Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science

Rational Cryptography: Novel Constructions, Automated Verification and Unified Definitions

Oana-Mădălina Ciobotaru

Dissertation
zur Erlangung des Grades
des Doktors der Naturwissenschaften (Dr. rer. nat.)
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes

December 2012
Saarbrücken

Thesis for obtaining the title of Doctor of Natural Sciences of the Faculties of Natural Sciences and Technology of Saarland University.

Dissertation zur Erlangung des Grades des Doktors der Naturwissenschaften der Naturwissenschaftlich-Technischen Fakultäten der Universität des Saarlandes.

Reporters / Berichterstatter:
Prof. Dr. Michael Backes (Saarland University and MPI-SWS)
Dr. Matteo Maffei (Saarland University)

Examination Board / Prüfungausschuss:
Prof. Dr. Raimund Seidel
Prof. Dr. Michael Backes
Dr. Matteo Maffei
Dr. Dario Fiore

Dean / Dekan:
Prof. Dr. Mark Groves

Date of Colloquium / Tag des Kolloquiums:
March 26, 2013. / 26. März, 2013.

Version from May 20, 2013. / Textfassung von 20. Mai, 2013.

Statement

I hereby swear in lieu of an oath that I have independently prepared this thesis and without using other aids than those stated. The data and concepts taken over from other sources or taken over indirectly are indicated citing the source. The thesis was not submitted so far either in Germany or in another country in the same or a similar form in a procedure for obtaining an academic title.

Oana-Mădălina Ciobotaru
Saarbrücken, December 2012

Acknowledgements

I consider myself very fortunate for being able to work with two distinguished academic advisors: Michael Backes and Matteo Maffei. During my graduate studies, they have taught me many invaluable things for which I am profoundly grateful. With patience and support, Michael inspired me to always improve and to give my best while pursuing my research. He gave me lots of trust and freedom to find and choose my path for research, which built my own confidence and ultimately allowed me to succeed. Matteo shared with me, very kindly and patiently, many subtle research insights related to my work. With great positive energy and enthusiasm, he taught me how to pursue my research effectively, while never loosing attention to important details.

I am grateful to Dominique Unruh, for his guidance in the beginning of my graduate studies. His kindness and cheerfulness, his striking dedication to clarity of thought and his tactful ability to share it with the others, have inspired me and shaped my first view on research. I am indebted for his patience and for many wonderful in-depth research topics he shared with me.

For creating a great working environment and a relaxed and friendly atmosphere, I would like to thank my colleagues and friends at Saarland University: Fabian Bendun, Matthias Berg, Markus Dürmuth, Fabienne Eigner, Dario Fiore, Martin Gagné, Sebastian Gerling, Christian Hammer, Cătălin Hriţcu, Aniket Kate, Boris Köpf, Stefan Lorenz, Joachim Lutz, Sebastian Meiser, Esfandiar Mohammadi, Kim Pecina, Manuel Reinert, Raphael Reischuk, Philipp von Styp-Rekowsky, Dominique Schröder, Malte Skoruppa and Milivoj Simeonovski. It was a pleasure and a privilege to be part of such a group.

For patiently proofreading various drafts of my research, I am grateful to Fabian, Matthias, Fabienne, Sebastian Meiser, Esfandiar, Kim, Manuel and Dominique. I am particularly thankful to Fabienne, Kim and Manuel for their invaluable patience and witty support which helped me "tame" ProVerif. For many wonderful conversations, on many research topics and beyond, for always being a good listener and a good and true friend, I would like to thank Esfandiar. I am grateful to Aniket for his patience, his kindness and thoughtfulness in providing many invaluable advices and for pedagogically sharing his knowledge on many research topics. To Sebastian Gerling, I am grateful for being a wonderful friend, colleague and office mate. His contagious good mood, his kindness, his continuous perseverance and optimism never cease to inspire and motivate me to be a better person. A special thank you is due to Bettina Balthasar for all her help, kindness and promptness with any administrative or logistic matter, many times

on a very short notice.

Finally, I would like to thank my parents and my sister for always believing in me. Their endless love and support were essential for all my achievements. This thesis is dedicated to them.

Saarbrücken, 20 December 2012 Oana Ciobotaru

Abstract

Rational cryptography has recently emerged as a very promising field of research by combining notions and techniques from cryptography and game theory, because it offers an alternative to the rather inflexible traditional cryptographic model. In contrast to the classical view of cryptography where protocol participants are considered either honest or arbitrarily malicious, rational cryptography models participants as rational players that try to maximize their benefit and thus deviate from the protocol only if they gain an advantage by doing so.

The main research goals for rational cryptography are the design of more efficient protocols when players adhere to a rational model, the design and implementation of automated proofs for rational security notions and the study of the intrinsic connections between game theoretic and cryptographic notions. In this thesis, we address all these issues.

First we present the mathematical model and the design for a new rational file sharing protocol which we call RatFish. Next, we develop a general method for automated verification for rational cryptographic protocols and we show how to apply our technique in order to automatically derive the rational security property for RatFish. Finally, we study the intrinsic connections between game theory and cryptography by defining a new game theoretic notion, which we call game universal implementation, and by showing its equivalence with the notion of weak stand-alone security.

Zusammenfassung

Rationale Kryptographie ist kürzlich als ein vielversprechender Bereich der Forschung durch die Kombination von Begriffen und Techniken aus der Kryptographie und der Spieltheorie entstanden, weil es eine Alternative zu dem eher unflexiblen traditionellen kryptographischen Modell bietet. Im Gegensatz zur klassischen Ansicht der Kryptographie, nach der Protokollteilnehmer entweder als ehrlich oder willkürlich bösartig angesehen werden, modelliert rationale Kryptografie die Protokollteilnehmern als rationale Akteure, die versuchen ihren Vorteil zu maximieren und damit nur vom Protokoll abweichen, wenn sie dadurch einen Vorteil erlangen.

Die wichtigsten Forschungsziele rationaler Kryptographie sind: das Design effizienterer Protokolle, wenn die Spieler ein rationale Modell folgen, das Design und die Implementierung von automatisierten Beweisen rationaler Sicherheitsbegriffe und die Untersuchung der intrinsischen Verbindungen zwischen spieltheoretischen und kryptographischen Begriffen. In dieser Arbeit beschäftigen wir uns mit all diesen Fragen.

Zunächst präsentieren wir das mathematische Modell und das Design für RatFish, ein neues rationales Filesharing-Protokoll. Dann entwickeln wir eine allgemeine Methode zur automatischen Verifikation rationaler kryptographischer Protokolle und wir zeigen, wie man unsere Technik nutzen kann, um die rationale Sicherheitseigenschaft von RatFish automatisch abzuleiten. Abschließend, untersuchen wir die intrinsische Verbindungen zwischen Spieltheorie und Kryptographie durch die Definition von game universal implementation, einem neuen spieltheoretischen Begriff, und wir zeigen die Äquivalenz von game universal implementation und weak stand-alone security.

Contents

Chapter 1

Introduction

Protocol security is important to have, however it is neither easy to define, nor to implement. Traditionally, cryptographic protocols are designed to achieve security properties in a "black and white" adversarial model, in which parties are either honest as they unconditionally follow the protocol, or they are compromised and as a consequence they may arbitrarily misbehave in order to break the security of the protocol. Even though the use of this classical adversarial model provides strong security guarantees, it may also lead to protocols that are extremely complicated, or very inefficient, and in some cases, even impossible to design.

Rational cryptography has recently emerged as a promising field of research by combining notions and techniques from cryptography and game theory because it offers an alternative to the rather inflexible traditional cryptographic model. In contrast to the classical view of cryptography where protocol participants are considered either honest or arbitrarily malicious, rational cryptography models participants as rational players that try to maximize their benefit and thus deviate from the protocol only if they gain an advantage by doing so. In the context of game theory, a protocol that gives rational participants no incentives for deviation constitutes an equilibrium. Rational cryptography is centered around such adapted computational equilibria concepts [60] and uses cryptographic notions and techniques as tools for building secure protocols with respect to rational participants.

An initial research focus for rational cryptography has been on defining [30, 33, 74] computational models for polynomially bounded rational participants, which are also called rational players. Later, the research focus has shifted towards designing protocols that rational participants have no incentive to deviate from, i.e., the protocols should fulfill some game theoretic notion of equilibrium. So far, most of these rational protocols fall into the category of rational secret sharing and rational secure multi-party computation [2, 39, 37, 48, 53, 63].

Currently, there are three research goals for rational cryptography: the design of more efficient protocols for the above rational model [8, 39], the design and implementation of automated proofs for rational security notions [9] and the study of the intrinsic connections between game theoretic and cryptographic notions [29, 55]. In this thesis,

we address all these issues.

First, we apply the rational model to file sharing, a popular P2P application. Existing file sharing applications, such as BitTorrent, provide remarkable efficiency and scalability, as well as adaptability to dynamic situations. However, none of them is secure against attacks from rational users, i.e., users that deviate from the protocol if doing so increases their benefits. We propose a rigorous model of rational seeders and leechers and we design and implement RatFish [8], a rational file sharing protocol. The distinctive feature of RatFish, however, is not only that it discourages the use of several selfish strategies, but that it comes with a formal proof that deviating from the recommended protocol is irrational for both seeders and leechers. In order to do this, we first characterize rational behaviors of leechers and seeders in file sharing protocols. We cast this intuition into a rigorous mathematical model, and we formally prove that our protocol is secure against deviations of rational parties, by showing that RatFish constitutes a Nash equilibrium.

Second, we provide a general technique for modeling rational cryptographic protocols in the applied pi calculus [9]. We further illustrate our approach by modeling a simplified version of the RatFish rational file sharing protocol [8] within the new framework and by providing its corresponding automatic proof of Nash equilibrium using ProVerif. To the best of our knowledge, this is the first time such an approach has been taken for general modeling of rationality and utilities and also for automatically verifying rational cryptographic protocols together with their associated game-theoretic properties. Our approach is both efficient and easy to implement.

Third, we investigate the intrinsic connection between game theory and cryptography [29]: More precisely, we study the equivalence relations between security notions and game theoretic notions. The first research result [55] regarding such equivalence relations has been centered around the "power of costly computation". Intuitively, the meaning of costly computation is that rational players interested in running a protocol may have an incentive to deviate from it if the cost of computation (e.g., number of steps needed to be performed or the size of memory used) is higher than a threshold they have decided upon. The main result of this study [55] is an equivalence relation between a very weak security notion, namely weak precise secure computation and a newly defined game theoretic notion, which was called strong universal implementation. However, it was left as an open question how to obtain further equivalence relations for stronger security notions.

In this thesis, we solve this open question by discarding the cost of computation. First, we define the notion of game universal implementation which we show is equivalent to weak stand-alone security. Thus, we are able to answer positively the open question from [55] regarding the existence of game-theoretic definitions that are equivalent to cryptographic notions for which the ideal world simulator does not depend on both the distinguisher and the input distribution. While developing the above mentioned result, we also show new equivalence relations among various existing security notions. Such results are important also on their own: Indeed, our main achievement in this study is a separation result between two variants of the universal composability (UC) security definition: 1-bit specialized simulator UC security and specialized simulator UC security. The separation result between these UC variants was stated as an open question [71] and

it comes in contrast with the well known equivalence result between 1-bit UC security and UC security.

The rest of this thesis is structured as follows: In Chapter 2 we present the mathematical model and the design for our rational file sharing protocol RatFish. In Chapter 3 we develop a general method for automated verification for rational cryptographic protocols and we show how to apply our technique in order to automatically derive the Nash equilibrium property for RatFish. In Chapter 4 we study some intrinsic connections between game theory and cryptography: We define a new game theoretic notion, game universal implementation, and we show its equivalence with the notion of weak stand-alone security. Our overall conclusions for this thesis are given in Chapter 5. Appendix A contains full Proverif listings related to Chapter 3.

Chapter 2

Rational File Sharing with RatFish

2.1 Introduction

Recently, the peer-to-peer (P2P) paradigm has emerged as a decentralized way to share data and services among a network of loosely connected nodes. Characteristics such as failure resilience, scalability and adaptivity to dynamic situations have popularized P2P networks in both academia and industry. The proliferation of P2P computing has also been propelled by popular applications, most notably file sharing protocols such as BitTorrent [32].

A crucial assumption underlying the design of such file sharing protocols is that users follow the protocol as specified; i.e., they do not try to bypass the design choices in order to achieve higher download rates, or to avoid uploading to the system at all. However, not all users are necessarily altruistic, and publicly available, modified BitTorrent clients like BitThief [73] or BitTyrant [90] can be used to strategically exploit BitTorrent's design to achieve a higher download while contributing less or nothing at all in return. While several minor protocol adaptations have been suggested to mitigate the attacks underlying these clients [105], the core weaknesses remain: In its current form, BitTorrent – and current file sharing protocols in general – offer better service to cheating clients, thereby creating incentives for users to deviate from the protocol; in turn, it further decreases the performance of honest clients. The task is thus to design a protocol that not only retains the remarkable characteristics of current file sharing protocols, but that is rational in the sense that it offers sufficient incentives for users to stick to the precise protocol specification. In more technical terms, this file sharing protocol should constitute an equilibrium state: Adhering to the protocol should optimize the benefits received by each individual participant, and any deviation from the protocol should result in a lower payoff for the cheating user.

2.1.1 Contributions

We contribute RatFish, a protocol for rational file sharing. RatFish is built upon the concepts and design choices that underlie BitTorrent, but it resolves the weaknesses that clients such as BitThief and BitTyrant exploit. We achieve this mostly by ensuring rational exchange of pieces between leechers and by having the tracker participate in the coordination of the downloads. In this context, an exchange is called rational if the participants have no incentive to deviate from it.

The distinctive feature of RatFish, however, is not that it discourages the use of several selfish strategies, but that it comes with a formal proof that deviating from RatFish is irrational for both seeders and leechers. In order to do this, we first characterize rational behaviors of leechers and seeders in file sharing protocols, building upon the concept of the recently emerging field of rational cryptography, in which users are defined as rational players in a game-theoretic sense. Intuitively, leechers are primarily interested in minimizing their download time and the amount of uploaded data, whereas seeders value the efficiency of the protocol in using their upload capacity. We cast this intuition into a rigorous mathematical model, and we formally prove that our protocol is secure against deviations of rational parties, by showing that it constitutes a Nash equilibrium. This holds even though RatFish allows users to dynamically leave and (re-)join. We prove this Nash equilibrium using a new proof technique that is of independent interest for rational cryptography: the step-by-step substitution of a deviating strategy with hybrid, semi-rational strategies.

We have built a prototype implementation of RatFish that demonstrates that RatFish is practical and efficient. We stress though that the purpose of RatFish is not to achieve performance improvements over existing protocols, but to establish a formal proof that under realistic conditions, such as dynamically joining users, no rationally-behaving user has an incentive to deviate from RatFish. The additional computational overhead of RatFish compared to BitTorrent is small: basic cryptographic primitives (symmetric encryptions and digital signatures schemes) are used, and the tracker is assigned additional tasks such as the coordination of downloads and the generation of user incentives. The communication overhead of a RatFish client is kept to a minimum.

2.1.2 Related Work

The performance of BitTorrent has been thoroughly studied [93, 17, 59, 92, 91]. All these works attest to the impressive performance of BitTorrent in the presence of honest participants; however, it has been noted [17] that the rate-based Tit-For-Tat policy of BitTorrent does not prevent nodes from uploading less content than they should serve (in all fairness), thereby creating an incentive for abuse of the protocol.

The behavior of BitTorrent in the presence of cheating peers was subsequently investigated [72, 90, 73, 101], revealing that cheating leads to a loss in overall performance for honest peers.

Our rigorous model of rational file sharing is grounded in the recently emerging field of rational cryptography, where users are assumed to only deviate from a protocol if doing

so offers them an advantage. Rational cryptography is centered around (adapted) notions of game theory such as computational equilibria [33]. A comprehensive line of work already exists that develops novel protocols for important cryptographic primitives such as rational secret sharing and rational secure multiparty computation [2, 37, 53, 48, 63, 39].

There has been already a variety of research aimed at making BitTorrent more robust against deviations of rationally-behaving users [105, 68, 91, 85, 102]. All these works provide stronger user incentives: they choke problematic connections [105], grant additional bandwidth to generously uploading neighbors [68], reward leechers that continue seeding after their download is completed [91], optimally distribute a seeder's bandwidth across swarms [85], and employ fair exchange protocols to stop leechers from aborting the protocol [102] early. These modified protocols, however, are still prone to rational attacks; in particular, none of these works reached a (Nash) equilibrium. There has been research [102] on how to ensure that certain deviations from selfish leechers or attacks of malicious peers cannot succeed, e..g., no peer can assume another authorized peer's identity. However, so far there is no work which ensures that deviating from the protocol cannot yield better results.

There is also previous work that strived to establish an equilibrium in the context of file sharing [93]. However, this equilibrium was severely restricted in that it was only guaranteed when rational parties were allowed to only tweak the protocol parameters, but not when they could deviate in larger ways.

More recent research such as BAR-B [3], Equicast [61], and FOX [96] aimed at deploying incentives and punishments such that obeying the protocol is the best strategy for every rational player. The first two protocols were shown to be strict Nash equilibria, i.e., a rational peer obtains no benefit from unilaterally deviating from the assigned strategy. The drawback is that their strict equilibrium solutions limit the design: the BAR-B system only permits a static set of users. Equicast requires the rate at which leechers join to precisely match the rate of which they leave and considers only restricted utility functions that do not take downloading time into account; moreover, these protocols require nodes to waste network bandwidth by sending garbage data to balance bandwidth consumption. FOX [96] establishes a stable Nash equilibrium, but again it only allows a static set of leechers; moreover, its rationality is not grounded on incentives but on fear of retaliation such that a single Byzantine node can cause the entire system to collapse. Somewhat orthogonal to our work are the file streaming applications BAR-Gossip [70] and FlightPath [69]. Both works show a Nash equilibrium (a strict one for BAR-GOSSIP, and an approximate one for Flightpath), but rational players are only interested in minimizing the amount of uploaded data and reducing jitter. While such time-independent utility functions are reasonable for streaming applications, they do not apply to the more sophisticated setting of rational file sharing, where minimizing the time to complete a download is usually the primary goal. Moreover, none of these five protocols considers seeders as part of the rational model. We conclude by saying that like our approach, none of these works offers resistance against Sybil attacks. A Nash equilibrium ensures that no individual user has an incentive to deviate. However, it conceptually does not take coalitions of users into account, rendering Sybil attacks

possible in most works on rationally secure protocols.

2.1.3 Outline

Section 2.2 provides a bird's eye view of the core ideas underlying how we create incentives in file sharing. Section 2.3 summarizes the concepts we use from rational cryptography and defines rational behaviors of seeders and leechers. Section 2.4 presents the RatFish protocol in detail. Section 2.5 contains the proof of equilibrium for RatFish; i.e., it shows that users cannot achieve a better payoff by deviating from the protocol. Section 2.6 discusses our experimental results. Section 2.7 concludes this chapter.

2.2 A Bird's Eye View on How to Rationalize P2P

For the sake of exposition, we provide a high-level overview of the core ideas underlying how we create incentives in file sharing. We briefly discuss which behaviors of seeders and leechers we consider rational, intuitively explain how to incentivize these behaviors, and finally discuss how an equilibrium is obtained for a small example protocol. In this section, we abstract away many important system's details and impose several assumptions to improve understanding. However, all these restrictions will be removed in section 2.4 where we present our RatFish protocol in its full generality.

In the following, we consider a single seeder S that intends to upload a file f to leechers $L_1, \ldots, L_M$. The file is split into pieces $f_1, \ldots, f_{M^2}$. In this exposition, we describe a simplistic protocol that proceeds in a sequence of $M + 1$ monolithic rounds. We assume that the seeder can upload exactly M pieces per round and that every leecher is able to upload and to download at least M pieces of the file in each round.

On the Rationality of Seeding. A seeder is a player that uploads without requesting reciprocation. Intuitively, it thus acts rationally if it uses its upload time and upload speed as efficiently as possible; i.e., for any fixed upload speed and time that the seeder spends within the system, the average download time for all leechers should be as small as possible. It is thus in the interest of the seeder to incentivize leechers to share parts of the file amongst each other as this increases the throughput of the whole system. As a consequence, the naive approach of uploading the whole file to an arbitrary leecher at once cannot yield a rationally secure protocol: This leecher may just complete the download and leave, causing some pieces of the file to be effectively lost from the system. Moreover, since there is only one seeder in this simplistic protocol and the number of leechers is known and does not change, there is no need for a third party, i.e., a tracker. In the simplistic protocol, the seeder sends each leecher L_i in each round j the piece $f_{j \cdot M+i}$.

On the Rationality of Leechers. Leechers aim to download the file as fast as possible while saving upload capacity. The protocol thus has to enforce leecher participation as they will otherwise just download and leave. We need to propose a suitable piece selection algorithm and a piece exchange mechanism that prevents parties from cheating each other. In our example, piece selection is easy: In each round j a leecher L_i holds a piece

$f_{j \cdot M+i}$ obtained from the seeder that no one else has. As the leecher can upload M pieces per round, he can exchange with the rest of the leechers their unique pieces. To ensure fair exchanges, leechers first exchange the pieces in encrypted form and subsequently send the corresponding decryption keys.

How an Equilibrium is Achieved. We conclude with some basic intuition on why no rational user has an incentive to deviate from the protocol. If all peers adhere to the protocol, the seeder will upload the file exactly once and stay in the system for M rounds. Each of the leechers will upload $M^2 - M$ pieces and complete its download after $M + 1$ rounds. It is easy to see that the average download time and hence the seeder's utility cannot be improved.

This outcome cannot be further improved for the leechers either: None of the leechers can download the file in less than $M + 1$ rounds since after round M each of them is missing at least $M - 1$ pieces. This holds because the protocol treats the rounds integrally. Otherwise, we could split a sufficiently big file into M^K pieces for some K and achieve a slightly reduced, optimal download time of $M + \frac{M^2}{M^K}$ using an analog algorithm. Moreover, since the seeder only provides M pieces to each of its peers, no leecher can obtain the full file without uploading at least $M^2 - M$ pieces in exchange for the pieces that it is missing from the seeder. This statement holds as no leecher can cheat during the rational piece exchange protocol: A leecher spends his upload capacity to receive an encrypted piece, hence he has no incentive not to send the much smaller decryption key to its partner. Thus, no party can improve its utility by deviating from the protocol.

2.3 A Game-theoretic Model for File Sharing

In this section, we propose a game-theoretic model for rationally secure file sharing. We start by reviewing central concepts from game theory and rational cryptography.

A *Bayesian game* $\Gamma = (\{T_i\}_{i=1}^n, \{A_i\}_{i=1}^n, Pr, \{u_i\}_{i=1}^n)$, also called *a game with incomplete information*, consists of *players* $1, \ldots, n$. The incomplete information is captured by the fact that the *type* for each player i (i.e., its private information) is chosen externally, from a set T_i, prior to the beginning of the game. Pr is a publicly known distribution over the types. Each player has a set A_i of possible *actions* to play and individual *utility functions* u_i. Actions are played either simultaneously or sequentially; afterwards, every player i receives a *payoff* that is determined by applying its utility function u_i to the vector of types received in the game, i.e., *profile types*, and the actions played, i.e., *action profile*.

Recent work has extended the traditional notion of a game to the requirements of cryptographic settings with their probabilistically generated actions and computationally-bounded running times. The resulting definition – called *computational game* [60] – allows each player i to decide on a probabilistic polynomial-time, in the security parameter, interactive Turing machine M_i (short PPITM). The machine M_i is called the *strategy* for player i. The output of M_i in the joint execution of these interactive Turing machines denotes the actions played by participant i.

Definition 1 (Computational Game)**.** *Let k be the security parameter and let $\Gamma = (\{T_i\}_{i=1}^n, \{A_i\}_{i=1}^n, Pr, \{u_i\}_{i=1}^n)$ be a Bayesian game. Then Γ is a* computational game *if the played action A_i of each participant i is computed by a PPITM M_i and if the utility u_i of each player i is polynomial-time computable.*

Because of the probabilistic strategies, the utility functions u_i now correspond to the expected payoffs. Thus, when there is no possibility for confusion, we overload the notation for u_i. However, when the utility we employ is not clear from the context, we denote by U_i the expected utility for party i.

Rationally behaving players aim to maximize these payoffs. In particular, if a player knew which strategies the remaining players intend to choose, he would hence pick the strategy that induces the most benefit for him. As this simultaneously holds for every player, we are looking for a so-called *Nash equilibrium*, i.e., a strategy vector where each player has no incentive to deviate from, provided that the remaining strategies do not change. Similar to the notion of a game, we consider a computational variant of a Nash equilibrium.

Definition 2 (Computational Nash Equilibrium)**.** *Let Γ be a computational game, where $\Gamma = (\{T_i\}_{i=1}^n, \{A_i\}_{i=1}^n, Pr, \{u_i\}_{i=1}^n)$ and let k be the security parameter. A* strategy vector *(or* machine profile*)* consisting of PPITMs $\overrightarrow{M} = (M_1, \ldots, M_n)$ is a *computational Nash equilibrium* if for all i and any PPITM M_i' there exists a negligible function ϵ such that

$$u_i(k, M_i', \overrightarrow{M_{-i}}) - u_i(k, \overrightarrow{M}) \leq \epsilon(k)$$

holds.

Here $u_i(k, M_i', \overrightarrow{M_{-i}})$ denotes the function u_i applied to the setting where every player $j \neq i$ sticks to its designated strategy M_j and only player i deviates by choosing the strategy M_i'. In the definition above, we call M_i a *computational best response* to $\overrightarrow{M_{-i}}$.

We finally define the *outcome* of a computational game as the transcript of all players' inputs and the actions each has taken. In contrast to strategy vectors, an outcome thus constitutes a finished game where every player can determine its payoff directly. A utility function is thus naturally defined on the outcome of a computational game: When applied to a strategy vector with its probabilistic choices, it describes the vector's expected payoff; when applied to an outcome of the game, it describes the exact payoff for this outcome.

2.3.1 A Game-theoretic Model for File Sharing Protocols

We now define the utility functions for seeders and leechers such that these functions characterize rational behavior in a file sharing protocol. We start by introducing common notation and some preliminaries.

Notation and Preliminaries. Following the BitTorrent convention, we call a player in the file sharing game a *peer*. The peers are divided into two groups: A *seeder* uploads to other peers a *file* f that it owns, whereas a *leecher* downloads f. To mediate the

communication among peers, we thus implicitly require a trusted party called the *tracker*. The tracker holds a signing key pair (pk, sk), and we assume that its IP address and public key pk are known to all peers.

The file f consists of pieces $f_1, \ldots, f_N$, each of length B bytes. The participants in the file sharing protocol hold the values $h_1 = h(f_1), \ldots, h_N = h(f_N)$, where h is a publicly known *hash function*. When deployed in practice, this publicly known information is distributed via a *metainfo file*. The tracker is only responsible for coordinating peers that are exchanging the same file. In order to stay close to a realistic setting, we allow different peers to have different upload and download capacities. Every seeder S_i has its individual *upload speed* $up_i^s(t, o)$ that depends on the time t and the outcome o. Note that a seeder does not download anything except for metadata; hence we do not need to consider the download speed of seeders. Similarly, every leecher L_i has individual *upload* and *download speeds* $up_i^l(t, o)$ and $down_i^l(t, o)$. We denote by $T_{i,\text{fin}}(o)$ the total time that leecher L_i spends downloading the file. In terms of measurement units, each of the upload and download capacities defined so far is considered as bytes per second. Additionally, the time is measured in seconds. To increase readability, we omit the outcome in all formulas whenever it is clear from the context. We also introduce the sets $L = \{i \mid L_i \text{ is a leecher}\}$ and $S = \{i \mid S_i \text{ is a seeder}\}$.

Rationally-behaving Seeders. A seeder uploads parts of the file to other peers without requesting reciprocation. Intuitively, a seeder is interested in using as efficiently as possible its upload time and upload speed. Thus for any fixed upload speed and time that the seeder spends within the system, the average download time for all leechers should be as small as possible. We express this preference by the following seeder's utility function.

Definition 3 (Seeder's Utility Function)**.** *We say that u_i^s is a* utility function for a seeder S_i *if for any two outcomes o, o' of the game with the same fixed upload speed up_i^s and fixed time T_i^s spent by S_i in the system, it holds that $u_i(o) \geq u_i(o')$ if and only if $\frac{1}{|L|}\sum_{i\in L} T_{i,\text{fin}}(o) \leq \frac{1}{|L|}\sum_{i\in L} T_{i,\text{fin}}(o')$.*

If S_i is the first seeder in the system, we implicitly require that S_i uploads the whole file at least once. Otherwise, it is not rational to share the file in the first place.

Rationally-behaving Leechers. Leechers aim at downloading the shared file as fast as possible; moreover, they also try to use as little upload capacity as possible. The relative weight of these two (typically contradictory) goals is given by a parameter α_i in the system measuring time units per capacity units, e.g., $sec^2/bytes$.

Definition 4 (Leecher's Utility Function)**.** *Let $\alpha_i \geq 0$ be a fixed value. We say that u_i^l is a* utility function for leecher L_i *if the following condition holds: For two outcomes o, o', L_i prefers outcome o to o' if and only if*

$$\alpha_i \cdot \int_0^{T_{i,\text{fin}}(o)} up_i^l(t, o)\, dt + T_{i,\text{fin}}(o) \leq \alpha_i \cdot \int_0^{T_{i,\text{fin}}(o')} up_i^l(t, o')\, dt + T_{i,\text{fin}}(o').$$

The value α_i corresponds to L_i's individual valuation for upload speed and time; e.g., if $\alpha_i = 0.5\frac{sec^2}{bytes}$, the leecher values time twice as much as the uploaded data.

In particular, this definition implies that a rationally-behaving leecher prioritizes completing the download over everything else: If the leecher does not download the file in outcome o, then $T_{i,\text{fin}}(o)$ equals infinity. If it downloads the file in some outcome o', then $T_{i,\text{fin}}(o')$ is finite and thus increases its payoff.

2.4 The RatFish Protocol

We now present the RatFish protocol. We start with the description of the tracker and proceed with the seeders and leechers, respectively.

2.4.1 The Protocol of the Tracker

Similar to BitTorrent, our tracker manages all valid IP addresses in the system and introduces new leechers to a set of neighbors. However, we assign the tracker additional tasks: First, our tracker is responsible for awarding each newcomer with seeding capacity equivalent to γ file pieces, for a tunable parameter γ. In practice, γ is a small constant number just big enough for the new leecher to participate in the system. As long as the seeders can provide γ pieces to each newly joining leecher, this value does not influence the existence of the Nash equilibrium.

Second, our tracker keeps track of which file pieces each peer owns at any given moment. This bookkeeping will be crucial for incentivizing peers to follow the RatFish protocol, for computing the deserved rewards and for answering queries about the leechers' availabilities. Third, the tracker imposes a forced wait for every leecher upon connecting, thereby preventing leechers from gaining advantages by frequently disconnecting and rejoining the protocol. Finally, if a leecher wishes to disconnect, the tracker provides a certificate on the most recent set of pieces the leecher has to offer. This allows leechers to later reconnect to RatFish and use their partially downloaded data, i.e., in order to cope with network disruptions. In the following, we describe the individual subprotocols of the tracker in detail. A rigorous description is given in Fig. 2.1, Fig. 2.2, Fig. 2.3 and Fig. 2.4.

The Connect Protocol. The tracker assigns every new leecher L_i a random subset of size H of all leechers that are currently in the system. This random subset corresponds to L_i's local neighborhood. The tracker sends this neighborhood information to L_i after T seconds. Once the forced wait is over, the leecher may start downloading γ free pieces from seeders. The rationale behind this forced wait is that granting newly joined leechers free pieces creates incentives for whitewashing, i.e., frequent disconnecting and rejoining. Intuitively, the forced wait is a simple defense against such a behavior. From a rational perspective, if a leecher joins the system only once, the induced small delay will not be hurtful; however, whitewashing by frequently rejoining will cause an accumulated delay that will result in a smaller payoff. The forced wait is achieved by having the tracker sign the leecher's connecting time and IP address. Such signed timestamps are exchanged between neighboors and are used to determine whether leechers are allowed to start uploading to eachother. Neighbors use the timestamps to determine whether they are

TrackerConnect($peer$)
If $peer$ is a seeder S_i, receive the seeder's upload speed up_i^s and store it. Else, do:

- – If a message PIECES$(a_1, \ldots, a_N, id_{r''}, sig_i'')$ is received from L_i, verify that sig_i'' is a valid signature on $(a_1, \ldots, a_N, id_{r''})$ for verification key pk and that $p = (a_1, \ldots, a_N, id_{r''})$ was previously stored. Each correct a_m, with $m \in \{1, \ldots, N\}$, is the binary representation of availability of piece j. If all above checks succeed, remove p from storage and set $A_i^m := a_m$, for all $m \in \{1, \ldots, N\}$ and select a random id_{r}.
  – Otherwise, if a message PIECES$(0, \ldots, 0)$ is received, select a random id_{r}.
- As soon as the current time T_c is larger than $T + T_p^i$, where T_p^i is the connecting time of the leecher, i.e., T_c is the time after the forced wait of T seconds, send L_i a random subset of size H of current leechers' IP addresses, corresponding to L_i's neighborhood. Moreover, compute $\mathsf{S}_{sk}(i, T_p^i)$, yielding a signature sig_i. Send TIME$(T_p^i, id_{\mathrm{r}}, sig_i)$ to L_i.

Figure 2.1: The protocol of the tracker with procedure Connect.

allowed to start uploading to each other. Thus as long as a user's IP address does not change, it can idle and become active again without being penalized by a forced wait, since the user's old signature on its IP address and time is still valid. This is detailed in Fig. 2.1.

The RatFish tracker has a mechanism for proving piece availability of rejoining leechers: it chooses a random rejoin ID id_{r} and signs it together with the departing leecher's piece availability. The tracker stores the availability status for leecher L_i and each piece m in the variable A_m^i. The algorithms of the trackers ensure that $A_m^i = 1$ if L_i has m-th piece of file f and $A_m^i = 0$ otherwise. The leecher uses id_{r} to prove its piece availability to the tracker upon rejoining the system. The rejoining id_{r} is then deleted from the tracker's memory preventing leechers from reconnecting twice using the same reconnect id_{r}. This is detailed in Fig. 2.1, Fig. 2.2 and Fig. 2.3.

The Reward Protocol. The reward system constitutes the crucial part of RatFish. The underlying idea is to reward only leechers who are exchanging. We only allow one exception to this rule: The leechers that have just connected to the tracker in the previous round are also entitled to a reward of γ pieces in the current round. Thus the seeders do not automatically upload to their neighborhood as in BitTorrent; rather they are told by the tracker whom to upload to.

To determine whether an exchange between L_i and L_j has indeed taken place, the tracker asks both L_i and L_j to acknowledge the exchange. If the acknowledgements succeed, the tracker internally increases the variables X_i and X_j, which corresponds

CheckExchange

Do the following steps unless one of their checks fails; abort in this case: (Below it is given only the protocol between the tracker and L_i. By symmetry, we can obtain the protocol between the tracker and L_j, where L_j is the exchange partner of L_i for the exchange described below.)

