bits long. We restrict ourselves to the case of prime $p$.

**Assumption 1** (Discrete Logarithm Assumption). *The* discrete logarithm assumption *holds with respect to $g$ if for a random $x \in \mathbb{Z}_p$ and for all ppt attackers $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr[\mathcal{A}(g^x) = x] \leq \mathsf{negl}(\lambda).$$

**Assumption 2** (Computational Diffie-Hellman Assumption). *The* computational Diffie-Hellman assumption *holds with respect to $g$ if for random $x, y \in \mathbb{Z}_p$ and for all ppt attackers $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr[\mathcal{A}(g^x, g^y) = g^{xy}] \leq \mathsf{negl}(\lambda).$$

**Assumption 3** (Inverse Diffie-Hellman Assumption). *The* inverse Diffie-Hellman assumption *holds with respect to $g$ if for a random $x \in \mathbb{Z}_p^*$ and its multiplicative inverse $x^{-1}$, and for all ppt attackers $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr\left[\mathcal{A}(g, g^x) = g^{x^{-1}}\right] \leq \mathsf{negl}(\lambda),$$

The inverse Diffie-Hellman assumption is known to be equivalent to the Computational Diffie-Hellman assumption [SS02, Theorem 6.4].

**Assumption 4** (Decisional Diffie-Hellman Assumption). *The* decisional Diffie-Hellman assumption *holds with respect to $g$ if for random $x, y, z \in \mathbb{Z}_p$ and for all ppt attackers $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr\left[\mathcal{A}(g^x, g^y, h) = b \;\middle|\; b \leftarrow \{0,1\}; h = \begin{cases} g^{xy} & \text{if } b = 0 \\ g^z & \text{if } b = 1 \end{cases}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

# Bibliography

[Accas] Tim Ruffing. Implementation of accountable assertion scheme. URL: https://github.com/real-or-random/accas/ (cit. on p. 38).

[ADA] "Cardano". URL: https://www.cardano.org/ (cit. on p. 3).

[AM04] Giuseppe Ateniese and Breno de Medeiros. "On the Key Exposure Problem in Chameleon Hashes". In: Security in Communication Networks (SCN) 2004. DOI: 10.1007/978-3-540-30598-9_12 (cit. on p. 30).

[AN.ON] "AN.ON". URL: https://anon.inf.tu-dresden.de/ (cit. on p. 53).

[And+13] Elli Androulaki, Ghassan O. Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. "Evaluating User Privacy in Bitcoin". In: Financial Cryptography and Data Security (FC) 2013 (cit. on pp. 2, 14, 51, 54, 89).

[And+14] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. "Secure Multiparty Computations on Bitcoin". In: Security and Privacy 2014. DOI: 10.1109/SP.2014.35 (cit. on pp. 3, 13, 20, 21).

[And17] Oleg Andreev. "Confidential Assets". 2017. URL: https://github.com/chain/chain/blob/confidential-spec/docs/protocol/specifications/ca.md#value-range-proof (cit. on p. 119).

[Ant17] Andreas M. Antonopoulos. "Mastering Bitcoin: Programming the Open Blockchain". 2nd edition. O'Reilly Media, 2017. ISBN: 978-1-491-95438-6. URL: https://github.com/bitcoinbook/bitcoinbook#mastering-bitcoin (cit. on p. 9).

[Bac+14] Michael Backes, Fabian Bendun, Ashish Choudhury, and Aniket Kate. "Asynchronous MPC with a Strict Honest Majority Using Non-equivocation". In: Principles of Distributed Computing (PODC) 2014. DOI: 10.1145/2611462.2611490 (cit. on pp. 19, 21, 46).

[Bac02] Adam Back. "Hashcash – A Denial of Service Counter-Measure". 2002. URL: http://www.hashcash.org/hashcash.pdf (cit. on pp. 11, 15).

[Bac13] Adam Back. Bitcoin development mailing list. July 5, 2013. URL: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-July/002857.html (cit. on p. 3).

[Bar+12] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. "Bitter to Better. How to Make Bitcoin a Better Currency". In: Financial Cryptography and Data Security (FC) 2012. DOI: 10.1007/978-3-642-32946-3_29 (cit. on pp. 2, 14, 51, 54, 83, 89).

[Bau+18] Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. "More Efficient Commitments from Structured Lattice Assumptions". In: Security and Cryptography for Networks (SCN) 2018. DOI: 10.1007/978-3-319-98113-0_20 (cit. on p. 115).

[BB89] Jurjen Bos and Bert den Boer. "Detection of Disrupters in the DC Protocol". In: EUROCRYPT 1989. DOI: 10.1007/3-540-46885-4_33 (cit. on pp. 53, 59).

[BCHSh] Josh Ellithorpe. "CashShuffle". Implementation of CoinShuffle [RMK14]. URL: https://cashshuffle.com/ (cit. on p. 54).

[BDG] "Bitcoin Developer Guide". URL: https://bitcoin.org/en/developer-guide (cit. on pp. 9, 23).

[Ben+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: Security and Privacy (S&P) 2014 (cit. on pp. 2, 3, 89, 93, 111).

[Ben+18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Scalable, Transparent, and Post-Quantum Secure Computational Integrity". 2018. IACR Cryptology ePrint Archive: 2018/046 (cit. on p. 118).

[BIP16] Gavin Andresen. "Pay to Script Hash". BIP 16. 2012. URL: https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki (cit. on p. 98).

[BIP32] Pieter Wuille. "Hierarchical Deterministic Wallets". BIP 32. 2012. URL: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki (cit. on p. 94).

[BIP65] Peter Todd. "OP_CHECKLOCKTIMEVERIFY". BIP 65. 2014. URL: https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki (cit. on pp. 20, 23, 25, 43).

[BIP141] Eric Lombrozo, Johnson Lau, and Pieter Wuille. “Segregated Witness (Consensus Layer). P2WPKH Nested in BIP16 P2SH”. BIP 141. 2015. URL: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki#p2wpkh-nested-in-bip16-p2sh (cit. on p. 98).

[Bis+14] George Bissias, A. Pinar Ozisik, Brian N. Levine, and Marc Liberatore. “Sybil-Resistant Mixing for Bitcoin”. In: Workshop on Privacy in the Electronic Society (WPES) 2014. DOI: 10.1145/2665943.2665955 (cit. on pp. 83, 89).

[Bitm17] Bitmain. “UAHF: A Contingency Plan Against UASF (BIP148)”. June 4, 2017. URL: https://blog.bitmain.com/en/uahf-contingency-plan-uasf-bip148/ (cit. on p. 17).

