View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by KITopen

Karlsruhe Institute of Technology

The Cryptographic Strength of Tamper-Proof Hardware

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Tobias Nilges

aus Siegen

Tag der mündlichen Prüfung:	16. Dezember 2015
Erster Gutachter:	Prof. Dr. Jörn Müller-Quade
Zweiter Gutachter:	Prof. Jesper Buus Nielsen, PhD

Acknowledgements

I want to deeply thank my advisor Jörn Müller-Quade for giving me the chance to pursue my PhD. He supported me all along the way and pointed me to interesting problems. We had many inspiring discussions, not limited to the topic of this thesis or cryptography. I also want to thank Jesper Buus Nielsen for taking the time and interest necessary to co-referee my thesis. And also for the After-TCC-Party in Warsaw.

The working atmosphere with my current and former colleagues was very enjoyable and I want to thank them all for the many fruitful discussions on- and off-topic, sometimes just lending an ear to let me get a better grasp of what I was trying to do.

Thanks to Masayuki Abe for giving me the opportunity to do an internship at NTT, providing me with a look outside of the box and giving me a new view on the topics I was working on. I learned a lot during those two months.

Last but not least, I want to express my heartfelt gratitude to Sabine, my friends and my family for their countless encouragements and their support over the years.

Abstract

Tamper-proof hardware has found its way into our everyday life in various forms, be it SIM cards, credit cards or passports. Usually, a cryptographic key is embedded in these hardware tokens that allows the execution of simple cryptographic operations, such as encryption or digital signing. The inherent security guarantees of tamper-proof hardware, however, allow more complex and diverse applications.

The Universal Composability (UC) framework provides very strong security guarantees, in particular for the case of protocol composition. To prove a protocol secure in this framework, a setup (like a common reference string or a public key infrastructure) usually has to be established by a trusted authority. In 2007, Katz showed that two mutually distrusting parties can instead use tamper-proof hardware as a setup assumption. In this scenario, one party creates and programs a tamper-proof hardware token. Katz showed that although the other party does not trust the token to operate according to the protocol, UC-secure two-party computation is possible. A trusted authority is thus no longer necessary. The security of the protocols is based on two physical assumptions: (1) The sender of the hardware token has no direct means of communication with the token (*isolation assumption*), and (2) the receiver of the hardware token cannot learn anything about the internal workings or secrets of the token apart from its input/output behavior (*tamper-resilience assumption*).

In a series of works, it was shown that stateful tamper-proof hardware allows statistically UC-secure non-interactive two-party computation, which means that even an unbounded adversary cannot break the security of the protocol. Here, by stateful hardware we mean that the token has a reliable internal state which can persistently store new information. This assumption, however, is relatively strong and does not accurately model most available realizations of tamper-proof hardware. It does e.g. not take into account that many tamper-proof hardware tokens rely on an external power source. Thus, a weaker model was proposed that allows a malicious adversary to reset the hardware arbitrarily.

Our contributions to this research area are manifold. On the one hand, we present optimal protocols with respect to the computational assumptions and number of tokens in existing models. On the other hand, we develop new models of tamper-proof hardware, and investigate the feasibility of UC-secure interactive and non-interactive two-party computation in our models. All our results assume untrusted tokens, i.e. a malicious sender can arbitrarily program the token. To achieve the above results, we develop new cryptographic tools and proof techniques. In a little more detail, our contributions are as follows:

Partially isolated stateful tamper-proof hardware.
We define two weaker models of stateful hardware where the isolation assumption is assumed to hold only in one direction, i.e. either the malicious sender is allowed to send messages to the token, or vice versa. We provide a full char-

acterization of these models with respect to the feasibility of (non-)interactive two-party computation, and show that our constructions are optimal with respect to the number of tokens that we need in the protocols.

Resettable tamper-proof hardware.
We improve upon previous work in the area of resettable, or equivalently stateless, tokens and provide optimal constructions for (non-)interactive two-party computation with respect to the number of tokens, and assume only one-way functions for our constructions.

Bounded-resettable tamper-proof hardware.
We define a slightly more restrictive model of resettable tamper-proof hardware that allows unconditionally UC-secure protocols, which are provably impossible with "standard" resettable hardware. In this model, we show the feasibility of statistically UC-secure interactive two-party computation, and also provide non-interactive protocols for several interesting cryptographic primitives.

Reusable resettable tamper-proof hardware.
We investigate the possibility of reusing a hardware token for several protocols and show that a reusable untrusted signature card can be used to UC-realize non-interactive two-party computation.

Zusammenfassung

Manipulationssichere Hardware ist ein Bestandteil des alltäglichen Lebens, beispielsweise in Form von SIM-Karten, Bankkarten oder auch Ausweisen. In diese sogenannten Token ist üblicherweise ein kryptographischer Schlüssel eingebettet, sodass einfache kryptographische Operationen wie digitales Signieren oder Verschlüsseln umgesetzt werden können. Die inhärenten Sicherheitgarantien von manipulationssicherer Hardware erlauben jedoch deutlich komplexere Anwendungen.

Universell komponierbare (UC) Sicherheit bietet starke Sicherheitsgarantien insbesondere für parallele und verschachtelte Protokollausführungen. Um ein Protokoll in diesem Modell als sicher zu beweisen muss ein sogenanntes Setup (wie etwa eine Public-Key Infrastruktur oder ein echt zufälliger Bitstring) üblicherweise von einer vertrauenswürdiger Instanz erstellt werden. Katz zeigte im Jahr 2007, dass zwei sich gegenseitig misstrauende Parteien mit manipulationssichere Hardware als Setup beliebige Berechnungen sicher durchführen können. Dabei erstellt und instanziiert eine Partei ein manipulationssicheres Token mit einem Programm. Katz konnte zeigen, dass UC-sichere Zweiparteienberechnung möglich ist, obwohl die empfangende Partei dem Token nicht vertraut. Es ist also keine vertrauenswürdige Instanz mehr nötig. Die Sicherheit des Protokolls basiert dabei auf zwei physikalischen Annahmen: (1) Der Sender der manipulationssicheren Hardware hat keine Möglichkeit, mit dieser zu kommunizieren, nachdem sie beim Empfänger angekommen ist (*Isolationsannahme*), und (2) der Empfänger der manipulationssicheren Hardware kann keine Daten aus der Hardware lesen, sondern nur über das Ein- und Ausgabeverhalten Informationen gewinnen (*Manipulationssicherheit*).

In einer Reihe von Arbeiten wurde gezeigt, dass zustandsbehaftete manipulationssichere Hardware sogar statistisch UC-sichere und nicht-interaktive Zweiparteienberechnungen erlaubt, also selbst unbeschränkte Angreifer die Sicherheit nicht brechen können. Zustandsbehaftete Hardware bedeutet hier, dass ein verlässlicher interner Speicher zur Verfügung steht, in den neue Informationen dauerhaft abgelegt werden können. Diese Annahme ist allerdings vergleichsweise stark und triff nicht auf jede Realisierung von manipulationssicherer Hardware zu. Es wird beispielsweise nicht abgebildet, dass die meisten manipulationssicheren Hardwaretoken von einer externen Stromquelle abhängen. Daher wurde ein schwächeres Modell vorgestellt, welches es einem bösartiger Empfänger zusätzlich erlaubt, die Hardware beliebig häufig neuzustarten.

Im Rahmen dieser Dissertation leisten wir vielfältige Beiträge zu diesem Forschungsbereich. Einerseits präsentieren wir Protokolle für bestehende Modell, welche optimal in Bezug auf die kryptographischen Annahmen und die Anzahl an verwendeten Token sind. Andererseits entwickeln wir neue Modellierungen von manipulationssicherer Hardware und untersuchen diese auf die Realisierbarkeit von interaktiver und nicht-interaktiver UC-sicherer Zweiparteienberechnung. Alle unsere Ergebnisse

gehen von nicht-vertrauenswürdiger Hardware aus, d.h. ein bösartiger Sender darf eine beliebige Funktion in der Hardware implementieren. Zum Erlangen unserer Ergebnisse entwickeln wir neue kryptographische Werkzeuge und Beweistechniken. Zusammengefasst leisten wir die folgenden Beiträge:

Teilweise isolierte zustandsbehaftete manipulationssichere Hardware.
Wir definieren zwei neue Modelle für zustandsbehaftete manipulationssichere Hardware, welche eine schwächere einseitige Isolation zwischen der Hardware und ihrem Sender beschreiben, d.h. entweder darf der Sender Nachrichten an die Hardware schicken, oder umgekehrt. Wir geben eine vollständige Charakterisierung dieser Modelle in Bezug auf Realisierbarkeit von (nicht-)interaktiver Zweiparteienberechnung an und zeigen, dass unsere Protokolle eine optimale Anzahl von manipulationssicherer Hardwaretoken nutzen.

Rücksetzbare manipulationssichere Hardware.
Wir entwickeln verbesserte Protokolle für (nicht-)interaktive Zweiparteienberechnung basierend auf zurücksetzbarer (oder äquivalent zustandsloser) manipulationssicherer Hardware und zeigen deren Optimalität bezüglich Anzahl der Hardwaretoken. Die einzige komplexitätstheoretische Annahme, die wir benötigen, sind Einwegfunktionen.

Begrenzt rücksetzbare manipulationssichere Hardware.
Wir entwickeln ein etwas eingeschränktes Modell rücksetzbarer Hardware, bei dem die Anzahl an Neustarts durch einen bösartigen Empfänger a priori begrenzt wird. In diesem Modell kann, im Gegensatz zu „normaler“ zurücksetzbarer Hardware, statistisch sichere Zweiparteienberechnung realisiert werden.

Wiederverwendbare zurücksetzbare manipulationssichere Hardware.
Wir definieren ein neues Modell für zurücksetzbare manipulationssichere Hardware, welches die Wiederverwendung von Hardware in mehreren Protokollen erlaubt. Dies ist üblicherweise im Rahmen von UC-Sicherheit nicht möglich. In diesem Modell erreichen wir effiziente (nicht-)interaktive Zweiparteienberechnung basierend auf einer nicht-vertrauenswürdigen Signaturkarte.

Contents

1. Introduction

Cryptographic tamper-proof hardware tokens are ubiquitous in everyday life. Tamper-proof chips are embedded in SIM-cards, Trusted Platform Modules, credit cards and even passports. When we talk about tamper-proof hardware, we usually assume two properties.

- *Isolation*: After the sender created the token, he has no possibility to directly communicate with the token.
- *Tamper-resilience*: The receiver cannot learn the program that the sender input into the token, apart from its input/output behavior.

Applications for such hardware tokens are manifold. In the area of software protection, tamper-proof hardware can be used to hide critical information of the software from the buyer, as shown by Goldreich and Ostrovsky [GO96]. The tamper-proof property of the token prevents a malicious buyer from copying the token, which in turn prevents unwanted redistribution of the program. A scenario similar to this is Pay-TV, where the broadcaster issues a tamper-proof token to the viewer. It is essential that the viewer cannot duplicate or manipulate such a token, otherwise he would be able to watch TV programs that he did not pay for.

Canetti et al. [CGGM00] show that tamper-proof hardware can also be used for identification purposes without releasing any information about the individual. Similarly, passports allow to create unforgeable digital signatures. The tamper-resilience property is required to prevent identity theft.

Another concept that is directly related to tamper-proof hardware are physically unclonable functions (PUFs) by Pappu [Pap01]. PUFs are functions that can be evaluated very efficiently, but have an intrinsic probabilistic behavior that cannot be cloned due to variations in the physical manufacturing process. The output of a PUF has high entropy and can therefore be applied to cryptographic protocols. On an abstract level, the behavior of a PUF is similar to a one-way function, but relies solely on physical properties.

All these scenarios have in common that the receiver of the token trusts the functionality that is stored on the token. In contrast, tamper-proof hardware was also proposed for electronic wallets by Chaum and Pedersen [CP93], where one party issues a small token that is on the one hand not trusted by the receiver, and

on the other hand cannot communicate with its sender. The physical separation of a party from its token allows strong security guarantees: [CP93] and following results [Bra94, CP94] show that tamper-proof hardware tokens can provide both privacy to its user and security of transactions for the issuing party.

With practical attacks on the tamper-resilience of hardware becoming increasingly sophisticated, e.g. [KJJ99, GMO01, SA03, Koc96], theoretical solutions to this problem were investigated. Ishai et al. [ISW03] present a solution that guarantees the security of a token program even if an adversary has access to some wires of the circuit representation of this program. This models a large class of side-channel attacks. Gennaro et al. [GLM$^+$04] investigate a model where the adversary may apply some function to the secret stored inside the token to influence the behavior of the program. The most general model was proposed by Micali and Reyzin [MR04], who presented a model where an adversary may specify a leakage function that is applied to the computation and releases specific information about the computation to the adversary, possibly including information about the secret keys. This model later spawned the cryptographic research area of leakage-resilient cryptography [DP08].

In 2001, Canetti [Can01] introduced a new security model for composable security of protocols, the Universal Composability (UC) framework. This model provides very strong security guarantees for protocols that are executed concurrently. Most security definitions did not consider concurrent executions of protocols, which led to protocols that are insecure when executed in complex environments such as the internet, see e.g. Goldreich and Krawczyk [GK96b] for the case of zero-knowledge. However, Canetti and Fischlin [CF01] showed that even simple functionalities such as commitment schemes cannot be realized in the UC framework without further setup assumptions.

Intrigued by this impossibility, Hofheinz et al. [HMQU05] investigated the feasibility of UC-secure computation with a trusted tamper-proof signature card as a setup assumption. They presented a UC-secure commitment protocol which, in combination with general results on UC-secure multi-party computation [CLOS02] yields UC-secure two-party computation. While this work already shows the power of tamper-proof hardware in the context of UC-secure computation, it was not until Katz [Kat07] proposed *untrusted* stateful tamper-proof hardware as a setup assumption for the UC framework that the concept was widely adopted. He showed that UC-secure two-party computation can be based on physical assumptions without requiring a trusted setup, in contrast to setups like a public key infrastructure or a common reference string. The result of Katz was improved in a series of works [MS08, CGS08, GIS$^+$10] until Döttling et al. [DKMQ11] showed that statistically UC-secure two-party computation is possible using a single untrusted and stateful hardware token.

Thus, the general focus shifted from using tamper-proof tokens for specific applications to investigating the cryptographic strength of such tokens with respect to two-party computation. Realizing secure two-party computation already allows to securely compute arbitrary functionalities. The problem of efficient and secure two-party computation is one of the most active areas in cryptographic research. In secure two-party computation, two parties, let us call them Alice and Bob, want to correctly compute a function f. Both have their own input, x_A and x_B, respectively, but neither of them wants to disclose the input to the other one. Additionally, the inputs of Alice and Bob are supposed to be independent. As a motivational example, consider Alice and Bob to be two millionaires that want to find out who has more

money. If Alice tells her amount to Bob, Bob can cheat by announcing a larger (or smaller) amount than he actually has. So this poses a problem for the independence of inputs. Additionally, Alice might not want Bob to know the exact amount of money, because she fears of being robbed. Thus, Alice also wants to have input privacy.

Yao [Yao82] was the first to propose a general solution for this problem via so-called garbled circuits. Intuitively, Alice generates an "encrypted" circuit of the function f with her own input hardwired into the circuit. Bob learns the decryption keys corresponding to his input via oblivious transfer (OT) and can thereby evaluate the garbled circuit with his input. A different approach was proposed by Goldreich et al. [GMW87], who use a more interactive protocol. They also use a circuit representation of the function f, but they share the inputs among the parties and evaluate this circuit gate by gate via oblivious transfer. Since then, countless improvements upon these initial protocols have been made, e.g. [DPSZ12, FJN$^+$13, LR14, LPSY15] to name a few recent works.

This rekindled interest in tamper-proof hardware also led to new protocols and applications outside of the UC framework, e.g. one-time programs [GKR08] and set intersection [HL08]. Recently, PUFs were also introduced into the UC framework as a setup assumption by Brzuska et al. [BFSK11] and Ostrovsky et al. [OSVW13], but it was shown that PUFs are not as powerful as stateful tamper-proof hardware tokens: Dachman-Soled et al. [DFK$^+$14] showed that if one allows maliciously created PUFs, OT and thus two-party computation is not possible.

1.1 Contribution of the Thesis

The central question of this thesis is whether weaker models of untrusted tamper-proof hardware can be used to UC-realize two-party computation. In the following, we thoroughly investigate the cryptographic strength of weaker hardware models derived from the original definition of [Kat07] with respect to UC-secure two-party computation. In a little more detail, we provide formal definitions and general feasibility results for the following models:

Partially isolated stateful tamper-proof hardware:
The isolation between the sender and the token is removed in one direction, i.e. the sender can send messages to the token, or vice versa.

Resettable tamper-proof hardware:
The receiver of the token is allowed to reset the state of the token.

Bounded-resettable tamper-proof hardware:
The receiver of the token is allowed to reset the state of the token only up to an a priori fixed bound.

Reusable resettable tamper-proof hardware:
The same hardware token can be used by the receiver for different cryptographic protocols.

Our results are summarized in Table 1.1. Results shaded in gray were already known and are added as a reference, the other results are either completely new or are an improvement over previous results. A checkmark in parentheses means

Model	computational 2PC		statistical 2PC	
	interactive	non-interactive	interactive	non-interactive
stateful	✓	✓	✓	✓
stateful (inc. com.)	✓	✓	✓	✓
stateful (outg. com.)	✓	✓	✗(✓)	✗
resettable	✓	✓	✗(✓)	✗
bounded-resettable	✓	✓	✓	(✓)
reusable resettable	✓	✓	✗(✓)	✗

Table 1.1: Overview of our results. A checkmark indicates the feasibility of computational/statistical two-party computation with/without interaction. A checkmark in parentheses indicates constructions of cryptographic primitives that do not imply two-party computation. Results shaded in gray are added for reference.

that constructive results for interesting functionalities exist in the corresponding area, but they do not imply computational/statistical (non-)interactive two-party computation. A detailed discussion both of the state of the art for each model as well as our corresponding results is given in the respective chapter.

1.2 Structure of the Thesis

This thesis is structured as follows. In each chapter, we first introduce a model, define tools and study limitations, and then present protocols for various cryptographic tasks. At the end of each chapter, our results are put into context with two-party computation. A brief summary of each chapter is given hereafter.

- In Chapter 2, we define the general notation and the security model for our proofs. We also present standard tools and definitions that will be used throughout this thesis. More exotic tools and definitions will be provided separately within the chapters.

- Chapter 3 gives an overview over UC-secure results in the stateful tamper-proof hardware model, and provides a more detailed introduction to UC-secure protocols based on tamper-proof hardware tokens.

- In Chapter 4, we define two new models of stateful tamper-proof hardware where the isolation assumption is weakened. These models cover the case of communication from the sender to the token and the case of communication from the token to the sender. The result was originally published in ICITS 2015 [DMQN15].

- In Chapter 5, we present our results in the resettable hardware model. These results are an improvement over Goyal et al. [GIS+10] and were originally published in TCC 2013 [DMMQN13] and ProvSec 2015 [DKMQN15].

- In Chapter 6, we define a new model for resettable tamper-proof hardware, where the number of resets is bounded a priori. This allows statistically secure OT, which is impossible with normal resettable hardware. This result was originally published in TCC 2015 [DKMN15].
- In Chapter 7, we propose a new model for resettable hardware that allows to reuse the hardware token in several protocols. Our results imply efficient UC-secure computation from untrusted signature cards.
- Chapter 8 summarizes our results and addresses some questions that arise from this work.

2. Preliminaries

In this chapter, we introduce the tools and notations that will be used throughout this thesis.

2.1 General Notations

We will first state some conventions. In this thesis, we investigate the feasibility of two-party protocols based on tamper-proof hardware. Unless stated otherwise, the protocol parties are called sender S and receiver R. We use the terms token sender, token issuer and token creator interchangeably.

The concatenation of two elements x and y is denoted by $x\|y$. For a value x, we define the following conventions. x^* denotes a value chosen or computed by an adversary and $\hat{x}$ denotes a value chosen or computed by the simulator.

An algorithm A that has access to an oracle algorithm $\mathcal{O}$ is denoted by $A^{\mathcal{O}}$. In our protocols, the notation $\mathsf{Scheme.Algorithm}(\cdot)$ denotes applying the $\mathsf{Algorithm}$-algorithm of the scheme Scheme.

By $x \leftarrow X$, we denote that x is drawn according to the distribution X via a probabilistic process. For two distributions X and Y, we denote the computational indistinguishability of X and Y by $X \underset{c}{\approx} Y$ and the statistical indistinguishability by $X \underset{s}{\approx} Y$. $\mathcal{U}_\kappa$ denotes the uniform distribution over $\{0,1\}^\kappa$.

In the case of *computational* cryptography, the security of cryptographic constructions is usually stated with respect to asymptotic parameters, meaning that for a large enough security parameter, the proven statement holds. We denote the security parameter by κ, unless explicitly stated otherwise.

Definition 2.1. *We call a function $f : \mathbb{N} \to [0,1]$* negligible *if there exists a value $n_0 \in \mathbb{N}$ for every polynomial $p(n)$ such that for every $n > n_0$, $f(n) < \frac{1}{p(n)}$. We will later use $\mathsf{negl}(\kappa)$ to signify an unspecified function that is negligible in the security parameter κ. Similarly, we call a function $f : \mathbb{N} \to [0,1]$* overwhelming*, if $1 - f$ is a negligible function.*

By definition, manipulation of a negligible function by a polynomial factor does not influence its asymptotic behavior, i.e. multiplication of a negligible function by a polynomial yields a negligible function.

By PPT, we denote a probabilistic polynomial time algorithm. We call an algorithm *efficient* if it runs in probabilistic polynomial time. In particular, any adversary against a computationally secure protocol must be efficient.

We say that x is polynomial in κ, or $x = \mathsf{poly}(\kappa)$, if there exist some constants α, c such that $x \leq \alpha\kappa^c$.

In the context of *statistical* security, we write $\Delta(x, y)$ for the statistical distance between x and y. The inner product of x and y is denoted as $\langle x \,|\, y \rangle$. By $\mathbb{F}_q$, we denote the finite field with q elements.

We canonically extend the notion of polynomials over a field $\mathbb{F}$ as follows. By $\mathbb{F}^n[X]$ we denote the set of all n-tuples of polynomials $p_1, \ldots, p_n \in \mathbb{F}[X]$. Each polynomial $p := (p_1, \ldots, p_n) \in \mathbb{F}^n[X]$ defines a function $\mathbb{F} \to \mathbb{F}^n$, $x \mapsto \big(p_1(x), \ldots, p_n(x)\big)$ whose degree is $\deg(p) := \max_{i=1}^{n}(\deg(p_i))$. We treat $\mathbb{F}^n[X]$ as an $\mathbb{F}$-linear vector space in the natural way.

2.2 Basic Primitives

In this section we define basic cryptographic primitives. We will later use one-way functions, pseudorandom functions and generators in our protocols. Also the concept of a secret sharing scheme will be useful.

2.2.1 One-Way Functions

A one-way function is the most basic primitive in the context of computational security. It describes a function that is easy to compute, but hard to invert.

Definition 2.2. *An efficiently computable function* $\mathsf{OWF} : \{0,1\}^* \to \{0,1\}^*$ *is called (strongly)* one-way *if for every PPT algorithm* $\mathcal{A}$

$$\Pr[\mathcal{A}(\mathsf{OWF}(\mathcal{U}_\kappa), \kappa) \in \mathsf{OWF}^{-1}(\mathsf{OWF}(\mathcal{U}_\kappa))] \leq \mathsf{negl}(\kappa).$$

2.2.2 Pseudorandom Generators and Functions

A pseudorandom generator basically expands a random string by some factor, thus creating a longer string that is indistinguishable from a truly random string of the same size.

Definition 2.3. *An efficiently computable function* $\mathsf{PRG} : \{0,1\}^\kappa \to \{0,1\}^{l(\kappa)}$ *with a polynomial* $l(\cdot)$ *and* $l(\kappa) > \kappa$ *is called a* pseudorandom generator *if for every PPT algorithm* $\mathcal{A}$

$$\Pr[\mathcal{A}(\mathsf{PRG}(\mathcal{U}_\kappa)) = 1] - \Pr[\mathcal{A}(\mathcal{U}_{l(\kappa)}) = 1] \leq \mathsf{negl}(\kappa).$$

Håstad et al. [HILL99] showed that a PRG can be constructed from any OWF. A PRG is somewhat limited in providing random access to a pseudorandom string. This problem is solved by pseudorandom functions.

Definition 2.4. *An efficiently computable function* $\mathsf{PRF} : \{0,1\}^n \times \{0,1\}^\kappa \to \{0,1\}^m$ *is called a* pseudorandom function *if for every PPT algorithm* $\mathcal{A}$

$$\Pr_{s \leftarrow \mathcal{U}_\kappa}[\mathcal{A}^{\mathsf{PRF}(\cdot,s)} = 1] - \Pr_{h \leftarrow H}[\mathcal{A}^h = 1] \leq \mathsf{negl}(\kappa),$$

where H *is the uniform function family* $\{h : \{0,1\}^\kappa \to \{0,1\}^\kappa\}$.

As Goldreich et al. [GGM86] showed, PRFs can be constructed from PRGs, hence PRFs can be constructed from OWFs.

2.2.3 Secret Sharing Schemes

A secret sharing scheme allows to split a value into several parts, the so-called shares, such that one share by itself does not reveal the original value. A very simple secret sharing scheme is to split a value $v \in \{0,1\}^\kappa$ into two parts $v_1, v_2 \in \{0,1\}^\kappa$ such that $v = v_1 \oplus v_2$.

Definition 2.5. *A* (k,n)-secret sharing scheme *for messages* $m \in \{0,1\}^\kappa$ *consists of two PPT algorithms* Share *and* Reconstruct *with the following properties:*

- $\mathsf{Share}(\kappa, m)$ *takes as input a message* m *and security parameter* κ *and outputs* n *shares* $(s_1, \ldots, s_n)$.
- $\mathsf{Reconstruct}(\mathsf{S})$ *takes as input a set* S *of shares and outputs* m *if* $|S| \geq k$.

It has to hold that $k-1$ *shares leave* m *undetermined, i.e. for every algorithm* $\mathcal{A}$*,*

$$\Pr[m \leftarrow \mathcal{A}(S, \kappa) \text{ with } |S| \leq k-1] \leq \mathsf{negl}(\kappa).$$

Shamir's secret sharing scheme [Sha79] satisfies this definition.

2.3 Commitment Schemes

We use several types of commitment schemes in this thesis. A commitment is a (possibly interactive) protocol between two parties and consists of two phases. In the commit phase, the sender commits to a value and sends the commitment to the receiver. The receiver must not learn the underlying value before the unveil phase, where the sender sends the unveil information to the receiver. The receiver can check the correctness of the commitment. A commitment must thus provide two security properties: a hiding property that prevents the receiver from extracting the input of the sender out of the commitment value, and a binding property that ensures that the sender cannot unveil a value other than the one he committed to.

Definition 2.6. *A* commitment scheme COM *between a sender* S *and a receiver* R *consists of two PPT algorithms* Commit *and* Open *with the following functionality.*

- Commit *takes as input a message* s *and computes a commitment* c *and unveil information* d.
- Open *takes as input a commitment* c*, unveil information* d *and a message* s *and outputs a bit* $b \in \{0,1\}$.

We require the commitment scheme to be correct, i.e. for all s*:*

$$\mathsf{Open}(\mathsf{Commit}(s), s, d) = 1$$

Throughout this theses, we mainly use statistically binding and computationally hiding commitments. The hiding property basically states that for all two messages, the receiver cannot distinguish which message is hidden in the commitment. We use a definition similar to IND-CPA security for encryption.

Definition 2.7. *We say that* $\mathsf{COM} = (\mathsf{Commit}, \mathsf{Open})$ *is* computationally hiding *if for every PPT algorithm* $\mathcal{A}_{\mathsf{R}}$*:*

$$\Pr[(s_0, s_1) \leftarrow \mathcal{A}_{\mathsf{R}}(\kappa); b \leftarrow \{0,1\}; (c,d) \leftarrow \mathsf{Commit}(s_b); b' \leftarrow \mathcal{A}_{\mathsf{R}}(c) \wedge b = b'] \leq \tfrac{1}{2} + \mathsf{negl}(\kappa).$$

Definition 2.8. *We say that* $\mathsf{COM} = (\mathsf{Commit}, \mathsf{Open})$ *is* statistically binding *if for every algorithm* $\mathcal{A}_{\mathsf{S}}$*:*

$$\Pr[(c, d, d', s, s') \leftarrow \mathcal{A}_{\mathsf{S}}(\kappa) \text{ s.t. } d \neq d' \wedge s \neq s' \wedge \mathsf{Open}(c,d,s) = \mathsf{Open}(c,d',s') = 1] \leq \mathsf{negl}(\kappa).$$

There are constructions of statistically binding commitments from one-way functions, e.g. Naor [Nao90]. The commitment from [Nao90] requires an interactive commit phase, where the receiver first draws a random value and sends it to the sender. We will call this value k and include it into the commit algorithm, i.e. we write $\mathsf{COM.Commit}(k, m)$.

Further, we need extractable commitments. Extractabilty is a stronger form of the binding property which states that the sender is not only bound to one input, but that there also exists an (efficient) extraction algorithm that extracts this value. Our definition of extractable commitments is derived from Pass and Wee [PW09].

Definition 2.9. *We say that* $\mathsf{COM} = (\mathsf{Commit}, \mathsf{Open})$ *is* extractable*, if there exists a PPT algorithm* Ext *that, given black-box access to any malicious PPT algorithm* $\mathcal{A}_{\mathsf{S}}$*, outputs a pair* $(\hat{s}, \tau)$ *such that*

- *(simulation)* τ *is identically distributed to the view of* $\mathcal{A}_{\mathsf{S}}$ *at the end of interacting with an honest receiver* R *in the commit phase,*
- *(extraction) the probability that* τ *is accepting and* $\hat{s} = \perp$ *is negligible, and*
- *(binding) if* $\hat{s} \neq \perp$*, then it is infeasible to open* τ *to any value other than* $\hat{s}$*.*

Extractable commitments can be constructed from any commitment scheme via additional interaction, see e.g. [Gol01, MOSV06]. The definition of extractable commitments implicitly allows the extractor to rewind the adversarial sender to extract the input. In some scenarios, especially in the context of concurrently secure protocols, it is necessary that the extractor can extract the input without rewinding. This is obviously impossible in the plain model, as a malicious receiver could employ the same strategy to extract the sender's input. Thus, some form of setup (e.g. tamper-proof hardware) is necessary to obtain straight-line extractable commitments.

Definition 2.10. *We say that* $\mathsf{COM} = (\mathsf{COM.Commit}, \mathsf{COM.Open})$ *is* straight-line extractable *if in addition to Definition 2.9, the extractor does not use rewinding.*

Another tool that we need is a trapdoor commitment scheme, where the sender can equivocate a commitment if he knows a trapdoor. We adapt a definition from Canetti et al. [CJS14].

Definition 2.11. *A* trapdoor commitment scheme TCOM *between a sender* S *and a receiver* R *consists of five PPT algorithms* KeyGen*,* TVer*,* Commit*,* Equiv *and* Open *with the following functionality.*

- KeyGen *takes as input a security parameter and creates a key pair* $(\mathsf{pk}, \mathsf{sk})$*, where* sk *serves as the trapdoor.*

- TVer *takes as input* pk *and* sk *and outputs* 1 *iff* sk *is a valid trapdoor for* pk.
- Commit *takes as input a message* s *and computes a commitment* c *and unveil information* d.
- Equiv *takes as input the trapdoor* sk, *message* s' *and commitment* c *and outputs an unveil information* d' *for* s'.
- Open *takes as input a commitment* c, *unveil information* d *and a message* s *and outputs a bit* $b \in \{0,1\}$.