- Upon receiving a message $\mathsf{HAS}(j, y)$ from a leecher L_i, send back 1 if $A_j^y = 1$, and 0 otherwise.
- Upon receiving a message $\mathsf{EXCHANGED}(j, y, x)$ from a leecher L_i, indicating that pieces f_x and f_y have been exchanged with L_j, send the message $\mathsf{ACKNOWLEDGE}(i, x, y)$ to L_j.
- Upon subsequently receiving the message $\mathsf{OK}(i, x, y)$ from L_j, set $X_i := X_i + 1$ and $A_i^y := 1$, and send the message $\mathsf{OK}(j, y, x)$ to L_i. X_i denotes the number of acknowledged exchanges of leecher L_i in the current round of T seconds. X_i is reset to 0 and computed anew each round.

Figure 2.2: The protocol of the tracker with procedure Check Exchange.

PeerLeave(i)

- Compute $sig_i' := \mathsf{S}_{sk}(A_i^1, \ldots, A_i^N, id_\mathrm{r})$. Store $(A_i^1, \ldots, A_i^N, id_\mathrm{r})$.
- Send the message $\mathsf{LEAVE}(id_\mathrm{r}, sig_i')$ to L_i and disconnect from L_i.

Figure 2.3: The protocol of the tracker with procedure PeerLeave.

RewardLeechers (called every T seconds, i.e., at the start of a new round)

- Award γ pieces to every leecher that joined in the previous round. Let *prev* be the number of these leechers.
- Compute for every leecher L_i its deserved percentage of seeders' upload speed:

$$r_i := \min\left\{\frac{X_i}{\sum_{k\in L} X_k}, \frac{1}{2}\right\}$$

- Let $\beta_i := r_i \cdot \left(\frac{\sum_{k\in S} up_k^s \cdot T}{B} - \gamma \cdot prev\right)$. For every i, assign a set of seeders to jointly offer β_i pieces to L_i such that the individual upload capacity of the seeders is respected, see Sect. 2.4.1. Send to every seeder the corresponding set of leechers and the number of pieces that these leechers should download from them.
- Set $A_i^y := 1$ when a seeder informs the tracker that f_y was downloaded by L_i. Reset $X_i := 0$ for all i.

Figure 2.4: The protocol of the tracker with procedure RewardLeechers.

to the number of file piece exchanges of L_i and L_j, respectively. The tracker moreover stores which pieces of the file the leechers now additionally know. This is detailed in Fig. 2.4. Details on the participation of the tracker in the exchange protocol are given in Sect. 2.4.3, where the rational exchange of pieces between leechers is explained.

Every round, i.e., after T seconds, the actual rewards are given out. The tracker distributes the seeders' upstream proportional to the number of exchanges every leecher made in the previous round. Hence, the more exchanges a leecher completed in a certain round, the larger the reward computed for him by the tracker, and hence the more download capacity the leecher receives from the seeders. At the beginning of a round, once the reward deserved by each leecher is computed, the old values X_i are reset to 0 and computed anew, according to the exchanges performed in the current round. A graphical illustration of the reward protocol is given in Fig. 2.5.

More precisely, Fig. 2.5 depicts a system with a tracker, four leechers and four seeders. During the latest complete round of exchanges, leecher L_1 has performed a number of 8 exchanges and leechers L_2, L_3 and L_4 have performed a number of 4,10 and 6 exchanges, respectively. In terms of existing notation, this can simply be written as $X_1 = 8$, $X_2 = 4$, $X_3 = 10$ and $X_4 = 6$. This means that for example L_1 has performed $28,57\%$ of all exchanges during the latest complete round. Taking into account that there are no new leechers currently connecting to the system, according to Fig. 2.5, at the beginning of next round, L_1 will receive for free $28,57\%$ of all the seeding capacity currently available

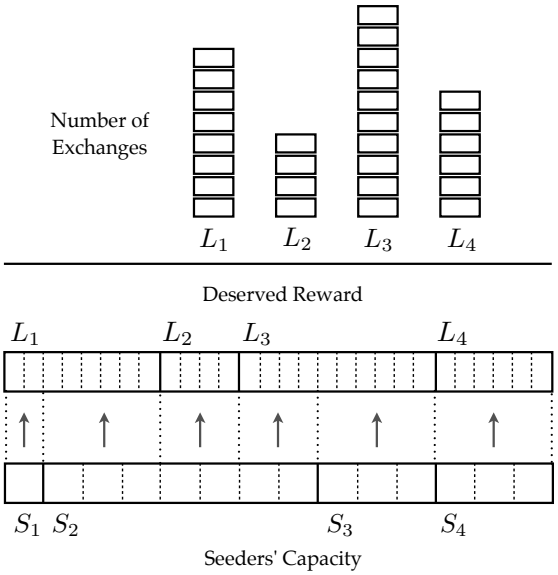


Figure 2.5: Schematic distribution of the rewards

in the system. Given the existing upload capacity of the seeders, this effectively implies that the tracker informs seeders S_1 and S_2 that they should upload 2 and respectively 6 file pieces for free to L_1. It is assumed that S_1 and S_2 have the complete file, so L_1 can choose which pieces he wants to download for free. The same reward distribution algorithm applies for the rest of the leechers and seeders.

2.4.2 The Protocol of the Seeder

Upon connecting, the seeder informs the tracker about the upload speed it is going to offer. The tracker adds the seeder's speed to the amount of free available upload capacity. As the tracker performs all the computations for determining the rewards, the seeder simply proceeds by uploading the number of file pieces to the leechers as instructed by the tracker. To keep the tracker's information about individual leechers up-to-date, the seeder informs the tracker whenever it uploads a piece to a leecher. A rigorous description is given in Fig. 2.6.

2.4.3 The Protocol of the Leecher

From a rational perspective, the leecher protocol is the most difficult to get right: while the tracker is honest and seeders partially altruistic, a leecher tries to bypass the incentives for uploading wherever reasonable. Intuitively, the exchange protocol looks like this: Leechers use the signed messages from the tracker to verify each other's join times. Also, when two leechers exchange data, the tracker participates in this exchange: Before two leechers start an exchange, they verify with the tracker that the other party holds the desired piece. If this check succeeds, the encryptions of the pieces agreed upon are

Seeding$_j$
Upon connecting, the seeder sends its upload speed up_j^s to the tracker. Additionally, in each round:

- Receive from the tracker a set M of leechers and the corresponding number of pieces ω_i that every leecher $L_i \in M$ should receive.
- Inform every leecher $L_i \in M$ how many pieces ω_i they are allowed to download.
- When a leecher $L_i \in M$ requests at most ω_i pieces by L_i (potentially incrementally in this round, i.e., it may ask for a few pieces first), send these pieces to L_i and send a message to the tracker that these pieces have been uploaded to L_i. Requests by leechers $L_j \notin M$ are ignored.

Figure 2.6: The protocol of the seeder S_j.

exchanged. Before they also send the key to each other to decrypt these messages, both leechers acknowledge the exchange to each other so that they get a higher reward.

The Connect Protocol. When a leecher connects to the tracker for the first time, it requests a local neighborhood. If the leecher rejoins, it additionally proves to the tracker that it already owns some pieces of the file by sending the signature received from the tracker at its last disconnect. When connecting to a seeder, the leecher requests pieces until its seeder's reward is depleted.

Upon contacting another leecher, it waits until both forced waits are over. Afterwards, both leechers exchange information such that they know which pieces they can request from each other. To keep track of the availability in its neighborhood, the leecher observes the messages that leechers broadcast to their local neighborhood, indicating which pieces of the file they have just downloaded. A detailed description is given in Fig. 2.7.

The Piece Exchange. The piece exchange protocol run between two leechers uses encryptions to ensure that no leecher can get a piece without completing the exchange phase. From a practical perspective, it is important to note that the key sizes are small compared to a file piece size. Thus the communication and storage overhead induced by the keys and cryptographic operations is kept manageable. Leechers additionally query the tracker to ensure that their exchange partners own a file piece they need. Moreover, leechers want their exchanges to be counted and rewarded. Thus, after the encryptions are exchanged, each leecher prompts the tracker to ask the other leecher for an acknowledgement. Intuitively, there is no incentive to deviate in this step as they still lack the key from the other party. Once the acknowledgement step is successfully completed, both leechers exchange the keys. If a leecher deviates from any of these steps, it is blacklisted by the other leecher. We stress that blacklisting is not required for the security proof; it solely constitutes a common technique in this setting to deal with

$\mathsf{LeecherConnect}_i(party)$
If *party* is the tracker, then:

1. If L_i rejoins the protocol, send $\mathsf{PIECES}(a_1, \ldots, a_N, id_\mathrm{r}, sig'_i)$ to the tracker where $a_m = 1$ if L_i owns the m-th piece of the file, id_r is the rejoin ID and sig'_i is the signature received when disconnecting from the system last time. If L_i is a new leecher, it sends $\mathsf{PIECES}(0, \ldots, 0)$.
2. Receive $\mathsf{TIME}(T_p^i, id_\mathrm{r}, sig_i)$ from the tracker – indicating the signed connecting time and ID, as well as a set of neighbors' IP addresses. Connect to them.

If *party* is a leecher L_j, do (abort if a check fails):

- Send the message $\mathsf{MYTIME}(T_p^i, sig_i)$ to L_j.
- Receive the message $\mathsf{MYTIME}(T_p^j, sig_j)$ from L_j. Verify that sig_j is a valid signature on (j, T_p^j) for pk and that $T_c > T_p^j + T$ holds.
- Send $\mathsf{AVAILABILITY}(a_1, \ldots, a_N)$ to L_j where $a_m = 1$ if L_i owns f_m.
- Receive the message $\mathsf{AVAILABILITY}(a'_1, \ldots, a'_N)$ from L_j.

Figure 2.7: The Connect protocol for leecher L_i.

LeecherAwarded
Whenever L_i is informed by a seeder S_j that it can download ω_i pieces, request up to ω_i pieces from S_j (potentially incrementally in this round, i.e., L_i may ask for a few pieces first), and download these pieces.

$\mathsf{Exchange}_i(f_x, j, y)$
If any of the following checks fails, blacklist L_j and abort.

- Send the message $\mathsf{HAS}(j, y)$ to the tracker and wait for a positive answer represented by a bit $b = 1$.
- Choose a random key $k_{j,x}$ and compute $c_{j,x} \leftarrow \mathsf{E}(k_{j,x}, f_x)$.
- Send $c_{j,x}$ to L_j and wait for c_y from L_j.
- Perform the following two steps in parallel and proceed once both steps are completed:
  - Send $\mathsf{EXCHANGED}(j, x, y)$ to the tracker and wait for $\mathsf{OK}(j, x, y)$ as response
  - If receiving $\mathsf{ACKNOWLEDGE}(j, y, x)$ from the tracker, reply with $\mathsf{OK}(j, y, x)$.
- Send the key $k_{j,x}$ to L_j.
- Upon receiving k_y from L_j, retrieve $f'_y \leftarrow \mathsf{D}(k_y, c_y)$ and verify $h_y = h(f'_y)$.
- Broadcast to the local neighborhood that you now own the piece y.

Figure 2.8: The Award and Exchange protocols for leecher L_i.

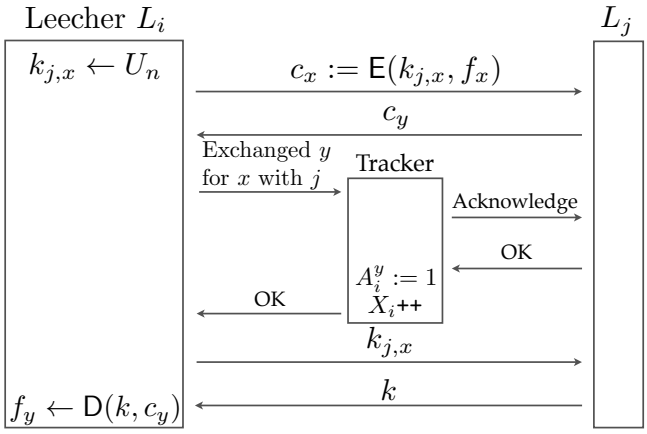


Figure 2.9: The core part of piece exchange protocol between two leechers

malicious parties. A detailed description is given in Fig. 2.8. Fair exchange protocols have been used in prior work to incentivize peers to fairly exchange information [102]. A fair exchange intuitively means that either both parties obtain what they want or none of them obtains something. In contrast to [102], however, RatFish needs to neither periodically renew cryptographic keys, nor implement a non-repudiable complaint mechanism to allow parties to prove possible misbehaviors; instead it relies on short acknowledgment messages for each recipient and on collecting these messages to monitor the file completion for the participants. In fact, RatFish implements a weaker version of fair exchange, which is called rational exchange [21, 103]. Intuitively, an exchange protocol is rational if the self-interested participating parties do not have an incentive to deviate. A schematic overview of the core part of the piece exchange protocol is provided in Fig. 2.9. More details on the differences between rational exchange and fair exchange can be found in Chapter 3.

2.5 Equilibrium Proof

In this section we prove that RatFish yields a computational Nash equilibrium; i.e., no leecher or seeder has an incentive to deviate from the protocol.

2.5.1 Underlying Assumptions

Recall that RatFish proceeds in rounds of T seconds. For simplicity, we assume that peers can join or leave only at the beginning or end of a round. This assumption can be easily enforced by letting the tracker force joining peers to wait until the first round after at least T seconds pass. We also assume that it is impossible to forge identities on the IP layer (e.g., by using appropriate authentication mechanisms). Additionally, at least one seeder is present in the beginning to bootstrap the system and that the overall seeding capacity does not exceed twice the overall upload capacity of the leechers; this bound on

the seeding capacity prevents the leechers from free riding, which is easy given enough seeding power. We moreover assume that each leecher's dedicated upload speed up_i^l is fully exhausted by other peers. This is equivalent to saying that there are enough leechers in the system such that each of them has enough neighbors for completely exhausting his upload capacity. These assumptions seem reasonable as the average seeders/leechers ratio is often close to 1:1 [18], and optimized centralized coordinators are capable of distributing upload capacities among different swarms [85]. Moreover, we assume that there exists a value δ such that for every leecher L_i the download speed $down_i^l$ is at most δ times larger than up_i^l. This assumption is not restrictive in the sense that we do not need to make any assumption regarding the magnitude of δ. Additionally, we assume keys do not contribute to the uploaded amount, since in practice, the size of the short keys is dominated by the size of the encrypted file piece. Moreover, we assume that each peer is only able to maintain one identity in the system. This in particular excludes Sybil attacks, in which multiple distinct identities are created by the same peer to subvert the reputation system of a P2P network. This assumption does not come as a surprise, since the Nash equilibrium conceptually does not defend against coalitions, rendering Sybil attacks possible in most works on rationally secure protocols. Regarding the cryptographic primitives, we assume that the signature scheme used by the tracker is secure against existential forgery under chosen-message attack and that the encryption scheme is semantically secure under chosen-plaintext attack. Finally, we assume that the encryption scheme is length preserving.

2.5.2 Proving the Nash Equilibrium

We finally show that RatFish constitutes a Nash equilibrium.

We first prove that a leecher deviating from the protocol cannot increase its utility by more than at most a negligible value, provided that no other party deviates. To show this, we determine two sets of possible *cheating actions* for leechers, which we call *independent actions* and *correlated actions*. Intuitively, the independent cheating actions can be *individually* replaced by honest actions without decreasing the utility, independent of the remaining parts of the deviating strategy. Correlated cheating actions are sensitive to the details of the deviating strategy: we can only replace a correlated cheating action by a corresponding honest action without decreasing the utility if all deviating actions that jointly influence the leecher's utility are *simultaneously* replaced in one round. We show that the only correlated cheating action is the refusal of acknowledgement for an exchange.

Our proof technique starts with an arbitrary deviating strategy M_i' and provides a proof in two steps: In the first step, we replace all independent cheating actions step-by-step; here, a step within a strategy denotes the computation performed within the strategy between two consecutive outputs. Slightly more formally, let M_i be the honest strategy for leecher L_i, M_i' a deviating strategy, and $\{H^*_{ack,j}\}_j$ the set of all strategies that in every step are either honest or do not acknowledge an exchange. Then our technique takes as input M_i' and yields a so-called *semi-honest* strategy $M_i^* \in \{H^*_{ack,j}\}_j$ that for every input and random tape outputs in every step the same action as M_i' whenever

possible, and plays honest otherwise. We then show that the semi-honest strategy M_i^* cannot yield a worse payoff than M_i'. The proof is based on the novel concept of hybrid concatenation of strategies.

We start by proving the following:

Lemma 5 (No Independent Cheating Actions of Leechers)**.** *Let γ be the number of uploaded pieces a newly joined leecher is awarded. Let M_i' be a deviating strategy of L_i and let M_i^* be the semi-rational strategy as defined above. Then for $\alpha_i \in [0, \frac{T}{\delta \cdot \gamma \cdot B}]$, we have*

$$u_i(k, M_i', \boldsymbol{M}_{-i}) - u_i(k, M_i^*, \boldsymbol{M}_{-i}) \leq \epsilon(k),$$

for some negligible function ϵ.

Proof. We consider two main cases, depending on the number of steps of M_i'. If M_i' does not terminate in a finite number of steps, then the time $T_{i,\text{fin}}(o)$ the leecher L_i has to spend in the system is infinite. Thus any strategy, including M_i^*, cannot give a worse payoff. If M_i' runs in $N \in \mathbb{N}$ steps, then we construct N hybrid concatenations and prove that for all $n = 0, \ldots, N-1$, there exist a negligible function ϵ_i^n such that it holds

$$u_i(k, M_i' \|_n^s M_i^*, \boldsymbol{M}_{-i}) \leq u_i(k, M_i' \|_{n+1}^s M_i^*, \boldsymbol{M}_{-i}) + \epsilon_i^n(k), \tag{2.1}$$

where by $M_i' \|_n^s M_i^*$ we denote the strategy M_i' with the last n steps replaced by last n steps of the rational strategy M_i^*.

In order to show this, we use induction on the steps of strategy M_i'. As there is no difference between the base case and the inductive step, we present below only the later. We examine all possible independent cheating actions which could have been performed by M_i' at step $n+1$. Intuitively, such independent cheating actions can be grouped into two main categories: related to the messages sent or related to the messages received. When a message is sent, there are three possible scenarios: sending no message, sending the correct message or sending a malformed message. When a message is received, there are also three possible scenarios: receiving no message, receiving the message and using it correctly for necessary computations or receiving the message and using it in a different way than described by the protocol.

In the following, we will concentrate on the case of messages which are sent. For the messages received, the proof follows a very similar approach.

It is clear that if at step $n+1$ the action performed is the action prescribed by M_i^*, then we have:

$$u_i(k, M_i' \|_n^s M_i^*, \boldsymbol{M}_{-i}) = u_i(k, M_i' \|_{n+1}^s M_i^*, \boldsymbol{M}_{-i}). \tag{2.2}$$

Below we give a detailed case distinction for the situation when the action is either to send no message or to send a malformed message.

Case 1: *M_i' does not announce the correct timestamp that it received from the tracker.*

Because L_i is required to provide a valid signature from the tracker on the timestamp, this deviation will be undetected only with a negligible probability, corresponding to the probability of constructing a selective forgery against the signature scheme. If L_i

is caught cheating, then the other peers do not let L_i connect to them. Therefore, we obtain that (2.1) holds.

Case 2: *M_i' does not announce any timestamp that it received from the tracker.*

As the other leechers follow the protocol as prescribed, they will not connect to a neighbor which did not announce his timestamp received from the tracker. Thus, in this case, leecher L_i cannot complete the file download and (2.1) trivially holds.

Case 3: *M_i' connects to leechers before the penalty wait of T seconds is over.*

Other leechers will not reply to L_i unless it provides a valid signature on a timestamp $T_p^i < T_c - T$. Since an invalid signature can be produced only with negligible probability, we have that (2.1) holds.

Case 4: *M_i' connects to leechers after the penalty wait of T seconds is over.*

The more time leecher L_i waits to connect to others after his penalty wait is over, the more his utility decreases as compared to the case when he follows the prescribed strategy. So (2.1) is true.

Case 5: *M_i' does not connect to leechers even after the penalty wait of T seconds is over.*

If L_i does not connect to other leechers in his neighborhood or L_i does not reply to their connect requests, then L_i will not be able to download the file at all or his upload capacity will not be fully utilized as it is the case when he follows the protocol. In both situations, his benefit by performing this deviation decreases compared to the prescribed run of the protocol. More formally, (2.1) holds.

Case 6: *M_i' accepts connections from leechers whose penalty wait is not over.*

Because those leechers will not connect to L_i, this does not change the outcome of the game. Therefore, (2.1) trivially holds.

Case 7: *M_i' does not accept connections from at least one leecher in its own neighborhood, even though leecher's penalty wait is over.*

Such a deviation would slow down L_i as his upload capacity would not be fully exhausted. Thus, we have (2.1).

Case 8: *M_i' announces to hold more pieces than it actually does.*

There are two possible cases. Either L_i tries to convince the tracker upon connecting that it holds more pieces, or it has sent wrong "HAVE x" messages. In the first case, L_i has to provide a valid signature from the tracker on wrong data, which is possible only with negligible probability. In the second case, this will not affect anything until some other leecher requests a piece f_x from L_i that it does not own yet. Then, in the exchange step, the tracker will send the other leecher the message that L_i does not have f_x. The exchange will be canceled and this will not allow L_i to increase his utility, thus fulfilling (2.1).

Case 9: *M_i' announces to hold fewer pieces than it actually does.*

This does not increase its utility, because the amount of pieces is used by other leechers to determine whether they can exchange with L_i. Fewer parties are willing to exchange with L_i if it claims to have fewer pieces. This case also fulfills (2.1).

Case 10: *M_i' does not make any announcement on the pieces it has.* Such a deviation from L_i would only slow it down as the other leechers would not connect to L_i for

performing new exchanges: L_i would potentially have no interesting pieces to offer. Hence, (2.1) holds.

Case 11: *When receiving an encrypted piece from a neighbor, M_i' does not send anything as a reply to the corresponding leecher.*

In this case, the utility of L_i would increase only if the leecher is able to guess the file piece from the encrypted message. Since the encryption scheme is semantically secure against chosen plaintext attack, the deviating leecher would succeed only with negligible probability so (2.1) is fulfilled.

Case 12: *In the exchange phase, M_i' sends a wrong encryption upon being requested to exchange piece f_x.*

Assume L_i does not hold piece f_x. Because other leechers only request what it announced with "HAVE x", this means it wrongly announced that it has too many pieces. This case is already discussed above. Therefore, L_i has the piece the other party requested and the other party expects a message of the length $|\mathsf{E}_k(f_x)|$. But then, L_i can also send the encryption of the requested piece, because sending a wrong encryption still requires it to upload exactly as much as uploading the right encryption. Thus, this deviation leaves L_i with (2.1).

Case 13: *M_i' requests an invalid reward from the tracker.*

By assumption, M_i' cannot send a message under the identity of another player (we assumed authentic channels). However, its request will not be acknowledged by any other party, as they are sticking to the protocol. Therefore, this does not increase its utility and (2.1) holds.

Case 14: *M_i' makes no request for reward, even though he is entitled to such a reward.* By performing this deviation, the overall download time for L_i will increase, thus trivially (2.1) holds.

Case 15: *In the exchange phase, M_i' does not send the key in the end.*

If the exchange reached the keys sending phase, since L_i's partner follows the protocol, it means that the two parties have the right piece for one another and they have already exchanged encryptions. By assumption, the key size does not increase the uploaded amount so if L_i sends the key it will not reduce his utility and (2.1) still holds.

Case 16: *M_i' reconnects after r rounds.*

Let o' be the outcome when L_i follows the reconnecting strategy $M_i' \|_n^s M_i^*$ and let o be the outcome when L_i follows $M_i' \|_{n+1}^s M_i^*$. Since the strategies of all other parties do not change and a leecher can reconnect only at the beginning of a new round, we have the following relation: $T_{i,\text{fin}}(o') = T_{i,\text{fin}}(o) + \tau$. The value τ is the additional time L_i spends in the system since for at least one round, L_i did not interact with any leecher. In a similar way, if U_i is the overall amount of uploaded data for outcome o, then the amount of uploaded data in o' is at most $U_i - \gamma \cdot B$, where $\gamma \cdot B$ is the number of free awarded bytes for a (re-)joining leecher. As a reminder, B represents the bytes size of a file piece.

We observe that in outcome o', after the rejoin, the leecher is missing at least $T \cdot up_i^l$ bytes compared to o. By assumption we know that the leecher never has a download speed larger than $\delta \cdot up_i^l$. Hence, the time τ the leecher needs to additionally stay in the

system is given by how fast he can download the missing data. This is at least:

$$\tau \geq \frac{T \cdot up_i^l}{\delta \cdot up_i^l} = \frac{T}{\delta}.$$

Proving (2.1) is equivalent to proving

$$T_{i,\text{fin}}(o) + \alpha_i \cdot U_i \leq T_{i,\text{fin}}(o) + \tau + \alpha_i \cdot (U_i - \gamma B).$$

This holds true if $\alpha_i \leq \frac{\tau}{\gamma \cdot B}$. The last inequality holds since by assumption $\alpha_i \in [0, \frac{T}{\delta \cdot \gamma \cdot B}]$ and $\tau \geq \frac{T}{\delta}$.

Case 17: *M_i' sends incompliant messages.*

Because such messages are ignored by other parties, this does not affect the utility, and (2.1) is fulfilled.

To summarize, we obtain that (2.1) holds for all $n \in \{0, \ldots, N-1\}$. By summating al these equations and taking into account that N is polynomial in k and also that the sum of polynomially many negligible functions is still negligible, we infer that there exists a negligible function ϵ' such that:

$$\begin{aligned} u_i(k, M_i', \boldsymbol{M}_{-i}) &= u_i(k, M_i' \|_0^s M_i^*, \boldsymbol{M}_{-i}) \\ &\leq u_i(k, M_i' \|_N^s M_i^*, \boldsymbol{M}_{-i}) + \epsilon'(k) \\ &= u_i(k, M_i^*, \boldsymbol{M}_{-i}) + \epsilon'(k). \end{aligned}$$

This concludes our proof. □

Thus far, we have transformed a deviating strategy M_i' into a semi-rational strategy M_i^* that uses only correlated cheating actions and does not decrease the payoff. In the second step, we replace all correlated cheating actions round-by-round until we reach the honest strategy M_i. We use a hybrid argument based on the hybrid concatenation of strategies to show that the honest strategy outperforms the semi-rational strategy for leechers.

Lemma 6 (No Correlated Cheating Actions of Leechers)**.** *Let M_i be the honest strategy for L_i, i.e., following the RatFish protocol and let M_i^* be the semi-rational strategy as defined above. Then*

$$u_i(k, M_i^*, \boldsymbol{M}_{-i}) - u_i(k, M_i, \boldsymbol{M}_{-i}) \leq \epsilon(k),$$

holds for some negligible function ϵ.

Proof. We again use the hybrid concatenation of strategies. However, now we need to replace correlated deviations with honest actions; hence the hybrids are constructed over rounds, and the notation $A\|_n^r B$ denotes that the last n rounds of strategy A are replaced by the last n rounds of strategy B. The proof starts with the last round and replaces the deviating strategy with the RatFish protocol round-by-round. We show by induction

that each such replacement will not decrease the utility, i.e., for every $n \in \{1, \ldots, R\}$, where R is the number of rounds for strategy M_i^*, there exists a negligible function ϵ_i^n such that:

$$u_i(k, M_i^* \|_{n-1}^r M_i, \boldsymbol{M}_{-i}) \leq u_i(k, M_i^* \|_n^r M_i, \boldsymbol{M}_{-i}) + \epsilon_i^n(k). \tag{2.3}$$

As a consequence, the payoff for the honest M_i cannot be exceeded.

For the base step, we examine the initial hybrid where no deviation has been removed. Clearly, it holds that $u_i(k, M_i^* \|_0^r M_i, \boldsymbol{M}_{-i}) = u_i(k, M_i^*, \boldsymbol{M}_{-i})$. For the inductive step, we have the induction hypothesis

$$u_i(k, M_i^* \|_{n-1}^r M_i, \boldsymbol{M}_{-i}) \leq u_i(k, M_i^* \|_n^r M_i, \boldsymbol{M}_{-i}) + \epsilon_i^n(k),$$

for some negligible function ϵ_i^n.

Now we consider the n-th hybrid: the last n rounds are all played according to M_i and all previous rounds are played according to M_i^*. If we compare this hybrid with the $n+1$st hybrid, the only difference is that the possibly deviating $n+1$-st round is replaced by an honest one. By definition, in round $n+1$, the strategy M_i^* says that L_i does not acknowledge $Z \geq 0$ correct exchanges, but his exchange partners acknowledge all exchanges made so far. If $Z = 0$, then M_i^* played in round n the honest strategy M_i, thus:

$$u_i(k, M_i^* \|_n^r M_i, \boldsymbol{M}_{-i}) = u_i(k, M_i^* \|_{n+1}^r M_i, \boldsymbol{M}_{-i}).$$

For the case where $Z > 0$, we investigate the properties of the utility function. To reach the phase of acknowledging the other's exchange, L_i first needs to upload the encrypted data. Therefore, the amount of uploaded data stays the same both in the deviating strategy and in the honest strategy. The only possibility to increase the utility is then to download more data in the same time span. In the following, we show this is not possible.

With M_i, the leecher's amount of downloaded data corresponding to one round of T seconds equals

$$B \cdot X_i + T \cdot (\sum_{k \in S} up_k^s) \cdot \min\left\{ \frac{X_i}{\sum_{k \in L} X_k}, \frac{1}{2} \right\}.$$

Note that in the second summand, we have the reward gained in the next round, because the tracker rewards exchanges always one round later. Furthermore, the tracker caps the reward to at most one half of the total seeders' upload, according to the protocol in Fig. 2.2. However, for a non-deviating player, it is impossible to obtain more than half of the overall exchanges, thus we have $\min\left\{ \frac{X_i}{\sum_{k \in L} X_k}, \frac{1}{2} \right\} = \frac{X_i}{\sum_{k \in L} X_k}$. Using M_i^* in round $n+1$, the leecher gains $Z \cdot B$ bytes less from the cheated exchanges, but he may obtain higher reward from the seeders.

This sums up to at most

$$B \cdot (X_i - Z) + T \cdot (\sum_{k \in S} up_k^s) \cdot \min\left\{ \frac{X_i}{(\sum_{k \in L} X_k) - Z}, \frac{1}{2} \right\}.$$

Indeed, the intuition for the first summand is that when leecher L_i does not acknowledge Z exchanges in a round, there is only a negligible probability that L_i will obtain the corresponding decryption keys for these exchanges. This holds as L_i has only honest exchange partners and they respond to a non-acknowledge of a correct exchange by blacklisting the deviating L_i and also by interrupting communication with L_i. The second summand gives the amount of upload reward that L_i receives from the seeders in a round of T seconds when he does not acknowledge Z exchanges. The value $\frac{X_i}{(\sum_{k\in L} X_k)-Z}$ gives the ratio of acknowledged exchanges for L_i to the total number of acknowledged exchanges. Therefore, his utility will not increase if

$$Z \cdot B + T \cdot (\sum_k up_k^s) \frac{X_i}{\sum_k X_k} \geq T \cdot (\sum_k up_k^s) \cdot \min\left\{ \frac{X_i}{(\sum_k X_k) - Z}, \frac{1}{2} \right\}. \tag{2.4}$$

To simplify the inequality more, we can express the number of exchanges X_k as the ratio of capacity used for uploading in one correct round divided by the size of a file piece. Formally, we have

$$X_k = \frac{T}{B} \cdot up_k^l, \tag{2.5}$$

for all $k \in L$.

Indeed, due to our assumption, for each leecher L_k there are enough neighbors in the system that can exhaust L_k's upload capacity. Due to the fact that values X_k, with $k \in \{1, \ldots, n\}$, are defined for the case when everybody is honest, and following the protocol implies exchanging with the neighbors until the full upload capacity is exhausted, we have that (2.5) holds.

We are now ready to distinguish two cases for the current proof. In the first case, the deviating leecher increases the benefit from the seeders, but stays below $\frac{1}{2}$ of the overall seeding capacity. In the second case, the reward is capped by the tracker at exactly half of this capacity. Note that those computations below rely on the assumption that seeding power is at most twice as large as leeching power.

Case 1: $\min\{\frac{X_i}{(\sum_k X_k)-Z}, \frac{1}{2}\} = \frac{X_i}{(\sum_k X_k)-Z}$. Then (2.4) evaluates to

$$\begin{aligned} &\frac{Z \cdot B \cdot (\sum_k X_k) \cdot ((\sum_k X_k) - Z)}{T \cdot (\sum_k up_k^s)} \geq Z \cdot X_i \\ \Leftrightarrow\ &\underbrace{\frac{(\sum_k up_k^l)}{(\sum_k up_k^s)}}_{\geq \frac{1}{2}} ((\sum_k X_k) - Z) \geq X_i. \end{aligned}$$

The last inequality holds as we are in case 1.

Case 2: $\min\{\frac{X_i}{(\sum_k X_k)-Z}, \frac{1}{2}\} = \frac{1}{2}$, then (2.4) evaluates to

$$Z \cdot B + T \cdot (\sum_k up_k^s) \frac{X_i}{\sum_k X_k} \geq \frac{T}{2}(\sum_k up_k^s)$$
$$\Leftrightarrow \underbrace{\frac{\sum_k up_k^l}{\sum_k up_k^s}}_{\geq \frac{1}{2}} Z \geq \frac{1}{2}(\sum_k X_k) - X_i.$$

The above inequality is fulfilled as we are in case 2. It follows that

$$u_i(k, M_i^* \|_n^r M_i, \boldsymbol{M}_{-i}) \leq u_i(k, M_i^* \|_{n+1}^r M_i, \boldsymbol{M}_{-i}) + \epsilon_i^{n+1}(k)$$

and this also concludes our induction proof. By summing up all the inequalities described by (2.3), where $n \in \{1, \ldots, R\}$ and taking into account that R is the number of rounds of a polynomially bounded strategy, we obtain there exists a negligible function ϵ' such that

$$\begin{aligned} u_i(k, M_i^*, \boldsymbol{M}_{-i}) &= u_i(k, M_i^* \|_0^r M_i, \boldsymbol{M}_{-i}) \\ &\leq u_i(k, M_i^* \|_R^r M_i^*, \boldsymbol{M}_{-i}) + \epsilon'(k) \\ &= u_i(k, \boldsymbol{M}) + \epsilon'(k). \end{aligned}$$

This concludes our proof. □

Showing that seeders have no incentive to deviate from the protocol is considerably easier than the corresponding statement for leechers, since seeders are considered partially altruistic. We show that as long as all leechers follow the protocol, a seeder cannot run a deviating strategy to improve its payoff.

Lemma 7 (No Seeder Deviation)**.** *There is no deviating strategy for any seeder that increases its payoff if all other parties follow the RatFish protocol.*

Proof. We prove that no matter how a seeder distributes its upload speed over the leechers, their average time to completion does not decrease. In particular, it implies the seeder's utility does not increase if its upload speed initially announced to the tracker decreases.