[BK14] Iddo Bentov and Ranjit Kumaresan. “How to Use Bitcoin to Design Fair Protocols”. In: CRYPTO 2014. DOI: 10.1007/978-3-662-44381-1_24 (cit. on pp. 3, 13, 20, 21).

[BKN17] Dan Boneh, Sam Kim, and Valeria Nikolaenko. “Lattice-Based DAPS and Generalizations: Self-Enforcement in Signature Schemes”. In: Applied Cryptography and Network Security (ACNS) 2017. DOI: 10.1007/978-3-319-61204-1_23 (cit. on p. 48).

[BL13] Foteini Baldimtsi and Anna Lysyanskaya. “Anonymous Credentials Light”. In: Computer and Communications Security (CCS) 2013. DOI: 10.1145/2508859.2516687 (cit. on p. 46).

[Blo70] Burton H. Bloom. “Space/Time Trade-offs in Hash Coding with Allowable Errors”. In: Communications of the ACM 13.7 (July 1970). DOI: 10.1145/362686.362692 (cit. on p. 34).

[BN06] Mihir Bellare and Gregory Neven. “Multi-Signatures in the Plain Public-Key Model and a General Forking Lemma”. In: Computer and Communications Security (CCS) 2006. DOI: 10.1145/1180405.1180453 (cit. on p. 80).

[Bon+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. “Random Oracles in a Quantum World”. In: ASIACRYPT 2011. DOI: 10.1007/978-3-642-25385-0_3 (cit. on p. 126).

[Bon+14] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A. Kroll, and Edward W. Felten. “Mixcoin: Anonymity for Bitcoin with Accountable Mixes”. In: Financial Cryptography and Data Security (FC) 2014 (cit. on pp. 82, 89).

[Bon+15] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. “SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies”. In: Security and Privacy (S&P) 2015 (cit. on p. 9).

[Bor+07] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. “Denial of Service or Denial of Security?” In: Computer and Communications Security (CCS) 2007. DOI: `10.1145/1315245.1315258` (cit. on pp. 53, 85).

[BPS17] Mihir Bellare, Bertram Poettering, and Douglas Stebila. “Deterring Certificate Subversion: Efficient Double-Authentication-Preventing Signatures”. In: Public-Key Cryptography (PKC) 2017. DOI: `10.1007/978-3-662-54388-7_5` (cit. on pp. 48, 49).

[BtcStats] BitcoinStats. URL: `http://bitcoinstats.com/network/propagation/` (cit. on p. 80).

[BtcWikiA] “Block Timestamp”. Bitcoin Wiki. URL: `https://en.bitcoin.it/w/index.php?title=Block_timestamp&oldid=51392` (cit. on p. 23).

[BtcWikiB] “Contracts”. Bitcoin Wiki. URL: `https://en.bitcoin.it/w/index.php?title=Contracts&oldid=50633` (cit. on pp. 20–22).

[BtcWikiC] “Mixing Services”. URL: `https://en.bitcoin.it/wiki/Category:Mixing_Services` (cit. on p. 82).

[Bün+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. “Bulletproofs: Short Proofs for Confidential Transactions and More”. In: Security and Privacy (S&P) 2018. DOI: `10.1109/SP.2018.00020` (cit. on pp. 92, 115, 118, 129).

[But13] Vitalik Buterin. “A Next-Generation Smart Contract and Decentralized Application Platform”. Ethereum Whitepaper. 2013. URL: `https://github.com/ethereum/wiki/wiki/White-Paper` (cit. on pp. 22, 46, 83).

[Byt11] ByteCoin. “Untraceable Transactions Which Can Contain a Secure Message Are Inevitable”. Bitcoin forum. Apr. 17, 2011. URL: `https://bitcointalk.org/index.php?topic=5965.0` (cit. on pp. 89, 94).

[Car+16] Miles Carlsten, Harry A. Kalodner, S. Matthew Weinberg, and Arvind Narayanan. “On the Instability of Bitcoin Without the Block Reward”. In: Computer and Communications Security (CCS) 2016. DOI: `10.1145/2976749.2978408` (cit. on p. 16).

[CBM15] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. "Riposte: An Anonymous Messaging System Handling Millions of Users". In: Security and Privacy (S&P) 2015. DOI: 10.1109/SP.2015.27 (cit. on p. 59).

[CF10] Henry Corrigan-Gibbs and Bryan Ford. "Dissent: Accountable Anonymous Group Messaging". In: Computer and Communications Security (CCS) 2010. DOI: 10.1145/1866307.1866346 (cit. on pp. 6, 52–55, 61, 77, 85–87).

[CFN88] David Chaum, Amos Fiat, and Moni Naor. "Untraceable Electronic Cash". In: CRYPTO 1988. DOI: 10.1007/0-387-34799-2_25 (cit. on pp. 10, 46).

[Cha] Chainalyis, Inc. URL: https://www.chainalysis.com/ (cit. on p. 2).

[Cha81] David Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". In: Communications of the ACM 24.2 (Feb. 1981). DOI: 10.1145/358549.358563 (cit. on p. 53).

[Cha82] David Chaum. "Blind Signatures for Untraceable Payments". In: CRYPTO 1982. DOI: 10.1007/978-1-4757-0602-4_18 (cit. on pp. 10, 83).

[Cha88] David Chaum. "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability". In: Journal of Cryptology 1.1 (1988). DOI: 10.1007/BF00206326 (cit. on pp. 5, 52, 53, 59, 70).

[CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. "Compact E-Cash". In: EUROCRYPT 2005. DOI: 10.1007/11426639_18 (cit. on p. 46).

[Chu+07] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. "Attested Append-only Memory: Making Adversaries Stick to Their Word". In: Symposium on Operating Systems Principles (SOSP) 2007. DOI: 10.1145/1294261.1294280 (cit. on pp. 19, 21, 46).

[CKS09] David Cash, Eike Kiltz, and Victor Shoup. "The Twin Diffie-Hellman Problem and Applications". In: Journal of Cryptology 22.4 (2009). DOI: 10.1007/s00145-009-9041-6 (cit. on pp. 62, 100).

[CL01] Jan Camenisch and Anna Lysyanskaya. "An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation". In: EUROCRYPT 2001. DOI: 10.1007/3-540-44987-6_7 (cit. on p. 46).