The algorithm Equiv *has to satisfy the following condition. For every PPT algorithm* $\mathcal{A}_{\mathsf{R}}$, *the following distributions are computationally indistinguishable.*

- (pk, c, d, s), *where* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{A}_{\mathsf{R}}(\kappa)$ *such that* $\mathsf{TVer}(\mathsf{pk}, \mathsf{sk}) = 1$ *and* $(c, d) \leftarrow \mathsf{Commit}(\mathsf{pk}, s)$
- (pk, c', d', s), *where* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{A}_{\mathsf{R}}(\kappa)$ *such that* $\mathsf{TVer}(\mathsf{pk}, \mathsf{sk}) = 1$, $(c', z) \leftarrow \mathsf{Commit}(\mathsf{pk}, \cdot)$ *and* $d' \leftarrow \mathsf{Equiv}(\mathsf{sk}, z, s)$

For example, the commitment scheme by Pedersen [Ped92] satisfies the above definition.

2.4 Digital Signature Schemes and Message Authentication Codes

2.4.1 Message Authentication Codes

Message authentication codes (MACs) provide an unforgeable message digest. Both parties have to share a common key to compute and verify the MAC. In our constructions, we require statistically secure MACs, e.g. the construction of Wegman and Carter [WC81].

Definition 2.12. *A* message authentication code MAC *consists of three PPT algorithms* $\mathsf{KeyGen}, \mathsf{Mac}$ *and* Verify *with the following functionality:*

- $\mathsf{KeyGen}(\kappa)$ *takes as input a security parameter* κ *and outputs a key* k.
- $\mathsf{Mac}(k, m)$ *takes as input a key* k *and a message* m *and outputs a tag* t.
- $\mathsf{Verify}(k, m, t)$ *takes as input a key* k, *a message* m *and a tag* t. *It outputs a bit* b.

We require correctness, i.e. for all m *and* $k \leftarrow \mathsf{KeyGen}(\kappa)$:

$$\mathsf{Verify}(k, m, \mathsf{Mac}(k, m)) = 1,$$

and statistical unforgeability of the MAC, i.e. for all $\mathcal{A}$:

$$\Pr[k \leftarrow \mathsf{KeyGen}(\kappa); (m^*, t^*) \leftarrow \mathcal{A}^{\mathsf{Mac}(k,\cdot)}; \mathsf{Verify}(k, m^*, t^*) = 1] \leq \mathsf{negl}(\kappa),$$

where m^* *was not sent to the MAC oracle.*

2.4.2 Digital Signatures

Similar to MACs, digital signatures allow to compute an unforgeable message digest. In contrast to MACs, however, digital signatures are an asymmetrical cryptosystem, i.e. the signer has a signing key sgk, and he can publish the verification key vk such that anyone can verify the correctness of a signature.

Definition 2.13. *A* digital signature scheme SIG *consists of three PPT algorithms* KeyGen, Sign *and* Verify.

- $\mathsf{KeyGen}(\kappa)$ *takes as input the security parameter* κ *and generates a key pair consisting of a verification key* vk *and a signature key* sgk.
- $\mathsf{Sign}(\mathsf{sgk}, m)$ *takes as input a signature key* sgk *and a message* m*, and outputs a signature* σ *on* m.
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$ *takes as input a verification key* vk*, a message* m *and a presumed signature* σ *on this message. It outputs 1 if the signature is correct and 0 otherwise.*

We require correctness, i.e. for all m *and* $(\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathsf{KeyGen}(\kappa)$*:*

$$\mathsf{Verify}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sgk}, m)) = 1.$$

For our constructions, the signature schemes have to fulfill the security property *existential unforgeability under chosen messages* (EUF-CMA), i.e. an adversary is not supposed to be able to forge a signature for any message of his choosing. In the EUF-CMA-security experiment, the experiment first executes the KeyGen algorithm to create a key pair $(\mathsf{vk}, \mathsf{sgk})$. The adversary $\mathcal{A}$ is given the verification key vk and access to a signature oracle $\mathcal{O}^{\mathsf{SIG.Sign}(\mathsf{sgk},\cdot)}$ that signs arbitrary messages. $\mathcal{A}$ wins the experiment if he manages to forge a valid signature σ^* for a message m^* without having queried the signature oracle with m^*.

A signature scheme SIG is called EUF-CMA-secure if no PPT adversary $\mathcal{A}$ wins the EUF-CMA-experiment with non-negligible probability. For the sake of simplicity, we require signature schemes with a deterministic verification procedure and succinct signature length (i.e. the length of σ does not depend on m). Standard hash-and-sign constructions based on any one-way function [NY89, Rom90] satisfy these requirements.

Additionally, we require the signing procedure to be deterministic. However, this is no restriction since the random coins used for signing can be chosen by a pseudorandom function, which is seeded by a part of the signing key.

2.4.2.1 Unique Digital Signatures

An additional property of some digital signature schemes is the uniqueness of the signatures. Our definition is taken from Lysyanskaya [Lys02]. Such schemes are known only from specific number theoretic assumptions.

Definition 2.14. *Let* SIG *be a digital signature scheme. A signature scheme is called* unique *if additionally to the properties of Definition 2.13 the following property holds. There exists no tuple* $(\mathsf{vk}, m, \sigma_1, \sigma_2)$ *such that* $\mathsf{SIG.Verify}(\mathsf{vk}, m, \sigma_1) = 1$ *and* $\mathsf{SIG.Verify}(\mathsf{vk}, m, \sigma_2) = 1$ *with* $\sigma_1 \neq \sigma_2$.

We point out that in the above definition, vk, σ_1, and σ_2 need not be created honestly by the respective algorithms, but may be arbitrary strings.

2.4.2.2 Sig-Com Schemes

In search for protocols based on weaker assumptions, Chung et al. [CPS13] define an interactive analogue of collision resistant hash functions from commitments and signatures, which can be based on one-way functions. In a sense, the interaction is a trade-off for a weaker assumption. They show that a tree based on signatures and commitments, a so-called *sig-com tree*, can be compressing and has a collision resistance property.

In their scheme, one party creates the signature and verification keys, and sends the verification key to the other party. The other party sends a commitment on its input and obtains a signature on the commitment, i.e. the party with the signature key acts as a signature oracle. It is necessary to separate the signature key from the input for the sig-com tree. Without this separation, the security of the signature scheme (and hence the collision resistance property) would no longer hold. The commitments to the input are necessary because otherwise the sender could abort depending on the received message. The commit-then-sign step can be applied sequentially to create a tree structure analogous to Merkle trees. In the following definition, the commitment is non-interactive, but this is only for ease of presentation. Any interactive statistically binding commitment scheme, e.g. [Nao90], can be used as well.

Definition 2.15 ([CPS13])**.** *Let* $\mathsf{SIG} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be an EUF-CMA-secure length-*κ *signature scheme and let* COM *be a non-interactive commitment scheme. Define a* Sig-Com scheme $\mathsf{SIG}' = (\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Verify}')$ *to be a triple of PPT algorithms defined as follows:*

- $\mathsf{KeyGen}' = \mathsf{KeyGen}$.
- $\mathsf{Sign}'(\mathsf{sgk}, m)$*: computes a commitment* $(c, d) \leftarrow \mathsf{COM.Commit}(m)$*, sets* $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, c)$ *and outputs* (σ, d).
- $\mathsf{Verify}'(\mathsf{vk}, m, \sigma, d)$*: output* 1 *iff* $\mathsf{SIG.Verify}(\mathsf{vk}, c, \sigma) = \mathsf{COM.Open}(c, d, m) = 1$.

Using a sig-com scheme, one can create a tree structure that is defined in the following.

Definition 2.16 ([CPS13])**.** *Let* $\mathsf{SIG} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be an EUF-CMA-secure length-*κ *signature scheme, let* COM *be a non-interactive commitment scheme, and let* $\mathsf{SIG}' = (\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Verify}')$ *be the sig-com scheme corresponding to* SIG *and* COM*. Let* $(\mathsf{vk}, \mathsf{sgk})$ *be a key pair of* SIG'*, and* s *be a bit string of length* 2^d*. A sig-com tree for* s *w. r. t.* $(\mathsf{vk}, \mathsf{sgk})$ *is a complete binary tree of depth* d*, defined as follows.*

- *A leaf* l_γ *indexed by* $\gamma \in \{0,1\}^d$ *is set as the bit at position* γ *in* s.
- *An internal node* l_γ *indexed by* $\gamma \in \bigcup_{i=0}^{d-1}\{0,1\}^i$ *satisfies that there exists some* r_γ *such that* $\mathsf{Verify}'\big(\mathsf{vk}, (l_{\gamma_0}, l_{\gamma_1}), l_\gamma, r_\gamma\big) = 1$*. (By* $l_{\gamma_0}, l_{\gamma_1}$ *we denote the left and right child of an inner node* l_γ*.)*

Note that sig-com trees have a collision resistance property in the following sense: no adversary with oracle access to a signature oracle $\mathcal{O}^{\mathsf{SIG.Sign}(\mathsf{sgk},\cdot)}$ can output a root and a sequence of signatures for both 0 and 1 for any leaf γ. This property stems from the binding property of the commitment and the unforgeability of the signature scheme.

2.5 Zero-Knowledge and Witness-Indistinguishable Proofs

We construct proof/argument systems for languages in $\mathcal{NP}$. Let a language $\mathcal{L}$ be in $\mathcal{NP}$ with witness relation $\mathcal{R}_{\mathcal{L}}$ and witness set $w_{\mathcal{L}}(x) = \{w : (x, w) \in \mathcal{R}_{\mathcal{L}}\}$. We distinguish between a proof system, which implies security against an unbounded corrupted prover, and an argument system that is secure against computationally bounded provers.

2.5.1 Zero-Knowledge

A zero-knowledge proof system is an interactive proof system that allows a prover to prove that an element x is in a language $\mathcal{L}$. Additionally, the zero-knowledge property guarantees that the witness w remains hidden from the verifier. This property is defined via a simulator which can simulate an accepting transcript that is indistinguishable from a real one without knowing the witness.

Definition 2.17. *A* zero-knowledge proof system *for a language $\mathcal{L} \in \mathcal{NP}$ consists of a pair of algorithms (P, V) such that there exist a PPT algorithm Sim and the following conditions hold.*

- *Completeness: For every $(x, w) \in \mathcal{R}_{\mathcal{L}}$,*

$$\Pr[\langle \mathsf{P}(w), \mathsf{V} \rangle(x) = 1] = 1.$$

- *Soundness: For every $x \notin \mathcal{L}$ and every malicious prover P^*,*

$$\Pr[\langle \mathsf{P}^*, \mathsf{V} \rangle(x) = 1] \leq \mathsf{negl}(|x|).$$

- *Computational Zero-Knowledge: For every $(x, w) \in \mathcal{R}_{\mathcal{L}}$ and every PPT verifier V^*, the distributions $\mathsf{Real} = \{\langle \mathsf{P}(w), \mathsf{V}^* \rangle(x)\}$ and $\mathsf{Ideal} = \{\mathsf{Sim}(x, \mathsf{V}^*)\}$ are computationally indistinguishable.*

In the context of resettable tamper-proof hardware, we rely on resettably-sound zero-knowledge arguments of knowledge for our proofs. In such a proof system, the verifier is resettable and has to reuse his randomness for each protocol run. In addition, the argument of knowledge property states that a convincing prover knows a witness. This is formalized by an extractor that, given black-box access to the prover, can extract the witness from the prover.

Definition 2.18. *A* resettably-sound zero-knowledge argument of knowledge system *(rsZKAoK) for a language $\mathcal{L} \in \mathcal{NP}$ consists of a pair of PPT algorithms (P, V), where the verifier V is resettable, such that there exist two PPT algorithms Sim and Ext and the following conditions hold.*

- *Completeness: For every $(x, w) \in \mathcal{R}_{\mathcal{L}}$,*

$$\Pr[\langle \mathsf{P}(w), \mathsf{V} \rangle(x) = 1] = 1.$$

- *Soundness: For every $x \notin \mathcal{L}$ and every malicious PPT prover P^* ,*

$$\Pr[\langle \mathsf{P}^*, \mathsf{V} \rangle(x) = 1] \leq \mathsf{negl}(|x|).$$

- *Computational Zero-Knowledge: For every* $(x, w) \in \mathcal{R}_{\mathcal{L}}$ *and every stateful or resettable PPT verifier* V^**, the distributions* $\mathsf{Real} = \{\langle \mathsf{P}(w), \mathsf{V}^* \rangle(x)\}$ *and* $\mathsf{Ideal} = \{\mathsf{Sim}(x, \mathsf{V}^*)\}$ *are computationally indistinguishable.*

- *Proof of Knowledge: For every* $x \in \mathcal{L}$ *and every PPT algorithm* P^**, there exists a negligible function* ν *such that* $\Pr[\mathsf{Ext}(x, \mathsf{P}^*) \in w_{\mathcal{L}}(x)] > \Pr[\langle \mathsf{P}^*, \mathsf{V} \rangle(x) = 1] - \nu$.

Recent constructions of rsZKAoK can be based on one-way functions [CPS13, BP13].

2.5.2 Witness-Indistinguishability

A notion that is weaker than zero-knowledge but still provides a meaningful functionality is witness-indistinguishabiliy. It basically states that a verifier cannot distinguish which witness the prover used to prove the statement, even if the verifier knows all witnesses. A popular application of witness-indistinguishable proofs are OR statements, where one part of the statement provides a trapdoor, that should be hidden from the verifier.

Definition 2.19. *A* witness indistinguishable argument of knowledge system *for a language* $\mathcal{L} \in \mathcal{NP}$ *consists of a pair of PPT algorithms* (P, V)*, such that there exist a PPT algorithm* Sim *and the following conditions hold.*

- *Completeness: For every* $(x, w) \in \mathcal{R}_{\mathcal{L}}$*,*

$$\Pr[\langle \mathsf{P}(w), \mathsf{V} \rangle(x) = 1] = 1.$$

- *Soundness: For every* $x \notin \mathcal{L}$ *and every malicious PPT prover* P^**,*

$$\Pr[\langle \mathsf{P}^*, \mathsf{V} \rangle(x) = 1] \leq \mathsf{negl}(|x|).$$

- *Witness-indistinguishability: For every* $w_1 \neq w_2$ *such that* $(x, w_1) \in \mathcal{R}_{\mathcal{L}}$*,* $(x, w_2) \in \mathcal{R}_{\mathcal{L}}$ *and every PPT verifier* V^**, the distributions* $\{\langle \mathsf{P}(w_1), \mathsf{V}^* \rangle(x)\}$ *and* $\{\langle \mathsf{P}(w_2), \mathsf{V}^* \rangle(x)\}$ *are computationally indistinguishable.*

- *Proof of Knowledge: For every* $x \in \mathcal{L}$ *and every PPT algorithm* P^**, there exists a negligible function* ν *such that* $\Pr[\mathsf{Ext}(x, \mathsf{P}^*) \in w_{\mathcal{L}}(x)] > \Pr[\langle \mathsf{P}^*, \mathsf{V} \rangle(x) = 1] - \nu$.

Witness-indistinguishable arguments/proofs of knowledge are also sometimes referred to as *witness-extractable*. Similar to the case of extractable commitments, one can also require the extractor to be straight-line, i.e. the extractor may not rewind the prover. Again, this requires an additional setup assumption and is not possible in the plain model.

Definition 2.20. *We say that a witness-indistinguishable argument/proof system is* straight-line witness-extractable *if in addition to Definition 2.19, the extractor does not use rewinding.*

2.6 Information Theory and Randomness Extraction

We use the standard notions of information theory. For statistical security, we usually need the min-entropy H_∞—i.e. the worst case entropy—instead of Shannon-entropy H.

Definition 2.21. *Let X,Y be two random variables.*

- *Information: for $x \in X$,*
$$I(x) = -\log \Pr[X = x].$$
- *Entropy:*
$$H(X) = \mathbb{E}[I(X)] = \sum_{x \in X} \Pr[X = x] I(\Pr[X = x]).$$
- *Conditional Entropy:*
$$H(X|Y) = \sum_{x \in X, y \in Y} \Pr[X = x, Y = y] \log \frac{\Pr[X=x]}{\Pr[X=x, Y=y]}.$$
- *Mutual Information:*
$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} \Pr[X = x, Y = y] \log \frac{\Pr[X=x, Y=y]}{\Pr[X=x]\Pr[Y=y]}.$$
- *Min-entropy:*
$$H_\infty(X) = -\log(\max_{x \in X} \Pr[X = x]).$$

In order to state the leftover hash lemma, we need several tools. First, we define the notion of average min-entropy and a lemma to estimate the average conditional min-entropy.

Definition 2.22 ([DORS08])**.** *Let X and Y be random variables. Then the* average conditional min-entropy *of X given Y is defined as*
$$\tilde{H}_\infty(X|Y) := -\log \mathbb{E}_{y \in Y}[2^{-H_\infty(X|Y=y)}].$$

The following chaining lemma to estimate the average conditional min-entropy will be useful when we want to apply the leftover hash lemma.

Lemma 2.23 ([DORS08])**.** *Let X,Y and Z be random variables. Then*

1. *For any $\delta > 0$, the conditional entropy $H_\infty(X|Y = y)$ is at least $\tilde{H}_\infty(X|Y) - \log \frac{1}{\delta}$ with probability at least $1 - \delta$ over the choice of y.*
2. *If Y has at most 2^λ possible values, then $\tilde{H}_\infty(X|(Y,Z)) \geq \tilde{H}_\infty((X,Y)|Z) - \lambda \geq \tilde{H}_\infty(X|Z) - \lambda$. In particular, $\tilde{H}_\infty(X|Y) \geq H_\infty((X,Y)) - \lambda \geq H_\infty(X) - \lambda$.*

The leftover hash lemma makes use of 2-universal hash functions. Basically, a 2-universal hash function is a randomly drawn function from a set of hash functions such that guarantees that the collisions on an input space are distributed uniformly.

Definition 2.24. *We say that $\mathcal{H} : X \to \{1, \ldots, m\}$ is 2-universal if for all $x, y \in X$ with $x \neq y$,*

$$\Pr_{h \leftarrow \mathcal{H}}[h(x) = h(y)] \leq \tfrac{1}{m},$$

where $h \leftarrow \mathcal{H}$ means that **h** *is selected uniformly at random from $\mathcal{H}$.*

We now state two versions of the generalized leftover hash lemma, which basically states that a 2-universal hash function is a good randomness extractor, i.e. it is possible to extract close to uniformly random strings from a leaky random source.

Lemma 2.25 (Generalized Leftover Hash Lemma [DORS08])**.** *Let any finite sets X, Y, Z and a tuple of random variables $(\hat{x}, z)$ with arbitrary joint distribution over $X \times Z$ be given. Let $\mathcal{H}$ be a family of 2-universal hash functions $\{h : X \to Y\}$, and let $h \leftarrow \mathcal{H}$ and $u \leftarrow Y$. Then:*

$$\Delta\Big((h(\hat{x}), h, z), (u, h, z)\Big) \;\leq\; \tfrac{1}{2}\sqrt{2^{-\tilde{H}_\infty(X \mid Z)} \cdot 2^{|Y|}}$$

Sometimes the following reformulation of Lemma 2.25 will be useful.

Lemma 2.26. *Let any finite sets X, Y, Z and a tuple of random variables $(\hat{x}, z)$ with arbitrary joint distribution over $X \times Z$ be given. Let $\mathcal{H}$ be a family of 2-universal hash functions $\{h : X \to Y\}$, and let $h \leftarrow \mathcal{H}$ and $u \leftarrow Y$. Then:*

$$\Delta\Big((h(\hat{x}), h, z), (u, h, z)\Big) \;\leq\; \tfrac{1}{2}\sqrt{\max_{e:Z\to X} \Pr[\hat{x} = e(z)] \cdot |Y|}$$

2.7 Security Model

We state and prove our results in the Universal Composability framework of Canetti [Can01]. Therefore, this section is meant to give a short overview over the formalizations of the framework. We also provide a short description of common primitives in the UC framework that will be used throughout this thesis. Non-standard and more uncommon primitives are modeled in the respective sections.

2.7.1 The Universal Composability Framework

The *Universal Composability* (UC) framework was introduced by Canetti [Can01] to solve the problem of defining security models for concurrent protocol executions. While we will not go into detail here, it is commonly known that many cryptographic protocols do not retain their security when executed in parallel with other protocols (or even the same protocol), see e.g. [GK96b] for the case of zero-knowledge.

Security is defined via the comparison of an *ideal model* and a *real model*. In the real model, a protocol Π between the protocol participants is carried out, while in the ideal model the parties only communicate with an ideal functionality $\mathcal{F}$ that is supposed to model the ideal security guarantees of the protocol. For an adversary $\mathcal{A}$ in the real protocol who coordinates the behavior of all malicious parties, there has to exist a simulator $\mathcal{S}$ for $\mathcal{A}$ in the ideal protocol. An environment $\mathcal{Z}$, which is plugged to both the real and the ideal protocol, provides the inputs to the parties and can read the outputs, but $\mathcal{Z}$ must not be able to distinguish these models. Thus, even with concurrently executed protocols (running in the environment) the security holds. Usually, we assume that $\mathcal{A}$ is a dummy adversary controlled by $\mathcal{Z}$,

which means that $\mathcal{Z}$ can adaptively choose its inputs depending on protocol messages it received and send messages on behalf of a (corrupted) protocol party.

Let $\mathsf{Real}_{\Pi}^{\mathcal{A}}(\mathcal{Z})$ denote the random variable describing the output of $\mathcal{Z}$ when interacting with the real model, and let $\mathsf{Ideal}_{\mathcal{F}}^{\mathcal{S}}(\mathcal{Z})$ denote the random variable describing the output of $\mathcal{Z}$ when interacting with the ideal model. UC-security is defined as follows:

Definition 2.27. *A protocol* Π UC-realizes *a functionality* $\mathcal{F}$ *if for any (PPT) adversary* $\mathcal{A}$*, there exists a PPT simulator* $\mathcal{S}$ *such that for any (PPT) environment* $\mathcal{Z}$*,*

$$\mathsf{Real}_{\Pi}^{\mathcal{A}}(\mathcal{Z}) \approx \mathsf{Ideal}_{\mathcal{F}}^{\mathcal{S}}(\mathcal{Z}).$$

Indistinguishability can be both statistically or computationally, so it is unspecified in the above definition. In this thesis, we only consider static corruption, i.e. $\mathcal{Z}$ is allowed to corrupt parties before the begin of the protocol. Similar to general multi-party security definitions, one can also investigate adaptive corruption where parties are adaptively corrupted during the protocol execution.

Let us briefly discuss how the protocol execution is modeled, because this is important when we model our ideal functionalities. In UC, all entities are modeled as interactive Turing machines. The environment is first invoked with an input z, and then sends a message to another party. This party is activated, processes the input and sends an output to another party (according to the protocol), which is then activated, and so on. If no message is sent by a party, $\mathcal{Z}$ is activated again.

The defining property of UC is its composability. Given two UC-secure protocols Π and Ψ, where Ψ UC-realizes $\mathcal{F}$ and Π accesses Ψ as a subprotocol, the universal composition theorem states that the security of Π remains intact. Π is said to run in the $\mathcal{F}$-hybrid model. This allows for a modular construction of protocols.

Theorem ([Can01])**.** *Let* $\mathcal{F}$ *be an ideal functionality and let* Ψ *be a protocol that UC-realizes* $\mathcal{F}$*. Then* Π^{Ψ} *UC-realizes* $\Pi^{\mathcal{F}}$*.*

Typically, the later described protocols realize some functionality while having access to an ideal token functionality. Our protocols are thus stated in a token-hybrid model.

2.7.2 Common Ideal Functionalities

Most ideal functionalities allow an adversary to schedule message delivery. In the rest of this thesis, unless explicitly stated otherwise, we omit these messages when describing the simulator to simplify the exposition. We now present common ideal functionalities for primitives that we will later realize.

2.7.2.1 Ideal Functionality for a Single Commitment

The ideal commitment functionality defined in Figure 2.1 models the properties we require from a commitment. The binding property is modeled by the fact that the functionality will only accept one `commit`-message. Obviously, the receiver has no possibility to learn the input before the unveil phase, thus the commitment is also hiding.

Functionality $\mathcal{F}_{\text{COM}}$

Implicitly parametrized by a domain of secrets S.

Commit phase:

1. Await an input $(\texttt{commit}, s)$ with $s \in S$ from the sender. Store s, send $(\texttt{committed})$ to the adversary and ignore any further `commit`-messages.
2. Await a message $(\texttt{notify})$ from the adversary. Then send $(\texttt{committed})$ to the receiver.

Unveil phase:

3. Await an input $(\texttt{unveil}, \hat{s})$ with $\hat{s} \in S$ from the sender. Then, store $\hat{s}$ and send $(\texttt{opened})$ to the adversary.
4. Await a message $(\texttt{output})$ from the adversary. Then, if $\hat{s} = s$, send $(\texttt{opened}, \hat{s})$ to the receiver; otherwise, send a special reject message $\perp$.

Figure 2.1: Ideal functionality for commitments.

2.7.2.2 Ideal Functionality for Zero-Knowledge

Let $\mathcal{R}_{\mathcal{L}}$ denote the witness relation for an $\mathcal{NP}$-language $\mathcal{L}$, i.e. a statement x lies in $\mathcal{L}$ if there exists a witness w such that $(x, w) \in \mathcal{R}_{\mathcal{L}}$. The ideal functionality for zero-knowledge in Figure 2.2 basically checks if the prover knows a correct witness. This implies soundness, and since the functionality only outputs one bit to the receiver, it is also zero-knowledge. To prove UC-security for a ZK protocol, this means that the simulator has to extract the witness.

Functionality $\mathcal{F}_{\text{ZK}}$

Implicitly parametrized with an $\mathcal{NP}$-language $\mathcal{L}$ and a corresponding $\mathcal{NP}$ problem instance x.

1. Await an input $(\texttt{witness}, w)$ from the sender. Store w and send $(\texttt{sent})$ to the adversary.
2. Await a message $(\texttt{verify})$ from the adversary. If $(x, w) \in \mathcal{R}_{\mathcal{L}}$, send $(\texttt{accept})$ to the verifier; else send $(\texttt{reject})$.

Figure 2.2: Ideal functionality for zero-knowledge proofs.

2.7.2.3 Ideal Functionality for Multiple Oblivious Transfer

For oblivious transfer, the ideal functionality in Figure 2.3 has to model two properties. On the one hand, the receiver privacy has to hold, i.e. the sender must not learn the choice bit of the receiver. This is ensured by our definition. On the other hand, the receiver should only learn one of the sender's inputs. This is also captured by our definition, because after the first input of the choice bit, a second input leads to an abort. Our definition captures the more complex case where the

sender directly inputs two vectors of inputs, and the receiver inputs a vector of choice bits.

Functionality $\mathcal{F}_{\mathrm{OT}}$

Implicitly parametrized by a sender input domain S and the number n of single OTs to be implemented. The variable *state* is initialized with wait.

Creation phase:

1. Await an input $(\texttt{create}, (s_0^{(1)}, s_1^{(1)}), \ldots, (s_0^{(n)}, s_1^{(n)}))$ with $(s_0^{(i)}, s_1^{(i)}) \in S \times S$ from the sender and store $(s_0^{(1)}, s_1^{(1)}), \ldots, (s_0^{(n)}, s_1^{(n)})$. Set *state* = sent, store s_0 and s_1 and send (created) to the adversary.
2. Await a message (delivery) from the adversary. Check if *state* = sent and abort if otherwise. Set *state* = ready and send (ready) to the receiver.

Choice phase:

3. Await an input $(\texttt{choice}, c)$ with $c = (c_1, \ldots, c_n) \in \{0,1\}^n$ from the receiver and store c. If *state* = ready, set *state* = done and send (chosen) to the adversary, otherwise abort.
4. Await an input (deliver) from the adversary. Send $(\texttt{output}, s_c)$ to the receiver.

Figure 2.3: Ideal functionality for multiple oblivious transfer.

In comparison to our definition of a weak one-time memory (cf. Section 4.2.3.2), the adversarial sender might change his inputs before the choice bits are fixed.

3. Stateful Tamper-Proof Hardware

This chapter provides a survey of results from the literature and the current state of the art concerning UC-secure protocols based on stateful tamper-proof hardware. It is meant to serve as an introduction to the topic of tamper-proof hardware in the UC-framework and does not contain an original contribution by the author.

Stateful tamper-proof hardware in the sense that we are interested in was first proposed by Katz [Kat07]. In this seminal work, he provides a formalization of a tamper-proof hardware token in the UC-framework and then shows that UC-secure two-party computation is possible using the hardware tokens as a setup assumption. The hardware token formalization introduced in [Kat07], as shown in Figure 3.1, is modeled as a wrapper functionality that can store a Turing machine. Thus, the sender of a token can create a program M and store it inside the token. The receiver can query the wrapper functionality with an input upon which the wrapper evaluates the stored program on the input and returns the output to the receiver. The stored program is allowed to keep a state by storing state information and reading it upon invocation.

Obviously, $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$ models the two major properties that are required from tamper-proof hardware: on the one hand, the token sender can only store a program in the token and has no further means to communicate with the token. Thus the isolation assumption holds. On the other hand, the token program is hidden from the receiver, who only has black-box access to the functionality. This guarantees (perfect) tamper-proofness. The wrapper functionality in Figure 3.1 differs slightly from the original definition of [Kat07] in the sense that we only consider the two-party case and do not consider tokens sent in two directions. Additionally, here and throughout the rest of this work, we omit all session and party identifiers to allow for a more concise presentation.

[Kat07] showed how to construct UC-secure commitments by exchanging 2 tokens bidirectionally. Based on UC-secure commitments, general UC-secure computation is feasible, e.g. via Canetti et al. [CLOS02]. This spawned a series of works where a wide variety of cryptographic protocols were realized based on untrusted stateful tamper-proof hardware, both with computational security (cf. Table 3.2) and statistical security (cf. Table 3.3). With stateful tamper-proof hardware tokens at their

Functionality $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$

Implicitly parametrized by a security parameter κ.

Creation:

1. Await an input $(\texttt{create}, \mathsf{M}, t)$ from the token issuer, where M is a deterministic Turing machine and $t \in \mathbb{N}$. Store (M, t) and send $(\texttt{created})$ to the adversary. Ignore all further `create`-messages.

2. Await a message $(\texttt{delivery})$ from the adversary. Then, send $(\texttt{ready})$ to the token receiver.

Execution:

3. Await an input $(\texttt{run}, w)$ from the receiver. If no `create`-message has been sent, return a special symbol $\perp$. Otherwise run M on w from its most recent state. When M halts without generating output or t steps have passed, send $\perp$ to the receiver; otherwise store the current state of M and send the output of M to the receiver.

Figure 3.1: The wrapper functionality by which we model stateful tamper-proof hardware. The runtime bound t is merely needed to prevent malicious token senders from providing a perpetually running program code M; it will be omitted throughout the rest of the chapter.

	interactive				non-interactive
	[Kat07]	[DNW09]	[JKSS10]	[FPS+11]	[GKR08]
Functionality	Com.	Com.	SFE	Set Intersection	OTP[1]
Tokens	2 (bidir.)	2 (bidir.)	1	1	$\Theta(\kappa)$
Rounds	$\Theta(1)$	$\Theta(1)$	$\Theta(\kappa)$	$\Theta(1)$	$\Theta(\kappa)$
Assumption	DDH	Dense PK	RO	SPRP[2]	OTM & OWF
Security	UC	UC	stand-alone	stand-alone	stand-alone

Table 3.2: Overview of computationally secure (non-)interactive two-party protocols from untrusted stateful hardware.

disposal that contain a one-time memory (OTM), Goldwasser et al. [GKR08] show that even non-interactive secure computation is possible. Damgård et al. [DNW09] present a protocol that UC-realizes a commitment, but they do not require specific number-theoretic assumptions. They actually consider a somewhat different hardware model, where the token is not completely isolated (cf. Chapter 4). There is also a protocol by Järvinen et al. [JKSS10] that directly uses untrusted tamper-proof hardware to improve the efficiency of secure function evaluation (SFE) protocols. Another functionality that has many applications in practice is set intersection, which was realized via untrusted tamper-proof hardware by Fischlin et al. [FPS+11], as an improvement over Hazay and Lindell [HL08], who require trusted tokens.