The statement holds as all other participants stick to their strategy: a seeder may deviate only by assigning leechers different weights than those received from the tracker. Let r be the last round in which the seeder S_j deviates and let up_i^s be its full upload speed. Let $\omega_1, \ldots, \omega_n$ be the weights on the upload speed to all leechers given by the tracker and let $\phi_1, \ldots, \phi_n$ be some other arbitrary weights. It holds that $\sum_{k=1}^n \phi_k \leq \sum_{k=1}^n \omega_k = 1$ and that we have $\omega_k, \phi_k \geq 0$ for all k. We obtain the following bound for the average download speed of all leechers in round r:

$$\begin{aligned} \frac{1}{|L|} \sum_{k \in L} (down_k^l + \omega_k \cdot up_i^s) &= \frac{1}{|L|} \left(\sum_{k \in L} (down_k^l) + up_i^s \right) \\ &\leq \frac{1}{|L|} \sum_{k \in L} (down_k^l + \phi_k \cdot up_i^s). \end{aligned}$$

Thus, if a seeder deviates from RatFish, it causes a decrease in the average download speed in round r for the leechers. Consequently, the average completion time for a leecher does increase when the seeder deviates, i.e., the seeder has no incentive to deviate from RatFish. □

We finally combine the results that neither leechers nor seeders have an incentive to deviate (the tracker is honest by assumption) to establish our main result.

Theorem 8 (Computational Nash Equilibrium)**.** *The RatFish protocol constitutes a computational Nash equilibrium if* $\alpha_i \in [0, \frac{T}{\delta \cdot \gamma \cdot B}]$ *for all* $i \in L$.

Proof. We show that for every participant in the protocol, a deviation increases the utility by at most a negligible value. Assume leecher L_i deviates from the strategy M_i described by the protocol using M_i'. Then, by Lemma 5, we have that there exists a sem-rational strategy $M_i^* \in \{H_{ack,j}^*\}_j$ such that for some negligible function ϵ_1 it holds

$$u_i(k, M_i', \boldsymbol{M}_{-i}) - u_i(k, M_i^*, \boldsymbol{M}_{-i}) \leq \epsilon_1(k).$$

However, by Lemma 6, we have that for all M_i^* and for some negligible function ϵ_2,

$$u_i(k, M_i^*, \boldsymbol{M}_{-i}) - u_i(k, \boldsymbol{M}) \leq \epsilon_2(k).$$

Therefore, we obtain that for all deviating strategies M_i' there exists a negligible function ϵ such that

$$u_i(k, M_i', \boldsymbol{M}_{-i}) - u_i(k, \boldsymbol{M}) \leq \epsilon(k).$$

Lemma 7 gives us an analogous result for seeders. Therefore, RatFish gives a computational Nash equilibrium. □

After we have shown how to derive by hand the rather long and intricate proof that RatFish represents a Nash equilibrium, an important observation is due. First, it is not clear at first glance that our case analysis technique is exhaustive. Unfortunately, since the protocol is rather complex in terms of number of different messages which are sent and received and also in terms of the side effects that they may incur, it is not clear how to perform an exhaustive case analysis by hand. Second, using a tool or a method off-the-shelf that performs an automatic verification of the game theoretic property, namely computational Nash equilibrium, for a protocol that involves cryptographic primitives and assumptions, has not been, to the best of our knowledge, attempted yet. Thus, in the current chapter we have seen an analysis using only known or adapted proofs techniques and we study how this analysis can be automated in Chapter 3.

2.6 Implementation and Performance Evaluation

In this section, we describe the implementation of RatFish and we experimentally evaluate its performance. We focus on the implementation and performance evaluation of the tracker, since the tracker took on several additional responsibilities and is now involved

in every exchange. In contrast to the tracker, seeders and leechers are largely unchanged when compared to BitTorrent: the exchange of encrypted pieces constitutes a small computational overhead, but leaves the network complexity that usually constitutes the performance bottleneck of P2P protocols essentially unchanged.

2.6.1 Implementation

The RatFish tracker was implemented using about 5000 lines of code in Java, thus ensuring compatibility with common operating systems. The implementation is designed to work with both UDP and TCP.

The messages sent in the protocol start with the protocol version number and message ID (which determines the length of the message), followed by the torrent ID, and additional information that depends on the type of message.

Afterwards, a task is created that processes the received message. This task is given to the threadpool executor – the main part of the RatFish tracker that also ensures parallelization. The threadpool sustains eight parallel threads and assigns new tasks to the next free thread. For instance, when the tracker receives a notification that a leecher joined the protocol, the task extracts the leecher's IP from this message and triggers the forced wait. After $T = 300$ seconds it replies with a digital signature for the leecher using an RSA-based signature scheme that signs SHA-1 hashes.

2.6.2 Experimental Setup

For the evaluation, we ran the RatFish tracker on a server with a 2-cores Intel Xeon CPU, 2GB of RAM, a 100MBit Internet connection and an Ubuntu SMP operating system with kernel version 2.6.28-18. We simulated a swarm with up to 50,000 peers, divided into neighborhoods of size 4. The simulated leechers send the same messages as a real leecher would, thus yielding an accurate workload measure for the tracker. Every leecher was configured to complete one exchange per second, and we chose the size of a piece to be 256 kB according to BitTorrent standards. Hence every leecher has a virtual upload speed of 256 kB/s. The size of the shared file is 50 MB, and the seeders upload one forth of the file per second in a round-robin fashion to their neighborhoods. The simulated clients are running on a separate machine. This allows us to measure network throughput. In our simulation, we need to pretend to the tracker that clients connect from different IPs. We thus used UDP in our experiments. Deploying RatFish in reality would presumably be based on TCP, which would slightly increase the network complexity.

2.6.3 Performance Evaluations

Fig. 2.10 depicts the results for our experiments. The main observation, shown in the left part of Fig. 2.10, is that even though we engage the tracker in every exchange in the swarm, the protocol scales well (a resource usage of 65% for 50,000 leechers). One can also observe that the computation complexity becomes a limiting factor, but we expect

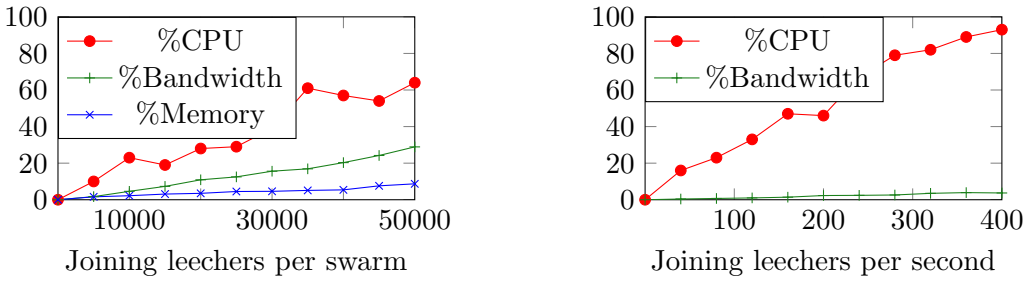


Figure 2.10: **Left:** Resource usage for a static number of leechers engaging in exchanges. **Right:** Resource usage for dynamically joining leechers.

this to change for more cores given our highly parallelized implementation. Memory was not a bottleneck in any experiment.

The right part of Fig. 2.10 considers the case where many leechers join at once, but no exchanges are happening. This study is important since the tracker's most expensive task is to sign the join time of leechers. In our experiments, the tracker was able to serve about 400 new leechers per second. Since the server has T seconds for signing in practical deployments, the signature computation would be scheduled in free CPU time and hence not delay ongoing exchanges. We also observed that the two measurements depicted in Fig. 2.10 on CPU usage are additive, e.g., a system with 30,000 leechers and 200 joining leechers per second uses 90% of the CPU.

2.7 Conclusion

We have proposed a file sharing protocol called RatFish and we have proven it is secure against deviations of rational users. We first characterized rational behaviors of leechers and seeders in file sharing protocols. Subsequently, we formally showed that no rational party has an incentive to deviate from RatFish; i.e., RatFish constitutes a Nash equilibrium. While the tracker in RatFish is assigned additional tasks compared to existing file sharing protocols such as BitTorrent, the communication overhead of a RatFish client compared to a BitTorrent client is minimal. We have demonstrated by means of a prototype implementation that RatFish is practical and efficient.

A currently open research area is creating a framework for automated verification of Nash equilibrium property for ProVerif. Indeed, while we have seen that such a framework would tremendously simplify and provide rigor to the existing hand-made proofs, there is no automated verification model yet which can be directly applied to RatFish. The need for automated verification actually extends further, to general rational cryptographic protocols: The only rational cryptographic protocol which has been subject of formal analysis is rational exchange [21, 103]. However, even for rational exchange protocols no automated analysis exists. In Chapter 3 we give the first framework for automated verification of rational cryptography by showing how rational protocols and their game-theoretic properties can be modeled in applied pi calculus.

Chapter 3

Automated Verification for RatFish

3.1 Introduction

As presented in Chapter 2, rational cryptographic solution concepts and methods are used for designing secure protocols with respect to a non-traditional adversarial model, called rational model, where participants do not arbitrarily misbehave for the sake of breaking the security goals of the protocol, but just for maximizing their benefit. The rational model is applicable to scenarios where participants have a predictable behaviour and we have studied the example of file sharing. The main benefits of using the rational model over the traditional "black and white" adversarial models, in which parties are either honest or arbitrarily compromised, is that rationally secure protocols are in general less complex, more efficient or even overcome classical impossibility results.

Formally proving rationality in the presence of cryptographic primitives is, however, neither an easy, nor a well studied task. Indeed, as highlighted in Chapter 2, proofs of Nash equilibrium for rational cryptographic protocols are lengthy, difficult, and error-prone, since they combine the complexity of game theory proofs with the complexity of cryptographic proofs. Specifically, a deviating party may perform arbitrary cryptographic operations and combine in an unexpected manner the messages received from the other parties. Despite the impressive progress in the automated analysis of cryptographic protocols, security proofs for rational cryptographic protocols are at present done by hand. The main problem is that the existing cryptographic protocol verifiers deal with a different adversary model, the so called Dolev-Yao adversary [34], which models an omnipresent and omnipotent network-level adversary that can overhear, intercept, and synthesise any message, is only limited by the security constraints of the employed cryptographic schemes, and unconditionally tries to break the security goals of the cryptographic protocol. In fact, the automated verification of rational cryptographic protocols, and in particular making utility functions accessible to existing cryptographic protocol verifiers, is recognized as an open challenge [5].

3.1.1 Contributions

In this chapter, we take the first steps towards the automated verification of rational cryptographic protocols. We model these protocols in the applied pi calculus [1], a well-established process calculus for the specification and analysis of cryptographic protocols. In order to make the security properties of these protocols amenable to automated verification, we formalize the concept of utility functions and Nash equilibrium as trace properties.

To exemplify our approach, we model the Ratfish rational file sharing protocol [8] in the applied pi calculus and analyse it using ProVerif [19], a state-of-the-art cryptographic protocol verifier. Ratfish relies on a fairly sophisticated utility function based on the length of messages, which expresses the interest of rational parties in completing the download of their files while minimizing the amount of uploaded data. We show how to manually encode a Nash equilibrium property into the ProVerif query language, based on the RatFish utility function. More explicitly, we use the concept of correspondence assertions [107], a formalism originally developed to state intended properties of cryptographic authentication protocols. The analysis of the resulting query is automated and takes about a minute.

To the best of our knowledge, this is the first attempt to leverage an automated cryptographic protocol verifier in the analysis of rational cryptographic protocols.

3.1.2 Related Work

In the following, we give a detailed review of the related work in the areas of automated theorem proving and model checking for fairness and related rationality properties.

Model Checking Nash Equilibria

The related research [77, 78] which comes closest to our goal in this chapter is describing how model checking methods can be used for automated verification of Nash equilibrium property for general distributed systems. When protocol specifications for each protocol participant can be modeled as a finite state machine, model checking algorithms exist [77] for verification of the Nash equilibrium property without coalitions.

In the follow-up paper [78], the framework is extended to reason also about Nash equilibrium in the presence of coalitions. More precisely, using the proposed framework, it can be inferred whether a protocol for distributed systems is an ϵ- f- q- Nash equilibrium, where ϵ is the maximum gain of a coalition in deviating from the protocol, f is the maximum number of Byzantine players and q is the maximum size of the coalition of rational players. The authors propose two model checking algorithms on top of the NuSMV model checker and implement the algorithms on two examples. However, in both of these examples, the representation of the underlying finite state mechanism comprises no more than 18 participants for which no more than 2 states and 3 actions are possible, or 6 participants for which no more than 3 states and 4 actions are possible. The verification algorithms may require up to 30 hours. Moreover, the examples modeled

contain no cryptographic primitives.

Another related approach is taken by Chen et al. [27] and they present an automatic verification technique for the modelling and quantitative analysis of probabilistic systems that incorporate competitive behaviour. Specifically, the authors present a model checking algorithm, implemented in the PRISM model checker, for verifying quantitative properties of stochastic multi-player games.

Our approach differs in three major ways from the models and techniques outlined above: First, our approach is aimed at automated verification of rational cryptographic protocols rather than verification of general distributed systems' protocol. This implies our model requires a way for defining semantics for the symbolic cryptography used within the protocols. For example, we need to model the fact that in case the attacker is able to gain access to the correct keys, then he can retrieve the corresponding secrets. We also need to model various security properties for the cryptographic schemes involved in the protocol, e.g., secrecy, authentication, unforgeability.

Second, we use ProVerif, an automated verification tool for security properties which allows for verification of an unbounded number of process replication. In contrast, the previously used approaches of employing model checkers do not have this capability.

Third, in case of rational players, even though they have the capability to arbitrarily deviate, they will do so only if they benefit from it. Thus, intuitively, since we investigate rational exchanges, we have to reason only about those traces where a rational player has the incentive to choose a different behavior than the prescribed one. On one hand, a model checker is more suitable to use for a brute-force search on the entire space of the traces. One the other hand, ProVerif allows us to reason about traces with certain properties, for example by using correspondence assertions. Thus, our approach is more suitable for situations where efficiency has priority. Indeed, verifying the rational security properties of the RatFish file sharing system using our framework takes less than two minutes.

Computation with Fairness

The rational exchange protocols which we automatically verify in this chapter are part of a wider line of research related to security and fairness. Indeed, rational exchanges and, more generally, rational cryptography, e.g. [8, 39, 48, 53, 63] aim at providing incentives for self interested participants to follow the protocol as designed. The most common utilities for such parties can be resumed intuitively as follows: "I am happy when I learn a certain secret value and if this happens, I am even happier when the other participants in the protocol do not learn the secret value I am interested in. However, I am unhappy when I do not learn the secret value I am interested in." Generally, rational cryptography proposes protocols that achieve roughly the following property : "I can learn the secret value I am interested in if and only if the other parties also learn the secret value(s) they are interested in."

In this context, it becomes clear that rational cryptography and different degrees of fairness (informally, for example, either all participants receive at least a certain "amount" of the correct output of the protocol or nobody does receive anything) are

well connected: Rational cryptographic problems can be viewed as instances of fairness problems which occur in the presence of rational players. Thus, below we review previous work on achieving and verifying fairness in various non-rational settings and afterwards we continue with an overview of the state-of-the-art verification for fairness properties in rational contexts with a focus on rational exchanges.

It is a known fact that under the appropriate assumptions, essentially any polynomial time function can be computed by a protocol in such a way that any desired security property is fulfilled, even in the presence of a malicious majority [43, 62, 110]. This holds as long as we do not consider the *complete fairness* property, i.e., either all participants receive the complete and correct output of the protocol, or nobody does. Indeed, it has been shown [31] that even a coin toss (and hence the boolean XOR) cannot be computed fairly with a dishonest majority. In particular, fairness becomes a non-trivial problem for the two party case.

On one hand, it has been shown that any arbitrary functionality can be implemented with complete fairness when the majority of protocol participants are honest [12, 16, 26, 94]. On the other hand, for the case of dishonest majority, it has been shown [40] that any functionality can be implemented with any number of corrupted parties, as long as only the weaker property of fairness with abort is required (i.e., the adversary in this case may abort the protocol when it receives its output). In the same line of work, another possibility to achieve some degree of fairness is by applying the gradual release technique[1], e.g. [14, 20, 46]. However, this will ensure only partial fairness [2]

Recently, it has been shown that the impossibility result from [31] for the dishonest majority does not apply to every polynomial time computable function. Indeed, an entire line of work, e.g., [49, 50, 51, 52] has been dedicated to classifying which functions can be computed with complete fairness in the presence of malicious (majority) of players and for which functions only partial fairness guaranties can be made.

In more detail, for the two party setting it has been shown [49] that the boolean functions AND and OR, as well as Yao's millionaires' problem [108] can be computed with complete fairness. These are just concrete example functions that can be computed with complete fairness. A general result [49] shows that any two-party function (over polynomial-size domains) which does not contain an "embedded XOR" (i.e. inputs x_0, x_1, y_0, y_1 such that $f(x_i, y_j) = i \oplus j$), can be computed with complete fairness. Moreover, the authors also prove that there exist two-party functions containing an "embedded XOR" that can be computed with complete fairness. The first study of implementing fairness in multiparty setting [51] gives a protocol for computing the 3-party majority function and the boolean OR with complete fairness, but each of them require at least logarithmic number of rounds and a private broadcast channel.

Another way to look at fairness is to understand which are the minimal assumptions that have to be fulfilled in order for this security property to hold when more than half of the participants are corrupted. When looking at the bigger picture, it is

[1] Gradual release informally means that at any point in the protocol, both the honest and the adversarial parties can obtain their output by performing similar amounts of effort.

[2] In case of coin flip, partial fairness means that the coin output by the correct parties is biased.

known [43, 62, 110] that for achieving computational security (without fairness) for any function with dishonest majority, it suffices for the participants to have access to an oblivious transfer functionality. Actually, it has been shown [62] that an oblivious transfer functionality is both necessary and sufficient for computational security, when there is a majority of dishonest participants. We say that oblivious transfer is a complete primitive for computational security without fairness.[3] The first complete primitive for fair computation for two-party and multi-party setting has been described in [38], but its input size and running time depend on the complexity of the target functionality, however, achieving completely fair computation using a more efficient primitive is possible [50]. The functionality used in the two-party case is a fair reconstruction procedure for secret sharing. The input to the primitive depends on the security parameter and on the output of the desired function to be computed fairly. One can compute any multiparty function with complete fairness [50]. However, an additional efficiency trade-off has to be considered between the input size and number of times the fair reconstruction procedure for secret sharing is invoked.

Formal Methods, Fair Exchange and Rational Exchange

From the field of fair computation, the results in this chapter have the most in common with fair exchange and rational exchange. Intuitively, an exchange between two parties mutually interested in a piece of information in possession of the other party is fair if the outcome of the exchange is such that either both parties get what they want or none of them does. It has been shown that in general fair exchange has strong limitations, i.e. it is impossible to implement without a third trusted party [84]. Various fair exchange protocols have been designed, e.g., [11, 15, 64, 65, 79, 104, 111]. In order to cope with the computational overhead of the trusted third party, the notion of optimistic fair exchange [6, 7] can be used instead. Optimistic fair exchange ensures that if all parties follow the protocol, all parties receive the information item they were interested in. Otherwise, in the improbable event that a party deviates, then the party catching this deviation can gather evidence from the protocol about the Byzantine behavior of the deviating party and take this to a judge or arbiter. Based on correct evidence, the judge rules in favor of the non-deviating participant, which as part of the protocol, is eligible for compensation.

The automated verification of fair exchange protocols is subject of active research, since the seminal works by Shmatikov and Mitchell [98, 99, 100] and by Kremer and Raskin [66, 64]. The former use *Mur*φ, a finite-state model checker for the verification of a fair exchange and contract signing protocols. Similar to the work presented in this chapter, they rely on a Dolev-Yao adversary to model possibly deviating parties, so that the adversary and the deviating parties share their knowledge. The latter use MOCHA, a model-checker for alternating transition systems, formalizing cryptographic protocols and their security properties in a game-based setting. The analysis is precise and covers a wide range of security properties, such as non-repudiation and timeliness. None of these

[3] As in the case of impossibility of realizing complete fairness in the plain model [31], there is also an impossibility result for realizing computational security, in general, in the plain model [28].

works, however, deals with rational cryptographic protocols and the associated utility functions.

An alternative to fair exchange is rational exchange. Intuitively, an exchange is rational if the participating parties have no incentives to deviate. Rational exchange protocols [103, 21] have already been subject of formal, but not automated, analysis. Alcaide [4] studies a few existing rational exchanges [103, 21] and enhances them with a game-theoretic model of utilities for the participating parties. She also presents an in-depth Bayesian analysis of various parameters related to the utilities in order to ensure Nash equilibrium. The proofs, however, are done by hand and it is not clear how to generalize that framework to arbitrary rational cryptographic protocols. Buttyán et al. [22] study the connections between the two different notions of fair exchange and rational exchange, which the authors redefine in terms of Nash equilibrium. They show that fairness implies rationality, but not viceversa. They also give an in-depth game-theoretic analysis of Syverson exchange protocol [103] and they conclude that this is a rational, but not a fair exchange protocol. The aim of their work is to define a sound and expressive game-theoretic model for designing and proving rationality for exchange protocols, but the analysis is not automated and tailored to the class of fair exchange protocols.

Fairness as a Game for Model Checking

Finally, automatic verification of fairness, modeled as a game between two players, has been also investigated in the context of model checking. For example, verifying whether a formula holds true can also be modeled as a two player game [67, 58]. On the one hand, the verifier aims to prove that the formula holds. On the other hand, the refuter aims to find a sub-formula that does not hold such that due to this the entire formula cannot hold. Such a game, called Hintikka game, has been used in connection with concurrent systems subject to fairness constraints, such as "Every action that is always enabled, is eventually executed."

3.1.3 Outline

This chapter is structured as follows: Section 3.2 reviews the applied pi calculus. Section 3.3 formalizes the concepts of rational cryptographic protocols and Nash equilibrium in applied pi-calculus. Section 3.4 illustrates our technique at work on RatFish. In Section 3.5 we give the conclusions for this chapter.

3.2 Applied Pi Calculus (Review)

In this section we briefly recall the syntax of applied pi calculus [1], we give an informal account of its operational semantics, and we introduce the notations used in the rest of the chapter.

The syntax of the calculus is shown in Table 3.1. Terms are defined by means of a *signature* Σ, which is a set of function symbols, each with an arity. The set of terms

$M, N ::=$	*terms*
a, b, c, m, n	*names*
x, y, z	*variables*
$f(M_1, \ldots, M_l)$	*function applications*
$P, Q, R, T ::=$	*processes*
0	*null process*
$P \mid Q$	*parallel composition*
$!P$	*replication*
$\nu n.P$	*name restriction*
if $M = N$ *then* P *else* Q	*conditional*
$N(x).P$	*message input*
$\bar{N}\langle M\rangle.P$	*message output*
$P\{N/x\}$	*substitution*
$ev\ M.P$	*event*

Table 3.1: Syntax of the calculus

is the term algebra built from names, variables, and function symbols in Σ applied to arguments. The language supports an arbitrary *equational theory* E for terms: we write $E \vdash M = N$ for equality and $E \nvdash M = N$ for inequality modulo E. Equational theories are an expressive tool to describe a variety of cryptographic operations: for instance, we can model symmetric-key cryptography with an equational theory E such that $E \vdash \mathsf{dec}(\mathsf{enc}(M, K), K) = M$.

Processes are defined as follows. The null process $\mathbf{0}$ does nothing and is usually omitted from process specifications; $P|Q$ executes P and Q in parallel; $!P$ generates an unbounded number of copies of p; $\nu n.P$ generates a fresh name n and behaves as P; *if* $M = N$ *then* P *else* Q behaves as P if $E \vdash M = N$ and Q otherwise; $c(x).P$ receives message N on channel c and behaves as $P\{N/x\}$; $\bar{c}\langle M\rangle.P$ outputs a message M on channel c and then behaves as P; $evM.P$ raises the event M and then behaves as P.

As usual, the scope of names and variables is delimited by restrictions and inputs. We write $fv(P)$ for the free variables and $fn(P)$ for the free names in a process P. We say that a process is closed if it does not have free variables. We let $\widetilde{M}$ denote an arbitrary sequence $M_1, \ldots, M_k$ of terms and $\nu\widetilde{n}$ a sequence $\nu n_1 \ldots \nu n_k$ of name restrictions.

As for the pi calculus, the operational semantics of the applied pi calculus is defined in terms of *structural equivalence* and *the reduction relation.* Structural equivalence ($P \equiv Q$) captures rearrangements of parallel compositions and restrictions, and the equational rewriting of the terms in a process. The reduction relation ($P \to Q$) formalizes the semantics of process synchronizations and conditionals.

Next, we introduce labeled transitions $P \xrightarrow{\alpha} Q$, where α has two possibilities: either $\alpha = ev\ M$ or α is an empty label. We require that $P \to Q$ if and only if there exist an α

as defined above such that $P \xrightarrow{\alpha} Q$. Finally we define execution traces in the context of the above defined semantics: A trace t captures all the events raised during process reduction. We write $t \in Tr(P)$ to denote the possibly infinite set of execution traces of P.

In the rest of this chapter we rather need the transitive and reflexive closure of the reduction relation $P \to Q$ and we denote it by $P \to^* Q$. Finally, we introduce the labeling for the transitive and reflexive closure of $P \to Q$. The requirement now is that $P \to^* Q$ if and only if there exists a trace t such that $P \xrightarrow{t}{}^* Q$.

3.3 Rational Cryptography in the Applied Pi Calculus

In this section, we show how to model rational cryptographic protocols and to formalize the concept of Nash equilibrium in the applied pi calculus.

Intuitively, we model rational cryptographic protocols as processes of the form $P = \nu\widetilde{n}.(P_1 \mid \ldots \mid P_m)$, where the sequence $\widetilde{n}$ of restricted names contains the cryptographic keys and the communication channels used by the protocol parties $P_1, \ldots, P_m$. In the following, for every $i \in \{1, \ldots, n\}$ we let $cin(P_i)$ and respectively $cout(P_i)$, denote the set of terms used as input and respectively as output channels in P_i. From a semantic perspective, in the following, unless states otherwise, we always assume all free names and bound names are distinct. Indeed, this is easy to achieve using a suitable alpha-renaming.

Definition 9 (Rational Cryptographic Protocol)**.** *A closed process P is called a rational cryptographic protocol if it has the form $P = \nu\widetilde{n}.(P_1 \mid \ldots \mid P_m)$ and for every $i \in \{1, \ldots, m\}$ it fulfills the following properties:*

1. *$cin(P_i) \cup cout(P_i) \subseteq \widetilde{n}$ (communication channels are private);*
2. *$cin(P_i) \subseteq \bigcup_{j \in \{1,\ldots,m\}, j \neq i} \{c_{j,i}\}$ and $cout(P_i) \subseteq \bigcup_{j \in \{1,\ldots,m\}, j \neq i} \{c_{i,j}\}$ (communication channels are all distinct);*
3. *for all contexts $C[]$, processes Q, and indexes j such that*
   (a) *$P_i = C[c_{j,i}(x).Q]$, we have that $Q = ev\ Recv(i, j, x, \ldots).Q'$ for some Q'.*
   (b) *$P_i = C[\overline{c_{i,j}}\langle M\rangle.Q]$, there exists a context $D[]$ such that*

$$C[] = D[ev\ Send(i, j, M, \ldots).[]].$$

Condition 1 requires the communication channels used by each protocol party to be globally restricted names, i.e., unknown to the adversary. Condition 2 names such channels: $c_{i,j}$ is the channel used by P_i to send messages to P_j. Condition 3 rules the usage of events: party i receiving message x from party j is tracked by the event $Recv(i, j, x, \ldots)$, which may take additional arguments as input, while party i sending message M to party j is tracked by the event $Send(i, j, M, \ldots)$.

In the model presented so far, all parties are honest and there is no way for them to deviate from the protocol. We model rational players, who may arbitrarily deviate from

the protocol if they have an interest in doing so, by putting them under the control of the adversary. This is achieved by replacing, for example, the corresponding process P_i with the null process and by removing all the secrets used in P_i from the initial sequence $\widetilde{n}$ of restricted names, including the communication channels. This allows the adversary to take control over rational player P_i and to act on P_i's behalf with honest parties.

Definition 10 (Party Corruption)**.** *Let $P = \nu\widetilde{n}.(P_1 \mid \ldots \mid P_m)$ be a rational cryptographic protocol. For every $i \in \{1, \ldots m\}$, we define the process transformation f_i as follows:*

$$f_i(P) = \nu\widetilde{n'}.(P_1 \mid \ldots \mid P_{i-1} \mid 0 \mid P_{i+1} \mid \ldots \mid P_m), \text{ where } \widetilde{n'} = \widetilde{n} \backslash fn(P_i).$$

We say that f_i is the corruption function *for process P_i.*

Rational parties deviate from the protocol if they have an interest in doing so, which is formally captured by the utility function. It is natural to express the utility function in terms of execution traces. Moreover, as we will see in the formal definition of utility function, it is sometimes sufficient to restrict our attention to the actions of a single rational player. For this reason, we define a projection operation $trace_i$ on traces, which given a trace t returns the subtrace of t composed of the events referring to party i. Since we use this function in a setting in which party i is under the control of the attacker and her events are also controlled by the adversary, they should not occur in the trace. Thus, we only look at the events of the parties exchanging messages with party i.

Definition 11 (Trace Projection)**.** *For a process P and for every $t \in Tr(P)$, we define the projection operation $trace_i$ as follows:*

$$trace_i(t) = \begin{cases} \alpha :: trace_i(t') & \text{if } t = \alpha :: t' \text{ and } \alpha \in Event(j,i) \\ trace_i(t') & \text{if } t = \alpha :: t' \text{ and } \alpha \notin Event(j,i) \\ \epsilon & \text{if } t = \epsilon \end{cases}$$

where for every $i, j \in \{1, \ldots, m\}$ with $i \neq j$, we denote:

$$Event(i,j) = \bigcup_{message\ M} \{Send(i,j,M), Receive(i,j,M)\}.$$

The definition above says that for a trace t, an event initiated by a party P_j and designated for party P_i is part of the trace projection $trace_i(t)$. However, all other events, where the pair (sender, recipient) is different from pairs of type (j,i), are not a part of $trace_i(t)$.

We are now ready to formalize the concept of utility in the applied pi calculus. The utility of party i is a function u_i mapping execution traces to real numbers. Utility functions can be *non-envious*, if they are only defined on the actions of the respective party, or *possibly envious*, if they take into account the actions of other parties.

From a technical perspective, there is a subtlety related to utility functions in computational settings that needs to be incorporated also into any utility definition

for applied pi calculus. Indeed, in the computational setting, when discussing utilities for honest participants for a non-deterministic protocol, we actually refer to *expected utilities*, as described in Section 2.3. If we try to translate the expected utility from the computational setting directly to the applied pi setting, then we have to compute the average of the *exact utility values* of each trace t for a given protocol. However, such an approach cannot be used in relation with automated verification tools as the number of traces may be exponential in the security parameter or even infinite.

The alternative is to construct in the applied pi setting another counterpart for the expected value from the computational setting. At first glance, it seems that a good candidate for utility in the symbolic world is the maximum of all exact utility values obtained for a non-deterministic honest protocol, if such a value exists. However, there are non-deterministic protocols where even when participants follow them honestly, the trace t which gives the highest benefit occurs only with negligible probability, while all other traces give only small benefit. To complicate matters even further, it can be the case that while a trace of an honest protocol gives a participant a high benefit, for all the others, the benefits are small. Thus, there are protocols that even when all participants are honest, there exists no trace that simultaneously maximizes the benefit for all of them. As we will see shortly, such a design choice for an honest protocol hinders us from defining the Nash equilibrium property in the applied calculus.

In order to avoid the situation described above, we define the *payoff* if and only if the honest protocol has the following property: For each participant i there exists a certain value r_i such that independent of where we are in the execution tree of an honest non-deterministic protocol, there is alway a sub-trace leading to the benefit r_i for participant i, and there is no sub-trace that leads to a higher benefit than r_i for participant i. More formally, we have:

Definition 12 (Utility and Rational Setting)**.** *Let $P \equiv \nu\widetilde{n}.(P_1 \mid \ldots \mid P_m)$ be a rational cryptographic protocol.*

- *The utility for party i is a function $u_i : Tr(P) \to \overline{\mathbb{R}}$ from traces to real numbers*[4]*.*
- *We say that u_i is a* non-envious utility *if for all traces $t \in Tr(P)$, $u_i(t) = u_i(trace_i(t))$; otherwise we say that u_i is a* possibly envious utility*.*
- *For every $k \leq m$, we call the pair $(P, (u_i)_{i\in\{1,\ldots,k\}})$ a* rational setting *with k rational players (the remaining players are said trusted and do not deviate from the protocol).*
- *If for every $i \in \{1, \ldots, m\}$, there exists $r_i \in \overline{\mathbb{R}}$ such that for every trace t and for every process Q with $P \xrightarrow{t}{}^{*} Q$ there exists $t' \in Tr(Q)$ such that $u_i(t :: t') = r_i$ and there is no $t'' \in Tr(Q)$ such that $u_i(t :: t'') > r_i$, then we say $(r_1, \ldots, r_m)$ is the payoff vector for P and r_i is the payoff for party i.*

We now formalize the concept of Nash equilibrium in the context of the applied pi calculus.

[4]We use $\overline{\mathbb{R}}$ to denote the set of real numbers extended with the infinity symbols, i.e., $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$.

Definition 13 (Nash Equilibrium)**.** *Let $(P, (u_i)_{i\in\{1,\ldots,k\}})$ be a rational setting with k rational players. We say that P is Nash equilibrium for $(u_i)_{i\in\{1,\ldots,k\}}$ if for every $i \in \{1,\ldots,k\}$, adversary A, and $tr \in Tr(f_i(P)|A)$, we have that $u_i(tr) \leq r_i$, where r_i is the payoff of party i.*

The framework presented so far captures the traditional adversarial model for rational cryptographic protocols, which comprises rational parties who may arbitrarily deviate from the protocol in order to maximize their payoffs. It is interesting to observe that we can easily modify our framework to deal with a stronger adversarial model, which combines rational players and Dolev-Yao adversary. We have just to modify the process modeling the rational cryptographic protocol (cf. Definition 9) in such a way that the communication channels are free, as opposed to restricted, and thus available to the adversary. In this way, protocol parties communicate over untrusted channels and rational parties can collude with the Dolev-Yao adversary to maximize their payoff.

3.4 Rational Exchange for File Sharing Protocols

We exemplify our verification framework on a simplified version of the RatFish rational file sharing protocol [8]. First we review this protocol together with the utilities associated with the protocol participants (Section 3.4.1). Next we model the protocol in the applied pi calculus (Section 3.4.2). Finally we show how to conclude the Nash equilibrium property using ProVerif [19], a state-of-the-art cryptographic protocol verifier (Section 3.4.3).

3.4.1 RatFish Protocol

Preliminaries. We start by giving a brief reminder on file sharing protocols. (More details are covered by Chapter 2.3) In the following, we call a player in the file sharing game a *peer*. The peers are of two types: A *seeder*, denoted S, uploads a *file* f that it owns to other peers, whereas a *leecher*, denoted L_i, downloads f. The communication among peers is mediated by a trusted party called the *tracker*, T. We assume that the tracker's IP address is known to all peers.

The file f consists of pieces $f_1, \ldots, f_N$, each of length B bytes. All participants in the file sharing protocol (i.e., leechers, seeders, tracker) hold the values $h_1 = h(f_1), \ldots, h_N = h(f_N)$, where h is a publicly known *hash function*. When deployed in practice, this publicly known information is distributed via a *metainfo file*. The tracker is only responsible for coordinating peers that are exchanging the same file. Different peers may have different upload and download capacities, which is usually expressed by the number of peers a leecher can upload to and download from, simultaneously.