[CL02] Miguel Castro and Barbara Liskov. “Practical Byzantine Fault Tolerance and Proactive Recovery”. In: ACM Trans. Comput. Syst. 20.4 (2002). DOI: 10.1145/571637.571640 (cit. on p. 10).

[Cle+12] Allen Clement, Flavio Junqueira, Aniket Kate, and Rodrigo Rodrigues. “On the (Limited) Power of Non-Equivocation”. In: Principles of Distributed Computing (PODC) 2012. DOI: 10.1145/2332432.2332490 (cit. on pp. 19, 21, 46).

[CM17] Nicolas T. Courtois and Rebekah Mercer. “Stealth Address and Key Management Techniques in Blockchain Systems”. In: International Conference on Information Systems Security and Privacy (ICISSP) 2017. DOI: 10.5220/0006270005590566 (cit. on pp. 89, 94).

[Coi] CoinMarketCap. URL: https://coinmarketcap.com/ (visited on 2018-04-19) (cit. on p. 1).

[Con+18] Mauro Conti, Sandeep Kumar E, Chhagan Lal, and Sushmita Ruj. “A Survey on Security and Privacy Issues of Bitcoin”. In: IEEE Communications Surveys and Tutorials 20.4 (2018). DOI: 10.1109/COMST.2018.2842460 (cit. on p. 9).

[Core15] Bitcoin Core. Version 0.11.0 release announcement. July 12, 2015. URL: https://bitcoin.org/en/release/v0.11.0 (cit. on p. 110).

[Core17] Bitcoin Core. “Technology Roadmap - Schnorr Signatures and Signature Aggregation”. Mar. 23, 2017. URL: https://bitcoincore.org/en/2017/03/23/schnorr-signature-aggregation/ (cit. on pp. 13, 26, 80, 92).

[CSS07] Christian Cachin, Abhi Shelat, and Alexander Shraer. “Efficient Fork-Linearizable Access to Untrusted Shared Memory”. In: Principles of Distributed Computing (PODC) 2007. DOI: 10.1145/1281100.1281121 (cit. on p. 42).

[CZK04] Xiaofeng Chen, Fangguo Zhang, and Kwangjo Kim. “Chameleon Hashing Without Key Exposure”. In: Information Security Conference (ISC) 2004. DOI: 10.1007/978-3-540-30144-8_8 (cit. on p. 30).

[Dai98] Wei Dai. Notes on b-money. Nov. 1, 1998. URL: http://www.weidai.com/bmoney.txt (cit. on p. 11).

[DFH18] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. “General State Channel Networks”. In: Computer and Communications Security (CCS) 2018. DOI: 10.1145/3243734.3243856 (cit. on p. 25).

[DK13] Larry A. Dunning and Ray Kresman. "Privacy Preserving Data Sharing With Anonymous ID Assignment". In: Transactions on Information Forensics and Security 8.2 (2013). DOI: 10.1109/TIFS.2012.2235831 (cit. on p. 59).

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. "Tor: The Second-Generation Onion Router". In: USENIX Security 2004. URL: https://www.usenix.org/conference/13th-usenix-security-symposium/tor-second-generation-onion-router (cit. on pp. 53, 81, 83, 92).

[Dod+08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam D. Smith. "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data". In: SIAM Journal on Computing 38.1 (2008). DOI: 10.1137/060651380 (cit. on p. 60).

[Dou02] John R. Douceur. "The Sybil Attack". In: Workshop on Peer-to-Peer Systems (IPTPS) 2002. DOI: 10.1007/3-540-45748-8_24 (cit. on p. 10).

[DRS] David Derler, Sebastian Ramacher, and Daniel Slamanig. "Short Double- and N-Times-Authentication-Preventing Signatures from ECDSA and More". In: European Symposium on Security and Privacy (EuroS&P). DOI: 10.1109/EuroSP.2018.00027 (cit. on p. 49).

[DRS90] Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. "Early Stopping in Byzantine Agreement". In: Journal of the ACM 37.4 (1990). DOI: 10.1145/96559.96565 (cit. on p. 55).

[DW13] Christian Decker and Roger Wattenhofer. "Information Propagation in the Bitcoin Network". In: Peer-to-Peer Computing (P2P) 2013. DOI: 10.1109/P2P.2013.6688704 (cit. on pp. 24, 80).

[DW15] Christian Decker and Roger Wattenhofer. "A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels". In: Self-Stabilizing Systems (SSS) 2015. DOI: 10.1007/978-3-319-21741-3_1 (cit. on pp. 2, 25).

[Dzi+19] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. "Perun: Virtual Payment Hubs over Cryptocurrencies". In: Security and Privacy (S&P) 2019. DOI: 10.1109/SP.2019.00020 (cit. on pp. 2, 25).

[Elements] Blockstream. "Elements". URL: https://elementsproject.org/ (cit. on p. 114).

[Elg84] Taher Elgamal. “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms”. In: CRYPTO 1984 (cit. on pp. 120, 123).

[Ell] Elliptic Enterprises Limited. URL: https://www.elliptic.co/ (cit. on p. 2).

[ES14] Ittay Eyal and Emin Gün Sirer. “Majority Is Not Enough: Bitcoin Mining Is Vulnerable”. In: Financial Cryptography (FC) 2014. DOI: 10.1007/978-3-662-45472-5_28 (cit. on p. 16).

[Fah+14] Sascha Fahl, Sergej Dechand, Henning Perl, Felix Fischer, Jaromir Smrcek, and Matthew Smith. “Hey, NSA: Stay Away from My Market! Future Proofing App Markets Against Powerful Attackers”. In: Computer and Communications Security (CCS) 2014. DOI: 10.1145/2660267.2660311 (cit. on pp. 19–21, 42).

[Fel+10] Ariel J. Feldman, William P. Zeller, Michael J. Freedman, and Edward W. Felten. “SPORC: Group Collaboration Using Untrusted Cloud Resources”. In: Operating Systems Design and Implementation (OSDI) 2010. URL: https://www.usenix.org/conference/osdi10/sporc-group-collaboration-using-untrusted-cloud-resources (cit. on pp. 19–21, 42).

[Fel+12] Ariel J. Feldman, Aaron Blankstein, Michael J. Freedman, and Edward W. Felten. “Social Networking with Frientegrity: Privacy and Integrity with an Untrusted Provider”. In: USENIX Security 2012. URL: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/feldman (cit. on pp. 19–21, 42).

[FG14] Christian Franck and Jeroen van de Graaf. “Dining Cryptographers are Practical”. 2014. arXiv: 1402.2269 [cs.CR] (cit. on p. 53).