As it turns out, stateful tamper-proof hardware also allows for statistically UC-secure protocols, which are impossible in the plain model even with stand-alone security. Moran and Segev [MS08] created a UC-secure commitment, where only one

[1] One-time programs.

[2] Strong pseudorandom permutation (e.g. AES is believed to be a SPRP).

	interactive			non-interactive	
	[MS08]	[GIS$^+$10]	[DKMQ11]	[GIS$^+$10]	[DKMQ11]
Functionality	Com.	OTM	OTM	OTM	OTM
Tokens	2 (bidir.)	$\Theta(\kappa)$	1	$\Theta(\kappa)$	2
Rounds	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Security	UC	UC	UC	UC	UC

Table 3.3: Overview of statistically secure (non-)interactive two-party protocols from stateful hardware.

party has to be able to create and send tokens. This setting is sometimes referred to as "David and Goliath" because of the obvious asymmetry concerning the ability to creating a token. Compared to previous UC-secure protocols, even those that use computational assumptions, this is an improvement. The result of [MS08] was later improved by Goyal et al. [GIS$^+$10], who showed how to realize OTM from several hardware tokens. Döttling et al. [DKMQ11] further improved the result such that only one token has to be sent to realize OTM, or two tokens if the interactive setup phase of the protocol is executed with a second token. They also show that this is optimal with respect to the number of tokens.

We want to shortly discuss the techniques that are used to obtain efficient UC-secure protocols based on stateful tamper-proof hardware. These techniques can also be partially applied for other hardware models, but independently of that, they are helpful for understanding some of the restrictions that apply to the other models. In the UC-framework, the simulator simulates the wrapper functionality for the corrupted party. This means that the simulator obtains the token code from a corrupted sender. This is the lever that the simulator can use to extract the senders input, because in contrast to an honest receiver, he can rewind the token functionality and execute it with different inputs. Against a corrupted receiver, the simulation of the wrapper functionality allows the simulator to observe all queries that the malicious receiver sends to the token. If necessary, the simulator can even adaptively change the behavior of the simulated token.

One scenario that we do not consider in this thesis is token wrapping. This describes the ability of a party to take a token that it received and embed it into a new token. On the one hand, token wrapping poses a thread to the security of protocols, as an adversary can encapsulate an intercepted token and obtain the queries of a receiver via the embedding token. It also renders the above mentioned proof technique useless, since the adversary does not know the code of the token it intercepts and thus the simulator will also not learn it. On the other hand, token wrapping allows protocols that are otherwise impossible to achieve with certain hardware tokens [GIMS10]. However, we believe that this scenario is not very realistic, as a wrapping attack can be prevented e.g. by physically fingerprinting a token. Honest token wrapping seems to require capabilities of the parties that we deem to be unrealistic.

Towards the goal of realizing (non-)interactive two-party computation, we can thus summarize that in the setting of computational security as well as in the setting of statistical security, both interactive and non-interactive general two-party computation is possible. We thus add the results to Table 3.4 to allow a comparison between all models that we discuss in this work. Please note that the optimal results concerning statistical security also imply optimal results in the computational

setting.

Model	computational 2PC		statistical 2PC	
	interactive	non-interactive	interactive	non-interactive
stateful	✓(1 token)	✓(2 tokens)	✓(1 token)	✓(2 tokens)
stateful (inc. com.)	✓	✓	✓	✓
stateful (outg. com.)	✓	✓	✗(✓)	✗
resettable	✓	✓	✗(✓)	✗
bounded-resettable	✓	✓	✓	(✓)
reusable resettable	✓	✓	✗(✓)	✗

Table 3.4: Feasibility of interactive and non-interactive two-party computation from stateful tamper-proof hardware.

4. Partially Isolated Stateful Tamper-Proof Hardware

In this chapter, we consider a weakened model of stateful tamper-proof hardware where a channel between the token and the token sender exists. All prior works on tamper-proof hardware have in common that the tokens are assumed to be completely isolated from their creator. In light of recent events, this assumption becomes questionable at the least, apart from the fact that maliciously created tokens could contain internal clocks, which can be exploited in conjunction with the activation time to send information into the device (or to make the abort behavior dependent on the activation time, which is not modeled in the UC-framework). This problem was already identified by Döttling et al. [DKMQ12], but left as an open problem.

We stress that the problem of a communication channel between the token and its sender lies skew to the well researched problems of leakage resilience [MR04, DP08, ADW09] and side-channel attacks on tamper-proof hardware, e.g. as considered in [CKM11, BCG$^+$11, PSW14] on the theoretical side and in e.g. [KJJ99, GMO01, SA03, Koc96] on the practical side, where a malicious token receiver tries to extract some of the contents of the token. In that scenario, the *tamper-resilience assumption* is weakened. In contrast, we consider a weakened *isolation assumption*. Damgård et al. [DNW08, DNW09] study a related scenario where the parties are only partially physically isolated, such that a low bandwidth covert channel remains. This model can be seen as a weaker variant of the two-prover proof systems [BOGKW88], and they show that based on computational assumptions, even partial physical separation allows UC-secure multi-party computation. The problem of communicating tokens was also identified by Rührmair and van Dijk [RvD13] in the context of physically uncloneable functions (PUFs), but they do not provide a protocol solution to this problem. Instead, they propose to use physical means to prevent the token from communicating with the sender.

Our contribution. In the following, we investigate the feasibility of (non-)interactive two-party computation in the scenario where an *unrestricted* channel between the token and its creator is considered. Of course, we have to introduce an additional restriction, because allowing unrestricted communication in both directions between the token and its creator obviously renders the token useless as a setup assumption. In essence, the model would collapse down to the stand-alone

setting, and from [CF01] we know that UC-secure computation is impossible in this case. Thus, there remain two different scenarios with unidirectional communication that can be considered to weaken the isolation assumption: either the tokens' creator can send messages to the tokens, or the tokens can send messages to their creator. While we deem the first case to be more realistic, we consider both cases. We emphasize that these one-way channels are available only for malicious parties and thus cannot be used by the honest parties during the protocol execution. The techniques that we use for our protocols are completely different from the ones used by Damgård et al. [DNW09], mainly due to our focus on statistical security and one-way communication.

We provide a broad characterization from a feasibility standpoint for both malicious incoming and outgoing communication between the tokens and their creator, with particular focus on one-time memory by Goldwasser et al. [GKR08]. OTM allows non-interactive two-party computation and is thus a very strong primitive. For our solutions, only one party has to be able to create hardware tokens. Let us briefly describe our main observations.

Communication from sender to token: In this scenario, the sender can completely change the behavior of the token at any point during the protocol run. Nevertheless, he can only adaptively change the token behavior depending on the information he learns. Thus, if we minimize or even remove any interaction between the sender and the receiver, the additional power of the sender boils down to changing the abort behavior of the token (independently of the actual protocol run). We present protocols for OT and OTM that make use of the fact that the sender does not learn enough information to maliciously influence the protocol run. Additionally, we give lower bounds for the number of tokens that are necessary to realize OTM in this setting, which confirms the optimality of our results with respect to the number of tokens used.

Communication from token to sender: This scenario is more restrictive than communication from the sender to the token, because the main idea of using hardware tokens in protocols is that the token cannot relay the queries from the receiver to the sender. This constraint is reflected by our results: it is impossible to realize statistically secure OT and OTM in such a setting. Nevertheless, we show how to realize a statistically UC-secure commitment in this setting and also give a construction of a computationally UC-secure (weak) OTM from a single token.

A preliminary version of these results was published in Dowsley et al. [DMQN15].

Structure of this chapter. The chapter is split into two parts. In Section 4.1, we consider the case of communication from the token sender to the token, while in Section 4.2, communication from the token to the token sender is examined. In both cases, we first formally define the model, study limitations and then present protocols for OTM. For each model, we also describe how our results relate to (non-)interactive two-party computation.

4.1 Communication from Token Issuer to Token

Incoming communication into tamper-proof hardware is a very strong security risk, and most protocols in the literature would completely loose their security guar-

antees if such communication were possible. This is mainly due to the fact that the model of tamper-proof hardware was specifically created to have an additional *isolated* party that can be used for interaction.

In real applications, however, there are many ways to send information into the hardware, e.g. using clocks, a powerful radio signal or modified data packages. While it is possible to prevent some forms of communication, e.g. by using a Faraday cage, in most real life scenarios, it seems infeasible to prevent all forms of communication. We consider it even difficult to guarantee a bound for the maximal amount of information that is transferred, and thus do not require any for our solutions.

Our results give a detailed characterization concerning the feasibility of OT(M) with and without computational assumptions in this scenario:

- Impossibility of statistically secure OTM from a single stateful hardware token.
- Statistically secure OTM from two stateful hardware tokens.
- Statistically secure OT from a single stateful hardware token.
- Computationally secure OTM from a single stateful hardware token.

Statistically secure OT and OTM can be obtained by adapting the existing protocol of Döttling et al. [DKMQ11] to the scenario of incoming communication. The computationally secure OTM protocol basically consists of a statistically UC-secure commitment scheme, which is then used to create a common reference string (CRS). Existing CRS-based UC-secure protocols, e.g. the commitment scheme from [CF01] and the OT of [PVW08], can be combined to realize an OTM from a single stateful token.

This section is structured as follows. In Section 4.1.1, we define an enhanced wrapper functionality that models incoming communication into the token. We then show our impossibility result in Section 4.1.2, which affirms the optimality of the main feasibility results concerning OTM that are presented in Section 4.1.3. A summarizing conclusion is given in Section 4.1.4.

4.1.1 Model

We modify the stateful token functionality $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$ such that it additionally allows the token issuer to send messages to the token (and thereby potentially modify the functionality). The functionality $\mathcal{F}_{\text{wrap}}^{\text{s-inc}}$ as shown in Figure 4.1 has an additional interface for a corrupted sender, which takes as input a message and executes the stored token program on this message. Note that we do not put a restriction on the length of the message that the sender can send (apart from being polynomial-size, otherwise the stored program cannot evaluate it).

While at first sight, our formalization might seem to put restrictions on the allowed modifications of the token program, the generality of this approach is guaranteed by e.g. having the sender input a universal Turing machine, and each update will then include a completely new program that is saved as the state.

4.1.2 Limitations

The first major restriction in the setting of incoming communication is that the token issuer can always switch off the token at any point in time. Thus, each functionality that can be realized in this model must necessarily be adapted to emulate

Functionality $\mathcal{F}_{\text{wrap}}^{\text{s-inc}}$

Implicitly parametrized by a security parameter κ.

Creation:

1. Await an input $(\texttt{create}, \mathsf{M}, t)$ from the token issuer, where M is a deterministic Turing machine and $t \in \mathbb{N}$. Store (M, t) and send $(\texttt{created})$ to the adversary. Ignore all further `create`-messages.

2. Await a message $(\texttt{delivery})$ from the adversary. Then, send $(\texttt{ready})$ to the token receiver.

Execution:

3. Await an input $(\texttt{run}, w)$ from the receiver. If no `create`-message has been sent, return a special symbol $\perp$. Otherwise run M on w from its most recent state. When M halts without generating output or t steps have passed, send $\perp$ to the receiver; otherwise store the current state of M and send the output of M to the receiver.

Update (adversarial sender only):

4. Upon receiving a message $(\texttt{update}, v)$ from a corrupted sender, run M on v for at most t steps. Save the current state of M.

Figure 4.1: The wrapper functionality by which we model stateful tamper-proof hardware that allows communication from the token issuer to the token. The runtime bound t is merely needed to prevent malicious token senders from providing a perpetually running program code M; it will be omitted throughout the rest of the section.

this behavior. From a feasibility standpoint, however, this does not necessarily rule out any functionality; the realized functionality is just a bit weaker than without the switch.

As we will see, there is nonetheless a non-trivial impossibility for the feasibility of OTM from a single stateful token, if the sender is allowed to send messages to the token. Intuitively, the reason for this is that the definition of OTM requires the sender inputs to be fixed at a certain point, and from that moment on, the receiver can decide when to query the OTM with his input. Through the communication channel, however, the sender can always change his inputs, even at a time when the inputs are supposed to be fixed.

Lemma 4.1. *There is no protocol Π_{OTM} using a single token that UC-realizes $\mathcal{F}_{\text{OTM}}$ in the $\mathcal{F}_{\text{wrap}}^{\text{s-inc}}$-hybrid model with statistical security.*

Proof. For the sake of contradiction assume that a correct and statistically secure OTM protocol Π_{OTM} from a single stateful token exists. Assume further that the parties' inputs are chosen as $s_0, s_1 \leftarrow \{0,1\}^\kappa$ and $c \leftarrow \{0,1\}$ uniformly at random. Further note that Döttling et al. [DKMQ11] show that OTM from a single stateful token requires interaction between the sender and the receiver.

Assume that the sender's privacy of the OTM protocol holds, i.e.

$$\begin{aligned} I(\mathsf{view}_\mathsf{R}; s_{1-c}) \leq \mathsf{negl}(\kappa) &\Leftrightarrow H_\infty(s_{1-c}) - H_\infty(s_{1-c}|\mathsf{view}_\mathsf{R}) \leq \mathsf{negl}(\kappa) \\ &\Leftrightarrow H_\infty(s_{1-c}|\mathsf{view}_\mathsf{R}) \geq \kappa - \mathsf{negl}(\kappa), \end{aligned}$$

where view_R is receiver's view of the protocol execution and $\mathsf{negl}(\kappa)$ is a function that is negligible in the security parameter.

By definition of the OTM functionality, the receiver can choose his input c at any point in time after he receives the token and the sender should not learn when the receiver inputs his choice bit into the OTM functionality. The receiver can choose his input c at a point in the future after receiving the token, when all initial communication between the parties is already finished, and then he interacts with the token to receive s_c.

Note that at the moment right before the receiver makes his choice c, the entropy of the inputs is still 1 from the point of view of all parties. Therefore, due to the sender's privacy, it holds that

$$H_\infty(s_0|\mathsf{view}_\mathsf{R}{}') \geq \kappa - \mathsf{negl}(\kappa)$$

and

$$H_\infty(s_1|\mathsf{view}_\mathsf{R}{}') \geq \kappa - \mathsf{negl}(\kappa),$$

where $\mathsf{view}_\mathsf{R}{}'$ is the receiver's view of the protocol execution until this point. Here, a malicious sender can use the channel into the token. He can forward his complete view to the token, which has to exist due to the impossibility of OTM from a single token without interaction as shown by [DKMQ11]. The token then gets to know all protocol interactions so far and due to the correctness of the OTM protocol (i.e., it works for any pair of inputs in $\{0,1\}^\kappa$), it is able, for almost any $s'_c \in \{0,1\}^\kappa$, to find a strategy to follow for the rest of the protocol that makes the receiver accept s'_c. Hence the values s_0 and s_1 are not fixed up to the point where the receiver inputs c. In the OTM functionality, however, the values s_0 and s_1 are fixed once the token is sent to the receiver, and thus we get a contradiction. □

In the case of a completely non-interactive protocol where the sender just issues two or more tokens to the receiver, the impossibility obviously no longer holds. This is the starting point for our studies in the following section.

4.1.3 Protocols

In this section, we present our results concerning the feasibility of OTM and OT in the scenario where the token issuer can relay messages to the token. As shown in Section 4.1.2, we cannot hope to achieve unconditionally secure non-interactive two-party computation from a single token, and also the sender can arbitrarily change the abort behavior. This definition of $\mathcal{F}_{\text{OTM}}^{\text{w-a}}$ in Figure 4.2 takes this into account.

The sender sends his two inputs s_0 and s_1 into the functionality, and the adversary is allowed to schedule the confirmation message (`ready`) to the receiver. After receiving the confirmation message, the OTM is considered delivered and the receiver can input his choice-bit c at any time. A malicious sender can use the overwrite interface to later change his inputs, but only up to the point when the OTM is

Functionality $\mathcal{F}_{\text{OTM}}^{\text{w-a}}$

Implicitly parametrized by a security parameter κ. The variable *state* is initialized with wait, the flag *abort* is set to 0. All messages apart from (`switch_on`) are ignored if $abort = 1$.

Delivery phase:

1. Await an input $(\texttt{create}, s_0, s_1)$ from the sender. Check if $state = \mathsf{wait}$ and $s_0, s_1 \in \{0,1\}^\kappa$, else abort. Set $state = \mathsf{sent}$, store s_0 and s_1 and send (`created`) to the adversary.
2. Await a message (`delivery`) from the adversary. Check if $state = \mathsf{sent}$ and abort if otherwise. Set $state = \mathsf{ready}$ and send (`ready`) to the receiver.

Choice phase:

3. Await an input $(\texttt{choice}, c)$ from the receiver. If $state = \mathsf{ready}$, set $state = \mathsf{done}$ and send $(\texttt{output}, s_c)$ to the receiver.

Overwrite (adversarial sender only):

4. Upon receiving a message $(\texttt{overwrite}, s_0', s_1')$ from the adversary, verify that $state = \mathsf{sent}$ and $s_0', s_1' \in \{0,1\}^\kappa$, otherwise abort. Set $s_0 = s_0'$ and $s_1 = s_1'$.
5. Upon receiving a message (`switch_on`) from the adversary, set $abort = 0$.
6. Upon receiving a message (`switch_off`) from the adversary, set $abort = 1$.

Figure 4.2: Ideal functionality for One-Time Memory with abort.

delivered, i.e. the state is set to ready. Independently of this, a malicious sender can always switch the functionality on or off.

Our starting point in Section 4.1.3.1 is using two tokens to obtain unconditionally secure OTM as defined above. In Section 4.1.3.2 we move on to sketch how to obtain unconditionally secure OT from a single token using the first result, and finally we present a computationally secure OTM protocol in Section 4.1.3.3 using only a single hardware token, thus giving a full characterization of the feasibility of OTM in the $\mathcal{F}_{\text{wrap}}^{\text{s-inc}}$-hybrid model.

4.1.3.1 Unconditionally Secure OTM from Two Tokens

We claim that the non-interactive version of an OTM protocol by Döttling et al. [DKMQ11] remains secure even in the case where the tokens can receive information from the token issuer. For completeness, the protocol is shown in Figure 4.3. The token sender programs two stateful tokens and sends them to the receiver, who then interacts with them to perform an OTM.

In more detail, one token serves as a commitment to the functionality (and thus the inputs) of the other token. It outputs the one-time-pad encrypted parameters $\tilde{a}, \tilde{B}$ of an affine function, which is stored on the second token. Additionally, it outputs the encrypted inputs of the sender $\tilde{s}_0, \tilde{s}_1$. Using his own input, the receiver can obtain a decryption key for one of these values from the second token and verify its correctness with the encrypted token functionality.

The OTM is considered delivered after the receiver obtains the encrypted parameters and the encrypted inputs. From that point on, the receiver can query the second token to obtain the output s_c whenever he wishes.

Protocol $\Pi^{\text{w-a}}_{\text{sOTM}}$

Let $\mathcal{T}_{rnd}$ and $\mathcal{T}_{inp}$ be two $\mathcal{F}^{\text{s-inc}}_{\text{wrap}}$ instances.

Setup phase:

1. Sender: Let $s_0, s_1 \in \{0,1\}^\kappa$ be the sender's inputs. Sample a matrix $B \leftarrow \mathbb{F}_2^{2\kappa \times 2\kappa}$ and a vector $a \leftarrow \mathbb{F}_2^{2\kappa}$ uniformly at random. Program two token functionalities T_{rnd} and T_{inp} as follows.
   - T_{rnd}: Set $ready = 1$.
     - Upon receiving a message $(\texttt{choice}, z)$ from the receiver, check if $ready = 1$ and abort otherwise. Set $ready = 0$, compute the matrix $V = az^T + B$ and send V to the receiver.
   - T_{inp}: Set $state = \mathsf{ready}$.
     - Upon receiving a message $(\texttt{matrix\_choice}, C)$ from the receiver, check if $state = \mathsf{ready}$ and $C \in \mathbb{F}_2^{\kappa \times 2\kappa}$, if not abort. Compute a matrix $G \in \mathbb{F}_2^{\kappa \times 2\kappa}$ that is complementary to C[a]. Set $\tilde{a} = Ca$, $\tilde{B} = CB$ and $state = \mathsf{committed}$. Send $(\texttt{committed}, G, \tilde{a}, \tilde{B})$ to the receiver.
     - Upon receiving a message $(\texttt{vector\_choice}, h)$ from the receiver, check if $state = \mathsf{committed}$ and $h \in \mathbb{F}_2^{2\kappa} \setminus \{0\}$, otherwise abort. Compute $\tilde{s}_0 = s_0 + GBh$ and $\tilde{s}_1 = s_1 + GBh + Ga$, set $state = \texttt{done}$ and send $(\texttt{output}, \tilde{s}_0, \tilde{s}_1)$ to the receiver.

   Send $(\texttt{create}, \mathsf{T}_{rnd})$ to $\mathcal{T}_{inp}$ and $(\texttt{create}, \mathsf{T}_{inp})$ to $\mathcal{T}_{inp}$.

Delivery phase:

2. Receiver: Let c be the receiver's input.
   - Choose a random matrix $C \leftarrow \mathbb{F}_2^{\kappa \times 2\kappa}$ and send $(\texttt{matrix\_choice}, C)$ to $\mathcal{T}_{inp}$. Let $(\texttt{committed}, G, \tilde{a}, \tilde{B})$ be the answer from $\mathcal{T}_{inp}$.
   - Choose a random vector $h \leftarrow \mathbb{F}_2^{2\kappa} \setminus \{0\}$ and send $(\texttt{vector\_choice}, h)$ to $\mathcal{T}_{inp}$. Let $(\texttt{output}, \tilde{s}_0, \tilde{s}_1)$ be the output.

Choice phase:

3. Receiver:
   - Choose $z \leftarrow \mathbb{F}_2^{2\kappa}$ uniformly at random subject to $z^T h = c$ and send $(\texttt{choice}, z)$ to $\mathcal{T}_{rnd}$. Let $(\texttt{output}, V)$ be the result.
   - Check if $CV = \tilde{a}z^T + \tilde{B}$. If yes, output $s_c = \tilde{s}_c + GVh$, otherwise abort.

[a] G is determined by κ vectors of length 2κ that are linearly independent and G spans a subspace of the kernel of C.

Figure 4.3: Statistically UC-secure protocol realizing $\mathcal{F}^{\text{w-a}}_{\text{OTM}}$ in the $\mathcal{F}^{\text{s-inc}}_{\text{wrap}}$-hybrid model [DKMQ11].

The fact that the protocol securely realizes $\mathcal{F}^{\text{w-a}}_{\text{OTM}}$ follows from a straightforward modification of the original security proof by Döttling et al. [DKMQ11, Theorem 8.2],

who considered the same protocol but with isolated tokens, and proved that it realizes $\mathcal{F}_{\text{OTM}}$ (without aborts) in the $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$-hybrid model.

Theorem 4.1. *The protocol $\Pi_{\text{sOTM}}^{\text{w-a}}$ in Figure 4.3 statistically UC-realizes $\mathcal{F}_{\text{OTM}}^{\text{w-a}}$ (cf. Figure 4.2) in the $\mathcal{F}_{\text{wrap}}^{\text{s-inc}}$-hybrid model.*

Proof sketch. The correctness as well as the security against a corrupted receiver follow directly from the proof of security in [DKMQ11].

In the case of the security against a corrupted sender, note that the OTM is considered delivered at the point when the receiver has obtained $(G, \tilde{a}, \tilde{B}, \tilde{s_0}, \tilde{s_1})$ from $\mathcal{T}_{inp}$. From that point on, $\mathcal{T}_{inp}$ does not participate in the protocol anymore, and there is no way for the sender or $\mathcal{T}_{rnd}$ to learn the values C and h. Thus, when the receiver queries $\mathcal{T}_{rnd}$, the sender cannot cheat by sending any information to the token that might allow to unveil the committed values differently. Thus, the only difference to the original proof from [DKMQ11] is that the sender can change the abort behavior of the tokens. In order to allow the simulator to simulate this behavior correctly, it can just internally run the token program with its current state on random inputs and observe if it aborts. Based on this, the simulator can adapt the abort behavior of $\mathcal{F}_{\text{OTM}}^{\text{w-a}}$ by switching the functionality on or off. □

Remark. The protocol $\Pi_{\text{sOTM}}^{\text{w-a}}$ realizes only a single OTM, but it can be modified straightforwardly using the same techniques as in [DKMQ11] to allow an (a priori) bounded number of sequential OTMs.

4.1.3.2 Unconditionally Secure OT from a Single Token

As shown in Lemma 4.1, it is not possible to obtain OTM from a single token in the $\mathcal{F}_{\text{wrap}}^{\text{s-inc}}$-hybrid model. Still, a protocol for oblivious transfer is not ruled out, and we will briefly sketch how the protocol from Section 4.1.3.1 can be adapted to give an OT protocol using only a single token.

First, note that the original protocol for OT from [DKMQ11] using a single token, on which the solution in Section 4.1.3.1 is based, is not secure in the $\mathcal{F}_{\text{wrap}}^{\text{s-inc}}$-hybrid model. In their scenario, they only require the token $\mathcal{T}_{rnd}$ from the protocol $\Pi_{\text{sOTM}}^{\text{w-a}}$, and the functionality of $\mathcal{T}_{inp}$ is carried out as an interactive protocol between the sender and the receiver. In our scenario, if the sender sends the matrix C to the token using the communication channel, the token can suddenly break the binding property of the commitment and thus adaptively change the output of the protocol.

To obtain an OT protocol in the $\mathcal{F}_{\text{wrap}}^{\text{s-inc}}$-hybrid model, we simply propose to use the token $\mathcal{T}_{inp}$ during the protocol and perform the functionality of the token $\mathcal{T}_{rnd}$ in an interactive protocol. Obviously, this does not result in an OTM, since the sender learns when the receiver obtains his output, but it is still an OT. While this approach requires the OT inputs to be fixed in advance of the protocol, this can be overcome by using standard techniques, i.e. using random inputs for the OT and derandomizing them afterwards [Bea95]. The security proof follows the same argumentation as the proof of Theorem 4.1.

4.1.3.3 Computationally Secure OTM from a Single Token

To complete the picture of the feasibility of OTM in the $\mathcal{F}_{\text{wrap}}^{\text{s-inc}}$-hybrid model, we now show how to realize $\mathcal{F}_{\text{OTM}}^{\text{w-a}}$ from a single token with computational security.

Our protocol $\Pi^{\text{w-a}}_{\text{cOTM}}$ in Figure 4.4 on a high level realizes a statistically secure commitment using a hardware token, which is then used to compute a common reference string (CRS) between the token sender and the token receiver. The CRS is then used with another UC-secure commitment and a UC-secure OT to realize the $\mathcal{F}^{\text{w-a}}_{\text{OTM}}$ functionality.

Protocol $\Pi^{\text{w-a}}_{\text{cOTM}}$

Let SIG be an EUF-CMA-secure signature scheme and COM a UC-secure commitment scheme in the CRS-hybrid model. Further let OT be a UC-secure OT protocol in the CRS-hybrid model and $\mathcal{T}$ be an instance of $\mathcal{F}^{\text{s-inc}}_{\text{wrap}}$.

Setup phase:

1. Sender: Generate a key pair $(\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathsf{SIG.KeyGen}(\kappa)$ and choose $x \leftarrow \mathbb{F}_2^\kappa$ uniformly at random. Further choose $v_{j,0} \leftarrow \mathbb{F}_2^\kappa$ uniformly at random for $j \in \{1, \ldots, \kappa\}$ and set $v_{j,1} = x \oplus v_{j,0}$. Create a token program T that behaves as follows.
   - Upon receiving a message $(\texttt{challenge}, k)$ from the receiver, check if $\mathit{state} = \mathsf{ready}$ and $k \in \{0,1\}$, else abort. Set $j = j + 1$; if $j = \kappa$, set $\mathit{state} = \mathsf{m\_committed}$. Send $(\texttt{commit}, v_{k_j})$ to the receiver.
   - Upon receiving a message $(\texttt{commit}, \mathsf{CRS}, \sigma)$ from the receiver, check if $\mathit{state} = \mathsf{m\_committed}$ and $\mathsf{SIG.Verify}(\mathsf{vk}, \mathsf{CRS}, \sigma) = 1$, otherwise abort. Set $\mathit{state} = \mathsf{i\_committed}$. Compute $(c_i, d_i) \leftarrow \mathsf{COM.Commit}(\mathsf{CRS}, s_i)$ for $i \in \{0,1\}$ and send $(\texttt{committed}, (c_0, c_1))$ to the receiver.
   - Upon receiving a message $(\texttt{output})$ from the receiver, check if $\mathit{state} = \mathsf{i\_committed}$, else abort. Execute the sender program of OT with input $(s_0 \| d_0, s_1 \| d_1)$ and CRS and forward the messages between the receiver and the sender program.

   Send $(\texttt{create}, \mathsf{T})$ to $\mathcal{T}$.

Delivery phase:

2. Receiver: Choose κ random bits $k_j \leftarrow \{0,1\}$, send $(\texttt{challenge}, k_j)$ to $\mathcal{T}$ for each $j \in \{1, \ldots, \kappa\}$ and obtain v_{j,k_j}. Choose $y \leftarrow \mathbb{F}_2^\kappa$ uniformly at random and send it to the sender.
3. Sender: Set $\mathsf{CRS} = x \oplus y$, compute $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, \mathsf{CRS})$ and send $(\{v_{j,0}, v_{j,1}\}_{j=1}^\kappa, \sigma, x)$ to the receiver.
4. Receiver: Check if $x = v_{j,0} \oplus v_{j,1}$ for all $j \in \{1, \ldots, \kappa\}$ and abort if that is not the case. Set $\mathsf{CRS} = x \oplus y$. Send $(\texttt{commit}, \mathsf{CRS}, \sigma)$ to $\mathcal{T}$ and let $(\texttt{committed}, (c_0, c_1))$ be the answer.

Choice phase:

5. Receiver: Let c be the receiver's input. Send $(\texttt{output})$ to $\mathcal{T}$ and execute the receiver program of OT with input c and CRS and forward the messages between $\mathcal{T}$ and the program. Let $s_c \| d_c$ be the output. Check if $\mathsf{COM.Open}(\mathsf{CRS}, s_c, c_c, d_c) = 1$, and output s_c if the check is passed.

Figure 4.4: Computationally UC-secure protocol realizing $\mathcal{F}^{\text{w-a}}_{\text{OTM}}$ in the $\mathcal{F}^{\text{s-inc}}_{\text{wrap}}$-hybrid model.

In more detail, the sender programs a token that is initialized with κ pairs of shares $(v_{j,0}, v_{j,1})$, such that $v_{j,0} \oplus v_{j,1} = x$ with x being the sender's input for the CRS. The token is sent to the receiver, who can query the token for one share of each pair. After that is done, the receiver sends his input y to the sender, who then computes the CRS as $\mathsf{CRS} = x \oplus y$ and additionally provides a signature σ on the CRS, such that the CRS can be verified by the token later. He sends $(\{\hat{v}_{j,0}, \hat{v}_{j,1}\}_{j=1}^{\kappa}, \sigma, x)$ to the receiver, who can verify that his shares coincide with the shares that the sender sent to him. For the simulation to go through, it is crucial that the simulator is able to extract the input of the adversarial party before giving his input, otherwise he cannot cheat in the rest of the protocol.