Utility Functions. We now describe the utilities associated with each peer type in RatFish. On the one hand, the seeders are interested in sharing a file they own with as many other leechers and as effectively as possible. More formally, in RatFish the utility of a seeder is inversely proportional to the average time a leecher needs to download the file f. On the other hand, leechers are interested in completing the download in the most

effective manner. Thus, their utilities are inversely proportional to the time they spend in the system until the download is complete and also to the amount of uploaded data.

In this section, we analyse a simplified version of Ratfish. Since the applied pi calculus does not incorporate a notion of time, we exclude the temporal aspects from the utility functions. In particular, we consider seeders as trusted parties and let the payoff of leechers be directly proportional to the difference between the amount of downloaded data and the amount of uploaded data. The leecher's utility function is formalized below:

Definition 14 (Leecher's Utility Function)**.** *Let $(P, (u_i)_{i\in\{1,\ldots,k\}})$ be a rational setting with k rational players. For a trace $t \in Tr(P)$, the utility u_i of a leecher L_i is defined as*

$$u_i(t) = \begin{cases} -\infty & \text{if the complete file was not downloaded} \\ \alpha_i + \beta_i \cdot (p_{down} - p_{up}) & \text{otherwise} \end{cases}$$

where

- *α_i represents the utility value if leecher L_i has downloaded the file and the number of uploaded file pieces equals the number of downloaded pieces;*
- *β_i represents the utility value gained by L_i for each piece downloaded without uploading any piece in return;*
- *p_{down} represents the number of pieces downloaded by L_i from other leechers as result of exchanges;*
- *p_{up} represents the number of pieces uploaded by L_i from other leechers as required by exchanges.*

Notice that the leecher's utility function is non-envious (cf. Definition 12).

Protocol Description. We are now ready to describe our automatic verification technique for rational cryptography at use on a RatFish example protocol. We assume a RatFish system with two leechers, a tracker and a seeder. The seeder owns a file which is split in four equally long pieces and the description of the file together with the hash values for the pieces are publicly available in a metafile. The protocol has *two phases*, the *setup phase* and the *exchange phase*.

In the setup phase, each of the leechers contacts the tracker in order to join the system. In turn, the tracker contacts the seeder and instructs him to upload two pieces of the file for free to each leecher, such that they can proceed to the next phase. The seeder replies to the tracker's request by contacting the two leechers indicated by the tracker, and by giving them 2 different pieces of the file, together with the indices of the file pieces that the other leecher has received. The seeder also notifies the tracker about the indices of the pieces received by each leecher. As part of our model, we assume that the seeder stays in the system as long as the file has not been yet downloaded by the leechers. This is in accordance with the assumptions described in Section 2.5 that ensure the existence of Nash equilibrium for RatFish.

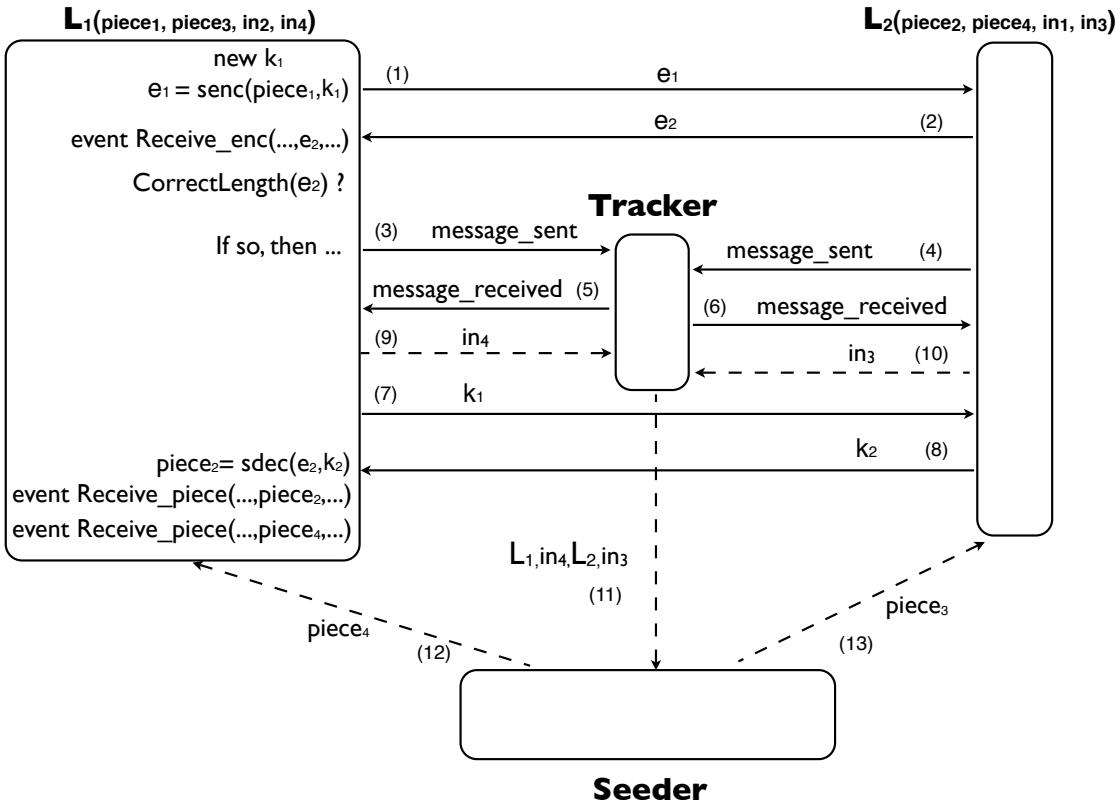


Figure 3.1: Exchange and Rewarding Protocol in RatFish

The exchange phase has *three steps*: the *agreement*, the *exchange* and the *rewarding*. In the agreement step, the leechers send the tracker the index of a piece they lack and the tracker acknowledges whether the other leecher has acquired it in the setup phase or not. If both leechers receive positive acknowledgements, then they proceed to the next step.

The exchange and rewarding phases are displayed in Figure 3.1. Since the protocol is symmetric, we describe it from the view point of one of the leechers, e.g., L_1. Assume L_1 received in the setup phase $piece_1$ and $piece_3$ together with the indices in_2 and in_4 of the missing pieces.[5] Hence L_1 is interested in the file $piece_2$, that according to the agreement step is owned by L_2. L_1 encrypts $piece_1$ with a fresh symmetric key k_1, sends the resulting ciphertext e_1 to L_2, and expects e_2 as a reply (messages (1)-(2)). As we assume that the encryption scheme is length preserving, L_1 checks that the length of the received ciphertext e_2 matches the one of the file pieces. As detailed below, this check is essential for establishing the Nash equilibrium property. Next, L_1 sends a confirmation to the tracker that the exchange of encryptions took place and he expects as a reply a confirmation that the other leecher has also acknowledged the exchange (messages (3)-(6)). The exchange phase is concluded by the exchange of the keys: L_1 sends his decryption key to L_2 and expects a correct key for decrypting e_2 in return (messages (7)

[5]These are represented as input parameters of L_1 in Listing 3.1.

and (8)).

The rewarding phase consists of a number of file pieces that the tracker uploads for free to each of the leechers, proportional to the number of exchanges they have performed. In practice, the rewarding is limited to the most active leechers since the bandwidth of the seeder is limited, the seeder may go offline before rewarding all the leechers, and so on. The exchange phase is depicted by dotted arrows. Each leecher contacts the tracker in order to receive their *reward* from the seeder. In this example, we assume the seeder can reward each leecher after they have completed the exchange with one free piece. More precisely, the protocol works as follows: the leechers send the tracker the indices of the file pieces they would like to receive as reward from the seeder (messages (9) and (10)). The tracker forwards both request to the seeder and adds the identity of the leechers (message (11)). The seeder replies by uploading to each of the leechers the file piece it has requested (messages (12) and (13)). Moreover, we observe that if both leechers follow the protocol as prescribed, then they both obtain the file at the end of the two phases. An important detail to observe here is that the communication between each pair of participants is conducted on secure channels.

3.4.2 Protocol Model in the Applied Pi Calculus

We modelled the RatFish protocol in ProVerif, a cryptographic protocol verifier that accepts as input applied pi calculus processes. Below we discuss the code of the leecher, from which one can derive the code of the other participants by symmetry. Listing 3.1 shows the code for the exchange and rewarding phases. Since the ProVerif implementation of RatFish protocol follows naturally from the protocol details specified in Section 3.4.1, we just discuss the most interesting aspects and we give the full implementation code in the appendix. For the sake of readability, we use distinct communication channels for each message exchange and name them according to the principals involved in the communication (e.g., c_{L1L2} is the channel used by L_1 to send messages to L_2). Similarly, we use only the events required for the Nash equilibrium property and use meaningful names.

A first important detail to discuss is our modeling of message length. For our purposes, it suffices to use four constants (i.e., nullary functions) for describing the length of messages, namely, *hashLength* for the length of hashed messages, *keyLength* for the length of keys, *pieceLength* for the length of file pieces, and *otherLength* for other lengths. In ProVerif, messages are typed and the type of these constants is Length. In the following, we let l range over constants of this type. The messages exchanged on the network have type *key* or they have the type Data and they may be of the form $hash(M)$, $senc(M, K)$, $data(M, l)$ and $keyD(M)$. The term $data(M, l)$ denotes bitstrings of length l and the term $keyD(M)$ denotes that type *key* is just a subtype of *Data*. To exemplify, $piece_i$ is actually of the form $data(filepiece_i, pieceLength)$, where $filepiece_i$ can be seen as the actual piece content. We use the function $length(\mathsf{Data}) : \mathsf{Length}$ to compute the length of a bitstring. This function is defined as follows: $length(hash(N)) = hashLength$, $length(senc(N, K)) = length(N)$ (encryptions are assumed to be length preserving), and $length(data(M, l)) = l$. Similarly, for messages K of type *key*, we have the function

Listing 3.1: Code for Leecher (Exchange phase)

```
let leecher_exchange( piece_1: Data, dex_2: index,
                      dex_4: index, phase_id: phase_id) =
   new k_1: key;
   let e_1: data = senc(piece_1, k_1) in
   out(c_L1L2, e_1);
   in(c_L2L1, e_2: Data);
   event Receive_enc(L1, L2, e_2, dex_2, phase_id);
   if CorrectLength(e_2) then
           out (c_L1t, message_sent);
           in  (c_tL1, =message_received);
           out(c_L1L2, k_1);
           in(c_L2L1, k_2: key);
           let piece_2: Data = sdec(e_2, k_2) in
           event Receive_piece(L1, L2, piece_2,
                                               phase_id).

let leecher_demand_reward( phase_id: phase_id) =
   out(c_L1t, dex_4);
   in(c_sL1, piece_4: Data);
   event Receive_piece(L1, L2, piece_4, phase_id).

let leecher(piece_1: Data, dex_1: index, piece_3: Data,
           dex_3: index, dex_2: index, dex_4: index) =
   (phase 1; leecher_exchange(piece_1, dex_2, dex_4,
                                               phase_1)) |
   (phase 1; leecher_demand_reward(phase_1)).
```

Listing 3.2: Nash Equilibrium Query for Deviating Leecher 2

```
query  dex:index, x:Data;
(attacker((filepiece_1,filepiece_3)) phase 1) ==>
    (event(Send_freepiece(sed, L2, data(filepiece_3,
                                pieceLength))) ==>
    event(Receive_enc(L1, L2, x, dex, phase_1)) &&
    CorrectLength(x)).
```

$length(keyD(K)) = keyLength$.

Finally, we briefly discuss how phases are modelled in ProVerif. The idea is that each part of the process runs in a certain phase and processes can only synchronize (i.e., exchange messages) with processes that are running in the same phase. Phases are denoted by sequential numbers starting from 0. We use two phases: phase 0 for the setup phase and phase 1 for the exchange and rewarding phase.

3.4.3 Automated Verification of Nash Equilibrium

Our overall goal is to verify whether or not a certain protocol constitutes a Nash equilibrium. The utility functions underlying the definition of Nash equilibrium (cf. Definition 13), however, are not supported by ProVerif, nor by any other cryptographic protocol verifier. Therefore, we have to manually encode the Nash equilibrium property based on the Ratfish utility function (cf. Definition 14) into a query accepted by ProVerif. Such an encoding is fairly easy and natural to define. We can easily see that the payoff of leecher i is $\alpha_i + \beta_i$, since the protocol allows each leecher to complete the download of the file by uploading only one piece and downloading one piece from the other leecher and one piece from the seeder in the rewarding phase.

Hence, we have to verify that a deviating party has no way to increase this payoff. The only way to do that would be to download the two missing pieces without uploading any piece. We can easily write a ProVerif query to make sure that this is not possible. Such a query is displayed in Listing 3.2 and is validated on a process modeling leecher L_2 under adversarial control. This query is based on the concept of correspondence assertions [107], which intuitively express relations among events that should hold in all execution traces.

In particular, the successful validation of this query ensures that if the attacker (i.e., leecher L_2) completes the download of $piece_1$ and $piece_3$, then it must have been the case that L_2 has received $piece_3$ from the seeder *sed* and sent a message with the correct length to L_1 The predicate $CorrectLength(x)$ is true if the length of x is the same as the one of file pieces, which is computed according to the previously defined function *length*. The code for this predicate is given in Listing 3.3.

Notice that the adversary has to send a message of the correct length to L_1, but not necessarily the encryption of the file piece L_1 is expecting. However, since the utility function is only defined on the number of exchanged file pieces, a deviating party has no incentive in doing so and, therefore, a rational party will follow the protocol and send

Listing 3.3: Definition of CorrectLength predicate

```
pred CorrectLength(Data).
clauses
    forall m: bitstring;
       CorrectLength(data(m, pieceLength));
    forall x: Data, ke: key;
       CorrectLength(x) -> CorrectLength(senc( x, ke)).
```

the correct encryption. We finally mention that the size of keys and hashes is considered negligible in Ratfish: this explains why the utility function is solely based on the number of exchanged file pieces. This is also the reason why Ratfish is a rational but not fair exchange protocol: in principal, a misbehaving leecher might wait for the key to decrypt the received piece (msg. 7) without sending back its own key (msg. 8). Leechers, however, have no incentive in doing that, since uploading them does not cost anything.

For completeness, the ProVerif scripts are available in Appendix A. The ProVerif code has been run on a machine with the following specifications: CPU Intel Core 2 Duo, 2GB of RAM and MAC operating system, version 10.5.8. Under these conditions, the queries are processed in less than 2 minutes.

3.5 Conclusion

In this chapter we have presented a formal framework for the automated verification of rational cryptographic protocols. We model protocols in the applied pi-calculus, provide symbolic definitions of utility and Nash equilibrium, and use ProVerif to automatically verify that a rational cryptographic protocol constitutes a Nash equilibrium. By analyzing an example of RatFish file sharing protocol, we show how a Nash equilibrium based on a complicated utility function can be captured by a set of trace properties amenable to automated verification.

In our presentation so far we have focused on practical aspects related to rational cryptography: We have shown how to design, implement and automatically verify RatFish, a rational file sharing system. In the following, we look at the field of rational cryptography from a different perspective: We are interested in uncovering the fundamental connections between the two worlds that form the basis of rational cryptography, i.e., game theory and cryptography. For this purpose, in the next chapter we investigate different notions from both worlds and deduce the equivalence relations among them.

Chapter 4

Bridging Security and Game Theory

4.1 Introduction

As the confluence of two different fields, rational cryptography allows for more appropriate security models for settings where the traditional "black and white" cryptographic model of either honest or arbitrarily malicious participants is too inflexible and cannot be applied realistically. In the previous two chapters we have seen how to design, implement and automatically verify rational cryptographic protocols. In the current chapter we focus on the study of the intrinsic connections between the two words that rational cryptography is built upon: game theory and cryptography. While such a research path is crucial for the understanding of the inherent similarities and differences between the two worlds, so far its study was mostly left as an open research field [55, 54].

Indeed, recently the view on cryptographic definitions has been extended [55] with the incipient study of the equivalence relation between the security notion of weak precise secure computation and a weak variant of the game-theoretic notion of universal implementation for a trusted mediator. However, it is still left as an open problem [55, 54] how to obtain such an equivalence result for stronger security notions.

4.1.1 Contribution

In this chapter we have a three fold contribution.

First, we relate the notion of weak stand-alone security[1] to the emerging game-theoretic concept of universal implementation [55, 54]. In contrast to previous work, for our result we use a variant of universal implementation that discards the cost of

[1]The difference between stand-alone security and weak stand-alone security is in the order of quantifiers. For stand-alone security, the simulator is universally quantified over all distinguishers and input distributions. As detailed in section 4.2, for our notion of weak security the simulator depends only on the distinguisher and not on the input distribution. This comes in contrast with [55], where the simulator for weak precise secure computation depends on both distinguisher and input distribution.

computation. We are thus able to answer positively the open question from [55, 54] regarding the existence of game-theoretic concepts that are equivalent to cryptographic security notions where the simulator does not depend on both the input distribution and the distinguisher.

Second, we study the propagation of weak security notion through the hierarchy of security definitions. More precisely, we show that the notion weak security composed under concurrent general composition is equivalent to 1-bit specialized simulator UC security, which is a variant of UC security. Together with our first result, this implies that weak stand-alone security and stand-alone security are not equivalent.

Third, we present a separation result between two variants of UC security: 1-bit specialized simulator UC security and specialized simulator UC security. This solves an open question from [71] and comes in contrast with the well known equivalence result between 1-bit UC security and UC security [24]. Both variants of the UC security notion are obtained from the UC security definition by changing the order of quantifiers [35, 71].[2] In order to obtain the separation result, we first show that the 1-bit specialized simulator UC security is equivalent to a seemingly weaker version of security, namely weak specialized simulator UC security.

The main proof technique used in our separation result is to employ a cryptographic tool called time-lock puzzles. Intuitively, this cryptographic tool can be used for comparing the computational power of two different polynomially bounded Turing machines. In order to achieve the separation result, we use time-lock puzzles from which we derive a result interesting also on its own, mainly a construction of a one-way function and a hard-core predicate.

4.1.2 Background and Related Work

The initial work [109] on general security definitions highlighted the need for a framework expressing security requirements in a formal way. The first formal definition of *secure computation* was introduced by Goldreich et al. [42]. The first approaches for formally defining security notions [45, 47] have taken into account only the stand-alone model. In this model, the security of the protocol is considered with respect to its adversary, in isolation from any other copy of itself or from a different protocol. However, there are simple protocols [36] that fulfill stand-alone security, but are no longer secure under parallel or concurrent composition.

Micali and Rogaway [82] introduced the first study of protocol composition, which the authors call *reducibility*. The first security definition expressed as a comparison with an ideal process, as well as the corresponding sequential composition theorem for the stand-alone model are provided in [13]. A general definition of security for evaluating a probabilistic function on the parties' inputs is given in [23]. It is shown that security is preserved under a subroutine substitution composition operation, which is a non-concurent version of universal composition: Only a single instance of the protocol is

[2] This means that in contrast to the UC security definition, the simulator may depend on the environment.

active at any point in time.

The framework of *universally composable security*, for short UC security [24] allows for specifying the requirements for any cryptographic task and within this framework protocols are guaranteed to maintain their security even in the presence of an unbounded number of arbitrary protocol instances that run concurrently in an adversarially controlled manner.

The notion of *specialized simulator UC* security has been introduced in [71] and it was shown that this is equivalent to *general concurrent composability* when the protocol under consideration is composed with one instance of any possible protocol. The definition of specialized simulator UC security is very much related to the definition of UC security, the only difference is the change in the order of quantifiers. Changing the order of quantifiers in the context of security definitions has been previously used in [35, 54, 55] for strengthening or weakening given security notions. We present a more detailed review about the existing implication relations among various security notions in section 4.5.

In parallel with the UC framework, the notion of *reactive security* has been developed [87, 56, 86, 88, 89]. The framework addresses for the first time *concurrent* composition in a computational setting: it is shown that security is preserved when a single instance of a subroutine protocol is concurrently composed with the calling protocol. The framework has been extended in [10] to deal with the case where the number of parties and protocol instances depends on the security parameter. More about the differences between reactive simulatability and universal composability notions can be read in the related work section from [24].

Our study of the relation between security and game-theoretic notions has been triggered by the recently emerging field of rational cryptography, where users are assumed to only deviate from a protocol if doing so offers them an advantage. Rational cryptography is centered around (adapted) notions of game theory such as computational equilibria [33]. A comprehensive line of work already exists developing novel protocols for cryptographic primitives such as rational secret sharing and rational secure multiparty computation [2, 37, 39, 48, 53, 63].

Historically, game theory and its computational aspects have been first studied in more detail in [83] (i.e., players are modeled as finite automata) and in [80] (players are defined as Turing machines). Later, [33, 106] study the rational cryptographic problem of implementing mediators using polynomially bounded Turing machines. Another direction, [54, 55, 97] considers that computation is costly for players and investigates how this affects their utilities and the design of appropriate protocols. In [97], a player's strategy is defined as a finite automaton whose complexity (i.e., number of states) influences players utilities. In [55, 54] similar considerations are made: both the input and the complexity of the machine (which is a Turing machine this time) are taken into account. This complexity can be interpreted, for example, as the running time or the space used by the machine for a given input. Their work develops a game-theoretic notion of protocol implementation and the authors show a special case of their definition is equivalent to a weak variant of precise secure computation.

4.1.3 Organization

This chapter is structured as follows: In section 4.2 we review the necessary security notions and in section 4.3 we revise the game-theoretic notion of universal implementation. In section 4.4 we prove our separation result between specialized simulator UC security and 1-bit specialized simulator UC security. In section 4.5 we show our equivalence relation between weak security under 1-bounded concurrent general composition and 1-bit specialized simulator UC security. In section 4.5.1 we present the equivalence between our weak security notion and the game-theoretic notion of strong universal implementation. In section 4.6 we present the conclusions for this chapter.

4.2 Review of Security Notions

In this chapter we consider all parties and adversaries run in polynomial time in the security parameter k and not in the length of input. In this section we review two models of security under composition: concurrent general composability and universal composability. Both frameworks require the notion of (computational) indistinguishability which we review below.

Definition 15 (Computational Indistinguishability)**.** *We call distribution ensembles* $\{X(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ *and* $\{Y(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ *computationally indistinguishable and we write* $X \equiv Y$*, if for every probabilistic polynomial time interactive Turing machine (PPITM)* $\mathcal{D}$ *there exists a function* ϵ*, negligible in* k*, and* $k_0 \in \mathbb{N}$ *such that for every* $z \in \{0,1\}^*$

$$|(Pr(\mathcal{D}(X(k,z)) = 1) - (Pr(\mathcal{D}(Y(k,z)) = 1)| < \epsilon(k),$$

for every $k \geq k_0$*.*

In the following, we call PPITM $\mathcal{D}$ *a* distinguisher*.*

A variant of this definition, which we call *indistinguishability with respect to a given adversary* $\mathcal{D}$ and we denote it by $\overset{\mathcal{D}}{\equiv}$, is analogous to the definition above, where "for every probabilistic distinguisher $\mathcal{D}$" is replaced with "for a distinguisher $\mathcal{D}$". Such a definition will be used in relation with our notion of weak security.

Definition 16 (Indistinguishability with respect to a Given Distinguisher)**.** *We say that the distribution ensembles* $\{X(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ *and* $\{Y(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ *are computationally indistinguishable with respect to a given PPITM* $\mathcal{D}$ *and we write* $X \overset{\mathcal{D}}{\equiv} Y$*, if there exists a function* ϵ*, negligible in* k*, and* $k_0 \in \mathbb{N}$ *such that for every* $z \in \{0,1\}^*$

$$|(Pr(\mathcal{D}(X(k,z)) = 1) - (Pr(\mathcal{D}(Y(k,z)) = 1)| < \epsilon(k),$$

for every $k \geq k_0$*.*

4.2.1 Universal Composability

The standard method for defining security notions is by comparing a real world protocol execution to an ideal world process execution. In the real world execution, a protocol interacts with its adversary and possibly with other parties. In the ideal world execution, an idealized version of the protocol (called ideal functionality) interacts with an ideal world adversary (usually called simulator) and possibly with other parties. The ideal functionality is defined by the security requirements that we want our protocol to fulfill.

On an intuitive level, given an adversary, the purpose of the simulator is to mount an attack on the ideal functionality; any PPITM distinguisher may try to tell apart the output of the interaction between the ideal functionality and the simulator and the output of the interaction between the protocol and its adversary. If for every adversary, a simulator exists such that the two outputs cannot be told apart by any PPITM distinguisher, then our initial protocol is as secure as the ideal functionality, with respect to what is called *the stand-alone model*. More formally we have:

Definition 17 (Stand-alone Security)**.** *Let ρ be a protocol, $\mathcal{F}$ an ideal functionality and k a security parameter. We say ρ* securely implements *$\mathcal{F}$ with respect to stand-alone security if for every PPITM real-model* adversary *$\mathcal{A}$ there exists a PPITM ideal-model* simulator *$\mathcal{S}$ such that for every protocol input x and for every auxiliary input z given to the adversary with $x, z \in \{0,1\}^{poly(n)}$, we have*

$$\{IDEAL_{\mathcal{S}}^{\mathcal{F}}(k,x,z)\}_{k\in\mathbb{N}} \equiv \{REAL_{\rho,\mathcal{A}}(k,x,z)\}_{k\in\mathbb{N}}.$$

By $IDEAL_{\mathcal{S}}^{\mathcal{F}}(k,x,z)$ we denote the output of $\mathcal{F}$ and $\mathcal{S}$ after their interaction and $REAL_{\rho,\mathcal{A}}(k,x,z)$ denotes the output of ρ and adversary $\mathcal{A}$ after their interaction.

If in Definition 17 we allow the simulator to depend also on the distinguisher, we obtain the notion of *weak stand-alone security*. More formally, we have the following definition:

Definition 18 (Weak Stand-alone Security)**.** *Let ρ be a protocol, $\mathcal{F}$ an ideal functionality and k a security parameter. We say ρ* computes *$\mathcal{F}$* with respect to weak stand-alone security *if for every PPITM real-model* adversary *$\mathcal{A}$ and for every PPITM distinguisher $\mathcal{D}$ there exist a PPITM simulator $\mathcal{S}$ such that for every $x, z \in \{0,1\}^{poly(n)}$ we have*

$$\{IDEAL_{\mathcal{S}}^{\mathcal{F}}(k,x,z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{REAL_{\rho,\mathcal{A}}(k,x,z)\}_{k\in\mathbb{N}}.$$

Sometimes, for brevity of notation, we compact the definition above into the relation

$$\{IDEAL(k,\mathcal{S},\mathcal{F})\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{REAL(k,\rho,\mathcal{A})\}_{k\in\mathbb{N}}. \tag{4.1}$$

There are examples [36] of protocols secure in the stand-alone model that do not remain secure even when two of its instances run concurrently. As a consequence, more stringent security definitions take into account that a protocol interacts not only with its adversary, but also with other (possibly an unbounded number of) protocols or with

(an unbounded number of) copies of itself. This is intuitively captured by the universal composability (UC) security framework [24].

The definition of universal composability follows the paradigm described above, however it introduces an additional adversarial entity which is called environment. The environment, usually denoted by $\mathcal{Z}$, is present in both the UC real world and UC ideal world. The environment represents everything that is external to the current execution of the real-world protocol or to the ideal functionality.

The main difference between the execution of UC real and UC ideal world, is that in the latter the ideal functionality cannot be directly accessed by the environment. Parties involved in the ideal execution give their inputs to the ideal functionality which computes some outputs and sends back these values. Since the ideal world parties perform no computation they are called the dummy parties for the ideal functionality. The ideal $\mathcal{F}$ together with its corresponding dummy parties represent an ideal process. The adversary for the protocol is not considered to be a part of the environment, but it could be controlled by the environment.

In order to determine whether a protocol securely implements a given task, first we define the ideal process for carrying out that task. Intuitively, in an ideal process for a given task, all parties give their inputs directly to the *ideal functionality for that task* which can be regarded as a formal specification of the security requirement of the task. According the universal composability security definition, a protocol securely implements a task if any damage that can be caused by an adversary while interacting with the protocol and the environment, can also be caused by an adversary interacting with the ideal process for that task and the environment. Intuitively, the entity assessing the amount of damage is the environment. Since there is no damage we can cause to the ideal functionality, the protocol considered must also be secure. We say that the protocol runs in a real-world model and the ideal functionality runs in the ideal-world model.

Real-world Protocols

More formally, let ρ be a cryptographic protocol. The real-world model for the execution of protocol ρ contains the following PPITMs: a PPITM $\mathcal{Z}$ called the environment, a set of PPITMs representing the parties running the protocol ρ and an adversary PPITM $\mathcal{A}$. We now have a more detailed look at each of these PPITMs and their interaction.

The environment $\mathcal{Z}$ represents everything that is external to the current execution of ρ and it is modeled as an PPITM with auxiliary input. Throughout the course of the protocol execution, the environment can provide inputs to parties running ρ and to the adversary. These inputs can be a part of the auxiliary input of $\mathcal{Z}$ or can be adaptively chosen by the environment. Also $\mathcal{Z}$ receives all the outputs that are generated by the parties and the adversary. The only interaction between the environment $\mathcal{Z}$ and the parties is when the environment sends the inputs and receives the outputs. Finally, at the end of the execution of ρ, the environment outputs all the messages received.

The adversary can receive inputs from $\mathcal{Z}$ at any moment during the protocol execution and it can send replies to $\mathcal{Z}$ at any time. In order to capture any possible adversarial behaviour, $\mathcal{A}$ and $\mathcal{Z}$ can communicate freely throughout the course of the protocol and

they can exchange information after any message sent between the parties and after any output made by a party.

Next, we look at the notion of corruption. By considering a PPITM P corrupted we mean that from that point on that adversary has access to all the inputs and communication messages send or received by P, and for any communication model, $\mathcal{A}$ can decide to alter such messages in any way it wants. Moreover, all the past incoming or outgoing messages of P are known to $\mathcal{A}$.

In order for $\mathcal{A}$ to corrupt a PPITM P, it first informs $\mathcal{Z}$ by sending it a corruption message $(corrupt, P)$. Thus $\mathcal{Z}$ is aware at any given moment about the corruption state of all PPITMs. Depending on the moment when the adversary $\mathcal{A}$ can corrupt a PPITM, there are two corruption models: static and adaptive. In the static corruption model, the adversary $\mathcal{A}$ is allowed to corrupt only in the beginning of the protocol, before the respective PPITMs receive their inputs from $\mathcal{Z}$. In contrast, if $\mathcal{A}$ is allowed to corrupt at any given moment during the protocol execution, then the adversary is called adaptive. Another way to look at the corruption model is by inspecting whether the adversary is passive, (i.e., only learns all inputs and communication messages a corrupted PPITM sends and receives), or if $\mathcal{A}$ is active. The latter case implies $\mathcal{A}$ is allowed to modify any input a corrupted PPITM gets and also any communication message sent.

In order to simplify the presentation, we use an equivalent definition for the static corruption model. As in the standard static case, the moment of corruption is fixed in the beginning, we can skip sending and receiving the corruption messages. Instead, we assume the corrupted PPITMs are fixed from the start and the adversary does not have to choose them. Then, the previous static adversary definition is equivalent to the latter formulation, which we use in this chapter.

Besides corruption, the adversary may interfere with the communication between honest parties. The most basic UC model ensures that all messages are handed to the adversary and the adversary delivers messages of its choice to all PPITMs. This model makes no assumption on the communication properties: authenticity, secrecy or synchrony of the messages delivered. For the more specialised models of authenticated, secure or synchronous communication, an ideal functionality is added to the basic model to capture the respective properties.

Authenticated communication assumes the adversary cannot alter content of messages without being detected. The synchronous communication model captures the property that messages are all delivered and without delay from the moment they were generated. The ideally secure communication model assumes the adversary receives all messages, but it has neither access to the content of communication, nor possibility to modify any message without breaking authenticity. In this model, the adversarial capabilities are limited to either delaying or not delivering some or all messages between the uncorrupted PPITMs.

When the protocol execution ends, $\mathcal{Z}$ outputs its view of that execution. This view contains messages that $\mathcal{Z}$ has received from the adversary $\mathcal{A}$ and outputs of all other PPITMs. Formally, $EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k, z)$ denote the output of $\mathcal{Z}$ in an execution of the protocol ρ with adversary $\mathcal{A}$ and environment $\mathcal{Z}$, where k is the security parameter and

z is the auxiliary input to the environment $\mathcal{Z}$. We denote by $EXEC_{\rho,\mathcal{A},\mathcal{Z}}$ the family of random variables $\{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}$.

Ideal Process and Ideal Functionalities

In order to formalize the ideal process, we do not want to define a different model, but we rather need to adapt to the one above. In the same way as in the real-world, the environment $\mathcal{Z}$ is the only PPITM that can send inputs at any moment to the ideal process parties and to the ideal adversary. In the case of the ideal process, the adversary is called the ideal simulator and is commonly denoted by $\mathcal{S}$. Moreover, $\mathcal{Z}$ receives all the outputs generated by the parties, as well the possible outputs of $\mathcal{S}$.

The first difference is that in the ideal model there exists a trusted party, the ideal functionality, that cannot be directly accessed by the environment. This works as follows: PPITMs involved in the ideal process give their inputs to the ideal functionality which computes outputs for each party and sends these values to them. Hence, the role of the ideal functionality is to receive inputs, perform computations and send results to the ideal PPITMs. As these PPITMs do not take an active role in the computation and just send inputs to and receive outputs from the ideal functionality, they are called dummy parties of the ideal functionality.

The second difference with the real-world model is that messages delivered by the adversary to dummy parties are ignored. In the ideal protocol the adversary sends corruption messages directly to the ideal functionality. The ideal functionality then determines the effect of corrupting a party. A typical response is to let the adversary know all the inputs received and outputs sent by the party so far.

The environment $\mathcal{Z}$ and the simulator $\mathcal{S}$ can communicate freely during the execution of the ideal process. Additionally, the ideal functionality informs the simulator every time it wants to output a message. If the simulator agrees, then the respective output is made. This is required by the UC ideal model in order to allow $\mathcal{S}$ to simulate the behavior of a UC real world adversary delaying messages or not sending some or all of the communication among real-world protocol PPITMs.

Similar to the real-world model, the environment $\mathcal{Z}$ outputs its view in the end of the ideal process execution. The view contains all the messages received from the simulator as well as all the messages that the dummy parties output to $\mathcal{Z}$. More formally, by $EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)$ we denote the output of $\mathcal{Z}$ in an execution of the ideal process with the trusted party $\mathcal{F}$, simulator $\mathcal{S}$ and environment $\mathcal{Z}$, where k is the security parameter and z is the auxiliary input to the environment $\mathcal{Z}$. We denote by $EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ the family of random variables $\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}$.

Protocol Emulation

We now define what it means that a real-world protocol ρ *emulates* with respect to UC security an ideal functionality $\mathcal{F}$. The environment $\mathcal{Z}$ is the PPITM deciding whether he can distinguish between the interactions he has with the protocols and their respective adversaries in the real and in the ideal world.

All the PPITMs used in either of the protocol executions for ρ or $\mathcal{F}$, including the environment $\mathcal{Z}$, are computationally bounded. Thus, it is sufficient if we formalize the notion of emulation in terms of computational indistinguishability. The environment $\mathcal{Z}$ will act as a distinguisher for the two protocol executions. Since all the information $\mathcal{Z}$ gains throughout its interaction is contained within the view $\mathcal{Z}$ outputs in the end, it is sufficient to compare the two views. Essentially, protocol ρ emulates $\mathcal{F}$ if for every adversary $\mathcal{A}$ there is an ideal simulator $\mathcal{S}$ such that for every environment $\mathcal{Z}$ the views of the two interactions are computationally indistinguishable.