[Fic18] Ádám Ficsór. “Mixing Unequal Inputs”. 2018. URL: https://github.com/nopara73/ZeroLink/issues/74 (cit. on p. 84).

[Fin04] Hal Finney. “RPOW - Reusable Proofs of Work”. Hashcash mailing list. Aug. 14, 2004. URL: http://hashcash.freelists.narkive.com/2WmeIozL/rpow-reusable-proofs-of-work (cit. on p. 11).

[Fin11] Hal Finney. “Re: Best Practice for Fast Transaction Acceptance – How High is the Risk?” Bitcoin forum. Feb. 13, 2011. URL: https://bitcointalk.org/index.php?topic=3441.msg48384#msg48384 (cit. on p. 41).

[FM00] Matthias Fitzi and Ueli M. Maurer. “From Partial Consistency to Global Broadcast”. In: Symposium on Theory of Computing (STOC) 2000. DOI : 10.1145/335305.335363 (cit. on p. 19).

[Fra14] Christian Franck. “Dining Cryptographers with 0.924 Verifiable Collision Resolution”. 2014. arXiv: 1402.1732 [cs.CR] (cit. on pp. 53, 59).

[Fre+13] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. “Non-Interactive Key Exchange”. In: Public-Key Cryptography 2013. DOI : 10.1007/978-3-642-36362-7_17 (cit. on pp. 62, 100).

[FT17] Ádám Ficsór and TDevD. “ZeroLink: The Bitcoin Fungibility Framework”. 2017. URL : https://github.com/nopara73/ZeroLink (cit. on p. 83).

[FWB15] Martin Florian, Johannes Walter, and Ingmar Baumgart. “Sybil-Resistant Pseudonymization and Pseudonym Change Without Trusted Third Parties”. In: Workshop on Privacy in the Electronic Society (WPES) 2015. DOI : 10.1145/2808138.2808145 (cit. on p. 56).

[GH12] Ilja Gerhardt and Timo Hanke. “Homomorphic Payment Addresses and the Pay-to-Contract Protocol”. 2012. arXiv: 1212.3257 [cs.CR] (cit. on p. 98).

[Gib16] Adam Gibson. “An Investigation into Confidential Transactions”. 2016. URL : https://github.com/AdamISZ/ConfidentialTransactionsDoc/raw/master/essayonCT.pdf (cit. on pp. 93, 101, 113, 127).

[GJ04] Philippe Golle and Ari Juels. “Dining Cryptographers Revisited”. In: EUROCRYPT 2004. DOI : 10.1007/978-3-540-24676-3_27 (cit. on pp. 53–55, 59).

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. “The Bitcoin Backbone Protocol: Analysis and Applications”. In: EUROCRYPT 2015. DOI : 10.1007/978-3-662-46803-6_10 (cit. on p. 11).

[GKL17] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. “The Bitcoin Backbone Protocol with Chains of Variable Difficulty”. In: CRYPTO 2017. DOI : 10.1007/978-3-319-63688-7_10 (cit. on p. 11).

[GM17] Matthew Green and Ian Miers. “Bolt: Anonymous Payment Channels for Decentralized Currencies”. In: Computer and Communications Security (CCS) 2017. DOI : 10.1145/3133956.3134093 (cit. on p. 25).

[Goe+03] Sharad Goel, Mark Robson, Milo Polte, and Emin Gün Sirer. "Herbivore: A Scalable and Efficient Protocol for Anonymous Communication". Tech. rep. 2003-1890. Cornell University, 2003. URL: https://ecommons.cornell.edu/handle/1813/5606 (cit. on p. 59).

[Gou99] Henry W. Gould. "The Girard-Waring Power Sum Formulas for Symmetric Functions and Fibonacci Sequences". In: Fibonacci Quarterly 37.2 (1999). URL: http://www.fq.math.ca/Issues/37-2.pdf (cit. on p. 60).

[Grin] "Grin". URL: https://www.grin-tech.org/ (cit. on pp. 110, 114).

[GRS96] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. "Hiding Routing Information". In: Information Hiding 1996. DOI: 10.1007/3-540-61996-8_37 (cit. on p. 53).

[GW11] Craig Gentry and Daniel Wichs. "Separating Succinct Non-Interactive Arguments from All Falsifiable Assumptions". In: Symposium on Theory of Computing (STOC) 2011. DOI: 10.1145/1993636.1993651 (cit. on p. 111).

[Has11] hashcash. "Re: Blind Bitcoin Transfers". Bitcoin forum. July 2, 2011. URL: https://bitcointalk.org/index.php?topic=12751.20 (cit. on pp. 5, 51, 52, 56, 78, 89).

[Hei+15] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. "Eclipse Attacks on Bitcoin's Peer-to-Peer Network". In: USENIX Security 2015. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman (cit. on p. 12).

[Hei+17] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. "TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub". In: Network and Distributed System Security (NDSS) 2017. URL: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/tumblebit-untrusted-bitcoin-compatible-anonymous-payment-hub/ (cit. on pp. 25, 82, 89).

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. "A Modular Analysis of the Fujisaki-Okamoto Transformation". In: Theory of Cryptography Conference (TCC) 2017. DOI: 10.1007/978-3-319-70500-2_12 (cit. on p. 126).

[HJP15] William Hart, Frederik Johansson, and Sebastian Pancratz. "FLINT: Fast Library for Number Theory". Version 2.5.2. 2015. URL: http://flintlib.org (cit. on p. 77).

[HN06] Danny Harnik and Moni Naor. "On Everlasting Security in the Hybrid Bounded Storage Model". In: International Colloquium on Automata, Languages and Programming (ICALP) 2006. DOI: 10.1007/11787006_17 (cit. on p. 121).

[Ho+08] Chi Ho, Robbert van Renesse, Mark Bickford, and Danny Dolev. "Nysiad: Practical Protocol Transformation to Tolerate Byzantine Failures". In: Networked Systems Design and Implementation (NSDI) 2008. URL: https://www.usenix.org/conference/nsdi-08/nysiad-practical-protocol-transformation-tolerate-byzantine-failures (cit. on p. 19).

[Jed16] Tom Elvis Jedusor. "Mimblewimble". https://scalingbitcoin.org/papers/mimblewimble.txt. 2016 (cit. on p. 110).

[JMV01] Don Johnson, Alfred Menezes, and Scott A. Vanstone. "The Elliptic Curve Digital Signature Algorithm (ECDSA)". In: International Journal of Information Security 1.1 (2001). DOI: 10.1007/s102070100002 (cit. on pp. 13, 26).