From that moment on the CRS is established, and the token is given CRS together with σ. Using this CRS, the token uses a UC-secure commitment COM to commit to the sender-inputs s_0, s_1 and sends the commitments to the receiver. At this point, the OTM is considered delivered to the receiver.

During the choice phase, the token and the receiver engage in a UC-secure OT protocol (again using the CRS), where the token uses as input the sender's input s_i and the corresponding unveil information d_i of the commitment c_i. Thus, the receiver obtains a value $s_c \| d_c$, which allows him to verify that the commitment actually contains the correct value.

Theorem 4.2. *The protocol $\Pi_{\mathrm{cOTM}}^{\text{w-a}}$ in Figure 4.4 computationally UC-realizes $\mathcal{F}_{\mathrm{OTM}}^{\text{w-a}}$ (cf. Figure 4.2) in the $\mathcal{F}_{\mathrm{wrap}}^{\text{s-inc}}$-hybrid model, given that CRS-based UC-secure OT exists.*

The proof of Theorem 4.2 is divided into two parts. Lemma 4.2 shows security against a corrupted sender, and Lemma 4.3 shows security against a corrupted receiver. The correctness of the protocol can be verified trivially, and CRS-based UC-secure OT was e.g. proposed by Peikert et al. [PVW08].

Lemma 4.2. *The protocol $\Pi_{\mathrm{cOTM}}^{\text{w-a}}$ in Figure 4.4 computationally UC-realizes $\mathcal{F}_{\mathrm{OTM}}^{\text{w-a}}$ (cf. Figure 4.2) in the $\mathcal{F}_{\mathrm{wrap}}^{\text{s-inc}}$-hybrid model against a corrupted sender, given that COM and OT are UC-secure against a corrupted sender.*

Proof. The simulator has to influence the creation of the CRS by obtaining x before it has to send y. Once that is accomplished, it can set the CRS such that it can extract the UC-commitments that are sent by the token. Thus, the simulator learns both inputs of the sender and can provide them to the ideal functionality. Let $\mathcal{A}_{\mathsf{S}}$ be the dummy adversary. We have to show that $\mathsf{Real}_{\Pi_{\mathrm{cOTM}}^{\text{w-a}}}^{\mathcal{A}_{\mathsf{S}}}(\mathcal{Z}) \underset{c}{\approx} \mathsf{Ideal}_{\mathcal{F}_{\mathrm{OTM}}^{\text{w-a}}}^{\mathcal{S}_{\mathsf{S}}}(\mathcal{Z})$ for any PPT-environment $\mathcal{Z}$. To that end, consider the simulator $\mathcal{S}_{\mathsf{S}}$ in Figure 4.5.

Indistinguishability of the simulation and a real protocol run follows by a simple hybrid argument.

Experiment 0: This is the real model.

Experiment 1: Identical to Experiment 0, except that $\mathcal{S}_1$ queries T^* to obtain all $v_{j,k}^*$, computes $\hat{x}$ and aborts if $\hat{x} \neq x^*$.

Experiment 2: Identical to Experiment 1, except that $\mathcal{S}_2$ chooses y as $\mathsf{CRS} \oplus x$ and uses the sender-simulators $\mathcal{S}_{\mathsf{S}}^{\mathsf{COM}}$ and $\mathcal{S}_{\mathsf{S}}^{\mathsf{OT}}$ for COM and OT to obtain $\hat{s}_b$ and $\hat{s}_b'$ for $b \in \{0,1\}$. It aborts if $\hat{s}_b' \neq \hat{s}_b$ for $b \in \{0,1\}$. This is the ideal model.

Simulator $\mathcal{S}_{\mathsf{S}}$

- Simulate $\mathcal{T}$ for $\mathcal{A}_{\mathsf{S}}$ and obtain T^*.
- (Setup) Simulated straightforwardly according to $\Pi^{\text{w-a}}_{\text{cOTM}}$.
- (Delivery) Proceed as follows:
  - In Step 2, execute the token program T^* with both messages $(\texttt{challenge}, 0)$ and $(\texttt{challenge}, 1)$ for each $j \in \{1, \ldots, \kappa\}$ by resetting the token to the appropriate state after each execution and obtain $v^*_{j,0}, v^*_{j,0}$ for each j. Compute the j messages $x^*_j = v^*_{j,0} \oplus v^*_{j,1}$ and set $\hat{x}$ as the value x^* that occurs the most. Set $\hat{y} = \mathsf{CRS} \oplus \hat{x}$, with CRS being a CRS that allows simulation of COM and OT, and send $\hat{y}$ to $\mathcal{A}_{\mathsf{S}}$.
  - Upon receiving a message $(\{v^*_{j,0}, v^*_{j,1}\}_{j=1}^{\kappa}, \sigma^*, x^*)$ in Step 3, check if $\hat{x} = x^*$ and $v^*_{j,0} \oplus v^*_{j,1} = x^*\ \forall j$, otherwise abort. Input $(\texttt{commit}, \mathsf{CRS}, \sigma^*)$ into T^* and obtain c^*_0, c^*_1. Start the sender-simulator $\mathcal{S}^{\mathsf{COM}}_{\mathsf{S}}$ to obtain the values $\hat{s}_0$ and $\hat{s}_1$ from c^*_0 and c^*_1, respectively. Start the sender-simulator for $\mathcal{S}^{\mathsf{OT}}_{\mathsf{S}}$ and obtain $\hat{s}'_0$ and $\hat{s}'_1$, abort if $\hat{s}_i \neq \hat{s}'_i$. Send $(\texttt{create}, \hat{s}_0, \hat{s}_1)$ to $\mathcal{F}^{\text{w-a}}_{\text{OTM}}$.
- (Choice) Simulated straightforwardly according to $\Pi^{\text{w-a}}_{\text{cOTM}}$.
- (Update) After each message m^* from $\mathcal{A}_{\mathsf{S}}$ to $\mathcal{T}$, run T^* on m^* and simulate a choice phase with T^* to determine the abort behavior. Send $(\texttt{switch\_on})$ or $(\texttt{switch\_off})$ to $\mathcal{F}^{\text{w-a}}_{\text{OTM}}$ accordingly.

Figure 4.5: Simulator against a corrupted sender in the protocol $\Pi^{\text{w-a}}_{\text{cOTM}}$.

Experiment 0 and Experiment 1 are indistinguishable since the simulator will extract the correct $\hat{x}$ with overwhelming probability. First note that if any of the values $v^*_{j,k} \neq \hat{v}_{j,k}$ or $v^*_{j,0} \oplus v^*_{j,1} \neq x$, then the receiver always aborts. Also, if the majority of the $\hat{v}_{j,k} = v^*_{j,k}$, then the extraction of $\hat{x}$ is correct. Thus the only remaining case is where for at least half of $j \in \{1, \ldots, \kappa\}$ it holds that $v_{j,0} = \hat{v}_{j,0}$ or $v_{j,1} = \hat{v}_{j,1}$, but not both. However, the probability that the opening check is passed is $\frac{1}{2}$, so the probability of opening $\left\lceil \frac{\kappa}{2} \right\rceil$ shares incorrectly without the receiver noticing is negligible in κ.

Experiments 1 and 2 are indistinguishable due to the UC-security of COM and OT. The sender-simulator extracts the inputs of the sender with overwhelming probability, so the simulator will not abort. A distinguishing environment $\mathcal{Z}$ can directly be used to break the UC-security of at least one of COM and OT. □

Lemma 4.3. *The protocol $\Pi^{\text{w-a}}_{\text{cOTM}}$ in Figure 4.4 computationally UC-realizes $\mathcal{F}^{\text{w-a}}_{\text{OTM}}$ (cf. Figure 4.2) in the $\mathcal{F}^{\text{s-inc}}_{\text{wrap}}$-hybrid model against a corrupted receiver, given that COM and OT are UC-secure against a corrupted receiver and SIG is EUF-CMA-secure.*

Proof. Let $\mathcal{A}_{\mathsf{R}}$ be the dummy adversary. We have to show that $\mathsf{Real}^{\mathcal{A}_{\mathsf{R}}}_{\Pi^{\text{w-a}}_{\text{cOTM}}}(\mathcal{Z}) \overset{c}{\approx} \mathsf{Ideal}^{\mathcal{S}_{\mathsf{R}}}_{\mathcal{F}^{\text{w-a}}_{\text{OTM}}}(\mathcal{Z})$ for any PPT-environment $\mathcal{Z}$.

Consider the simulator $\mathcal{S}_{\mathsf{R}}$ in Figure 4.6. In a first step, the simulator simulates $\mathcal{T}$ for $\mathcal{A}_{\mathsf{R}}$ and learns all queries. The simulator answers these queries with random values $\hat{v}_{j,k_j}$. Then, the adversary sends his input to the simulator, who adapts all

shares $\hat{v}_{j,1-k_j}$ to fit the CRS that $\mathcal{S}_\mathsf{R}$ needs for the simulation of the UC-secure protocols OT and COM.

Once that is accomplished, the simulator commits to random inputs $\hat{s}_0, \hat{s}_1$ and then obtains the choice bit c^* of $\mathcal{A}_\mathsf{R}$ using the CRS in the protocol OT. He inputs the choice-bit into $\mathcal{F}_{\text{OTM}}^{\text{w-a}}$ to obtain the real result and then again uses the CRS to equivocate the commitment corresponding to the choice bit to the real output.

Simulator $\mathcal{S}_\mathsf{R}$

- Simulate $\mathcal{T}$ for $\mathcal{A}_\mathsf{R}$.
- (Setup) Simulated straightforwardly according to $\Pi_{\text{cOTM}}^{\text{w-a}}$, without creating a token program T.
- (Delivery)
  - In Step 2, answer the κ challenges k_j with $\hat{v}_{j,k_j} \leftarrow \mathbb{F}_2^\kappa$ uniformly at random. Let y^* be the message from $\mathcal{A}_\mathsf{R}$. Choose $\hat{x} = \mathsf{CRS} \oplus y^*$ and set $\hat{v}_{j,1-k_j} = \hat{v}_{j,k_j} \oplus \hat{x}$.
  - Simulate Step 3 according to $\Pi_{\text{cOTM}}^{\text{w-a}}$, i.e. compute $\hat{\sigma} \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, \mathsf{CRS})$ and return $(\{\hat{v}_{j,0}, \hat{v}_{j,1}\}_{j=1}^{\kappa}, \hat{\sigma}, \hat{x})$.
  - Upon receiving a message $(\texttt{commit}, \mathsf{CRS}^*, \sigma^*)$ in Step 4, check if $\hat{\sigma} = \sigma^*$ and $\mathsf{SIG.Verify}(\mathsf{vk}, \mathsf{CRS}^*, \sigma^*) = 1$, otherwise abort. Proceed with the simulation as in $\Pi_{\text{cOTM}}^{\text{w-a}}$ for two random inputs $\hat{s}_0$ and $\hat{s}_1$.
- (Choice) Upon receiving a message ($\texttt{output}$) from $\mathcal{A}_\mathsf{R}$ in Step 5, start the receiver-simulator $\mathcal{S}_\mathsf{R}^\mathsf{OT}$ for OT and halt the simulation once $\mathcal{A}_\mathsf{R}$'s choice bit c^* is obtained. Input c^* into $\mathcal{F}_{\text{OTM}}^{\text{w-a}}$ and obtain the output s_{c^*}. Start the receiver-simulator $\mathcal{S}_\mathsf{R}^\mathsf{COM}$ of COM to compute the equivocation $\hat{d}_{c^*}$ of $\hat{c}_{c^*}$ to s_{c^*} and provide the receiver-simulator $\mathcal{S}_\mathsf{R}^\mathsf{OT}$ of OT with the input $\hat{s}_{c^*} \| \hat{d}_{c^*}$. Resume the simulation of $\mathcal{S}_\mathsf{R}^\mathsf{OT}$.

Figure 4.6: Simulator against a corrupted receiver in the protocol $\Pi_{\text{cOTM}}^{\text{w-a}}$.

Experiment 0: This is the real model.

Experiment 1: Identical to Experiment 0, except that $\mathcal{S}_1$ sets $\hat{x} = \mathsf{CRS} \oplus y^*$ and answers the queries for $(\hat{v}_{j,0}, \hat{v}_{j,1})$ accordingly.

Experiment 2: Identical to Experiment 1, except that $\mathcal{S}_2$ aborts if $\sigma^* \neq \hat{\sigma}$ and $\mathsf{SIG.Verify}(\mathsf{vk}, \mathsf{CRS}^*, \sigma^*) = 1$.

Experiment 3: Identical to Experiment 2, except that $\mathcal{S}_3$ starts the receiver-simulator $\mathcal{S}_\mathsf{R}^\mathsf{OT}$ to learn the choice bit.

Experiment 4: Identical to Experiment 3, except that $\mathcal{S}_4$ commits to uniformly random values $\hat{s}_0, \hat{s}_1$, starts the receiver-simulator $\mathcal{S}_\mathsf{R}^\mathsf{COM}$ and equivocates the commitment $\hat{c}_c^*$ to the value s_c^* obtained from $\mathcal{F}_{\text{OTM}}^{\text{w-a}}$.

From $\mathcal{Z}$'s view, Experiment 0 and Experiment 1 are identically distributed, since $\mathcal{Z}$ learns only one of the shares $v_{j,k}$ for each j and the shares are chosen uniformly at random. Experiments 1 and 2 are computationally indistinguishable given that SIG is EUF-CMA-secure, since $\mathcal{S}_2$ only aborts if the receiver sends a valid signature

to the token that $\mathcal{S}_2$ did not create. Experiment 2 and Experiment 3 are computationally indistinguishable by the UC-security of OT, i.e. a distinguishing $\mathcal{Z}$ can directly be used to break the UC-security of OT. The same argumentation holds for Experiment 3 and Experiment 4 and the UC-security of COM. □

Remark. Please note that the CRS can be used for several instances. Thus, this protocol can straightforwardly be enhanced to allow (using a PRF even an unbounded number of) sequential OTMs.

4.1.4 Relation to Two-Party Computation

Given our results, it seems very likely that all (completely) non-interactive protocols in the tamper-proof hardware model should in fact be secure even if the sender is allowed to send messages to the token. This is mainly due to the fact that the token does not need any additional information other than in the normal setting, while the sender will not receive any (possibly compromising) information that could be forwarded to the token. Nonetheless, we deem it to be interesting that even with interaction, some functionalities can be realized with tamper-proof hardware in this model. To summarize, we obtain the following results:

Statistically secure two-party computation: Our UC-secure OT protocol based on a single token from Section 4.1.3.2 can be combined with general feasibility results, e.g. [Kil88, IPS08], to obtain UC-secure two-party computation.

Statistically secure non-interactive two-party computation: Our adaption of the protocol of [DKMQ11] in Section 4.1.3.1 UC-realizes an OTM, which basically represents a non-interactive variant of OT. As above, this yields the claimed result. The impossibility from Section 4.1.2 shows that our solution is optimal with respect to the number of tokens used.

Computationally secure (non-interactive) two-party computation: In Section 4.1.3.3 we presented a protocol that realizes a computationally UC-secure OTM. Döttling et al. [DKMQ11] show that an OTM from a single token needs an interactive initialization phase. Thus, identically to the case of statistical security, to obtain a completely non-interactive solution, two tokens are necessary (e.g. by storing the sender functionality in the second token). Thus, our result for non-interactive UC-secure two-party computation is optimal with respect to the number of tokens used and of course directly implies computationally UC-secure two-party computation.

Our results are optimal with respect to the number of tokens used. In this section, we thus showed the results highlighted in Table 4.7.

4.2 Communication from Token to Token Issuer

Outgoing communication from a tamper-proof hardware token might seem to be a bit more unrealistic than incoming communication, because it can usually be assumed that the token is very limited in its ability to communicate messages to its creator. On the one hand, a token has only access to a weak power source and thus cannot send messages with high signal strength. On the other hand, it seems

Model	computational 2PC		statistical 2PC	
	interactive	non-interactive	interactive	non-interactive
stateful	✓	✓	✓	✓
stateful (inc. com.)	✓(1 Token)	✓(2 Tokens)	✓(1 Token)	✓(2 Tokens)
stateful (outg. com.)	✓	✓	✗(✓)	✗
resettable	✓	✓	✗(✓)	✗
bounded-resettable	✓	✓	✓	(✓)
reusable resettable	✓	✓	✗(✓)	✗

Table 4.7: Feasibility of interactive and non-interactive two-party computation from stateful tamper-proof hardware with communication from the sender to the token.

easier to shield the token itself such that any existing channels can be blocked. Nevertheless, communication cannot be ruled out in general and, as we will show, induces an ever greater security risk than incoming communication from a sender.

Our results show the following concering the feasibility of OTM:

- Impossibility of statistically secure OT and OTM.
- Statistically secure commitment from a single stateful token.
- Computationally secure OTM from a single stateful token.

The impossibility intuitively follows from the fact that for statistically secure OT and OTM, the model with outgoing communication essentially collapses to the plain model, because the sender obtains the complete view of the token. A statistically secure commitment is still possible, because the protocol can be designed such that the sender only learns critical information after he has opened the commitment. With computational assumptions, using the same techniques as in the case of incoming communication, a (weak) OTM can still be realized.

Similarly to Section 4.1, this section starts with a detailed description of the model in Section 4.2.1. We then show the limitations concering the feasibility of OTM in Section 4.2.2 and present our positive results in Section 4.2.3. In Section 4.2.4 we present our results in the context of two-party computation.

4.2.1 Model

In the scenario of outgoing communication from the token to the sender, we allow a malicious sender to construct a token that can send arbitrary messages to the sender. To model this in the UC-framework, we enhance the standard stateful wrapper functionality such that it provides an interface to the program M which allows M to send messages to the sender. We call this functionality $\mathcal{F}_{\text{wrap}}^{\text{s-out}}$ (cf. Figure 4.8).

Please note that this formalization is not completely in accordance with the principles of the UC-framework, in the sense that the token program is supposed to send two messages simultaneously. Normally, the sender would be invoked to process the

Functionality $\mathcal{F}_{\text{wrap}}^{\text{s-out}}$

Implicitly parametrized by a security parameter κ.

Creation:

1. Await an input $(\texttt{create}, \mathsf{M}, t)$ from the token issuer, where M is a deterministic Turing program and $t \in \mathbb{N}$. Store (M, t) and send $(\texttt{created})$ to the adversary. Ignore all further `create`-messages.

2. Await a message $(\texttt{delivery})$ from the adversary. Then, send $(\texttt{ready})$ to the token receiver.

Execution:

3. Await an input $(\texttt{run}, w)$ from the receiver. If no `create`-message has been sent, return a special symbol $\perp$. Otherwise run M on w. When M halts without generating output or t steps have passed, send $\perp$ to the receiver; otherwise store the current state of M and send the output of M to the receiver.

Transmission (adversarial sender only):

4. Whenever M makes an interface call $(\texttt{transmit}, m)$, send $(\texttt{transmission}, m)$ to the sender.

Figure 4.8: The wrapper functionality by which we model stateful tamper-proof hardware that allows communication from the token to the token issuer. The runtime bound t is merely needed to prevent malicious token senders from providing a perpetually running program code M; it will be omitted throughout the rest of the section.

input, and then indicate to the functionality that the computation can proceed. First, and foremost, this does not model the intended scenario. But even worse, it would introduce a feedback from the sender, resulting in a bidirectional channel between the sender and the token (although restricted in bandwidth). With such a bidirectional channel, known impossibility results, e.g. [CF01], directly rule out a UC-secure realization of any functionality, even using computational assumptions.

We use the simplified version defined above for the rest of the section to keep the protocol description more readable. Nevertheless, it is possible to formalize the scenario correctly in the UC-framework, and we will sketch this in the following. The token functionality is split into two parts: one program containing the actual functionality, and another program that serves as a buffer. Whenever the functionality is supposed to communicate a message to the sender, it instead sends it to the buffer machine, which stores the message and directly reactivates the normal machine. The sender can then query the buffer machine for all stored messages when he is activated, and proceed normally. Using this formalization, the channel from the sender to the token is removed, but the token can still channel messages to the sender.

4.2.2 Limitations

The main limitation in the $\mathcal{F}_{\text{wrap}}^{\text{s-out}}$-hybrid model is that the token can always send its complete view to the sender. Our next result shows that this makes protocols like OT or OTM impossible without computational assumptions. This is mainly due to the fact that in these protocols, it does not matter *when* the sender obtains the complete view; he can break the security in any case. For other functionalities, the order in which the sender and the token are interacting with each other can still allow secure protocols.

Lemma 4.4. *There is no protocol Π_{OT} using any number of tokens that UC-realizes $\mathcal{F}_{\text{OT}}$ in the $\mathcal{F}_{\text{wrap}}^{\text{s-out}}$-hybrid model with statistical security.*

Proofsketch. The basic idea is that the malicious tokens send their complete view to the sender after each interaction with the receiver. Thus, independently of whether the sender or some token receive the last protocol message, the combined view of the sender and the tokens is available to a malicious sender. This directly implies that an OT protocol with statistical security is not possible, because the whole model collapses to the two-party case in the stand-alone setting and known impossibilities hold [CF01]. Either the complete transcript of the exchanged messages (which is available to a malicious sender) uniquely determines the choice bit c of the receiver, or a malicious receiver can obtain both input bits (s_0, s_1), and in both cases the oblivious transfer security is broken. □

Remark. We remark that the crucial point here is that for oblivious transfer, it does not matter at which time the sender gets the complete view, i.e. it does not matter whether some token or the sender receives the last message. As soon as he learns the choice bit, the protocol is broken. This argumentation, however, does not rule out statistically UC-secure commitments.

4.2.3 Protocols

From the impossibility in Section 4.2.2 we know that statistically secure OT and OTM cannot be achieved. Commitments are not ruled out, and we will show an unconditionally secure commitment from communicating tokens in Section 4.2.3.1. Intuitively, the impossibility does not hold because in a commitment protocol based on communicating tamper-proof hardware, if the token is only queried after the sender has to send some committing information to the receiver, the sender cannot later equivocate the commitment. The token is only used to verify the commitment information.

Based on this commitment scheme, we show how to obtain a computationally secure OTM protocol based on communicating tokens in Section 4.2.3.2. Since the token can communicate any query of the receiver, we cannot prevent that the token issuer learns when the receiver obtains the output of the OTM. Thus we do not realize $\mathcal{F}_{\text{OTM}}$, but a variant that we call weak OTM.

4.2.3.1 Unconditionally Secure Commitment from a Single Token

We now present our construction for unconditionally secure commitments. The main idea behind our construction in Figure 4.9 is similar to the commitment scheme that is implicit in Section 4.1.3.3. The sender commits to a message s by creating

j pairs of shares $(v_{j,0}, v_{j,1})$ such that $v_{j,0} \oplus v_{j,1} = s$. Then, he stores these shares in a hardware token and locks it with a key. First, the receiver queries the sender for one of the shares $v_{j,b}$ for each j. During the unveil phase, the sender sends the key that unlocks the token to the receiver, so that he can verify that the answers from the sender are consistent with the stored values. Since this introduces a channel into the token, the key is of size κ, while the sender has to create 2κ pairs of shares for each message. This ensures that the key cannot contain enough information for the token to allow equivocation.

Protocol Π_{COM}

Let SShare be a $(\kappa, \frac{\kappa}{2}+1)$ secret sharing scheme, e.g. [Sha79], and let $\mathcal{T}$ be an instance of $\mathcal{F}_{\text{wrap}}^{\text{s-out}}$.

Commit phase:

1. Sender: Let s be the sender's input. Choose an opening key $k \leftarrow \mathbb{F}_2^\kappa$ uniformly at random. Generate κ shares $(s_1, \ldots, s_\kappa) \leftarrow \mathsf{SShare.Share}(s, \kappa)$. For each share s_i, choose 2κ random vectors $v_{j,0}^{(i)} \leftarrow \mathbb{F}_2^\kappa$ and set $v_{j,1}^{(i)} = s_i \oplus v_{j,0}^{(i)}$. Create a token program T with the following functionality.
   - Upon receiving a message $(\texttt{challenge}, I)$ from the receiver, check if $I \subset \{1, \ldots, \kappa\}$ and $|I| \leq \frac{\kappa}{2}$, otherwise abort. Set *committed* $= 1$ and output $(\texttt{open}, \{v_{j,0}^{(i)}, v_{j,1}^{(i)}\}_{i \in I, j \in \{1,\ldots,2\kappa\}})$.
   - Upon receiving a message $(\texttt{open}, \bar{k})$, check if *committed* $= 1$ and $\bar{k} = k$, otherwise abort. Send $(\texttt{opened}, \{v_{j,0}^{(i)}, v_{j,1}^{(i)}\}_{i \in \{1,\ldots,\kappa\}, j \in \{1,\ldots,2\kappa\}})$ to the receiver.

   Send $(\texttt{create}, \mathsf{T})$ to $\mathcal{T}$.
2. Receiver: Upon receiving a message $(\texttt{ready})$ from $\mathcal{T}$, select $2\kappa^2$ random bits $c_{i,j}$ for $i \in \{1, \ldots, \kappa\}$ and $j \in \{1, \ldots, 2\kappa\}$ and send them to the sender.
3. Sender: Let $(\{c_{i,j}\}_{i \in \{1,\ldots,\kappa\}, j \in \{1,\ldots,2\kappa\}})$ be the message from the receiver. Return $(\{v_{j,c_{i,j}}^{(i)}\}_{i \in \{1,\ldots,\kappa\}, j \in \{1,\ldots,2\kappa\}})$ to the receiver.
4. Receiver: Pick a random subset $I \subset \{1, \ldots, \kappa\}$ of size $\frac{\kappa}{2}$ and send $(\texttt{challenge}, I)$ to $\mathcal{T}$. Let $(\texttt{open}, \{\bar{v}_{j,0}^{(i)}, \bar{v}_{j,1}^{(i)}\}_{i \in I, j \in \{1,\ldots,2\kappa\}})$ be the result. Check if $v_{j,c_{i,j}}^{(i)} = \bar{v}_{j,c_{i,j}}^{(i)}$ for $i \in \{1, \ldots, \kappa\}$ and $j \in \{1, \ldots, 2\kappa\}$, if not abort.

Unveil phase:

5. Sender: Send all shares $(s_1, \ldots, s_\kappa)$ and the opening key k to the receiver.
6. Receiver: Send $(\texttt{open}, k)$ to $\mathcal{T}$ and obtain $\{v_{j,0}^{(i)}, v_{j,1}^{(i)}\}_{i \in \{1,\ldots,\kappa\}, j \in \{1,\ldots,2\kappa\}}$. Check if $s_i = v_{j,0}^{(i)} \oplus v_{j,1}^{(i)}$ for all i and j and abort if that is not the case. Output $s = \mathsf{SShare.Reconstruct}(s_1, \ldots, s_\kappa)$, abort if this fails.

Figure 4.9: Statistically UC-secure protocol realizing $\mathcal{F}_{\text{COM}}$ in the $\mathcal{F}_{\text{wrap}}^{\text{s-out}}$-hybrid model.

While this scheme is secure, it is not yet extractable. The UC simulator needs to extract all shares from the token to reconstruct the committed value, but he cannot learn the opening key in advance. In order to get extractability, instead of

committing to the message itself, we first use the $(\kappa, \frac{\kappa}{2}+1)$-Shamir's secret sharing scheme [Sha79] to create κ shares $(s_1, \ldots, s_\kappa)$ of the message, then commit to each share using the above scheme. The receiver has to ask the token during the commit phase to open $\frac{\kappa}{2}$ shares, so that he can verify if the answers from the sender are consistent with the messages from the token. Now, the simulator can rewind the token during this step and query it for all κ shares, thus being able to reconstruct the shared value.

Theorem 4.3. *The protocol Π_{COM} in Figure 4.9 statistically UC-realizes $\mathcal{F}_{\text{COM}}$ (cf. Section 2.7.2.1) in the $\mathcal{F}_{\text{wrap}}^{\text{s-out}}$-hybrid model.*

Proof. **Corrupted sender.** Consider the simulator in Figure 4.10. Let $\mathcal{A}_\mathsf{S}$ be the dummy adversary. We have to show that $\mathsf{Real}_{\Pi_{\text{COM}}}^{\mathcal{A}_\mathsf{S}}(\mathcal{Z}) \underset{s}{\approx} \mathsf{Ideal}_{\mathcal{F}_{\text{COM}}}^{\mathcal{S}_\mathsf{S}}(\mathcal{Z})$ for any PPT-environment $\mathcal{Z}$.

The strategy of the simulator is very straightforward. He just needs to query the token program T^* that the sender provides with two valid sets I_0 and I_1 to obtain $\frac{\kappa}{2}+1$ shares of s so that he can reconstruct the input of the sender during the commit phase.

Simulator $\mathcal{S}_\mathsf{S}$

- Simulate $\mathcal{T}$ for $\mathcal{A}_\mathsf{S}$ and obtain T^*.
- (Commit) Proceed as follows:
  - Simulate Step 2 straightforwardly according to Π_{COM}.
  - In Step 4, pick a set I_0 and run T^* with input $(\texttt{challenge}, I_0)$. Check consistency according to Π_{COM} and abort otherwise. Sample new sets $I_1, I_2, \ldots \neq I_0$ and run T^* until a check is passed. Let I_j denote the set where the check is passed. Use the shares from the query I_0 and the shares from the query I_j to reconstruct the message $\hat{s}$. Send $(\texttt{commit}, \hat{s})$ to $\mathcal{F}_{\text{COM}}$.
- (Unveil) Let k^* be the key from $\mathcal{A}_\mathsf{S}$ from Step 5. Simulate Step 6 according to Π_{COM} and let s^* be the output, but abort if $s^* \neq \hat{s}$. Send $(\texttt{unveil}, \hat{s})$ to $\mathcal{F}_{\text{COM}}$.

Figure 4.10: Simulator against a corrupted sender in the protocol Π_{COM}.

First, we show that the sampling of two sets I_0, I_j requires only an expected polynomial number of iterations, i.e. the simulation is efficient. Note that the probability that T^* answers any of the queries $I_0, I_1, \ldots$ correctly is the same for each query since they are chosen from the same distribution. Let p denote this probability. If p is not negligible, then the expected number of iterations needed to find a second query I_j that is answered correctly is clearly polynomial.

Now, observe that from $\mathcal{A}_\mathsf{S}$'s view, the simulation is indistinguishable from a real protocol run except for the possible abort during the unveil, where the extracted value is compared to the real value. However, with the two sets I_0, I_j, the unique message $\hat{s}$ can be fully recovered. Thus, either $\mathcal{A}_\mathsf{S}$ must have correctly guessed the challenges to T^* as well as the challenge to himself to create an unveil consistent with the challenges. The probability for this is $2^{-\kappa} \cdot 2^{-2\kappa} = 2^{-3\kappa}$, which is clearly negligible in κ. Or T^* has to guess the challenge that the receiver sends to $\mathcal{A}_\mathsf{S}$, but the probability for this is $2^{-2\kappa} \cdot 2^{\kappa} = 2^{-\kappa}$, where the factor 2^κ derives from the

key k^* of length κ. Combined, this shows that the simulation is successful with overwhelming probability.

Corrupted receiver. Consider the simulator in Figure 4.11. Let $\mathcal{A}_{\mathsf{R}}$ be the dummy adversary. We have to show that $\mathsf{Real}^{\mathcal{A}_{\mathsf{R}}}_{\Pi_{\mathrm{COM}}}(\mathcal{Z}) \underset{s}{\approx} \mathsf{Ideal}^{\mathcal{S}_{\mathsf{R}}}_{\mathcal{F}_{\mathrm{COM}}}(\mathcal{Z})$ for any PPT-environment $\mathcal{Z}$.