Definition 19 (UC Security)**.** *Let ρ be a protocol and $\mathcal{F}$ an ideal functionality. We say that ρ UC securely emulates $\mathcal{F}$ if for every PPITM adversary $\mathcal{A}$ there is a PPITM simulator $\mathcal{S}$ such that for every PPITM distinguisher $\mathcal{Z}$ and for every input $z \in \{0,1\}^*$, the two families of random variables $\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}$ and $\{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}$ are computationally indistinguishable.*

In the following we also use a relaxed version of this definition, where the order of quantifiers between the environment and the ideal-world simulator is reversed [71].

Definition 20 (Specialized Simulator UC Security)**.** *Let ρ be a protocol and $\mathcal{F}$ an ideal functionality. We say that ρ emulates $\mathcal{F}$ under specialized simulator UC security if for every probabilistic polynomial time adversary $\mathcal{A}$ and for every environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that for every distribution of auxiliary input $z \in \{0,1\}^*$, we have*

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \equiv \{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}$$

It had been shown [57] that the two notions defined above are not equivalent. In the above definition, the output of the environment is considered to be a string of arbitrary length. If the only change we make to the above definition is to consider environments $\mathcal{Z}$ that have a 1-bit output instead of an output containing the entire view, we obtain the notion of *1-bit specialized simulator UC security*. It has been an open problem [71] whether considering only environments with one bit output would produce an equivalent definition. In this chapter we show how to separate the notions of specialized simulator UC security and 1-bit specialized simulator UC security.

Definition 21 (1-bit Specialized Simulator UC Security)**.** *Let ρ be a protocol and $\mathcal{F}$ an ideal functionality. We say that ρ emulates $\mathcal{F}$ under 1-bit specialized simulator UC security if for every probabilistic polynomial time adversary $\mathcal{A}$ and for every 1-bit output environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that for every input $z \in \{0,1\}^*$, we have*

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \equiv \{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}.$$

If in the specialized simulator UC definition we let the simulator also depend on the distinguisher who is the only PPITM to establish whether the output of the executions in the real UC world and ideal UC world cannot be told apart, then we obtain the notion of *weak specialized simulator UC security*.

Definition 22 (Weak Specialized Simulator UC Security)**.** *Let ρ be a protocol and $\mathcal{F}$ an ideal functionality. We say that ρ emulates $\mathcal{F}$ under weak specialized simulator UC security if for every PPITM adversary $\mathcal{A}$, for every PPITM environment $\mathcal{Z}$ and for every PPITM distinguisher $\mathcal{D}$, there exists a PPITM simulator $\mathcal{S}$ such that for every distribution of input $z \in \{0,1\}^*$, we have*

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}.$$

In the revised version of [24] there is an extension of the UC model we reviewed above. This extension mainly considers that PPITM machines run in time polynomial in both the security parameter and the length of the input. While the extended model is seemingly more expressive in terms of adversarial attacks, it does not allow for fine grained separation between security notions (e.g., the separation result from [57] does not hold in the extended UC model). Another reason for choosing the original model is that, as it will be detailed in section 4.5, most of the UC results have been obtained in this model.

4.2.2 Weak Security under 1- bounded Concurrent General Composition

Given a security notion, there are two approaches to ensure the security properties of a protocol under composition. One way is to prove that the security property defined for the stand-alone case is preserved under composition. The other way is to define the security notion for the protocol directly under composition. The latter approach has the benefit that it captures the security property without having the drawback of a possible very strong and thus very restrictive stand-alone definition. Due to this reason we will focus on the second approach.

The concurrent general composability notion has been introduced by Lindell [71]. In this security model, a protocol ρ that is being investigated is run concurrently, possibly multiple times, with an arbitrary protocol π. The protocol π can be any arbitrary protocol and intuitively, it represents the network activity around ρ. There is another way to look at this: one can consider protocol π to be the external protocol that gives inputs and reads the outputs of the internal protocol ρ. As π is arbitrary, it can call multiple instances of ρ. However, we consider that different instances run independently from one another. The only correlation between them are the inputs and outputs, in the following way: the inputs for a certain run of ρ that are provided by π might depend on the previous inputs and outputs given and collected by π. Also, the messages of π may be sent concurrently to the execution of ρ. This composition of π with ρ is denoted as in the original notation by π^ρ.

As in the case of universal composability, in order to give the definition of security for ρ under concurrent general composition, we need to compare the execution of ρ with that of an ideal functionality so we have to define the real and the ideal world.

The computation in the ideal world is performed among the parties of π and a trusted party, playing the role of an ideal functionality $\mathcal{F}$. Thus, the messages considered in

the ideal world are standard messages between parties of π and ideal messages between π and $\mathcal{F}$. The protocol π is providing $\mathcal{F}$ with inputs and after performing necessary computations, $\mathcal{F}$ sends the results to parties of π. The ideal adversary is called a simulator, and as in the UC model, is denoted by $\mathcal{S}$. In addition to having full control over the parties it has corrupted (see also the case of real world adversary), the simulator controls the scheduling of the messages between the parties of π and if not otherwise mentioned, it can also arbitrarily read and change messages. An exception is represented by the messages between π and $\mathcal{F}$: they are ideally secure, so the simulator can neither read nor change them. This comes in contrast with the standard definition of UC ideal protocol execution, where it is not enforced that the channels between the trusted parties and the rest of the participants are ideally secure.

During the computation, the honest parties follow the instructions given by π and in the end they output on their outgoing communication tape whatever value is prescribed by π. The corrupted parties output a special corrupted symbol and additionally the adversary may output an arbitrary image of its view. Let z be the auxiliary input for the ideal-world adversary $\mathcal{S}$ and let the inputs vector for parties of π be $\bar{x} = (x_1, ..., x_m)$. Then the outcome of the computation of π with $\mathcal{F}$ in the ideal world (which we may also call $\mathcal{F}$-hybrid world) is defined by the output of all parties and $\mathcal{S}$ and is denoted by $\{HYBRID^{\mathcal{F}}_{\pi,\mathcal{S}}(k, \bar{x}, z)\}_{k\in\mathbb{N}}$.

The computation in the real world follows the same rules as the computation in the ideal world, only that this time there is no trusted party. Instead, each party of π has an PPITM that works as the specification of ρ for that party. Thus, all messages that a party of π sends to the ideal functionality in the ideal world are now written on the input tape of its designated PPITM. These PPITMs communicate with each other in the same manner as specified for the parties of ρ. After the computation is performed, the results are output by these PPITMs and the corresponding parties of π copy them on their incoming communication tapes. These messages are used by the parties of π in the same way as the messages output by $\mathcal{F}$ in the ideal-world. Similarly as above, in the real-world the adversary has full control over message delivery. There is one exception: any uncorrupted party of π can write and read directly to and from the input and respectively output tape of its designated PPITM without any interference from the adversary. Actually, the ideal adversary is not even aware of this taking place. This is similar to the UC communication between the environment and the real-world or ideal-world parties. Moreover, when we say that a real-world party is corrupted, we mean that a party of π and its corresponding PPITM are corrupted. This is not a restriction as an adversary that corrupts both a party of π and its PPITM can just fully control only one of them and let the other one follow its prescribed protocol.

Similarly to the ideal world, during the computation, the honest parties follow the instructions of π and their corresponding PPITM and in the end they output on their outgoing communication tape whatever value is prescribed by π. The corrupted parties output a special corrupted symbol and additionally the real-world adversary $\mathcal{A}$ may output an arbitrary image of its view. Let z be the auxiliary input for $\mathcal{A}$ and let the inputs vector be $\bar{x} = (x_1, ..., x_m)$. Then the outcome of the computation of π with

ρ in the real world is defined by the output of all parties and $\mathcal{A}$ and is denoted by $\{REAL_{\pi^\rho,\mathcal{A}}(k,\bar{x},z)\}_{k\in\mathbb{N}}$.

Independent of the world where the corruption takes place, the adversary could be static or adaptive. If the adversary is static, then the parties that are under the control of the adversary are fixed and do not depend on its auxiliary input or random tape. This is a restrictive definition of static corruption. However, the definition of adaptive corruption and the corresponding proof include the proof for a standard static corruption case. In the case of adaptive corruption, the adversary may decide during the protocol to arbitrarily corrupt a party, depending on the messages received so far. In both cases, once the adversary has corrupted a party then it learns all previous inputs and messages that the party received. From the moment of the corruption further on, the adversary has full control over the messages that the party sends. Moreover, we consider that the adversary fully controls the message scheduling: he decides if and when to deliver the messages between output tape of one party (or, more general, machine) to the input tape of another. As mentioned above, there is one exception: the adversary does not have any control over the messages that an uncorrupted party sends to its corresponding PPITM.

We are now ready to state the definition of security under concurrent general composition as in [71]. One can define security under concurrent general composition for the case that π make an unbounded number of calls to $\mathcal{F}$ and also for the case that π utilizes a constant number of calls to $\mathcal{F}$. Both cases are summarized in the following definition:

Definition 23 (Security under Concurrent General Composition)**.** *Let ρ be a protocol and $\mathcal{F}$ a functionality. Then, ρ* securely computes *$\mathcal{F}$* under concurrent general composition *if for every probabilistic polynomial-time protocol π in the $\mathcal{F}$-hybrid model that utilizes ideals calls to $\mathcal{F}$ and every PPITM real-model adversary $\mathcal{A}$ for π^ρ, there exists a PPITM hybrid-model adversary $\mathcal{S}$ such that for every $\bar{x}, z \in \{0,1\}^*$*

$$\{HYBRID^{\mathcal{F}}_{\pi,\mathcal{S}}(k,\bar{x},z)\}_{k\in\mathbb{N}} \equiv \{REAL_{\pi^\rho,\mathcal{A}}(k,\bar{x},z)\}_{k\in\mathbb{N}}.$$

If we restrict the protocols π to those that utilize at most ℓ ideal calls to $\mathcal{F}$, then ρ is said to securely compute $\mathcal{F}$ under ℓ-bounded concurrent general composition*.*

We also use a weak version of the security definition presented above.

Definition 24 (Weak Security under Concurrent General Composition)**.** *Let ρ be a protocol and $\mathcal{F}$ a functionality. Then, ρ* computes *$\mathcal{F}$* under concurrent general composition with weak security *if for every probabilistic polynomial-time protocol π in the $\mathcal{F}$-hybrid model that utilizes ideals calls to $\mathcal{F}$, for every PPITM real-model adversary $\mathcal{A}$ for π^ρ and for every PPITM distinguisher $\mathcal{D}$, there exists a PPITM hybrid-model adversary $\mathcal{S}$ such that for every $\bar{x}, z \in \{0,1\}^*$*

$$\{HYBRID^{\mathcal{F}}_{\pi,\mathcal{S}}(k,\bar{x},z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{REAL_{\pi^\rho,\mathcal{A}}(k,\bar{x},z)\}_{k\in\mathbb{N}}.$$

If we restrict the protocols π to those that utilize at most ℓ ideal calls to $\mathcal{F}$, then ρ is said to compute $\mathcal{F}$ under ℓ-bounded concurrent general composition with weak security*.*

4.3 Game-theoretic Definitions

In this section we define some game-theoretic concepts that we further need for establishing the equivalence between our notion of weak stand-alone security and a variant of the the strong univeral implementation notion given in [54] which we defined below. We build upon the notions already given in section 2.3.

We begin by extending the definition of a computational game presented in Chapter 2 such that it takes into account which are the utilities of a group of players participating in the prescribed protocol, or deviating from it. In the rest of the paper we denote by Z the set of players participating in such a coalition and we denote by u_Z and U_Z respectively, the utility and the expected utility for such a coalition. We also denote for example by M_Z the vector of strategies (or the PPTMs) that the parties in Z run (or are controlled by).

The definition of computational Nash equilibrium can be extended to the notion of *computational Nash equilibrium with immunity with respect to coalitions*: We require that the property in the definition of computational Nash equilibrium is fulfilled for all subsets Z of players, i.e., for all Z and all PPITM M'_Z controlling the parties in Z there exists a negligible function ϵ_Z such that $U_Z(k, M'_Z, \overrightarrow{M_{-Z}}) - U_i(k, \overrightarrow{M}) \leq \epsilon_Z(k)$ holds.

So far we have assumed that players communicate only among each other. We extend the notion of computational game to a *computational game with mediator*. The mediator is modeled by a PPITM denoted $\mathcal{F}$. Without loss of generality, we assume all communication passes between players and the trusted mediator (that can also forward messages among players).

Next we follow the approach from [55] to formalize the intuition that the machine profile $\overrightarrow{M} = (M_1, \ldots, M_n)$ implements a mediator $\mathcal{F}$: Each time a set of players want to truthfully provide a value (e.g., their input or type) to the mediator $\mathcal{F}$, they also want to run $\overrightarrow{M}$ using the same values. For each player i, let its type be $t_i = (x_i, z_i)$, where x_i is player's input and z_i is some auxiliary information about the state of the world.

Let $\Lambda^{\mathcal{F}}$ denote the machine that, given the type $t_i = (x_i, z_i)$ of player i, it sends x_i to the mediator $\mathcal{F}$, outputs as action the string it receives from $\mathcal{F}$ and halts. So $\Lambda^{\mathcal{F}}$ uses only input[3] x_i and ignores auxiliary information z_i. By $\overrightarrow{\Lambda^{\mathcal{F}}}$ we denote the machine profile where each player uses only $\Lambda^{\mathcal{F}}$. We ensure that whenever the players want to use mediator $\mathcal{F}$, they also want to run $\overrightarrow{M}$ if every time $\overrightarrow{\Lambda^{\mathcal{F}}}$ is a computational Nash equilibrium for the game $(G, \mathcal{F})$, then running $\overrightarrow{M}$ using the intended input is a computational Nash equilibrium as well.

Finally, we provide our definition for game-theoretic protocols implementing trusted mediators. We call our notion game universal implementation. A closely related notion, called strong universal implementation, has been previously defined [54]. On an intuitive level, the main difference between the existing notion and the new notion is that for

[3] As in [54], the games considered are canonical games of fixed input length n. Any game where there are only finitely many possible types can be represented (by corresponding padding of the input) as a canonical game for some length n.

strong universal implementation, parties consider computation to be costly (i.e., time or memory used for computation may incur additional costs in the utility of the users), while our notion basically regards computation as "for free". The naive intuition suggests that game universal implementation is a weaker notion than strong universal implementation. However, as we will see in Sect. 4.5.1, this intuition does not hold.

Definition 25 (Game Universal Implementation)**.** *Let $\perp_i$ be the PPITM ran by party i that sends no message (to the other parties or to the mediator) and outputs nothing. Let Games be a set of m-player games, $\mathcal{F}$ and $\mathcal{F}'$ be mediators and let $M_1, \ldots, M_m$ be PPITMs. We call $((M_1, \ldots, M_m), \mathcal{F}')$ a game universal implementation of $\mathcal{F}$ with respect to Games if for all $n \in \mathbb{N}$ and all games $G \in$ Games with input length n if $\overrightarrow{\Lambda}^{\mathcal{F}}$ is a computational Nash equilibrium in the mediated game $(G, \mathcal{F})$ with immunity with respect to coalitions, then the following three properties hold:*

- *(Preserving Equilibrium) $(M_1, \ldots, M_m)$ is a computational Nash equilibrium in the mediated machine game $(G, \mathcal{F}')$ with immunity with respect to coalitions;*
- *(Preserving Action Distributions) For each type profile $(t_1, \ldots, t_m)$, the output distribution induced by $\overrightarrow{\Lambda}^{\mathcal{F}}$ in $(G, \mathcal{F})$ is statistically close to the output distribution induced by $(M_1, \ldots, M_m)$ in $(G, \mathcal{F}')$;*
- *(Preservation of Best Response $\perp_i$) Additionally, for all $n \in \mathbb{N}$, all games $G \in$ Games with input length n and all $i \in \{1, \ldots, m\}$, if $\perp_i$ is a computational best response to $\overrightarrow{\Lambda}^{\mathcal{F}}_{-i}$ in $(G, \mathcal{F})$, then $\perp_i$ is a computational best response to $\overrightarrow{M}_{-i}$ in $(G, \mathcal{F}')$.*

4.4 Specialized Simulator UC Variants

Our main result in this section shows the separation between the notions of specialized simulator UC and 1-bit specialized simulator UC. The type of relation between these two notions was stated as an open problem by Lindell [71]. Furthermore, our result helps proving the relations among various other security notions, as explained in section 4.5.

4.4.1 On 1-bit Specialized Simulator UC

We start by showing that 1-bit specialized simulator UC (1-bit SSUC) is equivalent to weak specialized simulator UC (weak SSUC). This will give us a simpler alternative security notion that we can further work with.

Lemma 26 (Equivalence between 1-bit SSUC and weak SSUC)**.** *A protocol fulfills the 1-bit specialized simulator UC security if and only if it fulfills the weak specialized simulator UC security.*

Proof. Let protocol ρ and ideal functionality $\mathcal{F}$ be such that ρ is as secure as $\mathcal{F}$ with respect to 1-bit specialized simulator UC. We show this implies ρ as secure as $\mathcal{F}$ with

respect to weak specialized simulator UC security. Given a triple $(\mathcal{A}, \mathcal{Z}, \mathcal{D}^*)$ consisting of adversary, environment and distinguisher we have to provide a simulator $\mathcal{S}$ such that for every auxiliary input[4] z the following holds:

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}^*}{\equiv} \{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}. \tag{4.2}$$

Given $\mathcal{Z}$ and $\mathcal{D}^*$, we can construct a 1-bit output environment $\mathcal{Z}^{\mathcal{D}^*}$ in the following way: $\mathcal{Z}^{\mathcal{D}^*}$ internally runs a copy of $\mathcal{Z}$. When internal $\mathcal{Z}$ writes on its output tape, this is forwarded by $\mathcal{Z}^{\mathcal{D}^*}$ to an internal copy of $\mathcal{D}^*$. The output of $\mathcal{D}^*$ becomes the output of $\mathcal{Z}^{\mathcal{D}^*}$. Due to the hypothesis, there exist $\mathcal{S}$ such that for every auxiliary input z and for every distinguisher $\mathcal{D}$ we have

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}^{\mathcal{D}^*}}(k,z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{EXEC_{\rho,\mathcal{A},\mathcal{Z}^{\mathcal{D}^*}}(k,z)\}_{k\in\mathbb{N}}.$$

In particular

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}^{\mathcal{D}^*}}(k,z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}_{ind}}{\equiv} \{EXEC_{\rho,\mathcal{A},\mathcal{Z}^{\mathcal{D}^*}}(k,z)\}_{k\in\mathbb{N}},$$

where $\mathcal{D}_{ind}$ is the distinguisher that outputs whatever $\mathcal{D}^*$ outputs. As the simulator $\mathcal{S}$ can be used without modification in an interaction with $\mathcal{F}$ and the environment[5] $\mathcal{Z}$, the last relation is equivalent to (4.2). We conclude that ρ is as secure as $\mathcal{F}$ with respect to weak specialized simulator UC security.

The implication in the opposite direction is proven as follows. Given a pair $(\mathcal{A}, \mathcal{Z}_{1-bit})$ consisting of adversary and 1-bit output environment $\mathcal{Z}_{1-bit}$, we need to construct a simulator $\mathcal{S}$ such that for every auxiliary input z and for every distinguisher $\mathcal{D}$, we have

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}_{1-bit}}(k,z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{EXEC_{\rho,\mathcal{A},\mathcal{Z}_{1-bit}}(k,z)\}_{k\in\mathbb{N}}.$$

Given a 1-bit output environment $\mathcal{Z}_{1-bit}$, we can uniquely decompose it into an environment $\mathcal{Z}$ and a distinguisher $\mathcal{D}^*$ that given the view of $\mathcal{Z}$ outputs what $\mathcal{Z}_{1-bit}$ outputs.

Indeed, to each 1-bit environment $\mathcal{Z}_{1-bit}$ we can uniquely associate the environment $\mathcal{Z}$ that internally runs $\mathcal{Z}_{1-bit}$: when a party or adversary sends a message to $\mathcal{Z}$, the environment forwards it internally and replies back with the messages that the copy of $\mathcal{Z}_{1-bit}$ would reply. Analogously, when the internal copy of $\mathcal{Z}_{1-bit}$ wants to send a message to a party or to the adversary, the environment $\mathcal{Z}$ forwards this message to the corresponding party or adversary. Finally, $\mathcal{Z}$ gives as output the entire view of the interaction, i.e., all the inputs and messages it sent to the parties and to the adversary, all the outputs and messages it received from the other entities as well as the random bits used.

[4]Here and in the following "for every auxiliary input z" should be read as "for every distribution of auxiliary input z for $\mathcal{Z}$".

[5]Indeed, by construction $\mathcal{Z}^{\mathcal{D}^*}$ does not interact with an adversarial party (i.e., $\mathcal{S}$ or $\mathcal{A}$) after the simulation of internal $\mathcal{Z}$ is over.

Similarly, for each environment $\mathcal{Z}_{1-bit}$ we uniquely associate the distinguisher $\mathcal{D}^*$ as follows: after receiving the input, $\mathcal{D}^*$ internally simulates the environment $\mathcal{Z}_{1-bit}$ and emulates the rest of the entities in the protocol, including the adversary $\mathcal{A}$; $\mathcal{D}^*$ treats its input as the input for the copy of $\mathcal{Z}_{1-bit}$ and all other participants (either honest or corrupted) so $\mathcal{D}^*$ can use it to emulate the interaction between $\mathcal{Z}_{1-bit}$, $\mathcal{A}$ and the honest parties. The output bit of the simulated $\mathcal{Z}_{1-bit}$ in this context, becomes by definition the output bit of the distinguisher $\mathcal{D}^*$.

According to the definition of weak specialized simulator UC security, for $\mathcal{A}$, $\mathcal{Z}$, $\mathcal{D}^*$ there exists a simulator $\mathcal{S}$ such that for every auxiliary input z we have:

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}^*}{\equiv} \{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}.$$

As $\mathcal{D}^*$ has binary output (i.e., thus finite output), the above equation implies the two random variables

$$\{\mathcal{D}^*(EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z))\}_{k\in\mathbb{N},z\in\{0,1\}^*} \text{ and } \{\mathcal{D}^*(EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z))\}_{k\in\mathbb{N},z\in\{0,1\}^*}$$

are statistically close. Hence, for any computationally bounded distinguisher $\mathcal{D}$ and for any auxiliary input z the random variables

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}_{1-bit}}(k,z)\}_{k\in\mathbb{N}} \text{ and } \{EXEC_{\rho,\mathcal{A},\mathcal{Z}_{1-bit}}(k,z)\}_{k\in\mathbb{N}}$$

are indistinguishable and this concludes the proof. □

4.4.2 Separation Result

Next we separate the notions of weak specialized simulator UC and specialized simulator UC. For this we use a cryptographic tool called time-lock puzzles, originally introduced in [95].

Definition 27 (Time-lock puzzles)**.** *A probabilistic polynomial time algorithm $\mathcal{G}$ (problem generator) together with a probabilistic polynomial time algorithm $\mathcal{V}$ (solution verifier) represent a* time-lock puzzle *if the following holds:*
-sufficiently hard puzzles: for every probabilistic polynomial time algorithm B and for every $e \in \mathbb{N}$, there is some $f \in \mathbb{N}$ such that

$$\sup_{t\geq k^f, |h|\leq k^e} Pr[(q,a) \leftarrow \mathcal{G}(1^k,t) : \mathcal{V}(1^k,a,B(1^k,q,h)) = 1] \tag{4.3}$$

is negligible in k.
-sufficiently good solvers: there is some $m \in \mathbb{N}$ such that for every $d \in \mathbb{N}$ there is a PPITM algorithm C such that

$$\min_{t\leq k^d} Pr[(q,a) \leftarrow \mathcal{G}(1^k,t); v \leftarrow C(1^k,q) : \mathcal{V}(1^k,a,v) = 1 \wedge |v| \leq k^m] \tag{4.4}$$

is overwhelming in k.

Intuitively, a time-lock puzzle is a cryptographic tool used for proving the computational power of a PPITM. $\mathcal{G}(1^k, t)$ generates puzzles of hardness t and $\mathcal{V}(1^k, a, v)$ verifies that v is a valid solution as specified by a. The first requirement is that B cannot solve any puzzle of hardness t, with $t \geq k^f$, for some f depending on B, with more than negligible probability. The algorithm B may have an auxiliary input. This ensures that even puzzles generated using hardness t chosen by B together with a trap-door like auxiliary information (of polynomial length), do not provide B with more help in solving the puzzle.

The second requirement is that for any polynomial hardness value there exist an algorithm that can solve any puzzle of that hardness. It is important that the solution for any puzzle can be expressed as a string of length bounded above by a fixed polynomial.

As promoted by [95] and later by [57], a candidate family for time-lock puzzles which is secure if the RSA assumption holds, is presented next. A puzzle of hardness t consists of the task to compute $2^{2^{t'}} \mod n$ where $t' := \min(t, 2^k)$ and $n = p_1 \cdot p_2$ is a randomly chosen Blum integer. Thus, $\mathcal{G}(1^k, t) = ((n, \min\{t, 2^k\}), (p_1, p_2, \min\{t, 2^k\}))$, where n is a k-bit Blum integer with factorization $n = p_1 \cdot p_2$, and $\mathcal{V}(1^k, (p_1, p_2, t'), v) = 1$ if and only if $(v = v_1, v_2)$ and $v_1 \equiv 2^{2^{t'}} \mod n$ and $v_2 = n$.[6] Both solving the puzzle and verifying the solution can be efficiently done if p_1 and p_2 are known. From this point further we call these puzzles the Blum integer puzzles. An important property that we use in the following is that any Blum integer puzzle has a unique solution.

Before we state and prove our main separation result in theorem 32, we give as reminder the definition of hard-core predicates and then we state two properties related to them.

Definition 28 (Hard-Core Predicate)**.** *A hard-core predicate of a collection of functions $g_{k,t} : \{0,1\}^* \to \{0,1\}^*$ is a boolean predicate $HC : \{0,1\}^* \to \{0,1\}$ such that:*

- *there exists a probabilistic polynomial time algorithm E with $HC(x) = E(x)$, for every x;*
- *for every probabilistic polynomial time algorithm A and for every polynomial p, there exists k_p and t_p such that for every $k > k_p$ and $t > t_p$, we have*

$$Pr[A(1^k, t, g_{k,t}(x)) = HC(x)] < \frac{1}{2} + \frac{1}{p(k)}.$$

Now we are ready to state the two lemmas related to hard-core predicates. The first result shows that from a Blum integer time-lock puzzle we can construct a one-way function and a hard-core predicate.

[6] Without loosing any security of the initial definition of time-lock puzzles [95, 57], in addition to the value $2^{2^t} \mod n$, our solution for the puzzle $q = (t, n)$ contains also the value n. The full use of defining solutions in such a way, will become more clear when we define the one-way function based on time-lock puzzles: There is a one-to-one correspondence between the pair of values $(v = (2^{2^{t'}} \mod n, n), t)$ and $q = (t, n)$.

Lemma 29 (One-Way Function and Hard-Core Predicate from Blum Integer Time-Lock Puzzles)**.** *Let $(\mathcal{G}, \mathcal{V})$ be a Blum integer time-lock puzzle and let t be an integer. Let $S_{k,t}$ be the set of all correctly generated solutions $v = (2^{2^t} \bmod n, n)$ for puzzles q, where $q = (t, n)$ is the output of algorithm $\mathcal{G}$ when invoked with parameters 1^k and t. Then the collection of functions $\{f_{k,t} : S_{k,t} \to \{0,1\}^*\}_{(k\in\{0,1\}^*, t\in\{0,1\}^k)}$ and $\{g_{k,t} : S_{k,t} \times \{0,1\}^* \to \{0,1\}^*\}_{(k\in\{0,1\}^*, t\in\{0,1\}^k)}$ defined below are collections of one-way functions and the predicate $HC : \{0,1\}^* \to \{0,1\}^*$ defined below is a hard-core predicate for $\{g_{k,t}\}_{(k\in\{0,1\}^*, t\in\{0,1\}^k)}$. We alternatively call HC the hard-core predicate for $(\mathcal{G}, \mathcal{V})$. We define $f_{k,t}(2^{2^t} \bmod n, n) = (t, n)$ and for $v, r \in \{0,1\}^*$ such that $|v| = |r|$, let $g_{k,t}(v, r) = (f_{k,t}(v), r)$ and $HC(v, r) = \sum_{i=1}^{|v|} v_i \cdot r_i \mod 2$.*

Proof. First we prove that $\{f_{k,t}\}_{(k\in\{0,1\}^*, t\in\{0,1\}^k)}$ defined above is a collection of one-way functions. For every security parameter k, let $m(k)$ be the maximum number of bits that machine $\mathcal{G}$ can read from its randomness tape when invoked with security parameter k. Assume by contradiction that there exist adversary A and polynomial p such that for every integers k_p and t_p there exist $k \geq k_p$ and $t \geq t_p$ such that

$$Pr[A(1^k, t, (t, n)) = v : \mathcal{G}(1^k, t) = ((t, n), a),\ V(1^k, a, v) = 1] \geq \frac{1}{p(k)}.$$

If in the definition of the first property of time-lock puzzles, we take $e = 0$ and we use algorithm A for solving the puzzles, then we immediately obtain a contradiction so our assumption is false.

It is also clear that the property we have shown to hold for f, can be shown in a similar way to hold for g.

By following exactly the steps of the well known proof by Goldreich and Levin [41], which gives a hard-core predicate construction for any one-way function, it follows that $HC(v, r) = \sum_{i=1}^{k} v_i \cdot r_i \mod 2$ is a hard-core predicate for g and this concludes the proof. □

The second result is a straight forward consequence of the definition of hard-core predicates.

Lemma 30 (Distribution of Hard-Core Predicates)**.** *Let k be a security parameter. Then, for any given integer t, let $g_{k,t} : D_{k,t} \to \{0,1\}^*$ be a function such that $HC : \{0,1\}^* \to \{0,1\}$ is a hard-core predicate for the collection of functions $\{g_{k,t}\}_{k\in\{0,1\}^*, t\in\{0,1\}^k}$. Let $X(k, t)$ be the distribution of $(g_{k,t}(x), HC(x))$ and let $Y(k, t)$ be the distribution of $(g_{k,t}(x), U(x))$ with x taken from the domain $D_{k,t}$ and $U(x)$ being the uniform distribution on $\{0,1\}$. Then the ensembles $\{X(k,t)\}_{(k\in\{0,1\}^*, t\in\{0,1\}^k)}$ and $\{Y(k,t)\}_{(k\in\{0,1\}^*, t\in\{0,1\}^k)}$ are computationally indistinguishable.*

Proof. In definition 28 we choose an adversary A such that its output is independent of its input. More precisely, we take A that outputs 1 with constant probability c. This implies that the output distributions of A and HC are also independent. If we denote by $w_{k,t}(x)$ the probability that HC outputs 1 given x from a distribution $Output(1^k, t)$, then we obtain:

$$
\begin{aligned}
&Pr[A(1^k, t, g_{k,t}(x)) = HC(x) : x \leftarrow Output(1^k, t)] = \\
&= (Pr[A(1^k, t, g_{k,t}(x))] = 0) \cdot (Pr[HC(x)] = 0) + \\
&\qquad + (Pr[A(1^k, t, g_{k,t}(x))] = 1) \cdot (Pr[HC(x)] = 1) = \\
&= w_{k,t}(x)(2 \cdot c - 1) + 1 - c.
\end{aligned}
$$

Substituting this in the definition of hard-core predicate, we have that for every polynomial p, for all sufficiently large k and all sufficiently large t: $w_{k,t}(x)(2 \cdot c - 1) + 1 - c < \frac{1}{2} + \frac{1}{p(k)}$, which is equivalent to $w_{k,t}(x) < \frac{1}{2} + \frac{1}{p(k) \cdot (2c-1)}$ for large enough t and k. Since c is a constant, this implies that for large enough t and k, the probability $w_{k,t}(x)$, (where $x \leftarrow Output(1^k, t)$), is negligibly close to $\frac{1}{2}$ and this concludes our proof. □

Using lemmas 29 and 30, the following statement can be shown:

Lemma 31 (Weak SSUC Does Not Imply SSUC)**.** *Assume Blum integer time-lock puzzles exist. Then there are protocols that fulfill weak specialized simulator UC security but do not fulfill specialized simulator UC security.*

Proof. Let $(\pi, \mathcal{F})$ be a pair of protocol and ideal functionality as defined below. The only input the ideal functionality $\mathcal{F}$ requires is the security parameter 1^k. Then $\mathcal{F}$ sends a message to the adversary (i.e. ideal simulator $\mathcal{S}$) asking for its computational hardness. Using the reply value t' from $\mathcal{S}$ (which is truncated by $\mathcal{F}$ to maximum k bits), the ideal functionality invokes $Gen(1^k, t') \rightarrow (q', a')$ to generate a time-lock puzzle q' of hardness t', whose solution should verify the property a'. The puzzle q' is sent to $\mathcal{S}$ which replies with v'. Finally, $\mathcal{F}$ checks whether v' verifies the property a'. In case a' does not hold, $\mathcal{F}$ stops without outputting any message to the environment. Otherwise, for every value $i \in \{1, \ldots, k\}$, $\mathcal{F}$ generates a puzzle q_i of hardness $t_i = 2^i$. Let j be such that $2^j \leq t' < 2^{j+1}$, so $j \in \{1, \ldots, k\}$.

For the puzzle q_j, $\mathcal{F}$ computes the solution v_j. $\mathcal{F}$ can efficiently compute this solution as it knows the additional information a_j. Additionally, $\mathcal{F}$ chooses r uniformly at random from $\{0,1\}^{2k}$. Without loss of generality, we can assume the solution v of each puzzle q generated using the parameters 1^k and t has length $2 \cdot k$. Indeed, we can prepend with 0's the string v such that its length reaches $2 \cdot k$. It is easy to see that after this operation, the properties stated in lemma 29 still hold. The output of $\mathcal{F}$ to the environment is the tuple $(q_1, \ldots, q_k, r, HC(v_j, r))$, where HC is the hard-core predicate of $(\mathcal{G}, \mathcal{V})$ as given by lemma 29.

For each hardness t', we call $P(t')$ the distribution of the view of $\mathcal{Z}$ when interacting in the ideal world.

The real world protocol π, is defined similarly to $\mathcal{F}$, the only difference is the final output: π outputs to $\mathcal{Z}$ a tuple $(q_1, \ldots, q_k, r, b)$, with r randomly chosen from $\{0,1\}^{2k}$ and b randomly chosen from $\{0,1\}$. For each hardness t used by the adversary $\mathcal{A}$ when interacting with $\mathcal{Z}$, we call $R(t)$ the distribution of the view of $\mathcal{Z}$ when interacting in the real world.

The proof has two steps. First, we show that π is as secure as $\mathcal{F}$ with respect to weak specialized simulator UC security. Let $\mathcal{D}$ be a distinguisher of hardness $t_{\mathcal{D}}$ (i.e., it can solve puzzles of hardness less or equal to $t_{\mathcal{D}}$ with overwhelming probability but it cannot solve puzzles of hardness greater than $t_{\mathcal{D}}$ with more than negligible probability) and an adversary $\mathcal{A}$ of hardness $t_{\mathcal{A}}$. Let l be the minimum value such that $2^l > \max(t_{\mathcal{D}}, t_{\mathcal{A}})$. We now require that the simulator $\mathcal{S}$ has hardness t' such that $t' \geq 2^l$. As we will see next, this is one of the constraints necessary for making the two distributions $R(t')$ and $P(t)$ indistinguishable to $\mathcal{D}$.