[KB14] Ranjit Kumaresan and Iddo Bentov. "How to Use Bitcoin to Incentivize Correct Computations". In: Computer and Communications Security (CCS) 2014. DOI: 10.1145/2660267.2660380 (cit. on pp. 20, 21).

[KG17] Rami Khalil and Arthur Gervais. "Revive: Rebalancing Off-Blockchain Payment Networks". In: Computer and Communications Security (CCS) 2017. DOI: 10.1145/3133956.3134033 (cit. on pp. 2, 25).

[KKM14] Philip Koshy, Diana Koshy, and Patrick McDaniel. "An Analysis of Anonymity in Bitcoin Using P2P Network Traffic". In: Financial Cryptography (FC) 2014. DOI: 10.1007/978-3-662-45472-5_30 (cit. on pp. 2, 14, 51, 54, 89).

[KMB15] Ranjit Kumaresan, Tal Moran, and Iddo Bentov. "How to Use Bitcoin to Play Decentralized Poker". In: Computer and Communications Security (CCS) 2015. DOI: 10.1145/2810103.2813712 (cit. on p. 3).

[KNS16] Anna Krasnova, Moritz Neikes, and Peter Schwabe. "Footprint Scheduling for Dining-Cryptographer Networks". In: Financial Cryptography (FC) 2016. DOI: 10.1007/978-3-662-54970-4_23 (cit. on p. 59).

[Koc+12] Stéphan Kochen, Alexey Sokolov, Kyle Fuller, and James Wheare. "IRCv3.2 `server-time` Extension". IRCv3 Working Group. 2012. URL: https://ircv3.net/specs/extensions/server-time-3.2.html (cit. on p. 55).

[Kos+16] Ahmed E. Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. "Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts". In: Security and Privacy (S&P) 2016. DOI: 10.1109/SP.2016.55 (cit. on pp. 2, 22).

[KR00] Hugo Krawczyk and Tal Rabin. "Chameleon Signatures". In: Network and Distributed System Security (NDSS) 2000. URL: https://www.ndss-symposium.org/ndss2000/chameleon-signatures/ (cit. on pp. 20, 30, 31, 38).

[Kru+16] Johannes Krupp, Dominique Schröder, Mark Simkin, Dario Fiore, Giuseppe Ateniese, and Stefan Nürnberger. "Nearly Optimal Verifiable Data Streaming". In: Public-Key Cryptography (PKC) 2016. DOI: 10.1007/978-3-662-49384-7_16 (cit. on p. 30).

[KS97] Erich Kaltofen and Victor Shoup. "Fast Polynomial Factorization over High Algebraic Extensions of Finite Fields". In: International Symposium on Symbolic and Algebraic Computation (ISSAC) 1997. DOI: 10.1145/258726.258777 (cit. on pp. 76, 77).

[Kwo+17] Yujin Kwon, Dohyun Kim, Yunmok Son, Eugene Y. Vasserman, and Yongdae Kim. "Be Selfish and Avoid Dilemmas: Fork After Withholding (FAW) Attacks on Bitcoin". In: Computer and Communications Security (CCS) 2017. DOI: 10.1145/3133956.3134019 (cit. on p. 16).

[Ler13] Sergio Demian Lerner. "Re: Zerocoin: Anonymous Distributed E-Cash from Bitcoin". Bitcoin forum. Apr. 12, 2013. URL: https://bitcointalk.org/index.php?topic=175156.msg1823810#msg1823810 (cit. on p. 3).

[Lev+09] Dave Levin, John R. Douceur, Jacob R. Lorch, and Thomas Moscibroda. "TrInc: Small Trusted Hardware for Large Distributed Systems". In: Networked Systems Design and Implementation (NSDI) 2009. URL: https://www.usenix.org/conference/nsdi-09/trinc-small-trusted-hardware-large-distributed-systems (cit. on pp. 19, 21, 46).

[Li+17] Fei Li, Wei Gao, Gui-lin Wang, Ke-fei Chen, Dong-qing Xie, and Chunming Tang. "Double-Authentication-Preventing Signatures Revisited: New Definition and Construction from Chameleon Hash". Unclear publication status, accepted to "Frontiers of Information Technology and Electronic Engineering" according to journal website. 2017. URL: http://www.jzus.zju.edu.cn/iparticle.php?doi=10.1631/FITEE.1700005 (cit. on p. 49).

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. "The Byzantine Generals Problem". In: ACM Transactions on Programming Languages and Systems (TOPLAS) 4.3 (1982). DOI: 10.1145/357172.357176 (cit. on p. 10).

[Mal+17] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. "Concurrency and Privacy with Payment-Channel Networks". In: Computer and Communications Security (CCS) 2017. DOI: 10.1145/3133956.3134096 (cit. on pp. 2, 19, 25).

[Mau97] Ueli M. Maurer. "Information-Theoretically Secure Secret-Key Agreement by NOT Authenticated Public Discussion". In: EUROCRYPT 1997. DOI: 10.1007/3-540-69053-0_15 (cit. on p. 73).

[Max+19] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. "Simple Schnorr multi-signatures with applications to Bitcoin". In: Des. Codes Cryptogr. 87.9 (2019). DOI: 10.1007/s10623-019-00608-x (cit. on pp. 80, 92, 96, 101).

[Max13A] Gregory Maxwell. "CoinJoin: Bitcoin Privacy for the Real World". Bitcoin forum. Aug. 22, 2013. URL: https://bitcointalk.org/index.php?topic=279249 (cit. on pp. 5, 51, 52, 56, 78, 83, 89).

[Max13B] Gregory Maxwell. "CoinSwap: Transaction Graph Disjoint Trustless Trading". Bitcoin forum. Oct. 30, 2013. URL: https://bitcointalk.org/index.php?topic=321228.0 (cit. on p. 82).

[Max15] Gregory Maxwell. "Confidential Transactions". Technical notes. 2015. URL: https://people.xiph.org/~greg/confidential_values.txt (cit. on pp. 2, 6, 89, 93, 101, 113, 127, 129).

[Max18] Gregory Maxwell. "Taproot: Privacy Preserving Switchable Scripting". Bitcoin development mailing list. Jan. 23, 2018. URL: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2018-January/015614.html (cit. on pp. 98, 129).

[Mei+13] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. “A Fistful of Bitcoins: Characterizing Payments Among Men with No Names”. In: Internet Measurement Conference (IMC) 2013. DOI: 10.1145/2504730.2504747 (cit. on pp. 2, 14, 51, 54, 89).