The simulator uses random values instead of the real input for all shares. This is possible because he learns $\mathcal{A}_{\mathsf{R}}$'s challenge to the token and can then adaptively change the shares that are stored in the simulated token. This allows the simulator to equivocate the commitment to any value $\hat{s}$.

Simulator $\mathcal{S}_{\mathsf{R}}$

- Simulate $\mathcal{T}$ for $\mathcal{A}_{\mathsf{R}}$.
- (Commit) Upon receiving a message (`committed`) from $\mathcal{F}_{\mathrm{COM}}$, proceed as follows.
  - In Step 1, draw $2\kappa^2$ values $\hat{v}^{(i)}_j \leftarrow \mathbb{F}_2^\kappa$ uniformly at random.
  - Upon input $(\{c^*_{i,j}\}_{i\in\{1,\dots,\kappa\},j\in\{1,\dots,2\kappa\}})$ in Step 3, set $\hat{v}^{(i)}_{j,c^*_{i,j}} = \hat{v}^{(i)}_j$ and return the values $\hat{v}^{(i)}_{j,c^*_{i,j}}$ to $\mathcal{A}_{\mathsf{R}}$.
  - Upon input $(\texttt{challenge}, I^*)$ in Step 4, draw κ^2 values $\hat{v}^{(i)}_{j,1-c^*_{i,j}}$ for $i \in I$ uniformly at random and send $(\texttt{open}, \{\hat{v}^{(i)}_{j,c^*_{i,j}}, \hat{v}^{(i)}_{j,1-c^*_{i,j}}\}_{i\in I,j\in\{1,\dots,2\kappa\}})$ to $\mathcal{A}_{\mathsf{R}}$. Store the resulting shares $\hat{s}_1,\dots,\hat{s}_{\frac{\kappa}{2}}$ (with $\hat{s}_i = \hat{v}^{(i)}_{j,c^*_{i,j}} \oplus \hat{v}^{(i)}_{j,1-c^*_{i,j}}$).
- (Unveil) Upon receiving a message $(\texttt{opened}, \hat{s})$, proceed as follows.
  - In Step 5, compute the remaining $\frac{\kappa}{2}$ shares $\hat{s}_{\frac{\kappa}{2}+1},\dots,\hat{s}_\kappa$ such that $\mathsf{SShare.Reconstruct}(\hat{s}_1,\dots,\hat{s}_\kappa) = \hat{s}$. Additionally, set the values $\hat{v}^{(i)}_{j,1-c^*_{i,j}}$ such that $\hat{v}^{(i)}_{j,c^*_{i,j}} \oplus \hat{v}^{(i)}_{j,1-c^*_{i,j}} = \hat{s}_i$ for the remaining shares. Draw a random value $\hat{k} \leftarrow \mathbb{F}_2^\kappa$ and send $(\hat{s}_1,\dots,\hat{s}_\kappa,\hat{k})$ to $\mathcal{A}_{\mathsf{R}}$.
  - Upon receiving a message $(\texttt{open}, k^*)$ in Step 6, check if $\hat{k} = k^*$, if not abort. Output $\{\hat{v}^{(i)}_{j,0}, \hat{v}^{(i)}_{j,1}\}_{i\in\{1,\dots,\kappa\},j\in\{1,\dots,2\kappa\}}$.

Figure 4.11: Simulator against a corrupted receiver in the protocol Π_{COM}.

Indistinguishability of the simulation and the real protocol run follows from the fact that the shares are identically distributed in both runs, and the simulator only changes the token functionality adaptively, which $\mathcal{A}_{\mathsf{R}}$ notices only if he manages to query a valid message $(\texttt{open}, k^*)$ before having received $\hat{k}$. The probability for this event is negligible in κ. □

4.2.3.2 Computationally Secure OTM from a Single Token

One limitation of OTM in the $\mathcal{F}^{\text{s-out}}_{\text{wrap}}$-hybrid model is that the sender will learn the point in time when the receiver queries the OTM with his input. This cannot be prevented, because the token is allowed to send messages to the sender. Thus the resulting functionality that we realize is a weak OTM in the sense that the sender cannot change his inputs (as he might with OT), but still learns when the receiver inputs his choice bit (cf. Figure 4.12).

Functionality $\mathcal{F}_{\text{OTM}}^{\text{weak}}$

Implicitly parametrized by a security parameter κ. The variable *state* is initialized with wait.

Delivery phase:

1. Await an input $(\texttt{create}, s_0, s_1)$ from the sender. Check if *state* = wait and $s_0, s_1 \in \{0,1\}^\kappa$, else abort. Set *state* = sent, store s_0 and s_1 and send (created) to the adversary.
2. Await a message (delivery) from the adversary. Check if *state* = sent and abort if otherwise. Set *state* = ready and send (ready) to the receiver.

Choice phase:

3. Await an input $(\texttt{choice}, c)$ from the receiver. If *state* = ready, set *state* = done and send (notify) to the adversary. Send $(\texttt{output}, s_c)$ to the receiver.

Figure 4.12: Ideal functionality for weak One-Time Memory.

Similar to the case of incoming communication, we show a protocol for OTM where in a first step a CRS is computed via a fair coin toss. This CRS is then used to execute a UC-secure OT protocol between the receiver and the token. Instead of using a standard commitment for the CRS, we make use of our UC-secure commitment scheme Π_{COM} from Section 4.2.3.1 to fix the CRS. Due to the goal of using only one token, we cannot use the commitment scheme completely black-box and have to integrate the commitment protocol into the OTM protocol. To maintain a good readability of the protocol, we include which step of the protocol Π_{COM} is performed without explicitly stating the details. The formal description of the protocol is shown in Figure 4.13.

Theorem 4.4. *The protocol $\Pi_{\text{OTM}}^{\text{weak}}$ in Figure 4.13 computationally UC-realizes $\mathcal{F}_{\text{OTM}}^{\text{weak}}$ (cf. Figure 4.12) in the $\mathcal{F}_{\text{wrap}}^{\text{s-out}}$-hybrid model, given that CRS-based UC-secure OT exists.*

The proof of security follows the same argumentation as the security proof for the computationally secure OTM protocol in the $\mathcal{F}_{\text{wrap}}^{\text{s-inc}}$-hybrid model in Section 4.1.3.3.

Proof. **Corrupted sender.** The simulator has to influence the creation of the CRS by obtaining x before it has to send y. Once that is accomplished, it can set the CRS such that it can extract the UC-commitments that are sent by the token. Thus the simulator learns both inputs of the sender and can provide them to the ideal functionality. Let $\mathcal{A}_{\mathsf{S}}$ be the dummy adversary. We have to show that $\mathsf{Real}_{\Pi_{\text{OTM}}^{\text{weak}}}^{\mathcal{A}_{\mathsf{S}}}(\mathcal{Z}) \underset{c}{\approx} \mathsf{Ideal}_{\mathcal{F}_{\text{OTM}}^{\text{weak}}}^{\mathcal{S}_{\mathsf{S}}}(\mathcal{Z})$ for any PPT-environment $\mathcal{Z}$. To that end, consider the simulator $\mathcal{S}_{\mathsf{S}}$ in Figure 4.14.

Indistinguishability of the simulation and a real protocol run follows by a simple hybrid argument.

Experiment 0: This is the real model.

Experiment 1: Identical to Experiment 0, except that $\mathcal{S}_1$ starts the sender-simulator $\mathcal{S}_{\mathsf{S}}^{\Pi_{\text{COM}}}$ to obtain $\hat{x}$ and aborts if $\hat{x} \neq x^*$.

Protocol $\Pi_{\text{OTM}}^{\text{weak}}$

Let SIG be an EUF-CMA-secure signature scheme and COM a UC-secure commitment scheme in the CRS-hybrid model. Further let OT be a UC-secure OT protocol in the CRS-hybrid model and $\mathcal{T}$ be an instance of $\mathcal{F}_{\text{wrap}}^{\text{s-out}}$. Let Π_{COM} be the commitment scheme from Section 4.2.3.1.

Setup phase:

1. Sender: Let s_0, s_1 be the sender's input. Perform Step 1 of Π_{COM} and choose the opening key $k \leftarrow \mathbb{F}_2^\kappa$ and $x \leftarrow \mathbb{F}_2^\kappa$ uniformly at random. Use x as the commitment value. Generate a key pair $(\mathsf{sgk}, \mathsf{vk}) \leftarrow \mathsf{SIG.KeyGen}(\kappa)$. Enhance token program T' created according to the commit phase of Π_{COM} and add the following two functions to obtain T.
   - Upon receiving a message $(\texttt{commit}, \mathsf{CRS}, \sigma)$, check if $state = \mathsf{opened}$ and $\mathsf{SIG.Verify}(\mathsf{vk}, \mathsf{CRS}, \sigma) = 1$, otherwise abort. Set $state = \mathsf{i\_committed}$. Compute $(c_i, d_i) \leftarrow \mathsf{COM.Commit}(\mathsf{CRS}, s_i)$ and send $(\texttt{committed}, (c_0, c_1))$ to the receiver.
   - Upon receiving a message $(\texttt{output})$, check if $state = \mathsf{i\_committed}$, else abort. Execute the sender program of OT with input $(s_0 \| d_0, s_1 \| d_1)$ using CRS and forward the messages between the receiver and the sender program.

   Send $(\texttt{create}, \mathsf{T})$ to $\mathcal{T}$.
2. Receiver: Perform Step 2 of Π_{COM}.
3. Perform Step 3 of Π_{COM}.
4. Receiver: Perform Step 4 of Π_{COM}. Then, pick $y \leftarrow \mathbb{F}_2^\kappa$ uniformly at random and send it to the sender.

Delivery phase:

5. Sender: Compute $\mathsf{CRS} = x \oplus y$ and $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, \mathsf{CRS})$. Send the unveil information of Π_{COM} and (k, σ) to the receiver.
6. Receiver: Perform Step 6 of Π_{COM}. Set $\mathsf{CRS} = x \oplus y$ and send $(\texttt{commit}, \mathsf{CRS}, \sigma)$ to $\mathcal{T}$.

Choice phase:

7. Receiver: Let c be the receiver's input. Send $(\texttt{output})$ to $\mathcal{T}$ and execute the receiver program of OT with input c and CRS and forward the messages between $\mathcal{T}$ and the program. Let $s_c \| d_c$ be the output. Check if $\mathsf{COM.Open}(\mathsf{CRS}, s_c, c_c, d_c) = 1$, and output s_c if the check is passed.

Figure 4.13: Statistically UC-secure protocol realizing $\mathcal{F}_{\text{OTM}}^{\text{weak}}$ in the $\mathcal{F}_{\text{wrap}}^{\text{s-out}}$-hybrid model.

Experiment 2: Identical to Experiment 1, except that $\mathcal{S}_2$ chooses y as $\mathsf{CRS} \oplus x$ and uses the sender-simulators $\mathcal{S}_{\mathsf{S}}^{\mathsf{COM}}$ and $\mathcal{S}_{\mathsf{S}}^{\mathsf{OT}}$ for COM and OT to obtain $\hat{s}_b$ and $\hat{s}'_b$ for $b \in \{0, 1\}$. It aborts if $\hat{s}'_b \neq \hat{s}_b$ for $b \in \{0, 1\}$. This is the ideal model.

Experiment 0 and Experiment 1 are computationally indistinguishable since the sender-simulator $\mathcal{S}_{\mathsf{S}}^{\Pi_{\text{COM}}}$ will extract the correct $\hat{x}$ with overwhelming probability.

Simulator $\mathcal{S}_{\mathsf{S}}$

- Simulate $\mathcal{T}$ for $\mathcal{A}_{\mathsf{S}}$ and obtain T^*.
- (Setup) Start the sender-simulator $\mathcal{S}_{\mathsf{S}}^{\Pi_{\mathrm{COM}}}$ and extract the sender's input x^*. Set $\hat{y} = \mathsf{CRS} \oplus x^*$.
- (Delivery) Simulated straightforwardly according to $\Pi_{\mathrm{OTM}}^{\mathrm{weak}}$. After obtaining c_0^*, c_1^* from T^* in Step 6, use the sender-simulator $\mathcal{S}_{\mathsf{S}}^{\mathsf{COM}}$ to extract $\hat{s}_0$ and $\hat{s}_1$ from c_0^* and c_1^*, respectively. Send $(\texttt{create}, \hat{s}_0, \hat{s}_1)$ to $\mathcal{F}_{\mathrm{OTM}}^{\mathrm{weak}}$.
- (Choice) Simulated straightforwardly according to $\Pi_{\mathrm{OTM}}^{\mathrm{weak}}$.

Figure 4.14: Simulator against a corrupted sender in the protocol $\Pi_{\mathrm{OTM}}^{\mathrm{weak}}$.

Experiments 1 and 2 are indistinguishable due to the UC-security of COM and OT. The sender-simulator $\mathcal{S}_{\mathsf{S}}$ extracts the inputs of the sender with overwhelming probability, so the simulator will not abort. A distinguishing environment $\mathcal{Z}$ can directly be used to break the UC-security of at least one of COM and OT.

Corrupted receiver. Let $\mathcal{A}_{\mathsf{R}}$ be the dummy-adversary. We have to show that $\mathsf{Real}_{\Pi_{\mathrm{OTM}}^{\mathrm{weak}}}^{\mathcal{A}_{\mathsf{R}}}(\mathcal{Z}) \underset{c}{\approx} \mathsf{Ideal}_{\mathcal{F}_{\mathrm{OTM}}^{\mathrm{weak}}}^{\mathcal{S}_{\mathsf{R}}}(\mathcal{Z})$ for any PPT-environment $\mathcal{Z}$.

Consider the simulator $\mathcal{S}_{\mathsf{R}}$ in Figure 4.15. In a first step, the simulator simulates $\mathcal{T}$ for $\mathcal{A}_{\mathsf{R}}$ and learns all queries. He commits to a random value and obtains the receiver's input y. He then picks a CRS CRS that allows him to cheat in COM and OT, sets $x = \mathsf{CRS} \oplus y$ and uses the receiver-simulator of Π_{COM} to equivocate the commitment to the random value to y.

Once that is accomplished, the simulator commits to random inputs $\hat{s}_0, \hat{s}_1$ and then obtains the choice bit c^* of $\mathcal{A}_{\mathsf{R}}$ using the CRS in the protocol OT. He inputs the choice bit into $\mathcal{F}_{\mathrm{OTM}}^{\mathrm{weak}}$ to obtain the real result and then again uses the CRS to equivocate the commitments of COM corresponding to the choice bit to the real output.

Simulator $\mathcal{S}_{\mathsf{R}}$

- Simulate $\mathcal{T}$ for $\mathcal{A}_{\mathsf{R}}$.
- (Setup) Simulated straightforwardly according to $\Pi_{\mathrm{OTM}}^{\mathrm{weak}}$ with a random commitment value r. Obtain y^* in Step 4. Choose $\hat{x} = \mathsf{CRS} \oplus y^*$, where CRS is the CRS needed to correctly simulate COM and OTM.
- (Delivery) In Step 5, compute $\hat{\sigma} \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, \mathsf{CRS})$ and use the receiver-simulator $\mathcal{S}_{\mathsf{R}}^{\Pi_{\mathrm{COM}}}$ to equivocate r to $\hat{x}$. Execute the rest according to $\Pi_{\mathrm{OTM}}^{\mathrm{weak}}$.
- (Choice) Upon receiving a message $(\texttt{output})$ from $\mathcal{A}_{\mathsf{R}}$ in Step 7, start the receiver-simulator $\mathcal{S}_{\mathsf{R}}^{\mathsf{OT}}$ for OT and halt the simulation once $\mathcal{A}_{\mathsf{R}}$'s choice bit c^* is obtained. Input c^* into $\mathcal{F}_{\mathrm{OTM}}^{\mathrm{weak}}$ and obtain the output s_{c^*}. Start the receiver-simulator $\mathcal{S}_{\mathsf{R}}^{\mathsf{COM}}$ of COM to compute the equivocation $\hat{d}_{c^*}$ of $\hat{c}_{c^*}$ to s_{c^*} and provide $\mathcal{S}_{\mathsf{R}}^{\mathsf{OT}}$ with the input $\hat{s}_{c^*} \| \hat{d}_{c^*}$. Resume the simulation of $\mathcal{S}_{\mathsf{R}}^{\mathsf{OT}}$.

Figure 4.15: Simulator against a corrupted receiver in the protocol $\Pi_{\mathrm{OTM}}^{\mathrm{weak}}$.

Experiment 0: This is the real model.

Experiment 1: Identical to Experiment 0, except that $\mathcal{S}_1$ first commits to a random value r, later sets $\hat{x} = \mathsf{CRS} \oplus y^*$ and provides the corresponding unveil via the receiver-simulator $\mathcal{S}_{\mathsf{R}}^{\Pi_{\mathrm{COM}}}$.

Experiment 2: Identical to Experiment 1, except that $\mathcal{S}_2$ aborts if $\sigma^* \neq \hat{\sigma}$ and $\mathsf{SIG.Verify}(\mathsf{vk}, \mathsf{CRS}^*, \sigma^*) = 1$.

Experiment 3: Identical to Experiment 2, except that $\mathcal{S}_3$ starts the receiver-simulator $\mathcal{S}_{\mathsf{R}}^{\mathsf{OT}}$ to learn the choice-bit.

Experiment 4: Identical to Experiment 3, except that $\mathcal{S}_4$ commits to uniformly random values $\hat{s}_0, \hat{s}_1$, starts the receiver-simulator $\mathcal{S}_{\mathsf{R}}^{\mathsf{COM}}$ and equivocates the commitment $\hat{c}_c^*$ to the value s_c^* obtained from $\mathcal{F}_{\mathrm{OTM}}^{\mathrm{weak}}$.

From $\mathcal{Z}$'s view, Experiment 0 and Experiment 1 are statistically indistinguishable, since the protocol Π_{COM} is statistically UC-secure and a distinguishing $\mathcal{Z}$ could directly be used to break the UC-security of Π_{COM}. Experiments 1 and 2 are computationally indistinguishable given that SIG is EUF-CMA-secure since $\mathcal{S}_2$ only aborts if the receiver sends a valid signature to the token, although $\mathcal{S}_2$ did not create the signature. Experiment 2 and Experiment 3 are computationally indistinguishable by the UC-security of OT, i.e. a distinguishing $\mathcal{Z}$ can directly be used to break the UC-security of OT. The same argumentation holds for Experiment 3 and Experiment 4 based on the UC-security of COM. □

4.2.4 Relation to Two-Party Computation

In this section, we examined whether communication from the token to its sender still allows for UC-secure computation. Summarizing, our results imply the following.

Statistically secure (non-interactive) two-party computation: From the impossibility in Section 4.2.2, it is clear that general two-party computation with statistical security is impossible. This impossibility also directly rules out non-interactive solutions. Nonetheless, we showed in Section 4.2.3.1 that statistically UC-secure commitments can be obtained from a single token.

Computationally secure (non-interactive) two-party computation: In Section 4.2.3.2, we presented a computationally UC-secure OTM protocol, based on the above mentioned commitment scheme and a single hardware token. Again, the impossibility of Döttling et al. [DKMQ11] for OTM from a single token without interaction holds and thus storing the sender functionality in a second token yields an optimal and completely non-interactive solution.

Combined, we thus showed the results that are highlighted in Table 4.16.

Model	computational 2PC		statistical 2PC	
	interactive	non-interactive	interactive	non-interactive
stateful	✓	✓	✓	✓
stateful (inc. com.)	✓	✓	✓	✓
stateful (outg. com.)	✓(1 Token)	✓(2 Tokens)	✗(✓) Commitment (1 Token)	✗
resettable	✓	✓	✗(✓)	✗
bounded-resettable	✓	✓	✓	(✓)
reusable resettable	✓	✓	✗(✓)	✗

Table 4.16: Feasibility of interactive and non-interactive two-party computation from stateful tamper-proof hardware with communication from the token to its sender.

5. Resettable Tamper-Proof Hardware

At its inception, tamper-proof hardware for UC-secure computation was modeled as a wrapper that stores a Turing machine and maintains the state of the machine (cf. Chapter 3). This formalization, however, does not accurately model real tamper-proof hardware that is in use today, like e.g. smartcards or flash drives. These devices are usually dependent on an external power source that can be switched off during the operation of the device. In particular, this means that the tokens cannot reliably keep a state if the power source is maliciously manipulated. On closer inspection of UC-secure cryptographic protocols that make use of stateful tamper-proof hardware, it is apparent that a single successful *reset* of the token, i.e. preventing an update of the state during the computation, will completely break the security of these schemes.

Consider a protocol for OT where the receiver of the token can query the token exactly once with his choice bit to obtain the corresponding input (e.g. the protocol of Döttling et al. [DKMQ11]). Suppose that the receiver manages to prevent the token from storing the information that it has already been queried once. The receiver can then flip his choice bit and query the token a second time, thus learning both inputs and breaking the sender's security.

In order to obtain secure protocols while factoring in this so-called *resetting attack*, Chandran et al. [CGS08] introduce *stateless* tamper-proof hardware. It is modeled exactly like the stateful wrapper functionality, but the stored Turing machine cannot save its current state. For the rest of this chapter we will consider the equivalent model of *resettable* tamper-proof hardware, where the Turing machine is allowed to store its state, but a malicious receiver can reset that state. A discussion on this matter is deferred to Section 5.1. We will now give an overview over existing results based on resettable tokens.

Concerning statistical security, Goyal et al. [GIMS10] show that non-interactive commitments and OT with statistical security are impossible based on resettable hardware tokens alone. They provide a construction of an interactive commitment based on a single resettable token, but with stand-alone security only. This work was later improved by Damgård and Scafuro [DS13] who show how to obtain a UC-secure variant of unconditional commitments. Their approach is to transform any straight-

	interactive			non-interactive
	[GIMS10]	[DS13]	[GIMS10]	[GIMS10]
Functionality	Commitment	Commitment	OT	Commitments
Tokens	1	$\Theta(\kappa)$ (bidir.)		
Rounds	$\Theta(1)$	$\Theta(1)$	impossible	impossible
Security	stand-alone	UC		

Table 5.1: Overview of statistically secure (non-)interactive two-party protocols from resettable hardware.

line extractable commitment into a UC-secure one by using a generic compiler, and they also present a construction of a straight-line extractable commitment from a single resettable hardware token. However, the transformation leads to polynomially many resettable tokens for a UC-commitment (cf. Table 5.1). These results already cover most interesting functionalities from a feasibility point of view and we thus do not further investigate statistical security with "standard" resettable tokens.

Allowing computational assumptions, Chandran et al. [CGS08] achieve UC-secure commitments (and thus general two-pary computation) in this model based on enhanced trapdoor-permutations, but parties have to exchange tokens bidirectionally. Kolesnikov [Kol10] introduces an OT protocol based on resettable tokens that achieves covert security [AL07], where only one token has to be sent. Goyal et al. [GIS$^+$10] show how to construct UC-secure OT from bidirectionally exchanging polynomially many tokens while requiring collision-resistant hash-functions, or only one-way functions at the cost of increasing the round complexity. They also investigate non-interactive computation based on resettable hardware, and present a general feasibility result by providing a construction based on a single stateless token and one-way functions. The protocol is very inefficient because they use an obfuscation scheme in conjunction with a stateful token, which requires $\Theta(\kappa)$ rounds of interaction with the token. Very recently, a flaw in the OT protocol of Goyal et al. [GIS$^+$10] was found and fixed by Hazay et al. [HPV15]. Additionally, they provided a construction based on one-way functions that needs only two rounds and is thus round-optimal. A further optimized OT construction from resettable hardware was shown by Choi et al. [CKS$^+$14] who showed that exchanging two tokens and assuming the existence of verifiable random functions (VRFs) is sufficient for UC-secure OT. All of the above described works use black-box techniques, but [CKS$^+$14] also show that using non-black-box techniques, it is sufficient to only send a single token from sender to receiver to obtain UC-secure OT.

Our contribution. In this chapter, we improve upon the above mentioned previous results with regard to computational assumptions and the amount of tokens needed. Regarding interactive two-party computation, we present a generic UC-secure protocol compiler that transforms any protocol based on stateful tamper-proof hardware into a protocol based on resettable tamper-proof hardware. This yields very efficient constructions for UC-secure OT by applying our compiler to existing protocols, e.g. to [DKMQ11]. A preliminary version of this result was published in [DKMQN15]. We also investigate non-interactive UC-secure computation and present two protocols for UC-secure non-interactive computation. The results presented here are a revised and improved version of the work by Döttling et al. [DMMQN13], which independently of [CKS$^+$14] shows that non-black-box techniques are essential if one wants to obtain protocols where tokens are sent only in

	interactive					non-interactive	
	[CGS08]	[GIS$^+$10]	[Kol10]	[CKS$^+$14]	here	[GIS$^+$10]	here
Functionality	Com.	OT	OT	OT	OT	Obfuscation	CRS
Tokens	2 (bidir.)	$\Theta(\kappa)$ (bidir.)	1	2 (bidir.)	1	1	2
Rounds	$\Theta(\kappa)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(\kappa)$	$\Theta(1)$
Assumption	eTDP	CRHF[1]	OWF	VRF[2]	OWF	OWF	OWF
Security	UC	UC	Covert	UC	UC	stand-alone	UC

Table 5.2: Overview of computationally secure (non-)interactive two-party protocols from untrusted resettable hardware.

one direction. All our results in this chapter can be based on one-way functions and inherently make use of non-black-box techniques. Table 5.2 gives an overview of our results compared to the results in literature.

Our techniques. All of our constructions rely on non-black-box techniques, which allows us to send tokens into one direction only. Let us briefly discuss how UC-simulation usually works in the scenario of tamper-proof hardware, and how non-black-box techniques can be used to reduce the number of tokens. If tokens are sent by both parties, as e.g. in [CGS08, CKS$^+$14], the simulator obtains the token code and can learn the messages that are sent to the tokens from *both parties* and thereby produce a correct simulation. If a token is sent only in one direction, the simulator can no longer learn the inputs of the token issuer as described above. In the case of stateful tokens, the simulator can simply rewind the token to obtain the input. With resettable tokens, however, this strategy fails, because rewinding and resetting are equivalent in the sense that a successful rewinding strategy for the simulator directly implies a successful resetting strategy for the adversary. Here, non-black-box techniques can be used to obtain UC-security. First, note that resettably-sound zero-knowledge (rsZK) for non-trivial languages requires a non-black-box simulator [BGGL01]. Non-black-box in this context means that the simulator for the proof system requires access to the program of the verifier. In the context of UC-proofs, the simulator usually simulates the setup, i.e. the wrapper functionality, and thus learns the token code that the token issuer inputs into the wrapper. Therefore, the UC-simulator has access to the code of the verifier and the non-black-box simulator of the rsZK proof system can be used. One important aspect here is that the simulation of the resettably-sound zero-knowledge proof is typically straight-line, which means that the UC-simulator can use the rsZK-simulator directly.

Our approach is to have the token issuer program the token such that it outputs a secret once the receiver proves some statement to the token via a resettably-sound zero-knowledge proof. Resettable soundness is required for resettable tokens, otherwise the receiver as the prover essentially has the power of the ZK simulator. The UC-simulator obtains the token code and can therefore create a false proof for the token via the non-black-box simulator. With this false proof, the token outputs its secret and the simulator is able to produce a correct simulation.

In a little more detail, for our stateful to resettable token compiler, we add one

[1] A protocol based on OWF is also shown, but the round complexity increases to $\Theta(\kappa/\log(\kappa))$. Recently, it was shown by Hazay et al. [HPV15] that there is a subtle problem with this approach, and they present a solution with 2 rounds based on OWF.

[2] Verifiable random functions (VRFs) are only known from specific number-theoretic assumptions [MRV99, Lys02, Jag15]. They also present a protocol with similar properties based on a CRHF, but the number of OTs is bounded in this case.

additional round of interaction between the sender and the receiver to the protocol using a stateful token in which the sender obliviously authenticates the input of the receiver for the token. The receiver then uses a rsZKAoK to prove that he knows such an authentication on his input. In the simulation, the simulator can fake this proof, because he has access to the token code, and then obtain the output of the token without querying the sender.

A similar technique is also used for non-interactive secure computation. The token is used as a commitment on a random value which will later be used to generate a common reference string (CRS). The token will reveal the random value once the receiver proves that he sent his random value to the sender (or the other token). As before, the simulator can fake a proof, learn the random value of the token and adjust his random value to influence the CRS.

Structure of this chapter. The remaining part of the chapter is structured as follows. First, we formally define the model of resettable tamper-proof hardware in Section 5.1. We then prove some limitations in Section 5.2 and additionally give an overview over existing impossibility results in the area of resettable hardware tokens. In Section 5.3, we investigate efficient two-party computation from resettable hardware tokens and in Section 5.4, we cover the non-interactive case. Section 5.5 shows our results in context with previous works.

5.1 Model

We use a definition of *resettable* tamper-proof hardware very similar to the definition of stateless tamper-proof hardware by [CGS08, GIS+10]. For simplicity, we state the functionality for the two-party case where only a token issuer and a token receiver are present. The functionality allows the sender to wrap a program M in a hardware token, and "send" this token to the receiver, who in turn can query it an arbitrary (polynomial) number of times. While the program M can save its state, we allow an adversarial receiver to delete this state and thereby reset the program M to its initial state (cf. Figure 5.3).

In the sequel, unless stated otherwise, we will use the notation T for programs (given as code, Turing machine etc.) and $\mathcal{T}$ for the instance of the wrapper-functionality $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$ in which T runs. Compared to the definition of [CGS08, GIS+10], our formalization has the advantage that during an honest execution, the token can keep its state. While this might seem to be more powerful than stateless hardware, both models are in fact equivalent:

- The receiver can reset the resettable token after each query, clearly yielding a stateless token.
- The receiver can send the complete protocol transcript with each new query, thereby giving the stateless token a state similar to the one that is stored in the resettable token. Of course, the transcript has to be unforgeable in the sense that the receiver cannot arbitrarily create a valid transcript.

Our definition thus reduces the complexity of the description of protocols because we can omit the step of explicitly sending the transcript in each protocol step.

One additional aspect that we want to discuss here is whether it is possible to guarantee that a reset of the token actually deletes its state. This is important because a state where only part of the information is lost, but e.g. not the randomness,

Functionality $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$

Implicitly parametrized by a security parameter κ.

Creation:

1. Await an input $(\texttt{create}, \mathsf{M}, t)$ from the token issuer, where M is a deterministic Turing machine and $t \in \mathbb{N}$. Store (M, t) and send $(\texttt{created})$ to the adversary. Ignore all further `create`-messages.

2. Await a message $(\texttt{delivery})$ from the adversary. Then, send $(\texttt{ready})$ to the token receiver.

Execution:

3. Await an input $(\texttt{run}, w)$ from the receiver. If no `create`-message has been sent, return a special symbol $\perp$. Otherwise, run M on w from its most recent state. When M halts without generating output or t steps have passed, send $\perp$ to the receiver; otherwise store the current state of M and send the output of M to the receiver.

Reset (adversarial receiver only):

4. Upon receiving a message $(\texttt{reset})$ from a corrupted token receiver, reset M to its initial state.

Figure 5.3: The wrapper functionality by which we model resettable tamper-proof hardware. The runtime bound t is merely needed to prevent malicious token senders from providing a perpetually running program code M; it will be omitted throughout the rest of the chapter.

might allow an adversary to break the security of a scheme. To this end, one could for instance keep a certain information in RAM only, and whenever this information is missing, the token will erase its internal state completely.