The intuition is that in the ideal world $\mathcal{D}$ would have to solve a puzzle with hardness larger than $t_{\mathcal{D}}$ and learn the hard-core bit for such a puzzle. According to lemma 30, this hard-core bit is indistinguishable from a random bit, which is actually what the protocol π outputs to the environment.

More formally, let $(\mathcal{A}, \mathcal{Z}, \mathcal{D})$ be a triple of real world adversary, environment and distinguisher and let 1^k be the security parameter. Then, let e be such that the length of the messages sent by $\mathcal{Z}$ to $\mathcal{D}$ is bounded above by k^e. From (4.3), there exists $f_e^{\mathcal{D}}$ such that for every polynomial p there exists k_p^0 such that

$$\sup_{t \geq k^{f_e^{\mathcal{D}}}, |h| \leq k^e} Pr[(q', a') \leftarrow \mathcal{G}(1^k, t') : \mathcal{V}(1^k, a', \mathcal{D}(1^k, q', h)) = 1] < \frac{1}{p(k)}$$

for every $k > k_p^0$. This intuitively means that $\mathcal{D}$ can solve puzzles of hardness larger than $k^{f_e^{\mathcal{D}}}$ only with negligible probability. Given $\mathcal{A}$, in an analogue way we define $k^{f_e^{\mathcal{A}}}$ and k_p^1. With the notation used in the description of π and $\mathcal{F}$, it now becomes clear that we can take $t_{\mathcal{D}} = k^{f_e^{\mathcal{D}}}$ and $t_{\mathcal{A}} = k^{f_e^{\mathcal{A}}}$.

We construct $\mathcal{S}$ such that there exists a negligible function ϵ and k_2 such that for every $k \geq k_2$ and for every distribution of auxiliary input z we have:

$$|(Pr(\mathcal{D}(EXEC_{\mathcal{A},\pi,\mathcal{Z}}(k, z)) = 1) - (Pr(\mathcal{D}(EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z)) = 1)| < \epsilon(k). \tag{4.5}$$

We take k_2 such that for every $k \geq k_2$, it holds that $\max(t_{\mathcal{A}}, t_{\mathcal{D}}) < 2^k$.

For a given $t_{\mathcal{A}}$ and $t_{\mathcal{D}}$ and for l defined as above, let f' be such that for sufficiently large k, $2^l \leq k^{f'} \leq 2^k$. Let $\mathcal{S}$ be the simulator of hardness $k^{f'}$ that as first reply to $\mathcal{F}$ sends $t' := k^{f'}$. According to (4.4), there exists m such that for $d := f'$ there exists $C_{f'}$ such that

$$Pr[(q', a') \leftarrow \mathcal{G}(1^k, k^{f'}); v' \leftarrow C_{f'}(1^k, q') : \mathcal{V}(1^k, a', v') = 1 \wedge |v'| \leq k^m]$$

is overwhelming in k. When $\mathcal{F}$ sends a puzzle q' to $\mathcal{S}$, the simulator invokes $C_{f'}$ for $(1^k, q')$ and sends to $\mathcal{F}$ the output v' of $C_{f'}$. Internally, $\mathcal{S}$ simulates the adversary $\mathcal{A}$ and emulates the messages that the adversary would receive from $\mathcal{Z}$ and π as follows: When $\mathcal{F}$ requires the value of the computational hardness from $\mathcal{S}$, then $\mathcal{S}$ acts as π and requires the computational hardness from simulated $\mathcal{A}$. When $\mathcal{S}$ receives t from $\mathcal{A}$, then it invokes $Gen(1^k, t)$, obtaining output (q, a) and forwards to simulated $\mathcal{A}$ the puzzle q. Moreover, any message that internal $\mathcal{A}$ wants to send to the environment, $\mathcal{S}$ forwards it

to $\mathcal{Z}$. Any message for $\mathcal{A}$ coming from $\mathcal{Z}$ is immediately forwarded by $\mathcal{S}$ to the internally simulated adversary. This completes the construction of $\mathcal{S}$.

By construction, $\mathcal{S}$ solves the puzzle sent by $\mathcal{F}$ with overwhelming probability and hence the output of $\mathcal{F}$ to $\mathcal{Z}$ is $(q_1, \ldots, q_k, r, HC(v_j, r))$ with the same probability. The view of $\mathcal{Z}$ in the real world is $(1^k, t, q, v, (q_1, \ldots, q_k, r, b))$ and the view of $\mathcal{Z}$ in the ideal world is $(1^k, t, q, v, (q_1, \ldots, q_k, r, HC(v_j, r)))$. One may argue of course that the view of $\mathcal{Z}$ may or may not contain the values t, q, v, depending on the adversary $\mathcal{A}$. Also, additionally to the view(s) stated above, the environment could output the interaction that it has with $\mathcal{A}$ besides messages t, q, v. However, for the analysis of this proof, the views considered above are the worst case scenario that would allow a distinguisher to tell apart the two worlds.

By applying lemma 30 for the distinguisher $\mathcal{D}$ and polynomial p, there exists k_p and t_p, such that for every $k > k_p$ and $t > t_p$, the advantage of $\mathcal{D}$ for distinguishing between the distributions of $((q, r), b)$ and $((q, r), HC(v, r))$ (with $\mathcal{G}(1^k, t) \leftarrow (q, a)$, v the solution to q, b the random bit and r the uniformly distributed string of k bits) is less than $\frac{1}{p(k)}$. Hence, additionally to the previous constraints on k and t', we take k such that $k > k_p$ and $\max\{t_{\mathcal{A}}, t_{\mathcal{D}}, t_p\} < 2^k$ and t' such that $t' > \max\{t_{\mathcal{A}}, t_{\mathcal{D}}, t_p\}$. With this we can conclude that the real and the ideal world views are indistinguishable to $\mathcal{D}$.

Second, we prove that π is not as secure as $\mathcal{F}$ with respect to specialized simulator UC security. Intuitively, for every hardness $t_{\mathcal{S}}$ (polynomial in the security parameter k) of a simulator machine $\mathcal{S}$, there exists a distinguisher $\mathcal{D}_{\mathcal{S}}$ such that for every $t \leq t_{\mathcal{S}}$, $\mathcal{D}_{\mathcal{S}}$ can solve puzzles of hardness t. As we will see next, $\mathcal{D}_{\mathcal{S}}$ uses this property to distinguish with non-negligible probability between the environment's output distribution in the real and in the ideal world.

Formally, let $\mathcal{A}$ be the real world adversary that can solve puzzles of hardness $t_{\mathcal{A}}$ such that when receiving its input from the environment, it replies to π with $t_{\mathcal{A}}$ and the corresponding correct solution for the puzzle received. Let $\mathcal{Z}$ be the environment that just sends the security parameter to all parties (i.e., including the adversarial parties), receives their outputs and then outputs as view the messages received from the honest parties (i.e., protocol π in the real world or $\mathcal{F}$ in the ideal world). For every simulator $\mathcal{S}$, we show that there exists a distinguisher $\mathcal{D}_{\mathcal{S}}$ and a distribution for the auxiliary input z such that:

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}_{\mathcal{S}}}{\not\equiv} \{EXEC_{\pi,\mathcal{A},\mathcal{Z}}(k, z)\}_{k\in\mathbb{N}}.$$

Given $\mathcal{S}$ of hardness $t_{\mathcal{S}}$, we choose $\mathcal{D}_{\mathcal{S}}$ such that it can solve puzzles of hardness at least $t_{\mathcal{D}} = max(t_{\mathcal{S}}, t_{\mathcal{A}})$ with overwhelming probability in k. Such a $\mathcal{D}_{\mathcal{S}}$ exists according to (4.4). Additionally, after receiving the view of $\mathcal{Z}$, $\mathcal{D}_{\mathcal{S}}$ solves one by one each puzzle q_i included in that view that has associated hardness $t_i \leq t_{\mathcal{D}}$ and it obtains each time the corresponding correct and unique solution v_i with overwhelming probability. Then $\mathcal{D}_{\mathcal{S}}$ evaluates $HC(v_i, r)$. Lets call m the last bit in the output of the honest party (i.e., $\mathcal{F}$ or π) to $\mathcal{Z}$[7]. Next, $\mathcal{D}_{\mathcal{S}}$ checks if $m \neq HC(v_i, r)$ for all i as defined above. If this holds, then $\mathcal{D}$ outputs 1, otherwise it outputs 0.

[7]Due to the definition of $\mathcal{Z}$, the string m is also a part of the output of the environment.

If m is part of the view of the real world, then according to the definition of π, m is a random bit in $\{0,1\}$ so it is different from a given bit $HC(v_i, r)$ with probability $\frac{1}{2}$. This is equivalent to $\mathcal{D}_\mathcal{S}$ outputting 1 with probability $\frac{1}{2^{\log 2t_\mathcal{D}}} = \frac{1}{t_\mathcal{D}}$ when the view of $\mathcal{Z}$ is from the real world. Similarly, if m is part of the view of $\mathcal{Z}$ in the ideal world, then there exists at least an index i such that $HC(v_i, r)$ can be computed by $\mathcal{D}_\mathcal{S}$ and $m = HC(v_i, r)$; so $\mathcal{D}_\mathcal{S}$ outputs 1 with probability 0. This implies $\mathcal{D}_\mathcal{S}$ can distinguish at least with non-negligible probability $\frac{1}{t_\mathcal{D}}$ between the output distributions from the two worlds and this concludes the proof.[8] □

We are now ready to conclude that 1-bit specialized simulator UC security and specialized simulator UC security are not equivalent notions. By putting together the results from lemma 26 and from lemma 31 we obtain:

Theorem 32 (1-bit SSUC and SSUC Not Equivalent)**.** *Assume Blum integer time-lock puzzles exist. Then there are protocols secure with respect to 1-bit specialized simulator UC security which are not secure with respect to specialized simulator UC security.*

4.4.3 Discussion

The separation result presented in theorem 32 is conditioned on the existence of Blum integer time-lock puzzles, which in turn is based on the RSA assumption. To the best of our knowledge, the only other known time-lock puzzle constructions are possible in the random oracle model [76, 75]. However, these constructions cannot replace the Blum integer time-lock puzzle in our proof method for the separation lemma 31.

On one hand, the construction from [76] allows only a fixed linear gap between the time needed for generating and the time needed for solving a puzzle; with the Blum integer time-lock puzzles we can control the hardness of the puzzle. On the other hand, the puzzles from [75] allow for more fine tuning of the time gap, but it is not clear how to use them to construct the one-way functions that allowed us to conclude the separation lemma 31. This is the case since the constructions from [75] do not allow for generation of hard enough solutions in efficient time. We did not encounter this impediment in the case of Blum integer time-lock puzzles, since those puzzles were generated together with a trap-door which allowed for efficient computation of the solution.

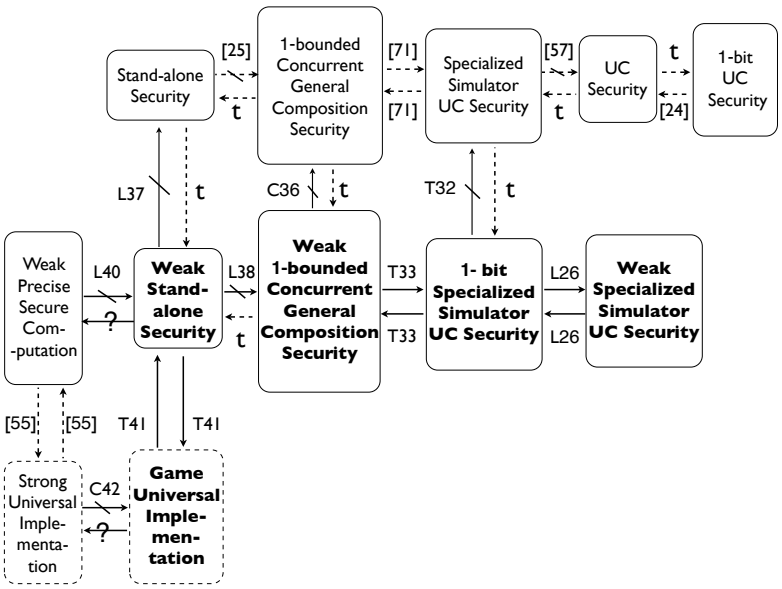
It is an open question how to construct a time lock puzzle based on general cryptographic assumption (e.g., the existence of one-way functions) or to show that such a construction cannot be used for our separation result.

4.5 Equivalence of Security Notions

Implication relations among various security notions with respect to computational security are depicted in Fig. 4.1. Previously existing notions are written in a regular font, while the notions defined in this chapter are written in a boldface font. The continuos line

[8] Since $\mathcal{D}$ is a polynomial time machine, its hardness $t_\mathcal{D}$ is also a polynomial in the security parameter k, so the function $\frac{1}{t_\mathcal{D}}$ is non-negligible.

Figure 4.1: Implication Relations among Computational Security Concepts



arrows depict relations we prove in this chapter[9]; all other relations have been previously known (the descriptor in the square brackets denotes the reference) or can be trivially derived (i.e., denoted by letter t). Continuous line frames highlight security notions, while dotted frames highlight game-theoretic concepts. Finally, open questions are marked by question marks.

It is a well-known result that UC security and 1-bit UC security are equivalent [24]. It has been also shown [57] that specialized simulator UC security does not imply UC security. Moreover, specialized simulator UC security is equivalent to security under 1-bounded concurrent general composition [71]. It has been shown [25] that stand-alone security does not imply specialized simulator UC security.[10] The implication in the opposite direction holds trivially. Similarly, it is trivial to see that universal composability implies specialized simulator UC security.

Our first result in this section proves that weak security under 1-bounded concurrent general composition is equivalent to 1-bit specialized simulator UC security. A similar proof technique has been used in [71], however, our proof requires more technicalities.

Theorem 33 (Equivalence between Weak 1-bounded CGC Security and 1-bit SSUC Security)**.** *Let ρ be a protocol and $\mathcal{F}$ an ideal functionality. We have that ρ implements $\mathcal{F}$ under weak 1-bounded concurrent general composition security, if and only if ρ securely computes $\mathcal{F}$ under 1-bit specialized simulator UC security.*

[9] A letter and number next to an arrow represent the theorem T or the lemma L or the corollary C where the respective result is shown in this paper.

[10] In order to preserve the symmetry and clarity of our picture, we have indicated that the result in [25] is that stand-alone security does not imply 1-bounded concurrent general composition. This is indeed a immediate consequence of combining the results from [25] and [71].

Proof. In the following we need *one-time information-theoretic message authentication codes* so we include the definition below.

Definition 34 (One-Time Information-Theoretic Message Authentication Code)**.** *A one-time information-theoretic message authentication code is a triple* $(Gen, Mac, Verify)$ *where* $Gen(1^n)$ *outputs a key* k*,* $Mac(k, x)$ *outputs a tag* t *(obtained using* k*) for the message* x *of length* n *and* $Verify(k, m, t)$ *outputs* 0 *or* 1*. The correctness property requires that* $\forall n, \forall k$ *in the range of* $Gen(1^n)$ *and* $\forall x \in \{0,1\}^n$ *we have* $Verify(k, x, Mac(k, x)) = 1$*.*

Moreover, the following security property is fulfilled. For every adversary $\mathcal{A}$ *such that*

$$Pr[(x', t') \leftarrow A(x, t) \wedge x' \neq x \wedge Verify(k, x', t') = 1 : \\ : x \leftarrow A(1^n), k \leftarrow Gen(1^n), t \leftarrow Mac(k, x)]$$

is negligible in n*.*

Next, we prove an important lemma which will allow us to conclude theorem 33.

Lemma 35 (Equivalence between Weak Security under 1-bounded Concurrent General Composition and Weak Specialized Simulator UC Security)**.** *Let* ρ *be a protocol and* $\mathcal{F}$ *an ideal functionality. Then* ρ *securely computes* $\mathcal{F}$ *under 1-bounded concurrent general composition with weak security if and only if* ρ *securely implements* $\mathcal{F}$ *under weak specialized simulator UC security.*

Indeed, putting together lemma 35 and lemma 26 we can conclude the theorem. □

And now we give in full detail the proof for lemma 35.

Proof. As expected, the more involved part of the proof is the implication from weak security under 1-bounded concurrent general composition to weak specialized simulator UC security. The reverse direction can be shown analogously to the proof existing in the initial version of [71].

Let $R_1, \ldots, R_m$ be the parties for ρ. Let $(\mathcal{A}, \mathcal{Z}, \mathcal{D})$ be a triple consisting of UC real world adversary (possibly adaptive), environment and distinguisher. We need to show there exists an UC ideal world simulator $\mathcal{S}$ such that the views of the environment in real world and in the ideal world cannot be distinguished by $\mathcal{D}$. The adversary $\mathcal{A}$ may not corrupt any party, in which case $\mathcal{A}$ is still capable of scheduling messages in the network. Additionally, in the UC model the only messages that $\mathcal{A}$ has no control of, even by scheduling, are the input messages that the environment $\mathcal{Z}$ writes directly on the input tapes of the parties and the output messages that $\mathcal{Z}$ reads directly from the parties' output tapes.

The intuition behind the proof is as follows: We use the fact that ρ composed with an instance of any protocol π (i.e., even one that has more parties than ρ) is secure and the security is of course in the sense of definition 24. We construct a protocol π for $m+2$ parties that besides the m parties of ρ has $P_{\mathcal{Z}}$ and $P_{\mathcal{A}}$ playing the role of $\mathcal{Z}$ and $\mathcal{A}$ respectively. In this way, we reduce the proof of weak specialized simulator UC security of ρ to weak security under 1-bounded concurrent general composition. As mentioned above, the adaptive adversary $\mathcal{A}$ could corrupt everyone or could corrupt no party and

act as a network adversary. Thus, the motivation behind using the two extra parties in the protocol π is to ensure there is always an honest entity and also a corrupted entity, the same way as in the UC model. In order to model the ideally secure channels that the specialized simulator UC (real/ideal) setting ensures by definition between $\mathcal{Z}$ and the parties of ρ, we use one-time pads and one-time authentication MACs in the concurrent general composition world between $P_{\mathcal{Z}}$ and the parties of ρ.

However, it is important to know how long should the keys be. They should suffice for all necessary encrypted and authenticated communication. Let q be a polynomial such that for every security parameter n and for every i the value $q(n)$ bounds above the length of encryption and authentication keys needed between each pair $P_{\mathcal{Z}}$ and P_i with $i \in \{1, \ldots, m\}$. We postpone until after the description of π why such polynomial q exists and how it is computed.

Formally, protocol π is described below and it can be used for both the real and the ideal concurrent general composability worlds.

1. *Inputs:* Each party P_i with $i \in \{1, \ldots, m\}$ receives a pair (k^i_{mac}, k^i_{enc}) of keys.[11] Party $P_{\mathcal{A}}$ receives the empty string λ as input. Party P_{m+1} receives an input z and also the tuples $((k^1_{mac}, k^1_{enc}), \ldots, (k^m_{mac}, k^m_{enc}))$.[12]

2. *Outputs:* The protocol outputs whatever $P_{\mathcal{Z}}$ outputs. The rest of the parties of π output an empty string λ.

3. *Instructions for P_i, with $i \in \{1, \ldots, m\}$:* When P_i receives $(input, x_i, t_i)$ from $P_{\mathcal{A}}$, it verifies the correctness of the tag. If verification succeeds, it computes $m_i = x_i \oplus k^i_{enc}$ and sends m_i either to its corresponding ITM that emulates R_i of ρ or to the functionality $\mathcal{F}$. (This choice depends on whether π is part of the composed protocol π^ρ or $\pi^{\mathcal{F}}$. As a reminder, independently of the channels model, an adversary in the concurrent general composability world cannot interfere in any way with the messages that an uncorrupted party of π wants to send to its associated ITM for ρ.) If verification fails, then P_i halts. When the ITM emulating R_i or when $\mathcal{F}$ respectively sends the output value y_i to P_i, then P_i computes $e_i = y_i \oplus k^i_{enc}$ and $v_i = MAC(k^i_{mac}, e_i)$ and sends the message $(output, e_i, v_i)$ to party $P_{\mathcal{Z}}$.

4. *Instructions for $P_{\mathcal{Z}}$:* Upon receiving an input value z, it uses it for internally invoking $\mathcal{Z}$. When internal $\mathcal{Z}$ wants to send a message $(input, m_i)$ to party i, then $P_{\mathcal{Z}}$ computes $x_i = m_i \oplus k^i_{enc}$ and $t_i = MAC(k^i_{mac}, x_i)$ and sends $(input, x_i, t_i)$ to P_i. When $P_{\mathcal{Z}}$ receives a message $(output, y_i, v_i)$ from party P_i, it first checks the

[11] For ease of notation, we use one encryption key and one MAC key per party P_i, as they can be considered long enough to encrypt and authenticate the entire communication between P_i and $P_{\mathcal{Z}}$. However, for each different encryption (authentication) that needs to be performed, a new part of the string k^i_{enc} (and k^i_{mac}, respectively) is used.

[12] The input of π may have any distribution and the indistinguishability between the real and the ideal concurrent general composability worlds would still be preserved. However, for this proof we restrict the inputs to encryption keys (i.e., they are uniformly distributed in $\{0,1\}^{q(k)}$) and MAC keys (i.e., they are generated with the *Gen* key generation algorithm).

correctness of the tag v_i. If verification succeeds, then $P_{\mathcal{Z}}$ computes $m_i = y_i \oplus k^i_{enc}$ and stores m_i. Otherwise, it halts. When internal $\mathcal{Z}$ wants to read the output tape of party i, then $P_{\mathcal{Z}}$ looks up if there is a message m_i stored from party P_i. If so, it writes m_i to corresponding tape of $\mathcal{Z}$, otherwise it just writes λ to $\mathcal{Z}$. Regarding the communication with its adversary, when $P_{\mathcal{Z}}$ receives a message from $\mathcal{Z}$ of the form $(\mathcal{Z}, \mathcal{A}, m)$, it forwards it to $P_{\mathcal{A}}$. Similarly when $P_{\mathcal{Z}}$ receives a message of the form $(\mathcal{A}, \mathcal{Z}, m)$ from $P_{\mathcal{A}}$, it forwards it internally to $\mathcal{Z}$.

5. *Instructions for $P_{\mathcal{A}}$:* This party has no predefined instructions. $P_{\mathcal{A}}$ is needed in order to provide a means of communication for the adversary of the general concurrent composition setting which in this model can only send messages through a corrupted party.[13]

We now explain how the polynomial q is chosen. Since the communication between $P_{\mathcal{Z}}$ and each of the parties P_i with $i \in \{1, \ldots, m\}$ has to be secure and authenticated, the length of the secret keys for the one-time pad and and for the one-time MAC should be long enough. The intuition is that the length of the encryption keys shared by $P_{\mathcal{Z}}$ and P_i is bounded above by the length of the longest string that machine $\mathcal{Z}$ can write plus the longest string that R_i can write. Since both machines are polynomially bounded and they are fixed before the protocol π is constructed, there exist a polynomial q_i such that $q_i(n)$ bounds from above the length of the common encryption keys for every security parameter n. Moreover, the length of the secret key needed for the authenticated messages between $P_{\mathcal{Z}}$ and P_i is at most as long as the one-time pad secret keys. Putting the above arguments together we conclude there exists a polynomial q such that $q(n) \geq max\{q_1(n), \ldots, q_m(n)\}$.

For the protocol π given above we construct an adversary $\mathcal{A}_\pi$ interacting with the composed protocol π^ρ. Intuitively, the task of $\mathcal{A}_\pi$ is to enable the communication among $\mathcal{Z}$ (invoked by $P_{\mathcal{Z}}$), $\mathcal{A}$ (invoked by the adversary $\mathcal{A}_\pi$) and the ITMs implementing ρ, in the same way as it happens in the UC real world. In order to make this work and for reasons explained above, the adversary A_π corrupts $P_{\mathcal{A}}$. We construct the adversary $\mathcal{A}_\pi$ as follows: It internally runs the code of the UC real world adversary $\mathcal{A}$ and if $\mathcal{A}$ corrupts a party R_i, then $\mathcal{A}_\pi$ corrupts the party P_i together with its corresponding ITM for computing ρ. The intuition is that $\mathcal{A}_\pi$ instructs the corrupted parties of π to run the protocol as before, while their corresponding corrupted ITMs follow the instructions of $\mathcal{A}$. The handling of messages by $\mathcal{A}_\pi$ is as follows:

1. Input messages $(input, x_i, t_i)$ sent by $P_{\mathcal{Z}}$ are forwarded *immediately* by $\mathcal{A}_\pi$ to P_i; Output messages $(output, e_i, v_i)$ sent by P_i are *immediately* forwarded by $\mathcal{A}_\pi$ to $P_{\mathcal{Z}}$.

   Moreover, as soon as party P_i is corrupted, its current state and all its previously received messages are sent by$\mathcal{A}_\pi$ to $\mathcal{A}$. The information that $\mathcal{Z}$ expects to receive

[13] This is in contrast to the UC model where even if none of the protocol parties is corrupted, the adversary can interact with the environment $\mathcal{Z}$.

upon corruption is sent by $\mathcal{A}_\pi$ to $P_\mathcal{Z}$. All messages received from this point on by P_i are forwarded by A_π to $\mathcal{A}$.

2. When $P_\mathcal{Z}$ sends a message $(\mathcal{Z}, \mathcal{A}, m)$ to party $P_\mathcal{A}$, then $\mathcal{A}_\pi$ forwards it to its internal run of $\mathcal{A}$ as if coming from $\mathcal{Z}$. The messages $(\mathcal{A}, \mathcal{Z}, m)$ that $\mathcal{A}$ wants to send to $\mathcal{Z}$ are forwarded by $\mathcal{A}_\pi$ to $P_\mathcal{Z}$;

3. All messages that $\mathcal{A}$ instructs a corrupted party R_i to send to an uncorrupted party R_j will be forwarded by $\mathcal{A}_\pi$ to the corresponding ITM of P_j as if coming from the corresponding ITM of P_i; However $\mathcal{A}$ schedules messages among parties R_i, $i \in \{1, \ldots, m\}$, $\mathcal{A}_\pi$ does the same for the messages among the corresponding ITMs of parties P_i, $i \in \{1, \ldots, m\}$.

4. The adversary $\mathcal{A}_\pi$ has no control over the messages between an uncorrupted P_i and its corresponding ITM for computing ρ.

After having defined protocol π and adversary $\mathcal{A}_\pi$, we prove that the output of $P_\mathcal{Z}$ in the execution of π^ρ (which we denote by $\{REAL_{\pi^\rho,\mathcal{A}_\pi}(k, \bar{z})|P_\mathcal{Z}\}_{k\in\mathbb{N}}$) and the output of $\mathcal{Z}$ in the UC real world are identically distributed. For every $z \in \{0,1\}^*$, let $\bar{z} = (z, k^1_{enc}, k^1_{mac}, \ldots, k^m_{enc}, k^m_{mac}), \lambda, (k^1_{enc}, k^1_{mac}), \ldots, (k^m_{enc}, k^m_{mac})$ be the vector where the first component is the input to $P_\mathcal{Z}$, the second component is the input to $P_\mathcal{A}$, and each of the other components is the input to a party P_i, for $i \in \{1, \ldots, m\}$.

We prove that for every $z \in \{0,1\}^*$, for every k^i_{enc} randomly chosen from $\{0,1\}^{q(n)}$ and for every k^i_{mac} generated by $Gen(1^{q(n)})$ we have:

$$\{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k, z)\}_{k\in\mathbb{N}} \equiv \{REAL_{\pi^\rho,\mathcal{A}_\pi}(k, \bar{z})|P_\mathcal{Z}\}_{k\in\mathbb{N}} \tag{4.6}$$

which as a special case, of course implies:

$$\{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k, z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{REAL_{\pi^\rho,\mathcal{A}_\pi}(k, \bar{z})|P_\mathcal{Z}\}_{k\in\mathbb{N}} \tag{4.7}$$

Our claim is based on the following facts: First, the inputs to parties are provided by $\mathcal{Z}$ in both models, as in the composed protocol π^ρ the party $P_\mathcal{Z}$ distributing the inputs is internally running $\mathcal{Z}$. Thus the input messages in both worlds are identically distributed. By construction, $\mathcal{A}_\pi$ follows the instructions of $\mathcal{A}$ (i.e., for network scheduling and for the corrupted messages among the corresponding ITMs for $P_1, \ldots, P_m$) and it also provides an internal perfect emulation for the view of $\mathcal{A}$. Once an honest party P_i receives an input, it immediately writes it on the input tape of its associated ITM for ρ. This implies that such a party with its ITM follows the same protocol as the corresponding party of ρ. We can now conclude that the view of $\mathcal{Z}$ in the UC real world for ρ and the view of $P_\mathcal{A}$ in the composed protocol π^ρ are identically distributed, so equation (4.6) follows.

According to the definition of weak security under 1-bounded general concurrent composition, we know that for the triple π, $\mathcal{A}_\pi$ and $\mathcal{D}$, there exists a polynomially bounded hybrid simulator $\mathcal{S}_\pi$ such that for every $\bar{z}$ defined as above we have:

$$\{HYBRID^\mathcal{F}_{\pi,\mathcal{S}_\pi}(k, \bar{z})\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{REAL_{\pi^\rho,\mathcal{A}_\pi}(k, \bar{z})\}_{k\in\mathbb{N}}. \tag{4.8}$$

We are now ready to construct a simulator $\mathcal{S}$ for the UC ideal world by using $\mathcal{S}_\pi$. We have to observe that in the hybrid world of concurrent general composition and in the UC real world the messages going over the network are the same. Intuitively, the new simulator $\mathcal{S}$ has to have a scheduling indistinguishable from that of $\mathcal{S}_\pi$ so the constructed simulator $\mathcal{S}$ internally invokes $\mathcal{S}_\pi$. As a short summary of the messages that have to be defined for $\mathcal{S}$: communication from $\mathcal{S}$ to $\mathcal{F}$, communication from $\mathcal{S}$ to $\mathcal{Z}$ and network scheduling (between parties of π and $\mathcal{F}$). As $\mathcal{S}$ internally runs $\mathcal{S}_\pi$, the constructed adversary has to provide an emulation for the entities that $\mathcal{S}_\pi$ is interacting with: the parties of $\pi^{\mathcal{F}}$.[14] Such an emulation of $\pi^{\mathcal{F}}$ consists of defining the input/output messages of the parties, the messages among $P_1, \ldots, P_m, P_{\mathcal{A}}, P_{\mathcal{Z}}$ and the messages from $P_1, \ldots, P_m$ to $\mathcal{F}$. The description of $\mathcal{S}$ is as follows:

1. Messages sent by $\mathcal{F}$ to $\mathcal{S}$ are forwarded to the internally simulated $\mathcal{S}_\pi$. The messages that internally emulated $\mathcal{S}_\pi$ wants to send to $\mathcal{F}$ are forwarded by $\mathcal{S}$ to $\mathcal{F}$. Similarly, the messages that internally emulated $\mathcal{S}_\pi$ sends to the internally simulated $P_{\mathcal{Z}}$ are forwarded by $\mathcal{S}$ to $\mathcal{Z}$. The messages that $\mathcal{Z}$ sends to $\mathcal{S}$ are forwarded internally to $\mathcal{S}_\pi$ as coming from $\mathcal{P}_{\mathcal{Z}}$.

2. Simulation of $P_{\mathcal{Z}}$: When $\mathcal{S}$ receives a message $(\mathcal{Z}, \mathcal{A}, m)$ from $\mathcal{Z}$, it sends it to the internally emulated $P_{\mathcal{A}}$ as if coming from the emulated $P_{\mathcal{Z}}$. When $\mathcal{S}_\pi$ instructs emulated $P_{\mathcal{A}}$ (which is a corrupted party) to send a message $(\mathcal{A}, \mathcal{Z}, m)$ to $P_{\mathcal{Z}}$, the simulator $\mathcal{S}$ forwards the same message to $\mathcal{Z}$.

3. Simulation of $P_{\mathcal{A}}$: As an uncorrupted party, $P_{\mathcal{A}}$ does not do anything, just receives messages from $\mathcal{P}_{\mathcal{Z}}$. These messages were actually sent by $\mathcal{Z}$ to $\mathcal{S}$. When internal $\mathcal{S}_\pi$ wants to corrupt emulated $P_{\mathcal{A}}$ (and this is actually the first party of π that $\mathcal{S}_\pi$ corrupts), then all that $\mathcal{S}$ needs to do is to send $\mathcal{S}_\pi$ all the messages it received from $\mathcal{Z}$.

4. In the UC ideal world, when an uncorrupted dummy party $\mathcal{D}_i$ receives an $(input, m_i)$ from the environment $\mathcal{Z}$, it immediately forwards the input value to $\mathcal{F}$. When $\mathcal{S}$ receives over the network such a message[15], it generates x_i randomly in the length of the received input and a MAC key k^i_{mac} with the corresponding generation algorithm, computes $t_i = MAC(k^i_{mac}, x_i)$ and internally sends the message $(input, x_i, t_i)$ to P_i as if coming from $P_{\mathcal{Z}}$. When $\mathcal{F}$ wants to send an output message (i.e., same discussion as above) to $\mathcal{D}_i$, the simulator $\mathcal{S}$ internally randomly generates y_i in the length of the output received over the network, then computes $v_i = MAC(k^i_{mac}, y_i)$ and sends message $(output, y_i, v_i)$ to $\mathcal{S}_\pi$ as if coming from the ideal functionality in $\pi^{\mathcal{F}}$.

   Whenever $\mathcal{S}_\pi$ corrupts a party P_i, we have one of the following 3 cases that define the behavior of $\mathcal{S}$:

[14] Observe that it is actually sufficient to simulate the parties of π without the messages sent by $\mathcal{F}$ as they can be forwarded by $\mathcal{S}$ from its communication with the ideal functionality.

[15] If the channels between the dummy parties and the ideal functionality are ideally secure, then the value received could also be encrypted, so what is forwarded should not depend on what is received.

-For a corrupted party P_i, that $\mathcal{S}_\pi$ wants to corrupt before a certain input is sent to it by $P_\mathcal{Z}$, the simulator $\mathcal{S}$ corrupts the corresponding dummy party $\mathcal{D}_i$, informs $\mathcal{Z}$ about it and generates a correct key pair (k^i_{enc}, k^i_{mac}) for encryption and authentication and gives them to $\mathcal{S}_\pi$.[16] When input value(s) x_i for $\mathcal{D}_i$ are received by $\mathcal{S}$ over the network[17], then $\mathcal{S}$ computes $y_i = x_i \oplus k^i_{enc}$ and $v_i = MAC(k^i_{mac}, y_i)$. Next, $\mathcal{S}$ sends (y_i, v_i) to $\mathcal{S}_\pi$ as coming from $P_\mathcal{Z}$. When an output o_i is sent by $\mathcal{F}$ to $\mathcal{D}_i$, then $\mathcal{S}$ computes $c_i = x_i \oplus k^i_{enc}$ and $t_i = MAC(k^i_{mac}, y_i)$ and sends (c_i, t_i) to simulated P_i as if coming from $P_\mathcal{Z}$.