[Mer87] Ralph C. Merkle. “A Digital Signature Based on a Conventional Encryption Function”. In: CRYPTO 1987. DOI: 10.1007/3-540-48184-2_32 (cit. on p. 15).

[MHG18] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. “Low-Resource Eclipse Attacks on Ethereum’s Peer-to-Peer Network”. 2018. IACR Cryptology ePrint Archive: 2018/236 (cit. on p. 12).

[Mie+13] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. “Zerocoin: Anonymous Distributed E-Cash from Bitcoin”. In: Security and Privacy (S&P) 2013. DOI: 10.1109/SP.2013.34 (cit. on pp. 2, 3, 89, 111).

[Mil+15] Andrew Miller, Ahmed E. Kosba, Jonathan Katz, and Elaine Shi. “Nonoutsourceable Scratch-Off Puzzles to Discourage Bitcoin Mining Coalitions”. In: Computer and Communications Security (CCS) 2015. DOI: 10.1145/2810103.2813621 (cit. on p. 16).

[Mil+19] Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. “Sprites and State Channels: Payment Networks that Go Faster than Lightning”. In: Financial Cryptography (FC) 2019. DOI: 10.1007/978-3-030-32101-7_30 (cit. on pp. 2, 25).

[MM17] Sarah Meiklejohn and Rebekah Mercer. “Möbius: Trustless Tumbling for Transaction Privacy”. In: Privacy-Enhancing Technologies (PoPETs) 2018.2. DOI: 10.1515/popets-2018-0015 (cit. on p. 83).

[MN06] Tal Moran and Moni Naor. “Receipt-Free Universally-Verifiable Voting with Everlasting Privacy”. In: CRYPTO 2006. DOI: 10.1007/11818175_22 (cit. on p. 121).

[MNF17] Felix Konstantin Maurer, Till Neudecker, and Martin Florian. “Anonymous CoinJoin Transactions with Arbitrary Values”. In: Trust, Security and Privacy in Computing and Communications (TrustCom) 2017. DOI: 10.1109/Trustcom/BigDataSE/ICESS.2017.280 (cit. on p. 84).

[MO15] Sarah Meiklejohn and Claudio Orlandi. “Privacy-Enhancing Overlays in Bitcoin”. In: Workshop on Bitcoin Research (BITCOIN) 2015. DOI: 10.1007/978-3-662-48051-9_10 (cit. on pp. 2, 14, 51, 54, 78, 80, 89).

[MRK17] Pedro Moreno-Sanchez, Tim Ruffing, and Aniket Kate. “PathShuffle: Credit Mixing and Anonymous Payments for Ripple”. In: Privacy-Enhancing Technologies (PoPETs) 2017.3. DOI: 10.1515/popets-2017-0031 (cit. on p. 52).

[MS02] David Mazières and Dennis Shasha. “Building Secure File Systems out of Byzantine Storage”. In: Principles of Distributed Computing (PODC) 2002. DOI: 10.1145/571825.571840 (cit. on pp. 19–21, 42).

[MZK16] Pedro Moreno-Sanchez, Muhammad Bilal Zafar, and Aniket Kate. “Linking Wallets and Deanonymizing Transactions in the Ripple Network”. In: Privacy-Enhancing Technologies (PoPETs) 2016.4. DOI: 10.1515/popets-2016-0049 (cit. on p. 52).

[NAH16] Till Neudecker, Philipp Andelfinger, and Hannes Hartenstein. “Timing Analysis for Inferring the Topology of the Bitcoin Peer-to-Peer Network”. In: Advanced and Trusted Computing (ATC) 2016. Bitcoin Network Monitor available at https://dsn.tm.kit.edu/bitcoin/. DOI: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0070 (cit. on p. 80).

[Nak08A] Satoshi Nakamoto. “Bitcoin P2P E-Cash Paper”. Cryptography mailing list. Oct. 31, 2008. URL: http://www.metzdowd.com/pipermail/cryptography/2008-October/014810.html (cit. on p. 1).

[Nak08B] Satoshi Nakamoto. “Bitcoin: A Peer-to-Peer Electronic Cash System”. 2008. URL: https://bitcoin.org/bitcoin.pdf (cit. on p. 1).

[Nar+16] Arvind Narayanan, Joseph Bonneau, Edward W. Felten, Andrew Miller, and Steven Goldfeder. “Bitcoin and Cryptocurrency Technologies - A Comprehensive Introduction”. Princeton University Press, 2016. ISBN: 978-0-691-17169-2. URL: https://press.princeton.edu/titles/10908.html (cit. on p. 9).

[Nau+19] Gleb Naumenko, Gregory Maxwell, Pieter Wuille, Alexandra Fedorova, and Ivan Beschastnikh. “Erlay: Efficient Transaction Relay for Bitcoin”. In: Computer and Communications Security (CCS) 2019. DOI: 10.1145/3319535.3354237 (cit. on p. 60).

[NC10] Michael A. Nielsen and Isaac L. Chuang. “Quantum Computation and Quantum Information”. 2nd edition. Cambridge University Press, 2010. ISBN: 978-1-107-00217-3. DOI: 10.1017/CBO9780511976667 (cit. on p. 124).

[NC17] Arvind Narayanan and Jeremy Clark. “Bitcoin’s Academic Pedigree”. In: Communications of the ACM 60.12 (Dec. 2017). DOI: 10.1145/3132259 (cit. on p. 9).

[Nic15] Jonas Nick. “Data-Driven De-Anonymization in Bitcoin”. Master’s Thesis. ETH Zürich, 2015. DOI: 10.3929/ethz-a-010541254 (cit. on pp. 2, 14, 51, 89).

[Nic19] Jonas Nick. “Insecure Shortcuts in MuSig”. Nov. 19, 2019. URL: https://medium.com/blockstream/insecure-shortcuts-in-musig-2ad0d38a97da (cit. on p. 81).

[NMM16] Shen Noether, Adam Mackenzie, and The Monero Research Lab. “Ring Confidential Transactions”. In: Ledger 1 (2016). DOI: 10.5195/ledger.2016.34 (cit. on pp. 2, 3, 110, 114).

[NxtSh] Description of CoinShuffle implementation in Nxt. URL: https://nxtplatform.org/what-is-nxt/coin-shuffling/ (cit. on p. 54).

[PARI] “PARI/GP”. The PARI Group. University of Bordeaux. URL: https://pari.math.u-bordeaux.fr/ (cit. on p. 76).