5.2 Limitations

In this section, we will present general limitations of the model of resettable tamper-proof hardware. Apart from two aforementioned results from the literature which rule out statistically secure OT and non-interactive commitments, we also present two restrictions that apply to computational security. First, if resettable tokens are sent only in one direction, then non-black-box techniques are necessary to obtain UC-security. Second, if a protocol based on resettable tokens is non-interactive, at least two tokens have to be sent to UC-realize most functionalities.

Let us first consider the case of information-theoretical security, where Goyal et al. [GIMS10] showed several impossibilities. The first concerns non-interactive commitments based on resettable hardware tokens. The bound on the entropy ensures an efficient token functionality.

Theorem ([GIMS10, Theorem 4.14](informal))**.** *Let Π_{COM} be a statistically secure non-interactive commitment protocol in the resettable hardware token model. Then either of the following holds:*

- *Violation of binding: There exists a strategy for the commitment sender to unveil the commitment to two different values with high probability.*
- *Violation of hiding: There exists an unbounded commitment receiver who can guess the commitment with high probability.*

[GIMS10] show that depending on the entropy enclosed in the token and the number of queries to the token, the cheating probability can reach at least $\frac{4}{5}$. The intuition behind this impossibility is that there exists an efficient learning algorithm that either extracts all of the entropy out of the token(s), thus breaking the hiding property, or this algorithm fails. If the algorithm fails, however, the sender can use the remaining entropy of the token(s) to unveil the commitment arbitrarily.

Goyal et al. [GIMS10] also show that unconditionally secure OT is not possible based on resettable hardware. The case of one-time memory (OTM) is directly implied by the above impossibility (OTM can be seen as a non-interactive variant of OT), but interaction might help. The authors show that interaction does not help by extending the notion of *accessible entropy* [HRVW09] to the resettable token setting. Let s_0, s_1 denote the sender's input and c denote the choice bit of the receiver.

Theorem ([GIMS10, Theorem 5.3](informal))**.** *Let Π_{OT} be a statistically secure protocol for OT in the resettable hardware token model. Then either of the following holds:*

- *Violation of sender-security: When the sender chooses s_0, s_1 uniformly at random from $\{0,1\}$ and interacts with the receiver, a malicious receiver can learn both s_0 and s_1 with high probability.*
- *Violation of receiver-security: When the receiver chooses c uniformly at random from $\{0,1\}$ and interacts with the sender, a malicious sender can correctly guess c with high probability.*

All that the malicious parties have to do is querying the resettable hardware more often than specified by the protocol to break the security; they can execute the rest of the protocol honestly. [GIMS10] first show that for OT, inaccessible entropy is necessary, and then they proceed to show that any OT protocol based on resettable hardware does not have inaccessible entropy because with the same (entropy) learning algorithm that was used before, (close to) all entropy can be accessed.

Summarizing, based on resettable hardware, statistically secure protocols can only be constructed for a limited class of functionalities, e.g. commitments [GIMS10, DS13]. In particular, non-interactive secure computation is completely ruled out. Furthermore, note that these impossibilities are not bound to UC-security but hold for any formalization of resettable hardware.

As it turns out, there are severe restrictions for protocol constructions even for the case of computational UC-security. We will first show that protocols where tokens are sent only in one direction have to make use of non-black-box techniques. As an example, we consider the ideal point function functionality, which models a simple case in which the simulator has to extract an input from the token. The point function functionality $\mathcal{F}_{\mathrm{PF}}$ is initialized by an input $\hat{x} \in \{0,1\}^n$ from the sender. The receiver can query $\mathcal{F}_{\mathrm{PF}}$ an arbitrary (polynomial) number of times with inputs x, receiving output $\mathsf{PF}_{\hat{x}}(x)$, where $\mathsf{PF}_{\hat{x}}(x) = 1$ if $x = \hat{x}$ and $\mathsf{PF}_{\hat{x}}(x) = 0$ otherwise.

Thus this impossibility implies a general impossibility for all protocols where such an extraction step is necessary.

Lemma 5.1. *There is no protocol Π_{PF} using any number of resettable hardware tokens $\mathcal{T}_1, \ldots, \mathcal{T}_n$ issued from the sender to the receiver that computationally UC-realizes $\mathcal{F}_{\mathrm{PF}}$ in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$-hybrid model and whose security is proven using only black-box techniques.*

Intuitively, the impossibility follows from the fact that the only advantage that the simulator has over a malicious token receiver is the fact that he learns the (possibly obfuscated or unreadable) program that the sender stores inside the token. If the UC-simulator is only allowed to rewind the program and is able to extract the input, a resetting receiver will also be able to extract the input from the token because rewinding and resetting are equivalent in this scenario. Thus the UC-simulator has to use the program in a non-black-box way to achieve UC-security.

Proof. Assume for the sake of contradiction that there exists a protocol Π_{PF} UC-realizing $\mathcal{F}_{\mathrm{PF}}$ and that there exists a black-box simulator $\mathcal{S}_{\mathsf{S}}$ against a corrupted sender $\mathcal{A}_{\mathsf{S}}$ such that for all PPT-environments $\mathcal{Z}$, $\mathsf{Real}_{\Pi_{\mathrm{PF}}}^{\mathcal{A}_{\mathsf{S}}}(\mathcal{Z}) \underset{c}{\approx} \mathsf{Ideal}_{\mathcal{F}_{\mathrm{PF}}}^{\mathcal{S}_{\mathsf{S}}}(\mathcal{Z})$. Such a simulator must be able to extract a point $\hat{x}$ from the malicious tokens $\mathcal{T}_1^*, \ldots, \mathcal{T}_n^*$ using only black-box techniques (i.e. rewinding).

We will now construct an environment $\mathcal{Z}^*$ that distinguishes between $\mathsf{Real}_{\Pi_{\mathrm{PF}}}^{\mathcal{A}_{\mathsf{S}}}(\mathcal{Z}^*)$ and $\mathsf{Ideal}_{\mathcal{F}_{\mathrm{PF}}}^{\mathcal{S}_{\mathsf{S}}}(\mathcal{Z}^*)$ by constructing a malicious receiver $\mathcal{A}_{\mathsf{R}}$ that extracts the secret $\hat{x}$ from the tokens $\mathcal{T}_1, \ldots, \mathcal{T}_n$.

$\mathcal{A}_{\mathsf{R}}$ is constructed such that he internally simulates $\mathcal{S}_{\mathsf{S}}$ and provides his interface with $\mathcal{T}_1, \ldots, \mathcal{T}_n$ to $\mathcal{S}_{\mathsf{S}}$. $\mathcal{A}_{\mathsf{R}}$ then outputs to $\mathcal{Z}^*$ whatever $\mathcal{S}_{\mathsf{S}}$ outputs. From the view of $\mathcal{S}_{\mathsf{S}}$, the simulation of $\mathcal{A}_{\mathsf{R}}$ is identically distributed to $\mathsf{Ideal}_{\mathcal{F}_{\mathrm{PF}}}^{\mathcal{S}_{\mathsf{S}}}(\mathcal{Z}^*)$. Thus, $\mathcal{A}_{\mathsf{R}}$ outputs the secret point $\hat{x}$ with overwhelming probability. On the other hand, any simulator $\mathcal{S}_{\mathsf{R}}$ has only black-box access to the point function $\mathsf{PF}_{\hat{x}}$ via $\mathcal{F}_{\mathrm{PF}}$. Thus $\mathcal{S}_{\mathsf{R}}$ succeeds to learn $\hat{x}$ only with negligible probability.

Therefore, $\mathcal{Z}^*$ can efficiently distinguish $\mathsf{Real}_{\Pi_{\mathrm{PF}}}^{\mathcal{A}_{\mathsf{R}}}(\mathcal{Z}^*)$ and $\mathsf{Ideal}_{\mathcal{F}_{\mathrm{PF}}}^{\mathcal{S}_{\mathsf{R}}}(\mathcal{Z}^*)$ for any PPT-simulator $\mathcal{S}_{\mathsf{R}}$, contradicting the UC-security of Π_{PF}. □

Independently of our result, a similar statement was shown for the restricted case of oblivious transfer from a single resettable tamper-proof hardware token by Choi et al. [CKS+14]. We state their theorem here for completeness; their proof follows the same argumentation as our proof of Lemma 5.1, but tailored to the case of OT.

Theorem ([CKS+14, Theorem 2](informal))**.** *There is no protocol Π_{OT} using one resettable token that UC-realizes $\mathcal{F}_{\mathrm{OT}}$ in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$-hybrid model and whose security is proven using only black-box techniques.*

Next, we state a lower bound for the number of resettable tokens that have to be sent for non-interactive two-party computation. For a completely non-interactive protocol, where the sender only sends tokens to the receiver, at least two tokens are necessary. If only one token is used, it can adaptively change its behavior depending on the inputs of the receiver. With two tokens, this is no longer possible: on the one hand, the receiver can cross-check the answers from both tokens, and on the other hand, none of the two tokens learns a complete protocol transcript.

Lemma 5.2. *There is no protocol Π_{PF} using one resettable token that computationally UC-realizes $\mathcal{F}_{\mathrm{PF}}$ in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$-hybrid model with no further communication.*

On an intuitive level, if such a protocol existed, the simulator against a corrupted receiver could be used by a malicious sender to program a malicious token that adaptively changes its behavior depending on the receiver's input. Thus, the environment can distinguish between the real protocol and the simulation by cleverly choosing the receiver's input.

Proof. For the sake of contradiction, assume there exists a protocol Π_{PF} that UC-implements $\mathcal{F}_{\mathrm{PF}}$ using a single (resettable) hardware token and no interaction (w.l.o.g. we can assume that messages from the sender to the receiver are sent together with the token). Let $\mathcal{A}_{\mathsf{R}}$ be the dummy adversary for the receiver. Since Π_{PF} is UC-secure, there exists a simulator $\mathcal{S}_{\mathsf{R}}$ such that for any PPT-environment $\mathcal{Z}$, $\mathsf{Real}^{\mathcal{A}_{\mathsf{R}}}_{\Pi_{\mathrm{PF}}}(\mathcal{Z}) \underset{c}{\approx} \mathsf{Ideal}^{\mathcal{S}_{\mathsf{R}}}_{\mathcal{F}_{\mathrm{PF}}}(\mathcal{Z})$.

We will now show that for every sender-simulator $\mathcal{S}_{\mathsf{S}}$, there exists a PPT-environment $\mathcal{Z}^*$ such that the distributions $\mathsf{Real}^{\mathcal{A}_{\mathsf{S}}}_{\Pi_{\mathrm{PF}}}(\mathcal{Z}^*)$ and $\mathsf{Ideal}^{\mathcal{S}_{\mathsf{S}}}_{\mathcal{F}_{\mathrm{PF}}}(\mathcal{Z}^*)$ are efficiently distinguishable, contradicting the UC-security of Π_{PF}. This $\mathcal{Z}^*$ creates a malicious token program T^* which behaves adaptively in the following sense. The token T^* internally simulates the simulator $\mathcal{S}_{\mathsf{R}}$ together with a malicious functionality $\mathcal{F}^*$, providing its interface with the receiver to $\mathcal{S}_{\mathsf{R}}$.

The malicious functionality $\mathcal{F}^*$ behaves as follows. Once it receives an input x for the first time, it checks whether x is equal to a secret random $\hat{x}_0$. If so, it will behave like the point function $\mathsf{PF}_{\hat{x}_0}$ in this call and all successive calls. If not, it will behave like a point function $\mathsf{PF}_{\hat{x}_1}$ (for a secret random $\hat{x}_1$) in this call and all successive calls. Observe now that, from the view of the receiver, the protocol Π_{PF} always implements a correct point function. However, a simulator $\mathcal{S}_{\mathsf{S}}$ must decide if it inputs $\hat{x}_0$ or $\hat{x}_1$ into the ideal functionality $\mathcal{F}_{\mathrm{PF}}$ without knowing the first input x of the receiver.

The environment $\mathcal{Z}^*$ can now distinguish between real and ideal as follows. It first flips an unbiased coin. If the outcome is 0, it provides $x = \hat{x}_0$ as input to the receiver, otherwise it provides $x = \hat{x}_1$ to the receiver. If $\mathcal{Z}^*$ is connected to the real experiment, then the output of the receiver behaves according to the specification of $\mathcal{F}^*$. In the ideal experiment, however, the output of the receiver behaves either like $\mathsf{PF}_{\hat{x}_0}$ or $\mathsf{PF}_{\hat{x}_1}$ (or completely different). Thus, with probability at least $\frac{1}{2}$, $\mathcal{Z}^*$ notices a difference. This contradicts the UC-security of Π_{PF}. □

These impossibilities justify that we do not consider statistically secure protocols based on resettable hardware, and also show that our constructions in the following sections are optimal with respect to the techniques used and the amount of tokens sent.

5.3 Computationally Secure Two-Party Computation

We will now show how to achieve computationally secure two-party computation using resettable tamper-proof hardware. Instead of building the required protocols from scratch, we build a compiler that transforms any protocol based on a stateful hardware token into a protocol that uses a resettable token. While our approach has the drawback that we need to introduce additional interaction into the protocol, the main benefit is that we can use existing and very efficient protocols based on

stateful hardware. Our compiler combined with existing results for stateful hardware implies protocols based on resettable hardware that are more efficient than known direct constructions in the resettable hardware model (cf. Section 5.3.4).

Let Π_s be a protocol that UC-realizes a functionality $\mathcal{F}$ based on a stateful tamper-proof hardware token. We need to make some assumptions on the structure of Π_s. W.l.o.g the protocol can be divided into the following phases.

1. Setup phase in which the token issuer sends a token program T to the ideal functionality $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$.

2. Communication phase between the sender and the receiver.

3. Invocation phase of the token by the receiver.

We will also assume that the input protocol Π_s has dummy messages `query` and `ack`, where an honest receiver sends the message `query` to the sender before querying the token, and waits until the sender replies with `ack` before proceeding. We do not require a corrupted receiver to send the `query`-message before querying the token; therefore any protocol Π_s can be converted into this form.

The main idea of our construction is as follows. We use the structure of the underlying protocol and add an authentication step in which the token receiver first has to have his token inputs authenticated by the token sender. It is important that the sender remains oblivious of the receiver's input. Then the receiver queries the token with his input and the authentication information to obtain his result. Care has to be taken not to introduce a channel from the token issuer to the token by the authentication.

Our general approach can be seen as transferring the state (that the token has to keep) to the token sender. However, we want to emphasize that there is no direct communication between the token and its creator, and the only state that the token issuer keeps is a counter for the number of messages. Imagine an OT protocol where the token receiver first has to provide the token issuer with his input (in an oblivious way), such that he can obtain an authentication on it. He then presents the authenticated input to the token, which outputs the corresponding OT output. For a second receiver input, the token issuer will deny the authentication and the token cannot be queried on both receiver inputs. Our solution thus does not prevent a malicious token receiver from carrying out reset attacks, but he cannot change his input after the reset, thus the attack will not yield any new information for him.

Our extensions of the protocol Π_s can be summarized as follows.

Addition to Phase 1: We enhance the functionality T by a verification step which ensures that only authenticated messages are accepted.

Addition to Phase 2: We do not alter the communication of sender and receiver in the protocol Π_s, but we need to introduce one additional round of interaction to authenticate the input of the receiver.

Addition to Phase 3: The token invocation requires an additional input (and, depending on the compiler, additional interaction) compared to Π_s.

In the following, we present two compilers. The first compiler as presented in Section 5.3.1 makes use of non-black-box techniques, which by Lemma 5.1 cannot be

avoided without making further assumptions. Additionally, we present a compiler in Section 5.3.2 that uses only a few UC seed OTs to authenticate the input, and removes the need for the rather inefficient resettably-sound zero-knowledge part of the first compiler. In Section 5.3.3, we present some optimizations of our compilers to increase the efficiency when several inputs have to be given to the resettable token, and in Section 5.3.4 we discuss the implications of using existing protocols based on stateful hardware with our compilers.

5.3.1 Compiler from Non-Black-Box Techniques

Our compiler $\mathcal{C}_{ZK}$ (cf. Figure 5.4) makes use of resettably-sound zero-knowledge arguments, for which non-black-box simulation is required. As we already shown in Lemma 5.1, an approach based on non-black-box techniques is necessary in any case. The compiler transforms a protocol Π_{s} in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$-hybrid model into a protocol Π_{r} in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$-hybrid model. It alters the underlying stateful protocol Π_{s} as follows. Before the execution of Π_{s}, a signature key pair $(\mathsf{vk}, \mathsf{sgk})$ for an EUF-CMA-secure signature scheme SIG and a key k for a statistically binding commitment scheme COM (cf. Definition 2.8) are created by the sender and sent to the receiver. Then the setup of Π_{s} is carried out. When the token code of the underlying protocol is sent to the wrapper functionality $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$, the sender chooses a seed s for a pseudorandom function and constructs a new token that checks via a resettably-sound zero-knowledge proof if the receiver knows a signature on a commitment to his input.

During the token invocation of the original protocol, we enhance the communication of the token and the receiver as follows. Instead of just sending an input inp to the token, the receiver first commits to its input inp and sends the commitment to the sender. The sender then computes a signature σ on the commitment c and sends the signature to the receiver. Now the receiver checks if the signature is valid and queries the token with his input. Additionally, the receiver starts a resettably-sound zero-knowledge argument of knowledge to prove to the token that he knows a signature on a commitment to the input. The token starts the verifier program of the resettably-sound zero-knowledge argument of knowledge and returns the output out of the underlying functionality on input inp if the verifier accepts.

We stress that it is essential that the token cannot learn the commitment c, otherwise both token and sender have correlated information which cannot be simulated, as token and sender might abort upon seeing a commitment with certain properties (e.g. first 10 bits are zero).

Theorem 5.1. *The compiled protocol $\Pi_{\mathrm{r}} \leftarrow \mathcal{C}_{ZK}(\Pi_{\mathrm{s}})$ with $\mathcal{C}_{ZK}$ as shown in Figure 5.4 UC-realizes $\mathcal{F}$ in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$-hybrid model, given that Π_{s} UC-realizes $\mathcal{F}$ in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$-hybrid model and one-way functions exist.*

We split the proof of Theorem 5.1 in two parts and prove security against a malicious sender in Lemma 5.3 and against a malicious receiver in Lemma 5.4. The proof strategy is to construct an adversary $\mathcal{A}'$ against Π_{s} in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$-hybrid model from an adversary $\mathcal{A}$ on the compiled protocol Π_{r} in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$-hybrid model, such that $\mathcal{A}$ cannot distinguish with which ideal functionality he is interacting. By the UC-security of the protocol Π_{s}, the protocol is robust against such an adversary, which implies that the compiled protocol is UC-secure as well.

Compiler $\mathcal{C}_{ZK}$

Let $\mathcal{F}$ be a two-party UC-functionality. Let COM denote a 2-message statistically binding commitment scheme and SIG an EUF-CMA-secure signature scheme. Let (P, V) be a rsZKAoK for the NP-language $\mathcal{L} = \{(\mathsf{vk}, k, \mathsf{inp}) \,|\, \exists\, \sigma, c, d : \mathsf{SIG.Verify}(\mathsf{vk}, c, \sigma) = 1 \wedge \mathsf{COM.Open}(k, c, d, \mathsf{inp}) = 1\}$. Further let PRF be a pseudorandom function and $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$ be a resettable hardware token.

Input:

A protocol Π_{s} UC-realizing $\mathcal{F}$ in the $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$-hybrid model.

Output:

A protocol Π_{r} UC-realizing $\mathcal{F}$ in the $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$-hybrid model.

Setup phase:

(before execution of Π_{s})

1. Sender: Generate a key pair $(\mathsf{sgk}, \mathsf{vk}) \leftarrow \mathsf{SIG.KeyGen}(\kappa)$ and choose a key $k \leftarrow \{0,1\}^{\kappa}$ for COM uniformly at random. Send (vk, k) to the receiver.
2. Receiver: Upon receiving a message (vk, k) from the sender, store vk and k.

Rewriting the token code:

3. Sender: Once the sender of Π_{s} is supposed to input a token code T into $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$, do the following.
   - Sample a seed $s \leftarrow \{0,1\}^{\kappa}$ for the pseudorandom function PRF.
   - Program a token code T' which upon receiving a message $(\texttt{input}, \mathsf{inp})$ from the receiver sets up a verifier V with input $(\mathsf{vk}, k, \mathsf{inp})$, random-tape $\mathsf{PRF}(\mathsf{inp})$ and runs V. It forwards the messages sent by V to the receiver and vice versa. If V rejects, T' aborts. If V accepts, T' starts the execution of T with input inp and forwards T's output to the receiver.
   - Send $(\texttt{create}, \mathsf{T}')$ to $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$.

Token invocation phase:

4. Receiver: Let inp be the receiver's input for the token. Compute $(c, d) \leftarrow \mathsf{COM.Commit}(k, \mathsf{inp})$ and send $(\texttt{query}, c)$ to the sender.
5. Sender: Upon receiving a message $(\texttt{query}, c)$ from the receiver, compute $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, c)$. Send $(\texttt{ack}, \sigma)$ to the receiver.
6. Receiver: Upon receiving a message $(\texttt{ack}, \sigma)$ from the sender, check if $\mathsf{SIG.Verify}(\mathsf{vk}, c, \sigma) = 1$, if not abort. Otherwise send $(\texttt{input}, \mathsf{inp})$ to the token. Setup a prover P with input $(\mathsf{vk}, \mathsf{inp}, \sigma, c, d)$ and run P. Forward the messages sent by P to the token and vice versa. Continue the receiver's computation from Π_{s} once the token outputs out.

Figure 5.4: Compiler from non-black-box techniques.

Please note that all required building blocks of our construction (i.e. statistically binding commitments, resettably-sound zero-knowledge arguments of knowledge,

EUF-CMA signatures and pseudorandom functions) can be constructed from one-way functions (cf. the corresponding paragraphs in Chapter 2).

Lemma 5.3. *The compiled protocol* $\Pi_{\mathrm{r}} \leftarrow \mathcal{C}_{ZK}(\Pi_{\mathrm{s}})$ *with* $\mathcal{C}_{ZK}$ *as shown in Figure 5.4 UC-realizes* $\mathcal{F}$ *in the* $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$*-hybrid model against a corrupted sender, given that* Π_{s} *UC-realizes* $\mathcal{F}$ *in the* $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$*-hybrid model,* COM *is computationally hiding and* (P, V) *is computationally zero-knowledge.*

Proof. We want to show that for every PPT-environment $\mathcal{Z}$, $\mathsf{Real}_{\Pi_{\mathrm{r}}}^{\mathcal{A}_{\mathsf{S}}}(\mathcal{Z})$ and $\mathsf{Ideal}_{\mathcal{F}}^{\mathcal{S}_{\mathsf{S}}}(\mathcal{Z})$ are computationally indistinguishable. Since Π_{s} is UC-secure, there exists a simulator $\mathcal{S}_{\mathsf{S}}$ such that $\mathsf{Real}_{\Pi_{\mathrm{s}}}^{\mathcal{A}'_{\mathsf{S}}}(\mathcal{Z}) \underset{c}{\approx} \mathsf{Ideal}_{\mathcal{F}}^{\mathcal{S}_{\mathsf{S}}}(\mathcal{Z})$. It remains to show that there exists an adversary-simulator $\mathcal{A}'_{\mathsf{S}}$, such that $\mathsf{Real}_{\Pi_{\mathrm{r}}}^{\mathcal{A}_{\mathsf{S}}}(\mathcal{Z})$ and $\mathsf{Real}_{\Pi_{\mathrm{s}}}^{\mathcal{A}'_{\mathsf{S}}}(\mathcal{Z})$ are computationally indistinguishable. Figure 5.5 shows an adversary-simulator $\mathcal{A}'_{\mathsf{S}}$ that uses $\mathcal{A}_{\mathsf{S}}$.

Adversary-Simulator $\mathcal{A}'_{\mathsf{S}}$

- Simulate $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$ for $\mathcal{A}_{\mathsf{S}}$ and obtain T^*.
- (Setup) Once $\mathcal{A}_{\mathsf{S}}$ sends a message $(\mathtt{setup}, \mathsf{vk}, k)$ in Step 1, store vk and k.
- (Rewriting the token code) In Step 3, construct from T^* a token-code $\mathsf{T}^\dagger$ with the following functionality.
  - Upon receiving a message $(\mathtt{input}, \mathsf{inp})$ from the receiver, run T^* with input $(\mathtt{input}, \mathsf{inp})$ up to the point when T^* expects the proof. Halt the computation of T^* and construct a corrupted verifier V^* from T^*. Run the non-black-box simulator Sim of (P, V) with input $(\mathsf{V}^*, \mathsf{vk}, \mathsf{inp})$ and obtain a new state for T^*. Proceed with the simulation of T^* from this point and let out be the result. Send out to the receiver.

  Send $(\mathtt{create}, \mathsf{T}^\dagger)$ to $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$.
- (Token invocation) Upon receiving a message $\mathtt{query}$ from the receiver in Step 5, compute $(\hat{c}, \hat{d}) \leftarrow \mathsf{COM.Commit}(k, 0)$. Send $(\mathtt{query}, \hat{c})$ to $\mathcal{A}_{\mathsf{S}}$. Let $(\mathtt{ack}, \sigma^*)$ be the output of $\mathcal{A}_{\mathsf{S}}$. Check if it holds $\mathsf{SIG.Verify}(\mathsf{vk}, \hat{c}, \sigma^*) = 1$, if not abort. Otherwise send $\mathtt{ack}$ to the receiver.

Figure 5.5: Adversary-simulator for a corrupted sender in the compiler $\mathcal{C}_{ZK}$.

Let $\mathcal{Z}$ be any PPT-environment. We will prove indistinguishability of $\mathsf{Real}_{\Pi_{\mathrm{r}}}^{\mathcal{A}_{\mathsf{S}}}(\mathcal{Z})$ and $\mathsf{Real}_{\Pi_{\mathrm{s}}}^{\mathcal{A}'_{\mathsf{S}}}(\mathcal{Z})$ by a series of hybrid experiments.

Experiment 0: Simulator $\mathcal{S}_0$ simulates $\mathsf{Real}_{\Pi_{\mathrm{r}}}^{\mathcal{A}_{\mathsf{S}}}$.

Experiment 1: Identical to Experiment 0, except that during invocation of the token, $\mathcal{S}_1$ does not setup and run a prover P with input $(\mathsf{vk}, \mathsf{inp}, \sigma^*, \hat{c}, \hat{d})$, but instead runs the non-black-box simulator Sim on the verifier V^* (as described in Figure 5.5) and uses the output as a new state for T^*.

Experiment 2: Identical to Experiment 1, except that the commitment $\hat{c}$ is computed by $(\hat{c}, \hat{d}) \leftarrow \mathsf{COM.Commit}(k, 0)$ instead of $(c, d) \leftarrow \mathsf{COM.Commit}(k, \mathsf{inp})$. From the view of $\mathcal{Z}$, this is identical to $\mathsf{Real}_{\Pi_{\mathrm{s}}}^{\mathcal{A}'_{\mathsf{S}}}$.

Indistinguishability of Experiment 0 and Experiment 1 follows directly from the computational zero-knowledge property of the argument system (P, V). If the commitment scheme COM is computationally hiding, Experiment 1 and Experiment 2 are computationally indistinguishable from the view of $\mathcal{Z}$ as well. This can be established by a simple reduction, where a $\mathcal{Z}$ distinguishing the two experiments can be used to to break the hiding-property of COM. □

In the following, we prove the second part of Theorem 5.1, namely the security against a corrupted receiver. This is the more demanding case since we have to show that the receiver will not manage to send two distinct inputs into the token that are evaluated.

Lemma 5.4. *The compiled protocol* $\Pi_{\mathrm{r}} \leftarrow \mathcal{C}_{ZK}(\Pi_{\mathrm{s}})$ *with* $\mathcal{C}_{ZK}$ *as shown in Figure 5.4 UC-realizes* $\mathcal{F}$ *in the* $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$*-hybrid model against a corrupted receiver, given that* Π_{s} *UC-realizes* $\mathcal{F}$ *in the* $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$*-hybrid model,* COM *is statistically binding,* PRF *is a pseudorandom function,* SIG *is EUF-CMA-secure and* (P, V) *is an argument of knowledge.*

Proof. We want to show that for every PPT-environment $\mathcal{Z}$, $\mathsf{Real}_{\Pi_{\mathrm{r}}}^{\mathcal{A}_{\mathsf{R}}}(\mathcal{Z})$ and $\mathsf{Ideal}_{\mathcal{F}}^{\mathcal{S}_{\mathsf{R}}}(\mathcal{Z})$ are computationally indistinguishable. Again, since Π_{s} is UC-secure, there exists a simulator $\mathcal{S}_{\mathsf{R}}$ such that $\mathsf{Real}_{\Pi_{\mathrm{s}}}^{\mathcal{A}'_{\mathsf{R}}}(\mathcal{Z}) \underset{c}{\approx} \mathsf{Ideal}_{\mathcal{F}}^{\mathcal{S}_{\mathsf{R}}}(\mathcal{Z})$. It remains to show that there exists an adversary-simulator $\mathcal{A}'_{\mathsf{R}}$ such that $\mathsf{Real}_{\Pi_{\mathrm{r}}}^{\mathcal{A}_{\mathsf{R}}}(\mathcal{Z})$ and $\mathsf{Real}_{\Pi_{\mathrm{s}}}^{\mathcal{A}'_{\mathsf{R}}}(\mathcal{Z})$ are computationally indistinguishable. Figure 5.6 shows an adversary-simulator $\mathcal{A}'_{\mathsf{R}}$ that uses $\mathcal{A}_{\mathsf{R}}$.

Adversary-Simulator $\mathcal{A}'_{\mathsf{R}}$

- Simulate $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$ for $\mathcal{A}_{\mathsf{R}}$.
- (Setup) In Step 1, generate a key pair $(\mathsf{sgk}, \mathsf{vk}) \leftarrow \mathsf{SIG.KeyGen}(\kappa)$ and choose $k \leftarrow \{0,1\}^{\kappa}$ uniformly at random. Send (vk, k) to $\mathcal{A}_{\mathsf{R}}$. Additionally, setup a random oracle H.
- (Rewriting the token code) Simulated straightforwardly according to $\mathcal{C}_{ZK}$.
- (Token invocation)
  - Upon receiving a message $(\mathtt{query}, c^*)$ from $\mathcal{A}_{\mathsf{R}}$ in Step 5, compute $\hat{\sigma} \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, c)$. Send a message $(\mathtt{query})$ to the sender and let $(\mathtt{ack})$ be the answer. Send $(\mathtt{ack}, \hat{\sigma})$ to $\mathcal{A}_{\mathsf{R}}$.
  - Upon receiving a message $(\mathtt{input}, \mathsf{inp})$ from $\mathcal{A}_{\mathsf{R}}$ in Step 6, setup a verifier program V with input $(\mathsf{vk}, \mathsf{inp})$ and random tape $\mathsf{H}(\mathsf{inp})$, and run V. Forward the messages between V and $\mathcal{A}_{\mathsf{R}}$ and abort if V rejects. If V accepts even though a tuple $(\mathsf{inp}', \mathsf{out}')$ has been stored with $\mathsf{inp}' \neq \mathsf{inp}$, abort. Further, if $\mathsf{inp}' = \mathsf{inp}$, output out'. If no tuple was stored, send inp to $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$ to obtain out and store the tuple $(\mathsf{inp}, \mathsf{out})$. Send out to $\mathcal{A}_{\mathsf{R}}$.

Figure 5.6: Adversary-simulator for a corrupted receiver in the compiler $\mathcal{C}_{ZK}$.

Consider the following series of hybrid experiments.

Experiment 0: Simulator $\mathcal{S}_0$ simulates $\mathsf{Real}_{\Pi_{\mathrm{r}}}^{\mathcal{A}_{\mathsf{R}}}$.