-For a corrupted party P_i, that $\mathcal{S}_\pi$ corrupts after a certain input is sent to P_i, but before the corresponding output is received, first the emulation from the case of uncorrupted input takes place. Thus, a message (y_i, v_i) has been already sent from $\mathcal{P}_\mathcal{Z}$ to P_i. When the corruption takes place, the simulator $\mathcal{S}$ corrupts the corresponding dummy party $\mathcal{D}_i$, informs $\mathcal{Z}$ about it and generates a correct key pair (k^i_{enc}, k^i_{mac}) for encryption and authentication. Then it sends the pair to $\mathcal{S}_\pi$, together with the correct input x_i in plain. When an output o_i is sent by $\mathcal{F}$ to $\mathcal{D}_i$, then $\mathcal{S}$ computes $c_i = x_i \oplus k^i_{enc}$ and $t_i = MAC(k^i_{mac}, y_i)$ and sends (c_i, t_i) to simulated P_i as if coming from $P_\mathcal{Z}$.

-For a corrupted party P_i that $\mathcal{S}_\pi$ corrupts after a certain input is sent to it and after the corresponding output is received, the simulator $\mathcal{S}$ corrupts the corresponding dummy party $\mathcal{D}_i$ and informs $\mathcal{Z}$ about the corruption.[18] Then $\mathcal{S}$ reads in plain the input and output values received by $\mathcal{D}_i$ and, using the simulated encrypted messages, computes the corresponding encryption keys which are sent to S_π as P_i input.

5. The following is valid only for honest parties P_i: When $\mathcal{S}_\pi$ delivers a message from P_i to the ideal functionality in $\pi^\mathcal{F}$, then $\mathcal{S}$ delivers the same message from $\mathcal{D}_i$ to $\mathcal{F}$.[19] When $\mathcal{S}_\pi$ delivers an output from P_i to $P_\mathcal{Z}$, then $\mathcal{S}$ delivers the output from $\mathcal{F}$ to $\mathcal{D}_i$.[20]

In order to conclude the proof we have to show that the output of the executions in both hybrid composition world and UC ideal world can be distinguished only with negligible probability. For this we detail the following three steps: a proof that the view of internally emulated $\mathcal{S}_\pi$ is identical with $\pi^\mathcal{F}$, a proof that the messages in the two worlds (hybrid composition and the UC ideal world) are identically distributed and finally, a proof that the delivery of output messages happens in the same time in both worlds.

[16] This simulates the information that $\mathcal{S}_\pi$ should learn from the newly corrupted (simulated) party.

[17] As $\mathcal{D}_i$ is corrupted, they are received from $\mathcal{Z}$ unencrypted.

[18] The simulation done by $\mathcal{S}$ for uncorrupted P_i receiving encrypted and authenticated input and output from $\mathcal{P}_\mathcal{Z}$ already took place.

[19] Actually, the simulator $\mathcal{S}_\pi$ has to make two deliveries (from $P_\mathcal{Z}$ to P_i and from P_i to the ideal functionality in $\pi^\mathcal{F}$), before $\mathcal{S}$ does the delivery of message from $\mathcal{D}_i$ to $\mathcal{F}$.

[20] Similarly as above, the simulator $\mathcal{S}_\pi$ has to make two deliveries (the output of $\mathcal{F}$ to P_i and from P_i to $P_\mathcal{Z}$), before $\mathcal{S}$ does its delivery from $\mathcal{F}$ to $\mathcal{D}_i$.

We start by analyzing how $\mathcal{S}$ internally emulates $\mathcal{S}_\pi$. It is easy to see that by construction $\mathcal{S}_\pi$, internally invoked by $\mathcal{S}$, gets and delivers the same messages as $\mathcal{S}_\pi$ does in the concurrent general composition world.

Next, we look at the messages sent between entities in both worlds. In the ideal UC world, the inputs are sent by $\mathcal{Z}$ and in the hybrid world with $\pi^{\mathcal{F}}$, the inputs are sent by $P_{\mathcal{Z}}$ who runs $\mathcal{Z}$. The messages that are sent between $P_{\mathcal{Z}}$ (running $\mathcal{Z}$) and $P_{\mathcal{A}}$ (corrupted and controlled by $\mathcal{S}_\pi$), are the same as the messages sent in the UC ideal world between $\mathcal{Z}$ and $\mathcal{S}$ who runs $\mathcal{S}_\pi$. In both worlds, the messages sent by parties to the ideal functionality are the same: the honest parties just forward their inputs and the corrupted parties are instructed by $\mathcal{S}_\pi$ and respectively by $\mathcal{S}$ running $\mathcal{S}_\pi$. We only need to show that the delivery of messages is the same in both worlds. Combining this claim with the proof above, we obtain that the outputs of both worlds are computationally indistinguishable.

In the following, we compare message delivery in both worlds. It is clear that the messages between the adversary and the environment $\mathcal{Z}$ or party $P_{\mathcal{Z}}$ running $\mathcal{Z}$ are identically delivered. The same holds for messages between the parties and the ideal functionality. We treat in more detail the case of inputs and outputs delivery. By definition, in the UC world, the input messages are written by $\mathcal{Z}$ directly on the input tapes of the protocol parties and for the honest parties, the adversary has no control over this step.[21] In the execution of $\pi^{\mathcal{F}}$, $P_{\mathcal{Z}}$ is distributing the inputs to the rest of the parties, but they are scheduled by $\mathcal{S}_\pi$, so we cannot know when they are delivered. However, we ensure that in both worlds an input of an honest party reaches the ideal functionality in the same time. Indeed, this holds as an honest dummy party $\mathcal{D}_i$ once it receives its input, it immediately sends it to the ideal functionality. As simulator $\mathcal{S}$ delivers this message only after internally simulated $\mathcal{S}_\pi$ has delivered the same message to $\mathcal{F}$, we have shown the claim.

Similarly, we show that an output message is delivered to $\mathcal{Z}$ and to the party $P_{\mathcal{Z}}$ in the same time. Both entities have basically the same instructions. We assume the machine environment $\mathcal{Z}$ reads all output tapes whenever it is activated. This gives the most power to the environment to distinguish between the delivery of messages. By construction, $\mathcal{S}$ sends an output of $\mathcal{F}$ to an honest dummy party $\mathcal{D}_i$ only when $\mathcal{S}_\pi$ sends the same output to $P_{\mathcal{Z}}$. Once it receives its output, the honest $\mathcal{D}_i$ immediately writes this value on its output tape (and this can be read by $\mathcal{Z}$ at any time). Analogously, $\mathcal{Z}$, (which is internally run by $P_{\mathcal{Z}}$), can read at any time the tape with output messages sent for it. So we have that also the outputs from the ideal functionality are delivered simultaneously in both worlds. This implies that for every $\bar{z}$ defined as before we have:

$$\{HYBRID^{\mathcal{F}}_{\pi,\mathcal{S}_\pi}(k,\bar{z})|P_{\mathcal{Z}}\}_{k\in\mathbb{N}}\equiv\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \tag{4.9}$$

[21] However, in the UC ideal world, immediately after receiving inputs, the honest dummy parties are activated and they write their inputs on the communication tape for the ideal functionality. As the simulator is responsible for the delivery of messages, in this way it will learn that inputs have been sent to the ideal functionality.

Thus, it holds that:

$$\{HYBRID^{\mathcal{F}}_{\pi,\mathcal{S}_\pi}(k,\bar{z})|P_{\mathcal{Z}}\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \tag{4.10}$$

By combining relations (4.7),(4.8) and (4.10), we can conclude the proof. □

As a consequence of theorem 32 and of theorem 33, we are now also able to compare the notion of 1-bounded concurrent general composition security [71] with our variant, i.e., weak 1-bounded concurrent general composition security.

Corollary 36 (Weak 1-bounded CGC and 1-bounded CGC Not Equivalent)**.** *Assume Blum integer time-lock puzzles exist. Then there are protocols secure with respect to weak 1-bounded concurrent general composition which are not secure with respect to 1-bounded concurrent general composition.*

Next we provide an application for theorem 33. In fact, we show that there are protocols that are secure with respect to weak stand-alone security but they are not secure anymore in the standard stand-alone model.

Lemma 37 (Weak Security Does Not Imply Stand-alone Security)**.** *If Blum integer time-lock puzzles exist, then there are protocols that fulfill the weak security notion, but do not fulfill the stand-alone security notion.*

Proof. From theorem 33, weak security under 1-bounded concurrent general composition is equivalent to 1-bit specialized simulator UC. As shown in [71], stand-alone security under 1-bounded concurrent general composition is equivalent to specialized simulator UC. According to theorem 32, the two UC variants are not equivalent. This implies weak security and stand-alone security are also not equivalent. One may wonder if the equivalence result between UC security and specialized simulator UC security that is known to hold in the extended UC model does not hinder the correctness of this result. However, this is not the case. On one hand, in the extended UC model, specialized simulator UC security and UC security are equivalent. Combining this with the well known result of equivalence between UC security and 1-bit UC security, we obtain that in the extended UC model, specialized simulator UC security and 1-bit specialized UC security are equivalent. This equivalence should not look surprising, as it is obtained in a more "permissive" adversarial UC model. On the other hand, the results obtained in this chapter show that there is at least one composition operation under which weak security and stand-alone security are not equivalent. □

The results of lemma 38 and lemma 40 complete the graphical survey of relations between security notions presented in Fig. 4.1.

Lemma 38 (Weak Stand-alone Security Does Not Imply Weak 1-bounded CGC Security)**.** *There exists a protocol π which is secure with respect to weak stand-alone model, but is not secure with respect to weak 1-bounded concurrent general composition security.*

Proof. The proof is analogous with the proof of theorem 6 presented in [25]. □

As shown in section 4.5.1, the next security result is essential for establishing the relation between the existing game-theoretic notion of strong universal implementation [55] and our notion of game universal implementation. As a preamble, we first give the intuition for weak precise secure computation. While the traditional notion of secure computation [44] requires only the worst case running time complexity of the ideal world simulator to match the running time of the real world adversary, weak precise secure computation [81] requires the complexity of the simulator to match the complexity of the real world adversary for each arbitrary distinguisher and input.

Definition 39 (Weak Precise Secure Computation)**.** *Let π be a protocol, $\mathcal{F}$ an ideal functionality and let $\mathcal{C}$ be the function that given a security parameter k, a polynomially bounded party Q and the view v of Q in the protocol π, it computes the time complexity of Q running with k and v. We say that π is a weak precise secure computation of $\mathcal{F}$ if there exists a polynomial p such that for every real world adversary $\mathcal{A}$, for every distinguisher $\mathcal{D}$ and for every input z, there exists an ideal world simulator $\mathcal{S}$, with $\mathcal{C}(k, \mathcal{S}, v) \leq p(k, \mathcal{C}(k, \mathcal{A}, \mathcal{S}(v)))$ such that :*

$$\{IDEAL(k, z, \mathcal{S}, \mathcal{F})\}_{k\in\mathbb{N}} \stackrel{D}{\equiv} \{REAL(k, z, A, \overrightarrow{M}\}_{k\in\mathbb{N}}.$$

Lemma 40 (Weak Precise Secure Computation Does Not Imply Weak Stand-alone Security)**.** *If Blum integer time-lock puzzles exist, then there exists a protocol π which is secure with respect to weak precise secure computation, but is not secure with respect to weak stand-alone security.*

Proof. The proof follows the general lines of the constructions that we have used for our main separation result in lemma 31, however, the details are much more straight forward.

Let π be such that on an input pair (k, t), where k is the security parameter, it truncates t to the first k bits obtaining t' and it generates a time-lock puzzle using $Gen(k, t') = (q', a')$. Then it sends q' to A and regardless of the reply received from the adversary, it outputs 1.

In the ideal world, on an input pair (k, t), the ideal functionality $\mathcal{F}$ behaves exactly like π with the only exception that it outputs 1 if and only if it receives from the adversary it interacts with, i.e., from $\mathcal{S}$, the correct solution v' from the puzzle q'. Otherwise $\mathcal{F}$ outputs 0.

Given π and $\mathcal{F}$ as defined above, first we show that π is not as secure as $\mathcal{F}$ with respect to weak stand-alone security. Assume by contradiction that weak stand-alone security property holds. Since in the real world π always outputs 1, in the ideal world the ideal functionality should output 1 with overwhelming probability. This in turn means that $\mathcal{S}$ should produce the correct solution for the puzzle sent by $\mathcal{F}$ with overwhelming probability. However, this should be the case independent of the input t'. For a fixed polynomially bounded simulator $\mathcal{S}$, there is a polynomially bounded hardness $t_{\mathcal{S}}$ for the puzzles that is can solve. However, by the definition of the time-lock puzzles, if the input $t' > t_{\mathcal{S}}$ then the simulator fails to reply correctly to the challenge sent by $\mathcal{F}$ with overwhelming probability. In conclusion, for a given simulator there is always an input

that the real and the ideal world are distinguishable with non-negligible probability, thus our assumption is false.

Second, we prove that π is as secure as $\mathcal{F}$ with respect to weak precise secure computation. In order to show this claim we make the following observation: From relation (4.4) in the definition of time-lock puzzles, we deduce that for every integer d there exists an integer p_d and a polynomial time solver C_d with run time at most p_d when solving puzzles of hardness k^d. By induction, it is easy to see that there is a polynomial *poly* such that for every d there is a PPITM solver C'_d such that the run time of C'_d is at most $poly(d)$ when solving puzzles of hardness k^d. This polynomial *poly* we can use as polynomial p in the definition of weak precise secure computation. Given an adversary $\mathcal{A}$, a distinguisher $\mathcal{D}$ and an input t, it is easy to see that, for each parameter k, if we take simulator $\mathcal{S}_k$ such that it has hardness at least t, then the real and the ideal world will be indistinguishable for $\mathcal{D}$. This concludes our proof. □

4.5.1 Relation Between 1-bit Specialized Simulator UC and Game Universal Implementation

In the following we prove an equivalence result between our notion of game universal implementation and our definition of weak stand-alone security. A similar result exists [55] and it shows the equivalence between with strong universal implementation [55] and weak precise secure computation. In order to prove this result, the authors consider a refined version for computational games, where the utility of the players may have strong correlations with the complexity of the computation they perform (e.g., time complexity, memory complexity, communication complexity or complexity of operations like reading inputs or copying messages). In this thesis, we discard the cost of computation and we are able to prove:

Theorem 41 (Game Universal Implementation and Weak Stand-alone Security)**.** *Let comm be a S communication mediator represented by the cryptographic notion of ideally secure channels. Let f be an m-ary function with the property that outputs the empty string to a party if and only if it had received the empty string from that party. Let $\mathcal{F}$ be a mediator that computes*[22] *f and let $\overrightarrow{M}$ be an abort-preserving computation of f.*[23] *Then $\overrightarrow{M}$ is a weak stand-alone secure computation of f with respect to statistical security*[24] *if and only if $(\overrightarrow{M}, comm)$ is a game universal implementation of $\mathcal{F}$ with respect to Games,*

[22]The ideal machine profile $\overrightarrow{\Lambda^{\mathcal{F}}}$ *computes* f if for all $n \in \mathbb{N}$, all inputs $\overrightarrow{x} \in (\{0,1\}^n)^m$, the output vector of the players after an execution of $\overrightarrow{\Lambda^{\mathcal{F}}}$ on input $\overrightarrow{x}$ is identically distributed to $f(\overrightarrow{x})$.

[23]$\overrightarrow{M}$ is an abort-preserving computation of f if for all $n \in \mathbb{N}$ and for all inputs $\bar{x} \in (\{0,1\}^n)^m$, the output vector of the players after an execution of $(\perp, \overrightarrow{M_{-Z}})$ on input $\bar{x}$ is identically distributed to $f(\lambda, \overrightarrow{x_{-Z}})$, where Z is a subset of all parties and λ is the empty string.

[24]We call $\overrightarrow{M}$ a weak stand-alone secure computation of f if the following two properties are fulfilled:

- For all $n \in \mathbb{N}$, all inputs $\overrightarrow{x} \in (\{0,1\}^n)^m$, the output vector of the players after an execution of $\overrightarrow{M}$ on input $\overrightarrow{x}$ is distributed statistically close to $f(\overrightarrow{x})$;
- For every adversary A and for every distinguisher D, there exists a simulator $\mathcal{S}$ such that for every

where Games is the class of games for which the utility functions of the players depend only on players types and on the output values and each of the player's P_i utility is smaller than a polynomial.

Proof. In this proof, we assume without loss of generality, that the ideal functionality $\mathcal{F}$ outputs to each of the parties in the ideal world the output of the computation for each of their input together with their input value. More formally, if party i sends $\mathcal{F}$ as input the value x_i, it receives from $\mathcal{F}$ as output $f_i(x_1, \ldots, x_m); x_i$, where by ";" we denote the concatenation of strings.

First we prove that if $\overrightarrow{M}$ is a weak secure computation of f with statistical security, then $(\overrightarrow{M}, comm)$ is a strong *Games* universal implementation of $\mathcal{F}$. Since both $\overrightarrow{M}$ and $\overrightarrow{\Lambda^{\mathcal{F}}}$ compute f, according to the respective definitions, it means they have statistically close output distributions. So the second property contained in the definition of game universal implementation has been proven. Next we prove that $\forall i \in \{1, \ldots, n\}$, there exists a negligible function ϵ_i and an integer k_i such that for all $k \geq k_i$:

$$U_i(k, \overrightarrow{M}) = U_i(k, \overrightarrow{\Lambda^{\mathcal{F}}}) + \epsilon_i(k). \tag{4.11}$$

Indeed, let $\overrightarrow{t}$ be the vector of inputs and $\overrightarrow{o}$ be the vector of outputs. Then we have:

$$\begin{aligned}
&|U_i(k, \overrightarrow{M}) - U_i(k, \overrightarrow{\Lambda^{\mathcal{F}}})| = \\
&= |\sum_{\overrightarrow{t}, \overrightarrow{o}} [Pr(REAL(k, \overrightarrow{M}) = \overrightarrow{o}) - Pr(IDEAL(k, \overrightarrow{\Lambda^{\mathcal{F}}}) = \overrightarrow{o})] \cdot u_i(k, \overrightarrow{t}, \overrightarrow{o})| \leq \\
&\leq [\sum_{\overrightarrow{t}, \overrightarrow{o}} |Pr(REAL(k, \overrightarrow{M}) = \overrightarrow{o}) - Pr(IDEAL(k, \overrightarrow{\Lambda^{\mathcal{F}}}) = \overrightarrow{o})|] \cdot p_i(k) \leq \\
&\leq \epsilon(k) \cdot p_i(k) = \\
&= \epsilon_i(k).
\end{aligned}$$

In the inequalities above, for readability we made the following notation: we denote by $REAL(k, \overrightarrow{M}$ the real world execution $REAL(k, z, A, \overrightarrow{M}\}_{k \in \mathbb{N}}$ for the stand-alone security, and we denote by $IDEAL(k, \overrightarrow{\Lambda^{\mathcal{F}}}$ the ideal world execution $IDEAL(k, z, \mathcal{S}, \mathcal{F})\}_{k \in \mathbb{N}}$ for the stand-alone execution.

Moreover, in order to conclude the inequalities above, we have used the following facts: The output distributions in the real and in the ideal world are finite and statistically close

input z, the following relation is fulfilled :

$$\{IDEAL(k, z, \mathcal{S}, \mathcal{F})\}_{k \in \mathbb{N}} \stackrel{D}{\equiv} \{REAL(k, z, A, \overrightarrow{M}\}_{k \in \mathbb{N}}.$$

In the second property, the indistinguishability relation can be further detailed with respect to perfect, statistical or computational security. If for example, in the second property the ensembles for the real and the ideal world are statistically indistinguishable, then we call the overall property as weak secure stand-alone computation with respect to statistical security.

and the utility function u_i is bounded above by a polynomial p_i. Hence, equation (4.11) holds.

Also, in an analogous manner, the following equation

$$U_Z(k, \overrightarrow{M}) = U_Z(k, \overrightarrow{\Lambda^{\mathcal{F}}}) + \epsilon_Z(k).$$

trivially holds, where Z can be any subset of players.

We are now ready to show that $\overrightarrow{M}$ fulfills the remaining two properties in the definition of game universal implementation. If $\overrightarrow{\Lambda^{\mathcal{F}}}$ is a computational Nash equilibrium in the mediated game $(G, \mathcal{F})$, assume by contradiction that $\overrightarrow{M}$ is not a Nash equilibrium in $(\overrightarrow{M}, comm)$. Then there exists a player i, a deviating strategy A_i, a polynomial p_i such that for every k_0 there exists $k \geq k_0$ with the property:

$$U_i(k, A_i, \overrightarrow{M_{-i}}) > U_i(k, \overrightarrow{M}) + \frac{1}{p_i(k)}. \tag{4.12}$$

Due to equation (4.11) and also due to the hypothesis that $\overrightarrow{\Lambda^{\mathcal{F}}}$ is a computational Nash equilibrium (this is where the following negligible function ϵ comes from) we additionally have:

$$\begin{aligned} U_i(k, \overrightarrow{M}) + \frac{1}{p_i(k)} + \epsilon(k) &= U_i(k, \overrightarrow{\Lambda^{\mathcal{F}}}) + \frac{1}{p_i(k)} + \epsilon_i(k) + \epsilon(k) \\ &\geq U_i(k, \mathcal{S}_i, \overrightarrow{\Lambda^{\mathcal{F}}_{-i}}) + \frac{1}{p_i(k)}, \end{aligned} \tag{4.13}$$

for every simulator $\mathcal{S}_i$. Thus we obtain there exists a polynomial p_i' such that $U_i(k, A_i, \overrightarrow{M_{-i}}) > U_i(k, \mathcal{S}_i, \overrightarrow{\Lambda^{\mathcal{F}}_{-i}}) + \frac{1}{p_i'(k)}$, for every $\mathcal{S}_i$. This is equivalent to the fact that the output distributions of $REAL(k, A_i, \overrightarrow{M_{-i}})$ and $IDEAL(k, \mathcal{S}_i, \mathcal{F})$ are not statistically close. Indeed, this can be shown in an analogous way as relation (4.11). So, for every $\mathcal{S}_i$, there exists a distinguisher $\mathcal{D}_i$ such that

$$\{IDEAL(k, \mathcal{S}_i, \mathcal{F})\}_{k\in\mathbb{N}} \overset{\mathcal{D}_i}{\not\equiv} \{REAL(k, A_i, \overrightarrow{M_{-i}}\}_{k\in\mathbb{N}}, \tag{4.14}$$

i.e., $\mathcal{D}_i$ (that can be also computationally unbounded) has non-negligible probability to distinguish between the ensembles above.

Let *Sim* be the set of all simulators $\mathcal{S}_i$ and let *Dist* be the set of all distinguishers $\mathcal{D}_i$ as described above. Let $\mathcal{D}$ be the distinguisher that runs every distinguisher in the set *Dist* and outputs 1 if and only if at least one of the distinguishers in the set *Dist* outputs 1. Such a $\mathcal{D}$ may be computationally unbounded, but is a viable distinguisher in relation with our definition of weak stand-alone computation with statistical security.

By the definition of weak security, for adversary A_i and for distinguisher $\mathcal{D}$, there exists a simulator $\mathcal{S}$ such that the following ensembles are statistically close distributed:

$$\{IDEAL(k, \mathcal{S}, \mathcal{F})\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{REAL(k, A_i, \overrightarrow{M_{-i}}\}_{k\in\mathbb{N}}$$

But since $\mathcal{D}$ also runs the distinguisher that can tell apart between the world with $\mathcal{S}$ and the world with A_i, the above relation is a contradiction with the definition of $\mathcal{D}$. Thus if $\overrightarrow{\Lambda^{\mathcal{F}}}$ is a computational Nash equilibrium, then so is $\overrightarrow{M}$. It can be shown in an analogous way, mainly by substituting the deviating player i with any set Z of deviating players, that if $\overrightarrow{\Lambda^{\mathcal{F}}}$ is immune to the coalition in Z, then $\overrightarrow{M}$ is also immune to the coalition in Z.

Finally, we look at the preservation of $\perp_i$ as a best response for a party i in $\overrightarrow{M}$. One one hand, As $\overrightarrow{M}$ is abort preserving and f has the empty string property, we have $U_i(\perp_i, \overrightarrow{M}_{-i}) = U_i(\perp_i, \overrightarrow{\Lambda^{\mathcal{F}}_{-i}})$. On the other hand, if playing $\perp_i$ is the best response to $\overrightarrow{\Lambda^{\mathcal{F}}_{-i}}$, (i.e., $\perp_i$ gives the highest utility for player i in the world with $\overrightarrow{\Lambda^{\mathcal{F}}_{-i}}$ up to a negligible value), due to the weak stand-alone secure computation security property and by using the same technique based on distinguishers' properties as in the preservation of computational Nash equilibrium above, the highest utility for party i in the real world is $U_i(\perp_i, \overrightarrow{M}_{-i})$ up to a negligible value. This concludes the implication that if $\overrightarrow{M}$ is a weak stand-alone secure computation of f with statistical security, then $(\overrightarrow{M}, comm)$ is a strong *Games* universal implementation of $\mathcal{F}$.

For the implication in the opposite direction, we follow the case-separation idea of the proof for theorem 4.2 (Information Theoretic Case) in [54] and we specify below the details.

Assume by contradiction that $\overrightarrow{M}$ is not a weak secure computation of f with statistical security. Thus, there exists a set Z of corrupted parties, there exists an adversary A_Z corrupting the parties in Z and a distinguisher D (possibly unbounded) such that for every simulator $\mathcal{S}$ there exists a polynomial $p_{\mathcal{S},Z}$ such that for every integer k_0 there exists $k \geq k_0$:

$$\begin{aligned} &Pr(D(k, REAL(k, A_Z, \overrightarrow{M}_{-Z})) = 1) - \\ &\qquad - Pr(D(k, IDEAL(k, \mathcal{S}, \mathcal{F})) = 1) > \frac{1}{p_{\mathcal{S},Z}(k)} \end{aligned} \tag{4.15}$$

As in [54], we distinguish between two cases:

Case 1: $A_Z = \perp_Z$

The proof idea in this case is to design a game in the class *Games*, with utilities depending on $\mathcal{D}$ such that $\overrightarrow{\Lambda^{\mathcal{F}}}$ is a computational Nash equilibrium (with immunity with respect to coalitions). By hypothesis, this implies that $\overrightarrow{M}$ is a computational Nash equilibrium (with immunity with respect to coalitions). However, for the constructed game, we obtain that $\perp_Z$ is the best response to $\overrightarrow{M_{-Z}}$, which represents a contradiction.

Let $d = Pr(D(k, IDEAL(k, \perp_Z, \mathcal{F})) = 1)$. We denote by $\overrightarrow{t}$ the inputs of the parties, which in game-theoretic terms correspond to the secret types of the players; and we denote by $\overrightarrow{o}$ the outputs of the parties, which in game-theoretic terms correspond to the actions taken by the players. In the following, by o_Z and by λ_Z respectively, we denote the output for parties in Z and the empty string corresponding to the output of the parties in Z.

Next, we define a game G such that for any subset of players $Z' \neq Z$, we have $u_{Z'} = 0$ and for the set Z we have:

$$u_Z(k, \overrightarrow{t}, \overrightarrow{o}) = \begin{cases} Pr(D(k, \overrightarrow{t}, \overrightarrow{o}) = 1) & \text{if } \overrightarrow{o_Z} = \lambda_Z \\ \\ d & \text{otherwise} \end{cases}$$

We show for the game G the strategy $\overrightarrow{\Lambda^{\mathcal{F}}}$ is a computational Nash equilibrium in the ideal world. Indeed, for any subset of players $Z' \neq Z$, we have that $U_{Z'}(k, \overrightarrow{\Lambda^{\mathcal{F}}}) = 0 = U_{Z'}(k, \mathcal{S}_{Z'}, \overrightarrow{\Lambda^{\mathcal{F}}_{-Z'}})$, for any simulator $\mathcal{S}$. For the set Z, on one hand we have $U_Z(k, \overrightarrow{\Lambda^{\mathcal{F}}}) = d$, as following the strategy $\overrightarrow{\Lambda^{\mathcal{F}}}$ does result in an "empty" output for the parties in Z only with negligible probability (i.e., if and only if the inputs to all the parties in Z are also the empty string). On the other hand, we have that:

$$U_Z(k, \mathcal{S}_Z, \overrightarrow{\Lambda^{\mathcal{F}}_{-Z}}) = \begin{cases} Pr(D(k, IDEAL(k, \perp_Z, \mathcal{F})) = 1) & \text{if } \mathcal{S}_Z = \perp_Z \\ \\ d + \epsilon_{\mathcal{S},Z}(k) & \text{otherwise,} \end{cases}$$

where for every $\mathcal{S}$, Z, $\epsilon_{\mathcal{S},Z}$ is a negligible function.

Hence $U_Z(k, \overrightarrow{\Lambda^{\mathcal{F}}}) + \epsilon_{\mathcal{S},Z}(k) \geq U_Z(k, \mathcal{S}_Z, \overrightarrow{\Lambda^{\mathcal{F}}_{-Z}})$, for every $\mathcal{S}_Z$. To summarize, $\overrightarrow{\Lambda^{\mathcal{F}}}$ is a Nash equilibrium with immunity with respect to coalitions. Adding the hypothesis of $(\overrightarrow{M}, comm)$ being a game universal implementation of $\mathcal{F}$, we obtain that $\overrightarrow{M}$ is a Nash equilibrium with immunity with respect to coalitions. But $\overrightarrow{M}$ and $\overrightarrow{\Lambda^{\mathcal{F}}}$ have statistically close output distributions, so similar to (4.11) we conclude $U_Z(k, \overrightarrow{M}) = d + \epsilon_Z(k)$. However, $U_Z(k, \perp_Z, \overrightarrow{M_{-Z}}) = Pr(D(k, REAL(k, \perp_Z, \overrightarrow{M_{-Z}})) = 1)$. By assumption (4.15), $Pr(D(k, REAL(k, A_Z, \overrightarrow{M_{-Z}})) = 1) > Pr(D(k, IDEAL(k, \mathcal{S}, \mathcal{F})) = 1) + \frac{1}{p_{\mathcal{S},Z}(k)}$, for every simulator $\mathcal{S}$. Thus, $U_Z(k, \perp_Z, \overrightarrow{M_{-Z}}) > d + \frac{1}{p_{\mathcal{S},Z}(k)} - \epsilon(k) = d + \frac{1}{p'_{\mathcal{S},Z}(k)}$.

As this contradicts the equilibrium property of $\overrightarrow{M}$, we conclude the first case.

Case 2: $A_Z \neq \perp_Z$

Without loss of generality, we assume that A_Z lets one of the players in Z output the entire view of the adversary A_Z. Indeed, we can construct A'_Z from A_Z such that besides the output for each of the parties in Z, the first player in Z also outputs $v' = A_Z(v)$. If we define the distinguisher D' such that

$$D'(k, REAL(k, A_Z(v), \overrightarrow{M}_{-Z}); v') = D(k, REAL(k, A_Z(v), \overrightarrow{M}_{-Z}))$$

and

$$D'(k, IDEAL(k, \mathcal{S}(v),)\overrightarrow{\Lambda^{\mathcal{F}}_{-Z}}; v') = D'(k, IDEAL(k, \mathcal{S}(v),)\overrightarrow{\Lambda^{\mathcal{F}}_{-Z}}),$$

then the property (4.15) fulfilled by D is also fulfilled by D'. So we can assume A_Z lets one of the players in Z output the entire view of A_Z.

Let $d = \underset{\mathcal{S}_Z}{sup}\ Pr(D(k, IDEAL(k, \mathcal{S}_Z, \mathcal{F})) = 1)$. We construct a game H in the following way. For any subset of players $Z' \neq Z$, the utility corresponding to the coalition Z' is 0, independent of the parties inputs and outputs. Then we define:

$$u_Z(k, \overrightarrow{t}, \overrightarrow{o}) = \begin{cases} d & \text{if } \overrightarrow{o_Z} = \lambda_Z \\ Pr(D(k, \overrightarrow{t}, \overrightarrow{o}, v) = 1) & \text{if } \exists\ o_{i_Z} = o'_{i_Z}; v \text{ and } \overrightarrow{o'_Z} \neq \lambda_Z \\ 0 & \textit{otherwise} \end{cases}$$

where for every $j_Z \neq i_Z$, $o'_{j_Z} = o_{j_Z}$.

We prove that for the game H defined above, $\perp_Z$ is the best response to $\overrightarrow{\Lambda^{\mathcal{F}}_{-Z}}$. Assume by contradiction this does not hold. Let the simulator $\mathcal{S}_Z^{best}$ be such that the strategy it implements for the parties in Z is the best response to $\overrightarrow{\Lambda^{\mathcal{F}}_{-Z}}$. This implies $\mathcal{S}_Z^{best} \neq \perp_Z$ and $U_Z(k, \mathcal{S}_Z^{best}, \overrightarrow{\Lambda^{\mathcal{F}}_{-Z}}) > U_Z(k, \perp_Z, \overrightarrow{\Lambda^{\mathcal{F}}_{-Z}}) + \frac{1}{p_Z(k)} = d + \frac{1}{p_Z(k)}$. From the last relation we can conclude that $IDEAL(k, \mathcal{S}_Z^{best}, \overrightarrow{\Lambda^{\mathcal{F}}_{-Z}})$ and $IDEAL(k, \perp_Z, \overrightarrow{\Lambda^{\mathcal{F}}_{-Z}})$ are not statistically close distributions.

Hence, the expected utility $U_Z(k, \mathcal{S}_Z^{best}, \overrightarrow{\Lambda^{\mathcal{F}}_{-Z}})$ can be computed using the second or the third branch of the definition of the utility function u_Z. Thus,

$$U_Z(k, \mathcal{S}_Z^{best}, \overrightarrow{\Lambda^{\mathcal{F}}_{-Z}}) \leq Pr(D(k, IDEAL(k, \mathcal{S}_Z^{best}, \mathcal{F})) = 1).$$

So $d < Pr(D(k, IDEAL(\mathcal{S}_Z^{best}, \mathcal{F})) = 1)$, which is a contradiction with the definition of d, so $\perp_Z$ is the best response to $\overrightarrow{\Lambda^{\mathcal{F}}_{-Z}}$.

By hypothesis of the current implication, we conclude $\perp_Z$ is the best response to $\overrightarrow{M_{-Z}}$. However, we show that $U_Z(k, A_Z, \overrightarrow{M_{-Z}}) > U_Z(k, \perp_Z, \overrightarrow{M_{-Z}}) + \frac{1}{p(k)}$, which is an obvious contradiction.

Indeed, on one hand:

$$\begin{aligned} U_Z(k, A_Z, \overrightarrow{M_{-Z}}) &= Pr(D(k, REAL(k, A_Z, \overrightarrow{M_{-Z}}))) > \\ &> \underset{\mathcal{S}_Z}{sup}\ Pr(D(k, IDEAL(k, \mathcal{S}_Z, \mathcal{F})) = 1) + \frac{1}{p_{\mathcal{S},Z}(k)} \\ &= d + \frac{1}{p_{\mathcal{S},Z}(k)}. \end{aligned}$$

On the other hand, $U_Z(k, \perp_Z, \overrightarrow{M_{-Z}}) = d$ due to the definition of the utility function u_Z, so the contradiction is obvious.