[PARI-Py] klinck. PARI-Python interface. URL: https://code.google.com/archive/p/pari-python/ (cit. on p. 76).

[PD] Joseph Poon and Thaddeus Dryja. “The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments”. Technical Report (draft). URL: https://lightning.network/ (cit. on pp. 2, 19, 25).

[Ped91] Torben P. Pedersen. “Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing”. In: CRYPTO 1991. DOI: 10.1007/3-540-46766-1_9 (cit. on pp. 101, 119, 120, 123, 126).

[Pin+17] Rafaël del Pino, Vadim Lyubashevsky, Gregory Neven, and Gregor Seiler. “Practical Quantum-Safe Voting from Lattices”. In: Computer and Communications Security (CCS) 2017. DOI: 10.1145/3133956.3134101 (cit. on p. 115).

[PJ72] William Wesley Peterson and Edward J. Weldon Jr. “Error-Correcting Codes”. 2nd edition. MIT Press, 1972. ISBN: 978-0-262-16039-1 (cit. on p. 60).

[Poe+18] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. “Confidential Assets”. In: Workshop on Bitcoin and Blockchain Research (BITCOIN) 2018. DOI: 10.1007/978-3-662-58820-8_4 (cit. on pp. 115, 118).

[Poe16] Andrew Poelstra. “Mimblewimble”. 2016. URL: https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf. More detailed than the notes by Jedusor [Jed16] (cit. on p. 110).

[Poe18] Bertram Poettering. “Shorter Double-Authentication Preventing Signatures for Small Address Spaces”. In: AFRICACRYPT 2018. DOI: 10.1007/978-3-319-89339-6_19 (cit. on p. 49).

[PRK98] Mohammad Peyravian, Allen Roginsky, and Ajay Kshemkalyani. “On Probabilities of Hash Value Matches”. In: Computers and Security 17.2 (1998). DOI: 10.1016/S0167-4048(97)82016-0 (cit. on p. 38).

[PS14] Bertram Poettering and Douglas Stebila. “Double-Authentication-Preventing Signatures”. In: European Symposium on Research in Computer Security (ESORICS) 2014. DOI: 10.1007/978-3-319-11203-9_25 (cit. on pp. 21, 29, 31, 47–49).

[PS17] Bertram Poettering and Douglas Stebila. “Double-Authentication-Preventing Signatures”. In: International Journal of Information Secucirty 16.1 (2017). Full version of [PS14]. DOI: 10.1007/s10207-015-0307-8 (cit. on pp. 21, 29, 31, 47–49).

[PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. “Analysis of the Blockchain Protocol in Asynchronous Networks”. In: EUROCRYPT 2017. DOI: 10.1007/978-3-319-56614-6_22 (cit. on p. 11).

[RH11] Fergal Reid and Martin Harrigan. “An Analysis of Anonymity in the Bitcoin System”. In: Workshop on Security and Privacy in Social Networks (SPSN) 2011. DOI: 10.1109/PASSAT/SocialCom.2011.79 (cit. on pp. 2, 14, 51, 54, 89).

[RKS15] Tim Ruffing, Aniket Kate, and Dominique Schröder. “Liar, Liar, Coins on Fire! – Penalizing Equivocation By Loss of Bitcoins”. In: Computer and Communications Security (CCS) 2015. DOI: 10.1145/2810103.2813686 (cit. on pp. x, 48).

[RMK14] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. “CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin”. In: European Symposium on Research in Computer Security (ESORICS) 2014. DOI: 10.1007/978-3-319-11212-1_20 (cit. on pp. 51, 52, 54, 56, 61, 77, 134).

[Rog15] Phillip Rogaway. “The Moral Character of Cryptographic Work”. Paper corresponding to an IACR Distinguished Lecture given at ASIACRYPT 2015. IACR Cryptology ePrint Archive: 2015/1162 (cit. on p. ix).

[Ros14] Meni Rosenfeld. "Analysis of Hashrate-Based Double Spending". 2014. arXiv: 1402.2009 [cs.CR] (cit. on p. 41).

[Sab13] Nicolas van Saberhagen. "CryptoNote v 2.0". 2013. URL: https://cryptonote.org/whitepaper.pdf (cit. on pp. 2, 3, 6, 89, 94, 110, 129).

[Sch91] Claus-Peter Schnorr. "Efficient Signature Generation by Smart Cards". In: Journal of Cryptology 4.3 (1991). DOI: 10.1007/BF00196725 (cit. on pp. 13, 26, 92).

[SEC1] Certicom Research. "Elliptic Curve Cryptography". SEC 1. Version 2.0. Standards for Efficient Cryptography Group, 2009. URL: http://www.secg.org/sec1-v2.pdf (cit. on p. 113).

[SEC2] Certicom Research. "Recommended Elliptic Curve Domain Parameters". SEC 2. Version 2.0. Standards for Efficient Cryptography Group, 2010. URL: http://www.secg.org/sec2-v2.pdf (cit. on pp. 13, 38, 120).

[SMZ14] Michele Spagnuolo, Federico Maggi, and Stefano Zanero. "BitIodine: Extracting Intelligence from the Bitcoin Network". In: Financial Cryptography (FC) 2014. DOI: 10.1007/978-3-662-45472-5_29 (cit. on pp. 2, 14, 51, 54, 89).

[Sno17] Edward Snowden. Twitter. Sept. 28, 2017. URL: https://twitter.com/Snowden/status/913544739542241282 (cit. on p. 129).

[Son11] Shinan Song. "Why I Left Sina Weibo". Chinese. July 14, 2011. URL: http://songshinan.blog.caixin.com/archives/22322 (cit. on p. 42).

[Spi13] Jeremy Spilmann. "Re: Anti DoS For Tx Replacement". Bitcoin development mailing list. Apr. 19, 2013. URL: https://www.mail-archive.com/bitcoin-development%40lists.sourceforge.net/msg02028.html (cit. on pp. 2, 19, 20, 25, 43).

[SS01] Ahmad-Reza Sadeghi and Michael Steiner. "Assumptions Related to Discrete Logarithms: Why Subtleties Make a Real Difference". In: EUROCRYPT 2001.

[SS02] Ahmad-Reza Sadeghi and Michael Steiner. "Assumptions Related to Discrete Logarithms: Why Subtleties Make a Real Difference". 2002. URL: http://www.semper.org/sirene/publ/SaSt_01.dh-et-al.long.pdf. Full version of [SS01] (cit. on p. 131).