Experiment 1: Identical to Experiment 0, except that the simulator $\mathcal{S}_1$ replaces the pseudorandom-function PRF by a random oracle H.

Experiment 2: Identical to Experiment 1, except for the following. $\mathcal{S}_2$ checks—after V accepts—if a tuple $(\mathsf{inp}', \mathsf{out}')$ has already been stored. If so and $\mathsf{inp}' \neq \mathsf{inp}$, it aborts. Moreover, if no such tuple exists, it will store $(\mathsf{inp}, \mathsf{out})$, where out is the output of the token. From the view of $\mathcal{Z}$, this is identical to $\mathsf{Real}^{\mathcal{A}'_{\mathsf{R}}}_{\Pi_{\mathrm{s}}}$.

Computational indistinguishability of Experiment 0 and Experiment 1 follows straightforwardly by the pseudorandomness property of the pseudorandom function PRF. Showing the computational indistinguishability of Experiment 1 and Experiment 2 is more involved.

We claim that Experiment 1 and Experiment 2 are computationally indistinguishable, provided that the argument system (P, V) fulfills the computational resettable soundness property, the commitment scheme COM is statistically binding and the signature scheme SIG is EUF-CMA secure.

Clearly, if $\mathcal{S}_2$ does not abort after V accepts, the view of $\mathcal{Z}$ is identical in Experiment 1 and Experiment 2. We will thus show that V aborts at most with negligible probability, establishing indistinguishability of Experiment 1 and Experiment 2.

Since the commitment scheme COM is statistically binding, the event that there exist two distinct unveils (c, d_1, inp_1) and (c, d_2, inp_2) with $\mathsf{COM.Open}(k, c, d_1, \mathsf{inp}_1) = 1$ and $\mathsf{COM.Open}(k, c, d_2, \mathsf{inp}_2) = 1$ has only negligible probability (over the choice of k). We can thus assume that each commitment c has a unique unveil (c, d, inp).

Assume now that the probability that $\mathcal{S}_2$ aborts after V accepts is non-negligible. We distinguish two cases:

1. The probability ϵ that $\mathcal{A}_{\mathsf{R}}$ successfully proves a false statement in one of the invocations of (P, V) is *non-negligible*.

2. The probability ϵ that $\mathcal{A}_{\mathsf{R}}$ successfully proves a false statement in one of the invocations of (P, V) is *negligible*.

In the first case, we can use a distinguishing $\mathcal{Z}$ to construct a corrupted prover P^* that breaks the soundness property of the argument system (P, V). P^* simulates $\mathcal{S}_1$ faithfully until the argument system (P, V) is started. Then, P^* announces the statement $(\mathsf{vk}, \mathsf{inp})^*$ and forwards all messages sent by $\mathcal{A}_{\mathsf{R}}$ to his own verifier V and vice versa. Clearly, from $\mathcal{A}_{\mathsf{R}}$'s (and thus $\mathcal{Z}$'s) view, $\mathcal{S}_1$ and P^*'s simulation are identically distributed. Thus, the chance of P^* successfully proving a false statement is at least ϵ, contradicting the soundness property of (P, V).

In the second case, we will argue that $\mathcal{A}_{\mathsf{R}}$ must be able to successfully forge a signature σ^* for a message c^*, contradicting the EUF-CMA security of SIG. We will therefor use $\mathcal{A}_{\mathsf{R}}$ to construct an adversary $\mathcal{B}$ that breaks the EUF-CMA property of SIG with non-negligible probability, leading to the desired contradiction. Let Ext be an extractor for the argument of knowledge (P, V). $\mathcal{B}$ simulates $\mathcal{S}_2$ faithfully except for the following. Instead of generating $(\mathsf{sgk}, \mathsf{vk})$ itself, it will use vk as provided by the EUF-CMA experiment. $\mathcal{B}$ uses $\mathcal{A}_{\mathsf{R}}$ and $\mathcal{Z}$ to construct a malicious prover P^* which simply consists of continuing the computation of $\mathcal{A}_{\mathsf{R}}$ and $\mathcal{Z}$ until the argument system terminates. $\mathcal{B}$ now runs the extractor Ext on P^* and obtains a witness (σ^*, c^*, d^*) for a statement $(\mathsf{vk}, \mathsf{inp}^*)$. If $\mathsf{SIG.Verify}(\mathsf{vk}, c^*, \sigma^*) = 1$, then $\mathcal{B}$ outputs the forge (c^*, σ^*) to the EUF-CMA experiment. Otherwise, it outputs $\perp$.

Clearly, from the view of $\mathcal{Z}$, both $\mathcal{S}_2$ and $\mathcal{B}$'s simulation are identically distributed. Since we assume that $\mathcal{S}_2$ aborts with non-negligible probability and P^* proves a true statement (except with negligible probability) the extractor Ext returns a witness (σ^*, c^*, d^*) with non-negligible probability. As we conditioned on the event that $\mathcal{S}_2$ aborts and the commitment c^* has a unique unveil, (c^*, σ^*) must be a valid forge with non-negligible probability. This, however, contradicts the EUF-CMA security of SIG, which concludes the proof. □

Remark. The above compiler can easily be extended to allow for multiple messages. In each step of the token invocation the token receiver has to query the sender on a commitment and provide a proof to the token that this commitment was signed by the sender. To allow multiple messages, a counter is added for each message such that the receiver cannot query the token "out-of-sync". If the token is reset, its counter will not match the counter of the sender and thus the token will abort.

5.3.2 Compiler from Seed-OTs

In some cases non-black-box techniques are not desired, i.e. for efficiency purposes. We present a second compiler $\mathcal{C}_{OT}$ in Figure 5.7 that is based on a small number of UC-secure seed-OTs. While at first sight this might seem to be a strong assumption, consider the following. Apart from being a common assumption in areas such as secure MPC [Yao82, GMW87, IPS08], a number of UC-secure OT protocols from resettable tamper-proof hardware have been proposed [GIS$^+$10, CKS$^+$14]. Thus, in the context of tamper-proof hardware, the assumption of seed-OTs is not far-fetched at all. Our compiler makes no additional assumptions, meaning the security of the resulting protocol solely relies on the input protocol and the realization of the OTs. For a detailed discussion concerning the implications and restrictions, see Section 5.3.4.

Similarly to the compiler from the previous section, the compiler $\mathcal{C}_{OT}$ adds a step to the underlying protocol Π_{s} that authenticates the token input. This time, the authentication is done using UC-secure OTs. Before the execution of Π_{s}, the token sender creates two random bit strings (s_0^i, s_1^i) for every bit i of the message inp that the receiver will input into the token. During the setup, the receiver obtains one of these random strings, namely $s^i_{\mathsf{inp}(i)}$, for each of his input bits. Thus, the receiver is committed to his input, while the sender does not learn anything about it. Additionally, the input is authenticated, because intuitively, the receiver cannot obtain a label $s^i_{1-\mathsf{inp}(i)}$ not pertaining to his input.

The token functionality is extended by storing all values $\left((s_0^1, s_1^1), \ldots, (s_0^\ell, s_1^\ell)\right)$ that the sender created. When the token is invoked on input $\left(\mathsf{inp}, (s^1_{j_1}, \ldots, s^\ell_{j_\ell})\right)$, the tokens checks that the authentication values are consistent with the input values of the OTs. If that is the case, the token will evaluate the underlying token functionality on inp and forward the output out.

Theorem 5.2. *The compiled protocol $\Pi_{\mathrm{r}} \leftarrow \mathcal{C}_{OT}(\Pi_{\mathrm{s}})$ with $\mathcal{C}_{OT}$ as shown in Figure 5.7 statistically UC-realizes $\mathcal{F}$ in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}/\mathcal{F}_{\mathrm{MOT}}$-hybrid model, given that Π_{s} UC-realizes $\mathcal{F}$ in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$-hybrid model.*

Proof. **Corrupted sender.** We want to show that $\mathcal{Z}$, $\mathsf{Real}^{\mathcal{A}_{\mathsf{S}}}_{\Pi_{\mathrm{r}}}(\mathcal{Z})$ and $\mathsf{Ideal}^{\mathcal{S}_{\mathsf{S}}}_{\mathcal{F}}(\mathcal{Z})$ are statistically indistinguishable for every environment. Since Π_{s} is UC-secure,

Compiler $\mathcal{C}_{OT}$

Let $\mathcal{F}$ be a two-party UC-functionality. Let $\ell = |m|$ be the input length of the token receiver's message m to the token in the protocol Π_{s}. Further let $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$ be a resettable hardware wrapper functionality and $\mathcal{F}_{\mathrm{MOT}}$ an OT functionality for at least ℓ sessions.

Input:

A protocol Π_{s} UC-realizing $\mathcal{F}$ in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$-hybrid model.

Output:

A protocol Π_{r} UC-realizing $\mathcal{F}$ in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$-hybrid model.

Setup phase:

(before execution of Π_{s})

1. Sender: Create 2ℓ random strings $S = ((s_0^1, s_1^1), \ldots, (s_0^\ell, s_1^\ell)), s_j^i \leftarrow \{0,1\}^\kappa$ and input them into the ℓ $\mathcal{F}_{\mathrm{MOT}}$-functionalities.
2. Receiver: Input $c^i_{\mathsf{inp}(i)}$ for $i \in \{1, \ldots, \ell\}$ into the corresponding $\mathcal{F}_{\mathrm{MOT}}$ and obtain $(s^1_{j_1}, \ldots, s^\ell_{j_\ell}), j_i \in \{0,1\}$.

Rewriting the token-code:

3. Sender: Once the sender of Π_{s} is supposed to input a token code T into $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$, do the following.
   - Construct a token-code T' which, upon receiving a message $(\mathtt{input}, \mathsf{inp}, (s^1_{j_1}, \ldots, s^\ell_{j_\ell})), j \in \{0,1\}$ from the receiver, checks if $s_j^i \in S$ for all $i \in \{1, \ldots, \ell\}$. If this is the case, it continues the execution of T with input inp and forwards whatever T outputs.
   - Send $(\mathtt{create}, \mathsf{T}')$ to $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$.

Token invocation:

4. Receiver: Send a message `query` to the sender.
5. Sender: Reply with a message `ack`.
6. Receiver: Send the message $(\mathtt{input}, \mathsf{inp}, (s^1_{j_1}, \ldots, s^\ell_{j_\ell}))$ to the token and continue the computation of the receiver from Π_{s} once the token outputs `out`.

Figure 5.7: Compiler from seed-OTs.

there exists a simulator $\mathcal{S}_{\mathsf{S}}$ such that $\mathsf{Real}_{\Pi_{\mathrm{s}}}^{\mathcal{A}'_{\mathsf{S}}}(\mathcal{Z}) \underset{s}{\approx} \mathsf{Ideal}_{\mathcal{F}}^{\mathcal{S}_{\mathsf{S}}}(\mathcal{Z})$. It remains to show that there exists an adversary-simulator $\mathcal{A}'_{\mathsf{S}}$ such that $\mathsf{Real}_{\Pi_{\mathrm{r}}}^{\mathcal{A}_{\mathsf{S}}}(\mathcal{Z})$ and $\mathsf{Real}_{\Pi_{\mathrm{s}}}^{\mathcal{A}'_{\mathsf{S}}}(\mathcal{Z})$ are indistinguishable. Figure 5.8 shows an adversary-simulator $\mathcal{A}'_{\mathsf{S}}$ that uses $\mathcal{A}_{\mathsf{S}}$.

The distributions $\mathsf{Real}_{\Pi_{\mathrm{r}}}^{\mathcal{A}_{\mathsf{S}}}(\mathcal{Z})$ and $\mathsf{Real}_{\Pi_{\mathrm{s}}}^{\mathcal{A}'_{\mathsf{S}}}(\mathcal{Z})$ are identically distributed once $\mathcal{A}'_{\mathsf{S}}$ obtains all labels that are input into $\mathcal{F}_{\mathrm{MOT}}$ because a normal protocol run is simulated with these labels.

Corrupted receiver. We want to show that for every environment $\mathcal{Z}$, $\mathsf{Real}_{\Pi_{\mathrm{r}}}^{\mathcal{A}_{\mathsf{R}}}(\mathcal{Z})$ and $\mathsf{Ideal}_{\mathcal{F}}^{\mathcal{S}_{\mathsf{R}}}(\mathcal{Z})$ are statistically indistinguishable. Again, since Π_{s} is UC-secure,

Adversary-Simulator $\mathcal{A}'_{\mathsf{S}}$

- Simulate $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$ and $\mathcal{F}_{\text{MOT}}$ for $\mathcal{A}_{\mathsf{S}}$ and obtain T^* and all inputs $((s_0^1, s_1^1), \ldots, (s_0^\ell, s_1^\ell))$.
- (Setup) Simulated straightforwardly according to $\mathcal{C}_{OT}$.
- (Rewriting the token code) In Step 3, proceed as follows.
  - Construct from T^* a token-code $\mathsf{T}^\dagger$ with the following functionality. Upon receiving a message $(\mathtt{input}, \mathsf{inp})$ from the receiver, select $(s^1_{\mathsf{inp}(1)}, \ldots, s^\ell_{\mathsf{inp}(\ell)})$ according to the input and run T^* on the input $(\mathtt{input}, \mathsf{inp}, (s^1_{\mathsf{inp}(1)}, \ldots, s^\ell_{\mathsf{inp}(\ell)}))$. Let out be the result. Send out to the receiver.
  - Send $(\mathsf{create}, \mathsf{T}^\dagger)$ to $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$.
- (Token invocation) Simulated straightforwardly according to $\mathcal{C}_{OT}$.

Figure 5.8: Adversary-simulator for a corrupted sender in the compiler $\mathcal{C}_{OT}$.

there exists a simulator $\mathcal{S}_{\mathsf{R}}$ such that $\mathsf{Real}_{\Pi_{\mathrm{s}}}^{\mathcal{A}'_{\mathsf{R}}}(\mathcal{Z}) \underset{s}{\approx} \mathsf{Ideal}_{\mathcal{F}}^{\mathcal{S}_{\mathsf{R}}}(\mathcal{Z})$. It remains to show that there exists an adversary-simulator $\mathcal{A}'_{\mathsf{R}}$ such that $\mathsf{Real}_{\Pi_{\mathrm{r}}}^{\mathcal{A}_{\mathsf{R}}}(\mathcal{Z})$ and $\mathsf{Real}_{\Pi_{\mathrm{s}}}^{\mathcal{A}'_{\mathsf{R}}}(\mathcal{Z})$ are indistinguishable. Figure 5.9 shows an adversary-simulator $\mathcal{A}'_{\mathsf{R}}$ that uses $\mathcal{A}_{\mathsf{R}}$.

Adversary-Simulator $\mathcal{A}'_{\mathsf{R}}$

- Simulate $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$ and $\mathcal{F}_{\text{MOT}}$ for $\mathcal{A}_{\mathsf{R}}$ and obtain the choice bits $\hat{c}_i, i \in \{1, \ldots, \ell\}$.
- (Setup) In Step 2, derive the input $\widehat{\mathsf{inp}}$ by concatenating the ℓ choice-bits.
- (Rewriting the token code) Simulated straightforwardly according to $\mathcal{C}_{OT}$.
- (Token invocation)
  - Upon receiving a message $(\mathtt{query})$ from $\mathcal{A}_{\mathsf{R}}$ in Step 5, send $(\mathtt{query})$ to the sender and let $(\mathtt{ack})$ be the answer. Send $(\mathtt{ack})$ to $\mathcal{A}_{\mathsf{R}}$.
  - Upon receiving a message $(\mathtt{input}, \mathsf{inp}^*)$ from $\mathcal{A}_{\mathsf{R}}$ in Step 6, check if $\widehat{\mathsf{inp}} = \mathsf{inp}^*$ and $s_j^i \in S \,\forall\, i$, if not abort. Send inp to $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$ and return the result out to $\mathcal{A}_{\mathsf{R}}$.

Figure 5.9: Adversary-simulator for a corrupted receiver in the compiler $\mathcal{C}_{OT}$.

The only difference between $\mathsf{Real}_{\Pi_{\mathrm{r}}}^{\mathcal{A}_{\mathsf{R}}}(\mathcal{Z})$ and $\mathsf{Real}_{\Pi_{\mathrm{s}}}^{\mathcal{A}'_{\mathsf{R}}}(\mathcal{Z})$ is the abort of $\mathcal{A}'_{\mathsf{R}}$ in case $\widehat{\mathsf{inp}} \neq \mathsf{inp}^*$. For this event to happen, $\mathcal{A}_{\mathsf{R}}$ has to guess a string $s_j^i \in S$ of length κ for any $i \in \{1, \ldots, \ell\}, j \in \{0, 1\}$. The probability for this event is obviously negligible in the security parameter κ. □

5.3.3 Optimizations

Recall that the compiler $\mathcal{C}_{ZK}$ can be straightforwardly extended to allow for multiple messages between token and receiver. However, this would lead to an inefficient zero-knowledge proof for each message. While most protocols in the literature query the token only a small constant number of times, it is still desirable to minimize the

number of proofs. Also, to change the compiler $\mathcal{C}_{OT}$ such that it allows for more than a single authenticated input requires increasing the fixed amount of UC-secure OTs that we have at our disposal, which is expensive in most scenarios.

We will address these shortcomings for the case of non-adaptive queries. By non-adaptive queries, we mean queries that can be fixed in advance, i.e. the queries do not depend on other queries to the token. First, please note that Agrawal et al. [AAG+14] showed that non-adaptive queries cannot exist for every protocol in the stateful token model, so we cannot hope to solve the problem of several expensive rsZK proofs completely. Nevertheless, a very simple strategy to optimize the efficiency of non-adaptive queries is to just concatenate all i messages into a single message and have the sender authenticate this message. However, this needs a lot of OTs and also the amount of data that has to be sent to the sender is very large, whereby the proof becomes more inefficient.

A more refined solution to the problem is the following. Instead of using the normal token input as the message that shall be authenticated by the sender, the receiver computes a Merkle tree [Mer88] with all non-adaptive messages in the leaves. Then, the sender authenticates the root of the Merkle tree and the receiver only has to use the compiler for the root message. From there on, for each of the initial non-adaptive messages, he sends the path through the tree and the corresponding message to the token which can verify that the path is consistent with the root.

This improvement leads to a single message of small size during the authentication step of $\mathcal{C}_{ZK}$ and $\mathcal{C}_{OT}$, respectively. Now we have introduced a new drawback into our solution: the Merkle tree relies on collision resistant hash functions. Considering our initial goal to achieve a compiler using only one-way functions, we replace the Merkle tree with the recent construction of *sig-com trees* [CPS13] (cf. Definition 2.16). The sig-com trees can be seen as an interactive variant of Merkle trees, but require only one-way functions due to the interaction.

We will briefly sketch how sig-com trees can be applied to our scenario. In addition to the normal setup, the token sender creates a key pair $(\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathsf{SIG.KeyGen}(\kappa)$ and extends the token functionality as follows. Upon receiving $(\mathtt{sign}, x)$ the token returns $\mathsf{SIG.Sign}(\mathsf{sgk}, x)$, basically implementing a signature oracle. Further, upon receiving $(\mathtt{check}, \mathsf{path}, \mathsf{root})$, the token checks that path constitutes a valid path given the root root of a sig-com tree. The verification key vk is given to the token receiver. The rest of the compiler is carried out as described in the previous sections. During the protocol run, instead of directly sending the non-adaptive messages to the sender, the receiver first uses the resettable token to create a signature tree and verifies each obtained signature using vk. Since all inputs are committed to in advance of the oracle calls, the token does not learn the inputs. Then the rest of the protocol proceeds normally: The sender authenticates the root of the sig-com tree, and the receiver has to present a path through the sig-com tree for each of the non-adaptive messages.

Simulation of this enhancement against a corrupted sender is quite simple. Since the commitments on the receiver inputs are never opened (but only used in zero-knowledge arguments of knowledge), the simulator can still just pick all-zero inputs, then use the token to create a corresponding sig-com tree, and proceed as before. Our indistinguishability proofs for the original compilers just carry over; otherwise the commitments on the receiver inputs would not be hiding. If the receiver is corrupted, the binding property of the commitments on his inputs and the collision resistance of the sig-com tree guarantee that the token can still be queried only

with messages that were authenticated by the sender. It follows again that our indistinguishability proofs for the original compilers just carry over.

5.3.4 Implications

In this section, we will briefly discuss the implications of applying our compiler to existing protocols. The main benefit of our approach is that it allows to design efficient protocols based on stateful hardware tokens and then compile them to the resettable setting. This is typically easier than directly constructing protocols based on resettable hardware.

We want to focus on UC-secure OT protocols. Previous constructions based on resettable tokens were either dependent on the fact that several hardware tokens had to be exchanged [GIS+10, CKS+14] or relied on stronger computational assumptions like enhanced trapdoor permutations, e.g. by using [CGS08] in conjunction with [CLOS02] to obtain OT. All of these constructions rely on black-box techniques, and thus Lemma 5.1 directly implies that these limitations are inherent.

Forestalling the results from Section 5.4, it is possible to use our non-interactive two-party computation protocol based on a single hardware token, which makes use of non-black-box techniques. While this solution requires only a single hardware token, it cannot be based on one-way functions. The reason is that—even though our construction relies only on one-way functions—realizing a UC-secure OT in the CRS-hybrid model, e.g. [PVW08], requires stronger assumptions than one-way functions.

In the context of stateful tamper-proof hardware, however, very efficient and even statistically secure constructions of OT are known. Of particular interest for us is the protocol of Döttling et al. [DKMQ11, DKMQ12] that provides statistically UC-secure OT from a single hardware token. If we plug their protocol into our compiler $\mathcal{C}_{ZK}$, we obtain the most efficient OT protocol based on resettable hardware to date. In particular, it allows us to use the optimization presented in Section 5.3.3 for non-adaptive queries if we settle for a random OT in a first step. Standard techniques can then be used to derandomize the OTs [Bea95]. Thus our result implies secure two-party computation from a single resettable hardware token based solely on one-way functions.

Applying the compiler $\mathcal{C}_{OT}$ to OT protocols clearly does not yield strong improvements, because we already need OT for the compiler itself. Also, statistically secure OT from resettable hardware is impossible [GIMS10]. Considering the goal of using resettable hardware, the UC-secure seed-OTs are thus either only computationally secure and realized by a resettable hardware token, or based on an additional assumption. While this technique would also imply an OT extension, i.e. deriving many OTs from a few OTs, current OT extension protocols, e.g. [IPS08, HIKN08, NNOB12], are far more efficient than applying $\mathcal{C}_{OT}$ to OT protocols. Instead, in conjunction with a statistically secure OT from a different physical assumption (e.g. our OT-protocol based on bounded-resettable hardware from Section 6.3.3), $\mathcal{C}_{OT}$ allows a very efficient transformation of statistically secure protocols to the $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$-hybrid model.

5.4 Computationally Secure Non-Interactive Two-Party Computation

In this section, we present protocols that realize computationally secure non-interactive two-party computation based on resettable hardware. This is obviously not implied by the result in Section 5.3 because our compiler introduces additional interaction between the token sender and the token receiver. Before we outline our solutions, let us first clarify the meaning of *non-interactive* two-party computation in the setting that we consider.

The general setting for our solutions is a protocol between two parties, the token sender and the token receiver. The token sender will program a token with some functionality and send it to the receiver, possibly with an additional message. The token receiver will then perform a two-party computation with the token, but without any further interaction with the token issuer.

Compared to interactive solutions, by using a non-interactive and resettable solution, one cannot hope to realize all two-party funtionalities. Instead, it is only possible to realize *resettable functionalities*, i.e. functionalities that remain secure even if the function is reset. This directly rules out functionalities like deterministic OT, since a malicious receiver of the resettable OT could just query the functionality on input $b = 0$ to obtain s_0, then reset the functionality, and input $b = 1$ to obtain s_1. As the protocol is non-interactive and the token itself cannot keep a state, such an attack cannot be prevented. Random oblivious transfer, on the other hand, is still possible. Intuitively, the inputs of the resettable party can be derived from a commitment on the input of the non-resettable party (e.g. via a PRF), thus each input will yield a different instance of the OT protocol, unless the non-resettable party can break the binding property of the commitment and present two unveils for a single commitment.

Thus, as a first step, we define resettable functionalities in Section 5.4.1 and show how they can be UC-realized in the CRS-hybrid model. Given this result, we present two protocols that enhance a resettable hardware token such that it has access to a UC-CRS. The first protocol, presented in Section 5.4.2, has an interactive setup phase between the token issuer and the token receiver during which a CRS is generated. It requires the sender to issue only one resettable token. Given our impossibility result from Section 5.2, an interactive setup combined with a single token is the best one can hope for. Then, in Section 5.4.3, we present a completely non-interactive protocol, i.e. the token issuer programs two tokens and sends them to the receiver, without additional interaction. The receiver negotiates a *resettable CRS* with the tokens, which is optimal again with respect to the impossibilty from Section 5.2.

5.4.1 Resettable Functionalities in the UC-Framework

In this section, we will introduce resettable UC-functionalities and the ideal functionalities for resettable hardware tokens. We first provide the definition of resettable two-party UC-functionalities. Let M be a Turing machine. The resettable functionality $\mathcal{F}_{\mathrm{2PC}}^{\mathrm{res}}$ specified by M is defined in Figure 5.10. For the sake of readability, we omit session and message identifiers.

As already discussed by Goyal and Sahai [GS09], resettable functionalities do not

Functionality $\mathcal{F}_{\text{2PC}}^{\text{res}}$

Implicitly parametrized by a security parameter κ and a Turing machine M.

1. Upon receiving a message $(\texttt{sender}, x_1)$ from the sender, store x_1, write x_1 on M's input tape and run M until it halts. Store the state of M. Accept no further inputs by the sender.
2. Upon receiving $(\texttt{receiver}, x_2)$ from the receiver, write x_2 on M's input tape and run M starting from the most recent state until it halts. Store the state of M. Read the message y from M's output tape and send y to the receiver.
3. Reset (Adversarial receiver only) Upon receiving `reset` from the receiver, reset the Turing machine M to its initial state.

Figure 5.10: Ideal functionality for resettable two-party computation.

cover all important cryptographic functionalities. One obvious counterexample is OT where the sender security breaks down if the functionality is reset. Nevertheless, a large class of cryptographic functionalities can be realized. This includes e.g. signatures where the key remains hidden, but also scenarios like database privacy as considered by Dwork [Dwo06], where the curator algorithm hides the actual database entries.

Goyal and Sahai [GS09] present a compiler that allows to securely compute any resettable functionality between a resettable and a non-resettable party. Their approach is to rebuild the general MPC protocol of Goldreich et al. [GMW87] and replace all zero-knowledge proofs by resettable and resettably-sound zero-knowledge proofs. Additionally, the resettable party has to generate its random coins via a pseudorandom function applied to the protocol transcript, similar to [CGGM00]. This leads to a deterministic protocol for each new input.

In spirit of the work of Canetti et al. [CLOS02] who show that based on a CRS, every functionality can be UC-realized, we replace some of the building blocks in the compiler of [GS09] by UC-secure variants that are realizable from a CRS. This allows us to remove an expensive precomputation phase from the protocol because our extraction of the inputs can be directly done via UC-commitments. [GS09], on the other hand, need a PRS simulator [PRS02] because they have to extract the inputs in a concurrent setting.

In more detail, the UC-secure resettability compiler proceeds as follows. In a first step, the non-resettable party (i.e. the token receiver) uses a non-interactive straightline-extractable commitment in the CRS-hybrid model to commit to his input x_{R} and random coins r_{R}. The resettable party (i.e. the token) applies a pseudorandom function to the message to obtain additional random coins r'_{R} and sends them to the receiver, who sets $r''_{\mathsf{R}} = r'_{\mathsf{R}} \oplus r_{\mathsf{R}}$. Then the token does the same with its input x_{S} and random coins r_{S}, except that the receiver has to provide a (resettably-sound) proof that it deterministically derived the random coins r'_{S} from r''_{R}. This is necessary to keep the receiver from choosing new random coins for the resettable party while using the same input.

Now, both parties have a fixed random tape and commitments to the inputs and random tape of the other party. The rest of the protocol proceeds as in [GS09], i.e. the secure computation phase of e.g. [GMW87] is carried out. Instead of using

normal zero-knowledge proofs to ensure that the parties behave according to the protocol, the token uses a straight-line-simulateable resettable zero-knowledge proof to prove its statements, while the receiver uses resettably-sound zero-knowledge proofs.

Protocol $\Pi_{\text{2PC}}^{\text{res}}$

Let $\mathcal{T}$ be a $\mathcal{F}_{\text{wrap}}^{\text{hybrid1}}$ or $\mathcal{F}_{\text{wrap}}^{\text{hybrid2}}$ instance, thus COM has access to a UC-CRS.

Setup phase:

1. Sender: Let x_1 be the sender's input. Sample a seed $s \in \{0,1\}^\kappa$ for the pseudorandom function PRF and program a token program T with the following functionality.
   - Upon receiving a message $(\texttt{commit}, c_{\mathsf{R}})$ from the receiver, compute $(r_2', r_1, r_{ver}) \leftarrow \mathsf{PRF}(c_{\mathsf{R}})$ and $(c_{\mathsf{T}}, d_{\mathsf{T}}) \leftarrow \mathsf{COM.Commit}(x_1, r_1)$. Send (r_2', c_{T}) to the receiver.
   - Upon receiving a message $(\texttt{rand}, r_1')$, start the verifier program V_{rs} for the language $\mathcal{L}_1$ with input $(r_1', r_2', c_{\mathsf{R}})$. If V_{rs} accepts, set $r_1'' = r_1' \oplus r_1$, otherwise abort.
   - Upon receiving a message $(\texttt{comp})$ from the receiver, send the next message m_1^j according to Π, start the prover program P for $\mathcal{L}_3$ with input $(x_1, r_1, r_1', c_{\mathsf{T}}, d_{\mathsf{T}}, m_1^j, T^j)$. Additionally, start the verifier program V_{rs} for $\mathcal{L}_2$ with input $(r_2', c_{\mathsf{R}}, m_{\mathsf{V}})$. If V_{rs} aborts, abort.
   - Upon receiving a message $(\texttt{comp}, m_2^j)$ from the receiver, start the verifier program V_{rs} for $\mathcal{L}_3$ with input $(r_2', c_{\mathsf{R}}, m_2^j, T^j)$. Abort if V_{rs} aborts, otherwise accept and set $j = j + 1$.

Commitment phase:

2. Receiver: Let x_2 be the receiver's input.
   a) Draw $r_2 \leftarrow \{0,1\}^\kappa$ uniformly at random and compute $(c_{\mathsf{R}}, d_{\mathsf{R}}) \leftarrow \mathsf{COM.Commit}(x_2, r_2)$. Send c_{R} to $\mathcal{T}$.
   b) Let (r_2', c_{T}) be the answer from $\mathcal{T}$. Set $r_2'' = r_2 \oplus r_2'$, derive $r_1' \leftarrow r_2''$ and send $(\texttt{rand}, r_1')$ to $\mathcal{T}$. Start the prover program P_{rs} for the language $\mathcal{L}_1$ with input $(x_1, r_1, r_1', r_2', c_{\mathsf{R}}, d_{\mathsf{R}})$.

Computation phase:

Let $m_1^j \leftarrow \Pi(T_j, x_1, r_1'')$ be the next message as per Π with T_j being the list of messages $(m_1^1, m_2^1, \ldots, m_1^{j-1}, m_2^{j-1})$ that have already been exchanged.