Our proof needs to clarify only one last point. What happens in the case that Z is the empty set, or to put it equivalently, A_Z is the empty adversary $\perp$. Then the assumption in equation (4.15) becomes

$$Pr(D(k, REAL(k, \perp, \overrightarrow{M})) = 1) - Pr(D(k, IDEAL(k, \mathcal{S}, \mathcal{F})) = 1) > \frac{1}{p_S(k)},$$

for every simulator $\mathcal{S}$. But this directly contradicts the fact that $\overrightarrow{M}$ and $\overrightarrow{\Lambda^{\mathcal{F}}}$ have statistically close output distributions.

Hence theorem 41 has been proven. □

So we have shown that by restricting the class of games to those for which the computation cost for parties during protocol execution is free, our variant of universal implementation becomes equivalent to more standard notions of security (i.e., where the simulator depends only on the distinguisher and not anymore on both the distinguisher and input). Finding such equivalences was stated as an open question in [55] and to the best of our knowledge, our work makes the first step towards answering it.

One may ask if our new notion of game universal implementation is a subcase of the existing notion of strong universal implementation [55]. Using Lemma 40, Theorem 41 and the equivalence result between strong universal implementation and weak precise secure computation [55], we obtain:

Corollary 42 (Non-Equivalence of Universal Implementation Variants)**.** *The notion of strong universal implementation does not imply the notion of game universal implementation.*

4.6 Conclusion

In this chapter we have established an equivalence result between a security notion (i.e., weak stand-alone security) and a game-theoretic notion (i.e., game universal implementation). In the process of deducing this result, we had a closer look at the implication relations among a wider set of security notions, including variants of the UC security definition. Along this path, an important result was the proof that two variants of the UC security definition where the order of quantifiers is reversed, namely 1-bit specialized simulator UC security and specialized simulator UC security, are not equivalent. This comes in contrast with the well known result that UC security and 1-bit UC security are equivalent [24] and solves the open question raised by Lindell [71] about the implication relation between the two above mentioned UC variants.

Chapter 5

Concluding Remarks

Rational cryptography has recently developed as a new field of research by combining notions and techniques from cryptography and game theory. In contrast to the traditional view of cryptography where protocol participants are considered either honest or arbitrarily malicious, rational cryptography models participants as rational players that try to maximize their benefit and thus deviate from the protocol only if in this way they obtain a benefit. In the context of game theory, a protocol that gives rational participants incentives to follow constitutes an equilibrium. Rational cryptography is centered around such (computational) equilibria concepts and uses cryptographic notions and techniques as tools for building secure protocols with respect to rational participants.

An initial research focus for rational cryptography has been on defining computational models for polynomially bounded rational participants (also called rational players) thus complementing the traditional cryptographic approach that considers participants to be either honest or arbitrarily malicious. Later, the research focus has shifted towards designing protocols that rational participants have no incentive to deviate from, i.e., the protocols should fulfill some game-theoretic notion of equilibrium. So far, most of these rational protocols fall into the category of rational secret sharing, rational secure multiparty computation and rational file sharing. File sharing is related to the recently emerging peer-to-peer paradigm which has seen the development of decentralized applications among loosely connected nodes that share data and services. Such applications (e.g., BitTorrent) became increasingly popular among users due to their efficiency, scalability, failure resistance and their adaptability to dynamically changing situations. However, most of such applications do not provide security against users that try to bypass the design choices of the underlying protocols in order to obtain higher advantages from the system (e.g., better download speed, downloading without reciprocating).

Designing rational cryptographic protocols

Our first goal in this thesis was to design, formally prove and implement RatFish [8], a new rational file sharing protocol. We build upon some concepts and design choices of the popular file sharing application BitTorrent but we resolve its well known weaknesses with

respect to users that try to deviate from the protocol. This we ensure by using rational exchanges of pieces among the users interested in downloading files and by having the tracker participate in the coordination of the downloads. However, our approach does not aim at incorporating countermeasures against known attacks, but we provide a rigorous game-theoretic model for the users and we formally prove the security of the system with respect to these well-defined rational users, i.e., we show RatFish is a computational Nash equilibrium. We have also implemented an actual RatFish tracker and we have shown, using experimental simulations, that possible overhead incurred by its secure design does not decrease its efficiency and scalability.

In spite of its provable rational security features and its tracker scalability, there are still challenges that lie ahead for RatFish and rational file sharing in general. Future directions to explore and tackle vary from pure theoretical approaches to actual implementation and deployment of the system into an experimental testbed.

A central question for future work on rational file sharing and also on rational cryptography is whether computational Nash equilibrium is a strong enough notion for real-world applications and threat models. For instance, an equilibrium that is robust against user coalitions might be more desirable. RatFish already provides suitable hooks for potential mitigation techniques against coalitions, e.g., by ensuring that players entering small coalitions can only increase their utilities by a small amount so that entering a coalition would be irrational as well in most cases. Still, this problem needs to be first investigated from a theoretical point of view. A well defined formal model has to be given for the possible behavior of coalition participants. Then new features have to be designed to deter RatFish users from entering coalitions. Such incentives could be, for example, the "fear" of being banned from the system for a long enough time if caught deviating. This, in turn, should be ensured by enhancing RatFish with the appropriate cryptographic techniques.

Automated verification of rational cryptographic protocols

While the application and deployment in real-life scenarios of rational protocols such as RatFish has the advantage of giving rational participants incentives to follow them, there are also technical drawbacks. Proofs of Nash equilibrium for rational cryptographic protocols are lengthy, difficult, and error-prone, since they combine the complexity of game theory proofs with the complexity of cryptographic proofs. Specifically, a deviating party may perform arbitrary cryptographic operations and combine in an unexpected manner the messages received from the other parties. Despite the impressive progress in the automated analysis of cryptographic protocols, security proofs for rational cryptographic protocols are at present done by hand. The main problem is that the existing cryptographic protocol verifiers deal with a different adversary model, the so called Dolev-Yao adversary [34], which models an omnipresent and omnipotent network-level adversary that can overhear, intercept, and synthesise any message and is only limited by the security constraints of the employed cryptographic schemes. In fact, the automated verification of rational cryptographic protocols, and in particular making utility functions accessible to existing cryptographic protocol verifiers, is recognized as

an open challenge [5].

In this context, we have initiated the line of research aimed at automated verification of rational cryptographic protocols [9]. We model these protocols in applied pi calculus [1], a well-established process calculus for the specification and analysis of cryptographic protocols. In order to make the security properties of these protocols amenable to automated verification, we formalize the concept of utility functions and Nash equilibrium as trace properties.

For exemplification of our approach, we have modeled the Ratfish rational file sharing protocol [8] in the applied pi calculus and analyzed it using ProVerif [19], a state-of-the-art cryptographic protocol verifier. Ratfish relies on a fairly sophisticated utility function based on the length of messages, which expresses the interest of rational parties in completing the download of their files while minimizing the amount of uploaded data. We show how to manually encode a Nash equilibrium property based on such a utility function into the ProVerif query language, which is based on the concept of correspondence assertions [107], a formalism originally developed to state properties of cryptographic authentication protocols. The analysis of the resulting query is automated and takes about a minute. To the best of our knowledge, this is the first attempt to leverage an automated cryptographic protocol verifier in the analysis of rational cryptographic protocols.

This work opens up a number of interesting research directions. First off, it would be interesting to extend existing cryptographic protocol verifiers and the underlying resolution algorithms in order to offer direct support for utility functions, as opposed to relying on manual encodings that are not always possible. Furthermore, refining the algorithms so to support sophisticated arithmetic relations seems to be crucial to express fine-grained utility functions, e.g., those underlying rational auction protocols.

The approach we have taken so far is to consider the communication network authenticated and secure. This can be smoothly adapted to deal with rational adversaries colluding with a standard Dolev-Yao attacker, by simply letting the communication channels be available to the adversary. It would also be interesting to consider a different adversarial model, in which the Dolev-Yao adversary does not necessarily collude with the rational parties, extending the security definitions and the verification algorithms accordingly.

Finally, the current framework for automated verification of rational cryptographic protocols can reason only about the Nash equilibrium property, which deals with unilateral deviations. It would be interesting to extend this framework to other properties, such as strong Nash equilibrium, which allows for arbitrary coalitions of rational players, and coalition-proof Nash equilibrium, in which players can freely discuss their strategies but cannot make binding commitments.

Equivalences between security and game-theoretic notions

The research detailed so far has highlighted that methods and notions from game theory can be used in cryptography and vice versa. This raises the question as to which extent notions from game theory and cryptography are related or which is the intrinsic connection

between the two fields. This is the motivation which triggered our work [29] on equivalence relations between security notions and game-theoretic notions. The first research result [55] regarding such equivalence relations has been centered around the "power of costly computation". The main result of this study [55] is an equivalence relation between a very weak security notion, namely weak precise secure computation and a newly defined game theoretic notion, which was called strong universal implementation. However, it was left as an open question how to obtain further equivalence relations, but for stronger security notions.

In this thesis, we discard the cost of computation and this allows us to go a step further: we define the notion of game universal implementation and we show it is equivalent to weak stand-alone security. Thus, we are able to answer positively the open question from [55] regarding the existence of game-theoretic definitions that are equivalent to cryptographic security notions for which the ideal world simulator does not depend on both the distinguisher and the input distribution. Additionally, we investigate the propagation of the weak stand-alone security notion through the existing security hierarchy, from stand-alone security to universal composability. The main achievement in this direction is a separation result between two variants of the UC security definition: 1-bit specialized simulator UC security and specialized simulator UC security. The separation result between the UC variants was stated as an open question [71] and it comes in contrast with the well known equivalence result between 1-bit UC security and UC security.

A future direction is to investigate the power of game theory in order to achieve security properties which are not possible otherwise in a pure cryptographic setting. More precisely, there are statements in cryptography for which it is known that under certain assumptions they do not hold. Such an impossibility statement is the secure and correct computation of an arbitrary function among a group of participants when more than half of them can freely deviate. A classical concrete example is the impossibility of achieving fair coin flipping with a dishonest majority [31]. Thus, as future research direction we consider defining the rational settings, i.e., classes of utilities for participants, for which cryptographic impossibility results such the one mentioned above become possible in the rational world. Moreover, a positive result with respect to rational coin toss would imply the existence of protocols achieving rational verifiable secret sharing and rational broadcast.

Bibliography

[1] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM, 2001.

[2] Ittai Abraham, Danny Dolev, Rica Gonen, and Joe Halpern. Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In *25th Annual ACM Symposium on Principles of Distributed Computing (PODC'06)*, pages 53–62. ACM, 2006.

[3] Amitanand S. Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. BAR fault tolerance for cooperative services. *Operating Systems Review*, 39(5):45–58, 2005.

[4] Almudena Alcaide. Phd thesis. Rational exchange protocols, 2008.

[5] Wihem Arsac, Giampaolo Bella, Xavier Chantry, and Luca Compagna. Validating security protocols under the general attacker. In *Foundations and Applications of Security Analysis*, chapter Validating Security Protocols under the General Attacker, pages 34–51. Springer, 2009.

[6] N. Asokan, Victor Shoup, and Michael Waidner. Asynchronous protocols for optimistic fair exchange. In *IEEE Symposium on Security and Privacy*, pages 86–99. IEEE Computer Society, 1998.

[7] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):593–610, 2000.

[8] Michael Backes, Oana Ciobotaru, and Anton Krohmer. RatFish: A file sharing protocol provably secure against rational users. In *15th European Symposium on Research in Computer Security (ESORICS'10)*, pages 607–625. Springer, 2010.

[9] Michael Backes, Oana Ciobotaru, and Matteo Maffei. Towards automated verification of rational cryptographic protocols, 2012. In Submission.

[10] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A general composition theorem for secure reactive systems. In *1st Theory of Cryptography Conference (TCC'04)*, pages 336–354. Springer, 2004.

[11] Feng Bao, Robert H. Deng, and Wenbo Mao. Efficient and practical fair exchange protocols with off-line ttp. In *IEEE Symposium on Security and Privacy*, pages 77–85. IEEE Computer Society, 1998.

[12] Donald Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.

[13] Donald Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2), 1991.

[14] Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority. In *9th Annual International Cryptology Conference (CRYPTO'89)*, pages 589–590. Springer, 1989.

[15] Michael Ben-Or, Oded Goldreich, Silvio Micali, and Ronald L. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.

[16] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th Annual ACM Symposium on Theory of Computing (STOC'88)*, pages 1–10. ACM, 1988.

[17] Ashwin R. Bharambe, Cormac Herley, and Venkata N. Padmanabhan. Analyzing and improving a BitTorrent network's performance mechanisms. In *The 25th IEEE Conference on Computer Communications (INFOCOM'06)*, pages 1–12. IEEE, 2006.

[18] Justin Bieber, Michael Kenney, Nick Torre, and Landon P. Cox. An empirical study of seeders in BitTorrent. Technical report, Duke University, 2006.

[19] Bruno Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 82–96. IEEE Computer Society, 2001.

[20] Dan Boneh and Moni Naor. Timed commitments. In *20th Annual International Cryptology Conference (CRYPTO'00)*, pages 236–254. Springer, 2000.

[21] Levente Buttyán and Jean-Pierre Hubaux. Rational exchange - a formal model based on game theory. In *2nd International Workshop of Electronic Commerce (WELCOM'01)*, pages 114–126. Springer, 2001.

[22] Levente Buttyán, Jean-Pierre Hubaux, and Srdjan Capkun. A formal model of rational exchange and its application to the analysis of Syverson's protocol. *Journal of Computer Security*, 12(3-4):551–587, 2004.

[23] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[24] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd IEEE symposium on Foundations of Computer Science (FOCS '01)*, pages 136–145. IEEE Computer Society, 2001.

[25] Ran Canetti and Marc Fischlin. Universally composable commitments. In *21st Annual International Cryptology Conference (CRYPTO'01)*, pages 19–40. Springer, 2001.

[26] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th Annual ACM Symposium on Theory of Computing (STOC'88)*, pages 11–19. ACM, 1988.

[27] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, David Parker, and Aistis Simaitis. Automatic verification of competitive stochastic systems. In *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference (TACAS'12)*, pages 315–330. Springer, 2012.

[28] Benny Chor and Eyal Kushilevitz. A zero-one law for boolean privacy. *SIAM Journal on Discrete Mathematics*, 4:36–47, 1991.

[29] Oana Ciobotaru. On the (non-) equivalence of UC security notions. In *6th International Conference on Provable Security (ProvSec2012)*, pages 104–124. Springer-Verlag, 2012.

[30] Allen Clement, Jeff Napper, Harry C. Li, Jean-Philippe Martin, Lorenzo Alvisi, and Michael Dahlin. Theory of bar games. In *26th Annual ACM Symposium on Principles of Distributed Computing (PODC'07)*, pages 358–359. ACM, 2007.

[31] Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *18th Annual ACM Symposium on Theory of Computing (STOC'86)*, pages 364–369. ACM, 1986.

[32] Bram Cohen. Incentives build robustness in BitTorrent. Technical report, bittorrent.org, 2003.

[33] Yevgeniy Dodis, Shai Halevi, and Tal Rabin. A cryptographic solution to a game theoretic problem. In *20th Annual International Cryptology Conference (CRYPTO'00)*, pages 112–130. Springer, 2000.

[34] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[35] Cynthia Dwork, Moni Naor, Omer Reingold, and Larry J. Stockmeyer. Magic functions. *Journal of the ACM*, 50(6):852–921, 2003.

[36] Uriel Feige. *Alternative Models For Zero Knowledge Interactive Proofs (PhD Thesis)*. PhD thesis, Weizmann Institute of Science, 1992.

[37] Joan Feigenbaum and Scott Shenker. Distributed algorithmic mechanism design: recent results and future directions. In *6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M'02)*, pages 1–13. ACM, 2002.

[38] Matthias Fitzi, Juan A. Garay, Ueli M. Maurer, and Rafail Ostrovsky. Minimal complete primitives for secure multi-party computation. *Journal of Cryptology*, 18(1):37–61, 2005.

[39] Georg Fuchsbauer, Johathan Katz, and David Naccache. Efficient rational secret sharing in standard communication networks. In *7th Theory of Cryptography Conference (TCC'10)*, pages 419–436. Springer, 2010.

[40] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.

[41] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *21st Anual ACM Symposium on Theory of Computing (STOC'89)*, pages 25–32. ACM, 1989.

[42] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th Annual Symposium on Foundations of Computer Science (FOCS'86)*, pages 174–187. IEEE Computer Society, 1986.

[43] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *19th Annual ACM Symposium on Theory of Computing (STOC'87)*, pages 218–229. ACM, 1987.

[44] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *19th Annual ACM Symposium on Theory of Computing (STOC'87)*, pages 218–229. ACM, 1987.

[45] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, 1994.

[46] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In *10th Annual International Cryptology Conference (CRYPTO'90)*, pages 77–93. Springer, 1990.

[47] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[48] Dov Gordon and Jonathan Katz. Rational secret sharing, revisited. In *5th Conference on Security and Cryptography for Networks (SCN'06)*, pages 229–241. Springer, 2006.

[49] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete fairness in secure two-party computation. *ACM Journal*, 58(6):24, 2011.

[50] S. Dov Gordon, Yuval Ishai, Tal Moran, Rafail Ostrovsky, and Amit Sahai. On complete primitives for fairness. In *7th Theory of Cryptography Conference (TCC'10)*, pages 91–108. Springer, 2010.

[51] S. Dov Gordon and Jonathan Katz. Complete fairness in multi-party computation without an honest majority. In *6th Theory of Cryptography Conference (TCC'09)*, pages 19–35. Springer, 2009.

[52] S. Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. *Journal of Cryptology*, 25(1):14–40, 2012.

[53] Joseph Halpern and Vanessa Teague. Rational secret sharing and multiparty computation: extended abstract. In *36th Annual ACM Symposium on Theory of Computing (STOC'04)*, pages 623–632. ACM, 2004.

[54] Joseph Y. Halpern and Rafael Pass. A computational game-theoretic framework for cryptography. Unpublished manuscript, 2010.

[55] Joseph Y. Halpern and Rafael Pass. Game theory with costly computation: Formulation and application to protocol security. In *Innovations in Computer Science (ICS'10)*, pages 120–142. Tsinghua University Press, 2010.

[56] Martin Hirt and Ueli Maurer. Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In *6th Annual ACM Symposium on Principles of Distributed Computing (PODC '97)*, pages 25–34. ACM, 1997.

[57] Dennis Hofheinz and Dominique Unruh. Comparing two notions of simulatability. In *2nd Theory of Cryptography Conference (TCC'05)*, pages 89–103. Springer, 2005.

[58] Xiaowei Huang and Ron van der Meyden. Model checking games for a fair branching-time temporal epistemic logic. In *Australasian Conference on Artificial Intelligence*, pages 11–20. Springer, 2009.

[59] M. Izal, G. Uroy-Keller, E.W. Biersack, P. A. Felber, A. Al Hamra, and L. Garces-Erice. Dissecting BitTorrent: Five months in torrent's lifetime. In *Passive and Active Measurements (PAM'04)*, pages 1–11. Springer, 2004.

[60] Jonathan Katz. Bridging game theory and cryptography: Recent results and future directions. In *5th Theory of Cryptography Conference (TCC'08)*, pages 251–272. Springer, 2008.

[61] Idit Keidar, Roie Melamed, and Ariel Orda. Equicast: Scalable multicast with selfish users. *Computer Networks*, 53(13):2373–2386, 2009.

[62] Joe Kilian, Eyal Kushilevitz, Silvio Micali, and Rafail Ostrovsky. Reducibility and completeness in private computations. *SIAM Journal on Computing*, 29(4):1189–1208, 2000.

[63] Gillat Kol and Moni Naor. Cryptography and game theory: Designing protocols for exchanging information. In *5th Theory of Cryptography Conference (TCC'08)*, pages 320–339. Springer, 2008.

[64] Steve Kremer and Olivier Markowitch. Fair multi-party non-repudiation protocols. *International Journal of Information Security*, 1(4):223–235, 2003.

[65] Steve Kremer, Olivier Markowitch, and Jianying Zhou. An intensive survey of fair non-repudiation protocols. *Computer Communications*, 25(17):1606–1621, 2002.

[66] Steve Kremer and Jean-François Raskin. A game-based verification of non-repudiation and fair exchange protocols. *Journal of Computer Security*, 11(3):399–430, 2003.

[67] Martin Lange and Colin Stirling. Model checking games for branching time logics. *Journal of Logic and Computation*, 12(4):623–639, 2002.

[68] Dave Levin, Katrina LaCurts, Neil Spring, and Bobby Bhattacharjee. BitTorrent is an auction: analyzing and improving BitTorrent's incentives. *Computer Communications Review (CCR)*, 38(4):243–254, 2008.

[69] Harry C. Li, Allen Clement, Mirco Marchetti, Manos Kapritsos, Luke Robison, Lorenzo Alvisi, and Michael Dahlin. FlightPath: Obedience vs. choice in cooperative services. In *8th USENIX Symposium on Operating Systems Design and Implementation (USENIX OSDI'08)*, pages 355–368. USENIX Association, 2008.

[70] Harry C. Li, Allen Clement, Edmund L. Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, and Michael Dahlin. BAR gossip. In *7th Symposium on Operating Systems Design and Implementation (USENIX OSDI'06)*, pages 191–204. USENIX Association, 2006.

[71] Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *44th Symposium on Foundations of Computer Science (FOCS'03)*, pages 394–403. IEEE Computer Society, 2003.

[72] Nikitas Liogkas, Robert Nelson, Eddie Kohler, and Lixia Zhang. Exploiting BitTorrent for fun (not for profit). In *5th International Workshop on Peer-to-Peer Systems (IPTPS'06)*, 2006.

[73] Thomas Locher, Patrick Moor, Stefan Schmid, and Roger Wattenhofer. Free riding in BitTorrent is cheap. In *5th Workshop on Hot Topics in Networks (HotNets'06)*, pages 85–90. ACM, 2006.

[74] Anna Lysyanskaya and Nikos Triandopoulos. Rationality and adversarial behavior in multi-party computation. In *26th Annual International Cryptology Conference (CRYPTO'06)*, pages 180–197. Springer, 2006.

[75] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Non-interactive time-stamping and proofs of work in the random oracle model. *IACR Cryptology ePrint Archive*, 2011:553, 2011.

[76] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In *31st Annual Cryptology Conference (CRYPTO'11)*, pages 39–50. Springer, 2011.

[77] Federico Mari, Igor Melatti, Ivano Salvo, Enrico Tronci, Lorenzo Alvisi, Allen Clement, and Harry C. Li. Model checking Nash equilibria in MAD distributed systems. In *Formal Methods in Computer-Aided Design (FMCAD'08)*, pages 1–8. IEEE Computer Society, 2008.

[78] Federico Mari, Igor Melatti, Ivano Salvo, Enrico Tronci, Lorenzo Alvisi, Allen Clement, and Harry C. Li. Model checking coalition Nash equilibria in MAD distributed systems. In *Stabilization, Safety, and Security of Distributed Systems, 11th International Symposium (SSS'09)*, pages 531–546. Springer, 2009.

[79] Olivier Markowitch, Dieter Gollmann, and Steve Kremer. On fairness in exchange protocols. In *5th Conference on Information Security and Cryptology (ICISC'02)*, pages 451–464. Springer, 2002.

[80] Nimrod Megiddo and Avi Wigderson. On play by means of computing machines: preliminary version. In *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning about Knowledge (TARK '86)*, pages 259–274. Morgan Kaufmann Publishers Inc., 1986.

[81] Silvio Micali and Rafael Pass. Local zero knowledge. In *38th Anual ACM Symposium on Theory of Computing (STOC'06)*, pages 306–315. ACM, 2006.

[82] Silvio Micali and Phillip Rogaway. Secure computation (abstract). In *1st Annual International Cryptology Conference (CRYPTO'91)*, pages 392–404. Springer, 1991.

[83] Abraham Neyman. Bounded complexity justifies cooperation in the finitely repeated prisoners' dilemma. *Economics Letters*, 19(3):227–229, 1985.

[84] Henning Pagnia and Felix C. Gartner. Technical Report. On the impossibility of fair exchange without a trusted third party, 1999.

[85] Ryan S. Peterson and Emin Gün Sirer. Antfarm: efficient content distribution with managed swarms. In *6th USENIX Symposium on Networked Systems Design and Implementation (USENIX NSDI'09)*, pages 107–122. USENIX Association, 2009.

[86] Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Cryptographic security of reactive systems. *Electr. Notes Theor. Comput. Sci.*, 32, 2000.

[87] Birgit Pfitzmann and Michael Waidner. A general framework for formal notions of secure systems. http://www.semper.org/sirene/lit., 1994.

[88] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In *7th ACM conference on Computer and Communications Security (CCS'00)*, pages 245–254. ACM, 2000.

[89] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pages 184–, 2001.

[90] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in BitTorrent? In *4th Symposium on Networked Systems Design and Implementation (USENIX NSDI'07)*, pages 1–14. USENIX Association, 2007.

[91] Michael Piatek, Tomas Isdal, Arvind Krishnamurthy, and Thomas Anderson. One hop reputations for peer to peer file sharing workloads. In *5th Symposium on Networked Systems Design & Implementation (USENIX NSDI'08)*, pages 1–14. USENIX Association, 2008.

[92] J. A. Pouwelse, P. Garbacki, D.H.J. Epema, and H. J. Sips. The BitTorrent p2p file-sharing system: Measurements and analysis. In *4th International Workshop on Peer-to-Peer Systems (IPTPS'05)*, pages 205–216. Springer, 2005.

[93] Dongyu Qiu and R. Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'04)*, pages 367–378. ACM, 2004.

[94] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st Annual ACM Symposium on Theory of Computing (STOC'89)*, pages 73–85. ACM, 1989.

[95] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology, 1996.

[96] Dave Levin Rob, Rob Sherwood, and Bobby Bhattacharjee. Fair file swarming with FOX. In *5th International Workshop on Peer-to-Peer Systems (IPTPS'06)*, 2006.

[97] Ariel Rubinstein. Finite automata play the repeated prisoner's dilemma. *Journal of Economic Theory*, 20(1-3):83–96, 1986.

[98] Vitaly Shmatikov and John C. Mitchell. Analysis of a fair exchange protocol. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'00)*. The Internet Society, 2000.

[99] Vitaly Shmatikov and John C. Mitchell. Analysis of abuse-free contract signing. In *4th International Conference on Financial Cryptography (FC'00)*, pages 174–191. Springer, 2000.

[100] Vitaly Shmatikov and John C. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science*, 283(2):419–450, 2002.

[101] J. Shneidman, D. Parkes, and L. Massoulié. Faithfulness in internet algorithms. In *Workshop on Practice and Theory of Incentives and Game Theory in Networked Systems (PINS'04)*, pages 220–227. ACM, 2004.

[102] Michael Sirivianos, Xiaowei Yang, and Stanislaw Jarecki. Robust and efficient incentives for cooperative content distribution. *Transactions On Networking*, 17(6):1766–1779, 2009.

[103] Paul F. Syverson. Weakly secret bit commitment: Applications to lotteries and fair exchange. In *Computer Security Foundations Workshop (CSFW'98)*, pages 2–13. IEEE Computer Society, 1998.

[104] Tom Tedrick. Fair exchange of secrets. In *Advances in Cryptology (CRYPTO'84)*, pages 434–438. Springer, 1984.

[105] R.W. Thommes and M.J. Coates. BitTorrent fairness: Analysis and improvements. In *4th IEEE Workshop on the Internet, Telecommunications and Signal Processing (WITSP'05)*. IEEE, 2005.

[106] Amparo Urbano and Jose Vila. Computationally restricted unmediated talk under incomplete information. *Economic Theory*, 23(2):283–320, 2004.

[107] Thomas Y.C. Woo and Simon S. Lam. A semantic model for authentication protocols. In *IEEE Symposium on Security and Privacy*, pages 178–194. IEEE Computer Society, 1993.

[108] Andrew C. Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science (FOCS'82)*, pages 160–164. IEEE Computer Society, 1982.

[109] Andrew C. Yao. Theory and application of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science (FOCS '82)*, pages 80–91. IEEE Computer Society, 1982.

[110] Andrew C. Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science (FOCS'86)*, pages 162–167. IEEE Computer Society, 1986.

[111] Jianying Zhou and Dieter Gollmann. A fair non-repudiation protocol. In *IEEE Symposium on Security and Privacy*, pages 55–61. IEEE Computer Society, 1996.

Appendix A

Complete Listings

```
type index.
type Data.
type key.
type Length.
type haltmessage.
type participant.
type phase_id.

const phase_0: phase_id.
const phase_1: phase_id.

const dex_1: index.
const dex_2: index.
const dex_3: index.
const dex_4: index.

free dedicw: channel.

free c_L1L2: channel[private].
free c_L2L1: channel[private].

free c_L1L2_2: channel[private].
free c_L2L1_2: channel[private].

free c_L1s: channel[private].
free c_sL1: channel[private].
free c_sL1_2: channel[private].
free c_L2s: channel[private].
free c_sL2: channel[private].
```

```
free c_sL2_2: channel[private].

free c_L1t: channel[private].
free c_tL1: channel[private].
free c_L1t_2: channel[private].
free c_tL1_2: channel[private].
free c_L1t_3: channel[private].
free c_L1t_4: channel[private].

free c_L2t: channel[private].
free c_tL2: channel[private].
free c_L2t_2: channel[private].
free c_tL2_2: channel[private].
free c_L2t_3: channel[private].
free c_L2t_4: channel[private].

free c_ts: channel[private].
free c_ts_2: channel[private].
free c_st: channel[private].

free  filepiece_1: bitstring [private].
free  filepiece_2: bitstring [private].
free  filepiece_3: bitstring [private].
free  filepiece_4: bitstring [private].

const hashLength: Length.
const pieceLength: Length.
const keyLength: Length.
const otherLength: Length.

const continue: haltmessage.
const stop: haltmessage.
const message_sent: haltmessage.
const message_received: haltmessage.

const L1: participant.
const L2: participant.
const trac: participant.
const sed: participant.

fun s(Length): Length.
fun data(bitstring, Length): Data.
fun length(Data): Length.
```

```
fun pair(Data, Data): Data.
fun conc(bitstring, bitstring): bitstring.
fun hash(Data): Data.
fun senc(Data, key): Data.
fun mes(Data): bitstring.
fun sdec(Data, key): Data.
fun keyD(key): Data.

equation forall ke: key; length(keyD(ke)) = keyLength.
equation forall blm: Data; length(hash(blm)) = hashLength.
equation forall ll: Length, m: bitstring; length(data(m, ll))=ll.
equation forall mm: bitstring, ll: Length, ke: key;
                length(senc(data(mm, ll), ke)) = ll.
equation forall m: bitstring, l: Length; mes(data( m, l)) = m.
equation forall m: Data, k: key; sdec(senc(m,k), k) = m.


pred CorrectLength(Data).
clauses
    forall m: bitstring; CorrectLength(data(m, pieceLength));
    forall x: Data, ke: key;
                  CorrectLength(x) -> CorrectLength(senc( x, ke)).

event Receive_enc(participant, participant, Data, index, phase_id).
event Receive_key(participant, participant, key, phase_id).
event Send(participant, participant, bitstring, phase_id).
event Send_key(participant, participant, key, phase_id).
event Receive_piece(participant, participant, bitstring, phase_id).
event Send_freepiece(participant, participant, Data).

query  dex:index, x: Data;
(attacker((filepiece_1,filepiece_3)) phase 1) ==>
(event(Send_freepiece(sed, L2, data(filepiece_3, pieceLength)))
==> event(Receive_enc(L1, L2, x, dex, phase_1)) && CorrectLength(x)).

let leecher_exchange(l1: participant, l2: participant, c_l1t_2:channel,
c_tl1:channel, c_l1l2:channel, c_l2l1:channel, c_l1t_3:channel,
c_tl1_2:channel, c_l1l2_2:channel, c_l2l1_2:channel, c_l1t_4: channel,
somefilepiece_1: Data, in2: index, in4: index, phase_id: phase_id) =

  out(c_l1t_2, in2);
  in(c_tl1, (=continue, id_2: participant));
  new k_1: key;
```

```
  let e_1: Data = senc(somefilepiece_1, k_1) in
  out(c_l1l2, e_1);
  in(c_l2l1, e_2: Data);
  event Receive_enc(l1, l2, e_2, in2, phase_id);
  if CorrectLength(e_2) then
       out (c_l1t_3, message_sent);
       in (c_tl1_2, =message_received);
       out(c_l1l2_2, k_1);
       event Send_key(l1, l2, k_1, phase_id);
       in(c_l2l1_2, k_2: key);
       event Receive_key(l1, l2, k_2, phase_id);
       let somefilepiece_2: Data = sdec(e_2, k_2) in
       event Receive_piece(l1, l2, mes(somefilepiece_2), phase_id);
       out(c_l1t_4, in4);
       0
  else
       0.


let leecher_demand_reward(l1: participant, l2: participant,
                        c_sl1_2: channel, phase_id: phase_id) =

      in(c_sl1_2, somefilepiece_4: Data);
      event Receive_piece(l1, l2, mes(somefilepiece_4), phase_id);
      0.


let leecher(l1: participant, l2: participant, ppiece_1: Data,
                   dex_1: index, ppiece_3: Data, dex_3: index,
                   dex_2: index, dex_4: index) =

(phase 1; leecher_exchange(l1, l2, c_L1t_2, c_tL1, c_L1L2,
c_L2L1, c_L1t_3, c_tL1_2, c_L1L2_2, c_L2L1_2, c_L1t_4,
ppiece_1, dex_2, dex_4, phase_1))
|
(phase 1;  leecher_demand_reward(l1, l2, c_sL1_2, phase_1)).


let compromised(other_id: participant) =

out (dedicw, (other_id, c_L1L2, c_L1L2_2, c_L2L1, c_L2L1_2,
c_sL2, c_sL2_2, c_L2s, c_tL2, c_tL2_2, c_L2t, c_L2t_2, c_L2t_3,
c_L2t_4, filepiece_2, filepiece_4));
```

```
0.


let tracker(idd1: participant, iin1: index, iin3: index,
            idd2: participant,  iin2: index, iin4: index)=

(phase 1;
in(c_L1t_2, =iin2);
in(c_L2t_2, =iin1);
out(c_tL1, (continue, idd2));
out(c_tL2, (continue, idd1));
in(c_L1t_3, =message_sent);
in(c_L2t_3, =message_sent);
out(c_tL1_2, message_received);
out(c_tL2_2, message_received);
in(c_L1t_4, =iin4);
in(c_L2t_4, =iin3);
out(c_ts_2, (idd1, iin4, idd2, iin3));
0
).

let seeder(iid1: participant, iid2: participant, pp1: Data,
inn1: index, pp2: Data, inn2: index, pp3: Data, inn3: index,
pp4: Data, inn4: index)=

(phase 1;
in(c_ts_2, (=iid1, =inn4, =iid2, =inn3));
event Send_freepiece(sed, iid1, pp4);
out(c_sL1_2, pp4);
event Send_freepiece(sed, iid2, pp3);
out(c_sL2_2, pp3);
0).

process
 let piece_1: Data = data(filepiece_1, pieceLength) in
 let piece_2: Data = data(filepiece_2, pieceLength) in
 let piece_3: Data = data(filepiece_3, pieceLength) in
 let piece_4: Data = data(filepiece_4, pieceLength) in

(
 (leecher(L1, L2, piece_1, dex_1, piece_3, dex_3, dex_2, dex_4))
 |
 (compromised(L2))
```

```
 |
 (tracker(L1,  dex_1, dex_3, L2, dex_2, dex_4) )
 |
 (seeder(L1, L2, piece_1, dex_1, piece_2, dex_2, piece_3, dex_3,
  piece_4, dex_4))
)
```