[SS12] Dominique Schröder and Heike Schröder. "Verifiable Data Streaming". In: Computer and Communications Security (CCS) 2012. DOI: 10.1145/2382196.2382297 (cit. on p. 30).

[SS15] Dominique Schröder and Mark Simkin. "VeriStream – A Framework for Verifiable Data Streaming". In: Financial Cryptography (FC) 2015. DOI: 10.1007/978-3-662-47854-7_34 (cit. on p. 30).

[ST87] T. K. Srikanth and Sam Toueg. "Simulating Authenticated Broadcasts to Derive Simple Fault-Tolerant Algorithms". In: Distributed Computing 2.2 (1987). DOI: 10.1007/BF01667080 (cit. on pp. 55, 74).

[Sun14] sundance. "Byzantine Cycle Mode: Scalable Bitcoin Mixing on Unequal Inputs". 2014. URL: https://github.com/sundance30203/dubmix/raw/master/doc/bcm.pdf (cit. on p. 84).

[Syt+14] Ewa Syta, Henry Corrigan-Gibbs, Shu-Chun Weng, David Wolinsky, Bryan Ford, and Aaron Johnson. "Security Analysis of Accountable Anonymity in Dissent". In: Transactions on Information and System Security (TISSEC) 17.1 (2014). DOI: 10.1145/2629621 (cit. on pp. 6, 52–54, 61, 85, 86).

[Sza08] Nick Szabo. "Bit gold". Dec. 27, 2008. URL: https://unenumerated.blogspot.de/2005/12/bit-gold.html (cit. on p. 11).

[Sza97] Nick Szabo. "Formalizing and Securing Relationships on Public Networks". In: First Monday 2.9 (1997). DOI: 10.5210/fm.v2i9.548 (cit. on p. 13).

[TFEU] Council of the European Union. "Treaty of the Functioning of the European Union. Article 128(1)". In: Official Journal of the European Union (Oct. 26, 2012). URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.C_.2012.326.01.0001.01.ENG#d1e3385-47-1 (cit. on p. 10).

[Tod14A] Peter Todd. "Near-Zero Fee Transactions with Hub-and-Spoke Micropayments". Bitcoin development mailing list. Dec. 12, 2014. URL: https://www.mail-archive.com/bitcoin-development@lists.sourceforge.net/msg06576.html (cit. on p. 19).

[Tod14B] Peter Todd. "Stealth Addresses". Bitcoin development mailing list. Jan. 6, 2014. URL: https://www.mail-archive.com/bitcoin-development@lists.sourceforge.net/msg03613.html (cit. on pp. 89, 94).

[Tor] The Tor Project. "Tor". URL: https://www.torproject.org/ (cit. on pp. 81, 83, 92).

[TRL12] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz. "Theory and Practice of Bloom Filters for Distributed Systems". In: IEEE Communications Surveys and Tutorials 14.1 (2012). DOI: 10.1109/SURV.2011.031611.00024 (cit. on p. 34).

[TS16] Florian Tschorsch and Björn Scheuermann. "Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies". In: IEEE Communications Surveys and Tutorials 18.3 (2016). DOI: 10.1109/COMST.2016.2535718 (cit. on p. 9).

[Unr13] Dominique Unruh. "Everlasting Multi-Party Computation". In: CRYPTO 2013. DOI: 10.1007/978-3-642-40084-1_22 (cit. on p. 121).

[Unr14] Dominique Unruh. "Revocable Quantum Timed-Release Encryption". In: EUROCRYPT 2014. DOI: 10.1007/978-3-642-55220-5_8 (cit. on p. 126).

[VR15] Luke Valenta and Brendan Rowan. "Blindcoin: Blinded, Accountable Mixes for Bitcoin". In: Workshop on Bitcoin Research (BITCOIN) 2015. DOI: 10.1007/978-3-662-48051-9_9 (cit. on pp. 82, 89).

[Was18] zkSNACKs. "Wasabi Wallet". 2018. URL: https://wasabiwallet.io/ (cit. on p. 83).

[Whi+02] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. "An Integrated Experimental Environment for Distributed Systems and Networks". In: Operating System Design and Implementation (OSDI) 2002. URL: http://www.usenix.org/events/osdi02/tech/white.html (cit. on pp. 52, 76).

[Wir16] Aaron van Wirdum. "Rejecting Today's Hard Fork, the Ethereum Classic Project Continues on the Original Chain: Here's Why". Bitcoin Magazine. July 20, 2016. URL: https://bitcoinmagazine.com/articles/rejecting-today-s-hard-fork-the-ethereum-classic-project-continues-on-the-original-chain-here-s-why-1469038808/ (cit. on p. 17).

[WMN18] Pieter Wuille, Gregory Maxwell, and Gleb Naumenko. "Minisketch: A Library for BCH-based Set Reconciliation". 2018. URL: https://github.com/sipa/minisketch (cit. on p. 60).

[Wol+12] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. “Dissent in Numbers: Making Strong Anonymity Scale”. In: Operating Systems Design and Implementation (OSDI) 2012. URL: https://www.usenix.org/conference/osdi12/technical-sessions/presentation/wolinsky (cit. on p. 85).

[WSF13] David Isaac Wolinsky, Ewa Syta, and Bryan Ford. “Hang with Your Buddies to Resist Intersection Attacks”. In: Computer and Communications Security (CCS) 2013. DOI: 10.1145/2508859.2516740 (cit. on pp. 53, 85).

[Wui13] Pieter Wuille. “libsecp256k1: Optimized C Library for EC Operations on Curve secp256k1”. 2013. URL: https://github.com/bitcoin/secp256k1 (cit. on p. 38).

[XMR] “Monero”. URL: https://monero.org/ (cit. on pp. 3, 92, 93, 114).

[XXW13] Xiang Xie, Rui Xue, and Minqian Wang. “Zero Knowledge Proofs from Ring-LWE”. In: Cryptology and Network Security (CANS) 2013. DOI: 10.1007/978-3-319-02937-5_4 (cit. on p. 115).

[ZEC] Electric Coin Company. “Zcash”. URL: https://z.cash/ (cit. on p. 3).

[Zie+15] Jan Henrik Ziegeldorf, Fred Grossmann, Martin Henze, Nicolas Inden, and Klaus Wehrle. “CoinParty: Secure Multi-Party Mixing of Bitcoins”. In: Conference on Data and Application Security and Privacy (CODASPY) 2015. DOI: 10.1145/2699026.2699100 (cit. on pp. 54, 56, 83, 89).