3. Receiver: Send $(\texttt{comp})$ to $\mathcal{T}$ and let m_1^j be the next message from $\mathcal{T}$.
   a) Start the verifier program V for $\mathcal{L}_3$ with input $(r_1', c_{\mathsf{T}}, m_1^j, T^j)$. For every verifier message m_{V}, start the prover program P_{rs} for $\mathcal{L}_2$ with input $(x_2, r_2, r_2', c_{\mathsf{R}}, d_{\mathsf{R}}, m_{\mathsf{V}})$. If V aborts, abort.
   b) Compute $m_2^j \leftarrow \Pi(T_2^j, x_2, r_2'')$ and send $(\texttt{comp}, m_2^j)$ to $\mathcal{T}$. Start the prover program P_{rs} for the language $\mathcal{L}_3$ with input $(x_2, r_2, r_2', c_{\mathsf{R}}, d_{\mathsf{R}}, m_2^j, T^j)$.

Figure 5.11: Computationally UC-secure protocol realizing $\mathcal{F}_{\text{2PC}}^{\text{res}}$ in the $\mathcal{F}_{\text{wrap}}^{\text{hybrid1}}/\mathcal{F}_{\text{wrap}}^{\text{hybrid2}}$-hybrid models.

A formal description of the protocol is given in Figure 5.11. $\mathcal{F}_{\text{wrap}}^{\text{hybrid1}}$ and $\mathcal{F}_{\text{wrap}}^{\text{hybrid2}}$ are our hybrid token functionalities that provide the token program with access to a UC-CRS. Let COM be a UC-secure commitment scheme in the CRS-model and PRF be a pseudorandom function with output length ℓ. Further, let $(\mathsf{P}_{rs}, \mathsf{V}_{rs})$ be resettably-sound zero-knowledge arguments of knowledge and (P, V) be a zero-knowledge proof system. Let Π be a passively secure implementation of the resettable functionality $\mathcal{F}$. Define the languages as follows:

$$\mathcal{L}_1 = \{(r_1', r_2', c) \mid \exists x, r, d \text{ s.t. } \mathsf{COM.Open}(c, d, (x, r)) = 1 \wedge r_1' = r_2 \oplus r_2'\}$$
$$\mathcal{L}_2 = \{(r', c, m_{\mathsf{V}}) \mid \exists x, r, d \text{ s.t. } \mathsf{COM.Open}(c, d, (x, r)) = 1 \wedge m_{\mathsf{V}} \leftarrow \mathsf{V}(r \oplus r')\}$$
$$\mathcal{L}_3 = \{(r', c, m, T) \mid \exists x, r, d \text{ s.t. } \mathsf{COM.Open}(c, d, (x, r)) = 1 \wedge m \leftarrow \Pi(T, x, r \oplus r')\}$$

The proof of security is a straightforward modification of the proof of [GS09] and thus omitted. The technically most important aspect is the fact that a standard zero-knowledge proof in combination with a resettably-sound zero-knowledge argument of knowledge showing the correctness of the verifier messages results in a straight-line simulatable resettable zero-knowledge proof (in contrast to existing solutions like [CGGM00]). Thus, the simulator can extract the commitments of the other party, and instead of equivocating them, it just fakes the proofs.

In the following sections we will present two constructions that realize $\mathcal{F}_{\text{wrap}}^{\text{hybrid1}}$ and $\mathcal{F}_{\text{wrap}}^{\text{hybrid2}}$, i.e. the token has access to a UC-secure CRS.

5.4.2 Solution Using One Token with Interaction

Our first solution is based on a single untrusted hardware token, but needs an interactive setup. Since Lemma 5.2 states that interaction is needed, if we are to use only one token, an interactive setup phase and a non-interactive online phase is the best one can hope for.

As explained in Section 5.4.1, it is sufficient to obtain a CRS between the token and the token receiver to implement $\mathcal{F}_{\text{2PC}}^{\text{res}}$. In Figure 5.12, we define an intermediate wrapper functionality that provides such a CRS. It is defined analogously to the resettable hardware wrapper functionality (cf. Section 5.1), with the additional property that it samples a uniformly random string and provides both parties with the CRS.

Please note that by sending the common reference string CRS to the token issuer, we model an artifact that arises in our protocol. For the realization of the resettable functionality, it suffices that the token itself and its receiver have access to the CRS. However, this also models that the protocol has to be interactive, as otherwise the CRS would have to be resettable (cf. Section 5.4.3).

Before we proceed with the formal description of the protocol (cf. Figure 5.13), we will outline the main ideas behind the construction. On an intuitive level, in an interactive setup phase we perform a Blum coin-toss [Blu81] between the token receiver and the token. In the following execution phase, the receiver can just query the token without any interaction with the token issuer.

One can imagine the token as being a commitment containing the sender's input y for the coin toss, but it is locked with a password, so that the receiver cannot access the token unless he learns the password. The password is the preimage a to a value b under a one-way function OWF. The receiver has to provide his input x

Functionality $\mathcal{F}_{\text{wrap}}^{\text{hybrid1}}$

Implicitly parametrized by a security parameter κ. Choose the common reference string CRS uniformly at random of length ℓ.

Creation:

1. Await an input $(\texttt{create}, \mathsf{T}, t)$ from the token issuer, where T is a deterministic Turing machine and $t \in \mathbb{N}$. Store (T, t) and give T read access to CRS. Send $(\texttt{created})$ to the adversary.
2. Await a message $(\texttt{delivery})$ from the adversary. Then, send $(\texttt{ready})$ to the token receiver.

Execution:

3. Await an input $(\texttt{run}, w)$ from the receiver. If no $\texttt{create}$-message has been sent, return a special symbol $\perp$. Otherwise, run T on w from its most recent state. When T halts without generating output or t steps have passed, send $\perp$ to the receiver; otherwise store the current state of T and send the output of T to the receiver.
4. Upon receiving a message $(\texttt{crs})$ from the receiver, send $(\texttt{crs}, \mathsf{CRS})$ to both the token issuer and receiver.

Reset (adversarial receiver only):

5. Upon receiving a message $(\texttt{reset})$ from a corrupted token receiver, reset T to its initial state.

Figure 5.12: The wrapper functionality by which we model a resettable tamper-proof hardware token that provides a CRS to both token issuer and token receiver. The runtime bound t is merely needed to prevent malicious token senders from providing a perpetually running program code M; it will be omitted throughout the rest of the section.

to the sender, who will then compute a signature σ on the resulting CRS $x \oplus y$ and send the signature to the receiver together with the preimage a and his input y.

Care has to be taken that the token does not obtain the value a because this would induce a channel between the token issuer and the token after the issuer has already seen the receiver's input. Thus the token receiver performs a (resettably-sound) proof that he is in possession of the password a. This eliminates the possible channel. If the receiver succeeds in convincing the token, the token will reveal y'. The receiver just has to check if $y = y'$, and if that is the case, he is convinced that the token issuer was actually committed to his input. The receiver can compute the CRS and provide the token with the signature and the CRS. A formal description of the protocol follows.

Theorem 5.3. *The protocol $\Pi_{\text{wrap}}^{\text{hybrid1}}$ in Figure 5.13 computationally UC-realizes $\mathcal{F}_{\text{wrap}}^{\text{hybrid1}}$ (cf. Figure 5.12) in the $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$-hybrid model, given that one-way functions exist.*

To give a clearer presentation of the proof, we prove Theorem 5.3 in two steps. Security against a corrupted sender is shown in Lemma 5.5, while security against

Protocol $\Pi_{\text{wrap}}^{\text{hybrid1}}$

Let κ be a security-parameter. Let $\mathcal{T}$ be a $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$ instance, further let OWF be a one-way function and (P, V) be a resettably-sound zero-knowledge argument of knowledge for the language $\mathcal{L} = \{b \,|\, \exists\, a \in \{0,1\}^\kappa \text{ s.t. } b \leftarrow \mathsf{OWF}(a)\}$. Further let SIG be an EUF-CMA-secure signature scheme.
Let $\ell = \mathsf{poly}(\kappa)$ be the desired length of the CRS.

Setup phase:

1. Sender: Let a program M be the sender's input. Choose $a \leftarrow \{0,1\}^\kappa$ uniformly at random and set $b \leftarrow \mathsf{OWF}(a)$. Choose a $y \leftarrow \{0,1\}^\ell$ uniformly at random and generate a key pair $(\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathsf{SIG.KeyGen}(\kappa)$. Program a resettable token program T with the following functionality:
   - Upon receiving a message $(\texttt{unveil})$ from the receiver, run the verifier program V with input b. Forward the messages between the receiver and V. If V rejects, output $\perp$; otherwise send y to the receiver.
   - Upon receiving a message $(\texttt{verify}, \mathsf{CRS}, \sigma)$, check if $\mathsf{SIG.Verify}(\mathsf{vk}, \mathsf{CRS}, \sigma) = 1$, if not abort.
   - Upon receiving input $(\texttt{run}, w)$ from the receiver, run M on input w starting from its most recent state, output whatever M outputs, store the new state of M and wait for the next message $(\texttt{run}, w)$.

   Send $(\texttt{create}, \mathsf{T})$ to $\mathcal{T}$ and send (vk, b) to the receiver.
2. Receiver: Wait for the message $(\texttt{ready})$ from $\mathcal{T}$. Choose $x \leftarrow \{0,1\}^\ell$ uniformly at random and send x to the sender.
3. Sender: Set $\mathsf{CRS} = x \oplus y$, compute $\sigma = \mathsf{SIG.Sign}(\mathsf{sgk}, \mathsf{CRS})$ and send (y, a, σ) to the receiver. Output CRS.
4. Receiver: Check if $\mathsf{OWF}(a) = b$, if not abort. Send $(\texttt{unveil})$ to the token and run the prover program P with input b and witness a. Forward the messages between P and $\mathcal{T}$. Let y' be the output of $\mathcal{T}$. If $y \neq y'$, abort; otherwise set $\mathsf{CRS} = x \oplus y$ and send $(\texttt{verify}, \mathsf{CRS}, \sigma)$ to $\mathcal{T}$. Output CRS.

Execution Phase:

5. Receiver: Let w be the receiver's input. Send $(\texttt{run}, w)$ to $\mathcal{T}$ and output whatever $\mathcal{T}$ outputs.

Figure 5.13: Computationally UC-secure protocol realizing $\mathcal{F}_{\text{wrap}}^{\text{hybrid1}}$ in the $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$-hybrid model.

a corrupted receiver is shown in Lemma 5.6. Together with the observation that all building blocks that we use can be realized from one-way functions, the claim follows.

Lemma 5.5. *The protocol $\Pi_{\text{wrap}}^{\text{hybrid1}}$ in Figure 5.13 computationally UC-realizes $\mathcal{F}_{\text{wrap}}^{\text{hybrid1}}$ (cf. Figure 5.12) in the $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$-hybrid model against a corrupted sender, given that (P, V) is computationally zero-knowledge.*

Proof. We will prove computational UC security for the case of a corrupted sender. Let $\mathcal{A}_\mathsf{S}$ be the dummy-adversary for a corrupted sender and let Sim be the non-

black-box simulator for the argument system (P, V) that takes as input a statement b and the code V^* of a malicious verifier. The simulator $\mathcal{S}_\mathsf{S}$ is given in Figure 5.14. We have to show that for all PPT environments $\mathcal{Z}$, $\mathsf{Real}^{\mathcal{A}_\mathsf{S}}_{\Pi^{\text{hybrid1}}_{\text{wrap}}}(\mathcal{Z}) \approx_c \mathsf{Ideal}^{\mathcal{S}_\mathsf{S}}_{\mathcal{F}^{\text{hybrid1}}_{\text{wrap}}}(\mathcal{Z})$.

Simulator $\mathcal{S}_\mathsf{S}$

- Simulate $\mathcal{T}$ for $\mathcal{A}_\mathsf{S}$ and obtain T^*.
- (Setup) Upon receiving a message (vk, b^*) from $\mathcal{A}_\mathsf{S}$ in Step 1, proceed as follows.
  - Use the token code T^* to construct a malicious verifier V^* for (P, V). V^* basically simulates the zero-knowledge step of T^* and outputs the state of T^* after the zero-knowledge step is over.
  - Start the non-black-box simulator Sim with input (V^*, b^*) to obtain a new token state for T^* and proceed with the protocol simulation as specified in $\Pi^{\text{hybrid1}}_{\text{wrap}}$ until T^* outputs y^*.
  - Send a message $(\texttt{crs})$ to $\mathcal{F}^{\text{hybrid1}}_{\text{wrap}}$ and let $(\texttt{crs}, \mathsf{CRS})$ be the answer. Set $\hat{x} = \mathsf{CRS} \oplus y^*$ and send $\hat{x}$ to $\mathcal{A}_\mathsf{S}$ as the message in Step 2.
  - Upon receiving a message (y'^*, a^*, σ^*) from Step 3, check if $y^* = y'^*$ and $\mathsf{OWF}(a^*) = b^*$, otherwise abort. Otherwise perform Step 4 by sending (CRS, σ^*) to T^*.
  - Send $(\texttt{create}, \mathsf{T}^*)$ to $\mathcal{F}^{\text{hybrid1}}_{\text{wrap}}$.
- (Execution) Simulated straightforwardly according to $\Pi^{\text{hybrid1}}_{\text{wrap}}$.

Figure 5.14: Simulator for a corrupted sender in the protocol $\Pi^{\text{hybrid1}}_{\text{wrap}}$.

The simulator follows the following strategy. It obtains the possibly malicious token program from $\mathcal{A}_\mathsf{S}$ and uses this code to construct a malicious verifier for the argument system (P, V). Once this is done, $\mathcal{S}_\mathsf{S}$ applies the non-black-box simulator for the argument system to the verifier program and generates a fake proof that it knows a preimage of b under OWF. The transcript of the simulation is integrated into the token program such that it will continue to output the sender's input y. Then the simulator can choose his input depending on the CRS from $\mathcal{F}^{\text{hybrid1}}_{\text{wrap}}$ and proceed with the protocol as specified in $\Pi^{\text{hybrid1}}_{\text{wrap}}$.

We will now show indistinguishability of the following experiments.

Experiment 0: This is the real model.

Experiment 1: Identical to Experiment 0, except that when the receiver sends an `unveil`-message to T^*, $\mathcal{S}_1$ runs the non-black-box simulator Sim on the verifier V^* (as described in Figure 5.14) instead of letting the prover P interact with T^*. $\mathcal{S}_1$ then uses the output of Sim as the most recent state s to continue the computation of T^*.

Experiment 2: Identical to Experiment 1, except that $\mathcal{S}_2$ runs the `unveil`-phase of T^* before interacting with $\mathcal{A}_\mathsf{S}$, thereby obtaining y^*. Set $\hat{x} = \mathsf{CRS} \oplus y^*$, where CRS is a uniformly random common reference string. This is the ideal model.

The indistinguishability of Experiment 0 and Experiment 1 follows from the computational zero-knowledge property of the argument system (P, V) (cf. Lemma 5.5.1). Experiments 1 and 2 are identically distributed as the interactions of the receiver with $\mathcal{T}$ and $\mathcal{A}_{\mathsf{S}}$ are independent of one another in both experiments and thus exchangeable.

Lemma 5.5.1. *From $\mathcal{Z}$'s view, Experiment 0 and Experiment 1 are computationally indistinguishable, provided that the argument system (P, V) is computational zero-knowledge.*

Proof. Fix a PPT-environment $\mathcal{Z}$. Assume for the sake of contradiction that $\mathcal{Z}$ distinguishes Experiment 0 and Experiment 1 with non-negligible advantage ϵ. We will construct a malicious verifier V^* and a distinguisher $\mathcal{D}$ from $\mathcal{Z}$, such that $\mathcal{D}$ distinguishes the random variables $\langle \mathsf{P}(a), \mathsf{V}^* \rangle(b)$ and $\mathsf{Sim}(b, \mathsf{V}^*)$ with advantage ϵ for some a and b.

Fix the random tape of $\mathcal{Z}$ such that $\mathcal{Z}$ with these fixed coins distinguishes Experiment 0 and Experiment 1 with non-negligible advantage ϵ. By a simple averaging argument, such coins must exist. Let (a, b) with $\mathsf{OWF}(a) = b$ be the fixed tuple that corresponds with this $\mathcal{Z}$. The malicious verifier V^* is constructed as in the description of the simulator.

The distinguisher $\mathcal{D}$ is constructed as follows. It behaves like $\mathcal{S}_1$, but does not compute the new state for $\mathsf{T}^*$by itself, instead it uses the challenge of the distinguishing experiment as the state. Clearly, if $\mathcal{D}$'s input is distributed as $\langle \mathsf{P}(a), \mathsf{V}^* \rangle(b)$, then $\mathcal{Z}$'s view is distributed identical to Experiment 0. If, on the other hand, $\mathcal{D}$'s input is distributed according to $\mathsf{Sim}(b, \mathsf{V}^*)$, then $\mathcal{Z}$'s view is distributed identical to Experiment 1. Thus $\mathcal{D}$ and V^* contradict the zero-knowledge property of (P, V). ■

This concludes the proof of Lemma 5.5. □

We move on to show UC-security against a corrupted receiver.

Lemma 5.6. *The protocol $\Pi_{\text{wrap}}^{\text{hybrid1}}$ in Figure 5.13 computationally UC-realizes $\mathcal{F}_{\text{wrap}}^{\text{hybrid1}}$ (cf. Figure 5.12) in the $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$-hybrid model against a corrupted receiver, given that OWF is a one-way function, SIG is an EUF-CMA-secure signature scheme and (P, V) is an argument of knowledge.*

Proof. We have to show that $\mathsf{Real}^{\mathcal{A}_{\mathsf{R}}}_{\Pi^{\text{hybrid1}}_{\text{wrap}}}(\mathcal{Z}) \approx_c \mathsf{Ideal}^{\mathcal{S}_{\mathsf{R}}}_{\mathcal{F}^{\text{hybrid1}}_{\text{wrap}}}(\mathcal{Z})$ for all PPT environments $\mathcal{Z}$. Let $\mathcal{A}_{\mathsf{R}}$ be the dummy adversary for a corrupted receiver. Let Ext be the knowledge-extractor for the argument of knowledge-system (P, V). The simulator $\mathcal{S}_{\mathsf{R}}$ against a corrupted receiver is depicted in Figure 5.15.

The simulator behaves very similar to an honest sender, but by simulating the hybrid functionality, he is able to adapt the token output $\hat{y}$ of the simulated resettable token after he received the input x^* of a corrupted receiver. Thus he can set the value $\hat{y} = \mathsf{CRS} \oplus x^*$ to match the output of the ideal functionality $\mathcal{F}_{\text{wrap}}^{\text{hybrid1}}$.

We show the indistinguishability of a real protocol execution and the output of the simulator in a series of hybrid experiments.

Experiment 0: This is the real model.

Experiment 1: Identical to Experiment 0, except that $\mathcal{S}_1$ outputs $\perp$ if $\mathcal{A}_{\mathsf{R}}$ sends an `unveil`-query to $\mathcal{T}$ for which the verifier program V accepts although $\mathcal{A}_{\mathsf{R}}$ has not sent his input x^*.

Simulator $\mathcal{S}_{\mathsf{R}}$

- Simulate $\mathcal{T}$ for $\mathcal{A}_{\mathsf{R}}$. Let Q denote the set of token inputs that $\mathcal{A}_{\mathsf{R}}$ inputs into the token.
- (Setup) Simulate Step 1 straightforwardly according to $\Pi_{\text{wrap}}^{\text{hybrid1}}$ with $\hat{b} \leftarrow \mathsf{OWF}(\hat{a})$. Let x^* be the answer from $\mathcal{A}_{\mathsf{R}}$ in Step 2. For Step 3, send $(\texttt{crs})$ to $\mathcal{F}_{\text{wrap}}^{\text{hybrid1}}$ and let $(\texttt{crs}, \mathsf{CRS})$ be the answer. Set $\hat{y} = \mathsf{CRS} \oplus x^*$, compute $\hat{\sigma} \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, \mathsf{CRS})$ and output $(\hat{y}, \hat{a}, \hat{\sigma})$ to $\mathcal{A}_{\mathsf{R}}$.
- (Token) Simulate the token in Step 4 as follows.
  - If $\mathcal{A}_{\mathsf{R}}$ sends a `unveil`-message to $\mathcal{T}$ although he did not send an x^* beforehand, output $\perp$ regardless if V accepts or not; otherwise output $\hat{y}$.
  - If $\mathcal{A}_{\mathsf{R}}$ sends a message $(\texttt{verify}, \mathsf{CRS}^*, \sigma^*)$ to $\mathcal{T}$ and has not yet sent an x^* or $\mathsf{SIG.Verify}(\mathsf{vk}, \mathsf{CRS}^*, \sigma^*) = 0$, output $\perp$.
- (Execution) If $\mathcal{A}_{\mathsf{R}}$ sends a tuple $(\texttt{run}, w)$ to $\mathcal{T}$ in Step 5, but he has not yet sent a tuple $(\mathsf{CRS}^*, \sigma^*)$ to $\mathcal{T}$ with $\mathsf{SIG.Verify}(\mathsf{vk}, \mathsf{CRS}^*, \sigma^*) = 1$, output $\perp$. Otherwise forward $(\texttt{run}, w)$ to $\mathcal{F}_{\text{wrap}}^{\text{hybrid1}}$ and output whatever $\mathcal{F}_{\text{wrap}}^{\text{hybrid1}}$ outputs.
- (Reset) Whenever $\mathcal{A}_{\mathsf{R}}$ sends a message $(\texttt{reset})$ to $\mathcal{T}$, send $(\texttt{reset})$ to $\mathcal{F}_{\text{wrap}}^{\text{hybrid1}}$ and reset the state of $\mathcal{T}$.

Figure 5.15: Simulator for a corrupted receiver in the protocol $\Pi_{\text{wrap}}^{\text{hybrid1}}$.

Experiment 2: Identical to Experiment 1, except that $\mathcal{S}_2$ sets $\hat{y} = \mathsf{CRS} \oplus x^*$, where CRS is a uniformly chosen common reference string.

Experiment 3: Identical to Experiment 2, except that $\mathcal{S}_3$ aborts if $\mathcal{A}_{\mathsf{R}}$ sends a a tuple $(\texttt{verify}, \mathsf{CRS}^*, \sigma^*)$ to $\mathcal{T}$, for which it holds that $\mathsf{SIG.Verify}(\mathsf{vk}, \mathsf{CRS}^*, \sigma^*) = 1$ and CRS^* has not been signed by $\mathcal{S}_3$. This is the ideal model.

Experiment 0 and Experiment 1 are computationally indistinguishable given that the one-way function OWF is strongly one-way and (P, V) is an argument of knowledge (cf. Lemma 5.6.1). Experiment 1 and Experiment 2 are identically distributed as both x^* and CRS are uniformly and independently distributed. The indistinguishability of Experiment 2 and Experiment 3 follows directly from the EUF-CMA-security of SIG.

Lemma 5.6.1. *From $\mathcal{Z}$'s view, Experiment 0 and Experiment 1 are computationally indistinguishable, given that* OWF *is a one-way function and the argument system* (P, V) *is an argument of knowledge.*

Proof. Experiment 0 and Experiment 1 are identically distributed conditioned on the event that $\mathcal{A}_{\mathsf{R}}$ does not convince V that it possesses an a such that $\mathsf{OWF}(a) = b$ before sending his own coins x^* to the sender. Thus, a $\mathcal{Z}$ distinguishing between Experiment 0 and Experiment 1 must succeed in making $\mathcal{A}_{\mathsf{R}}$ convince $\mathcal{T}$ of this before receiving such a value a from the sender.

Assume that $\mathcal{Z}$ causes this event with non-negligible probability ϵ. We will construct an adversary $\mathcal{B}$ from $\mathcal{Z}$ that inverts the one-way function OWF with non-negligible probability. Let $m = \mathsf{poly}(\kappa)$ be an upper bound on the number of `unveil`-messages that $\mathcal{A}_{\mathsf{R}}$ sends to $\mathcal{T}$. Let b' be the image on which $\mathcal{B}$ is supposed to

invert OWF. $\mathcal{B}$ first chooses an index $i \in \{1, \ldots, m\}$ uniformly at random. It then simulates the real protocol to $\mathcal{Z}$, except that it sets $b = b'$ instead of generating b by $b \leftarrow \mathsf{OWF}(a)$. The simulation proceeds until the i-th call of $\mathcal{A}_\mathsf{R}$ to $\mathcal{T}$.

If the computation of $\mathcal{Z}$ continued after this point, the subsequent messages passed by $\mathcal{A}_\mathsf{R}$ would correspond to the messages of a malicious prover P^* for the argument system (P, V). Thus, $\mathcal{B}$ can construct P^* from the state of the halted $\mathcal{Z}$ as follows: P^* basically continues the simulation of $\mathcal{Z}$ at its current state and forwards messages between $\mathcal{A}_\mathsf{R}$ and an external verifier V. $\mathcal{B}$ can now take the code of P^* and run the extractor $\mathsf{Ext}(b, \mathsf{P}^*)$. Let $\hat{a}$ be the output of Ext. $\mathcal{B}$ outputs a and terminates.

First, notice that from $\mathcal{Z}$'s view, this simulation is identically distributed to a real protocol run. Thus, the event that $\mathcal{A}_\mathsf{R}$ convinces $\mathcal{T}$ that it possesses a preimage a of b' happens with probability at least ϵ. With probability at least $\frac{1}{m}$, the index i chosen by $\mathcal{B}$ matches the index of the proof where this event happens. Therefore, $\Pr[\langle \mathsf{P}^*, \mathsf{V} \rangle = 1] \geq \frac{\epsilon}{m}$.

Since (P, V) is an argument of knowledge, given a successful proof the extractor will extract a witness for the statement, i.e. a preimage a^* of b', with overwhelming probability. In more detail, it holds that $\Pr[\mathsf{Ext}(b', \mathsf{P}^*) \in w_\mathcal{L}(b)] > \Pr[\langle \mathsf{P}^*, \mathsf{V} \rangle = 1] - \nu \geq \frac{\epsilon}{m} - \nu$ for some negligible extraction error ν. Thus, with probability $\frac{\epsilon}{m} - \nu$, which is non-negligible, $\mathcal{B}$ outputs a preimage a^* of b' under the one-way function OWF, thus breaking the one-wayness property of OWF. ■

This concludes the proof of Lemma 5.6. □

5.4.3 Solution Using Two Resettable Tokens Without Interaction

The solution presented in the previous section needs an interactive setup phase. This is directly implied by Lemma 5.2 since we used only one $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$-instance. We now show that it is possible to achieve non-interactive two-party computation for resettable functionalities using only two $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$-instances. This is optimal with respect to the number of tokens used and also with respect to the techniques we use, i.e. non-black-box simulation.

Before we discuss the protocol, we present an additional intermediate wrapper functionality $\mathcal{F}_{\text{wrap}}^{\text{hybrid2}}$ (cf. Figure 5.16) similar to the one from Section 5.4.2, but with the additional property that the CRS is resettable by the token receiver. This cannot be prevented if we do not allow interaction because both tokens are resettable and thus each CRS that is created between the tokens and the receiver must also be resettable. The functionality in Figure 5.16 thus samples a new CRS after each reset using a random oracle.

This hybrid wrapper functionality can straightforwardly be used to implement $\mathcal{F}_{\text{2PC}}^{\text{res}}$ as described in Section 5.4.1. We move on to show our realization of $\mathcal{F}_{\text{wrap}}^{\text{hybrid2}}$ in the $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$-hybrid model. The idea behind the protocol is similar to the interactive solution from Section 5.4.2, but with some important changes. If we were to use the protocol based on a single token and just store the sender functionality in another token, the receiver could learn the password a and then reset the tokens and choose his input adaptively. Thus, we now require the receiver to first compute a commitment c to his input x. He then provides x and a resettably-sound proof to the first token which ensures that c is a proper commitment, and obtains a signature

Functionality $\mathcal{F}_{\text{wrap}}^{\text{hybrid2}}$

Implicitly parametrized by a security parameter κ. Let H be a random oracle that maps to strings of length ℓ.

Creation:

1. Await an input $(\texttt{create}, \mathsf{T}, t)$ from the token issuer, where T is a deterministic Turing machine and $t \in \mathbb{N}$. Set the common reference string $\mathsf{CRS} = H(1)$. Store (T, t) and give T read-access to CRS. Send $(\texttt{created}, \mathsf{CRS})$ to the adversary.
2. Await a message $(\texttt{delivery})$ from the adversary. Then, send $(\texttt{ready}, \mathsf{CRS})$ to the token receiver.

Execution:

3. Await an input $(\texttt{run}, w)$ from the receiver. If no `create`-message has been sent, return a special symbol $\perp$. Otherwise, run T on w from its most recent state. When T halts without generating output or t steps have passed, send $\perp$ to the receiver; otherwise store the current state of T and send the output of T to the receiver.

Reset (adversarial receiver only):

4. Upon receiving a message $(\texttt{reset}, j)$ from a corrupted token receiver, set the common reference string to $\mathsf{CRS} = H(j)$, send $(\texttt{crs}, \mathsf{CRS})$ to the corrupted receiver and reset T to its initial state.

Figure 5.16: The wrapper functionality by which we model a resettable tamper-proof hardware token that provides a CRS to the token and the token receiver. The runtime bound t is merely needed to prevent malicious token senders from providing a perpetually running program code M; it will be omitted throughout the rest of the section.

σ on his commitment together with a value y that is derived from the commitment using a PRF.

The signature σ and the commitment c are used to unlock the second token via another resettably-sound proof. The proof states that the receiver knows a signature on the commitment. If the second token accepts the proof, it will compute the value y' again by applying a PRF. The receiver compares both values and if they match, he knows that the tokens behave consistently. The formal description of the protocol is given in Figure 5.17.

We will now show the following theorem.

Theorem 5.4. *The protocol $\Pi_{\text{wrap}}^{\text{hybrid2}}$ in Figure 5.17 computationally UC-realizes $\mathcal{F}_{\text{wrap}}^{\text{hybrid2}}$ (cf. Figure 5.16) in the $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$-hybrid model, given that one-way functions exist.*

The proof is split into a security proof against a corrupted sender (cf. Lemma 5.7) and a proof against a corrupted receiver (cf. Lemma 5.8). All primitives that we use in the protocol can be realized from one-way functions, thus the claim follows.

Lemma 5.7. *The protocol $\Pi_{\text{wrap}}^{\text{hybrid2}}$ in Figure 5.17 computationally UC-realizes $\mathcal{F}_{\text{wrap}}^{\text{hybrid2}}$ (cf. Figure 5.16) in the $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$-hybrid model against a corrupted sender, given*

Protocol $\Pi_{\text{wrap}}^{\text{hybrid2}}$

Let PRF be a pseudorandom function with output length ℓ, COM be a 2-message statistically binding commitment scheme and SIG be an EUF-CMA-secure signature scheme with deterministic signing algorithm. Let $(\mathsf{P}_1, \mathsf{V}_1)$ be a resettably-sound argument of knowledge for the language $\mathcal{L}_1 = \{(x, c) \mid \exists\, d \text{ s.t. } \mathsf{COM.Open}(c, d, x) = 1\}$ and $(\mathsf{P}_2, \mathsf{V}_2)$ be a resettably-sound argument of knowledge for the language $\mathcal{L}_2 = \{(\mathsf{vk}, c) \mid \exists\, \sigma \text{ s.t. } \mathsf{SIG.Verify}(\mathsf{vk}, c, \sigma) = 1\}$. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two instances of $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$.

Setup phase:

1. Sender: Let a program M be the sender's input. Sample a seed $s \leftarrow \{0,1\}^\kappa$ for the pseudorandom function PRF, a key k for COM and generate a key pair $(\mathsf{sgk}, \mathsf{vk}) \leftarrow \mathsf{SIG.KeyGen}(\kappa)$. Program two token functionalities T_1 and T_2 as follows.
   - T_1: Choose a random tape of appropriate length and set a flag *ready* $= 0$.
     - Upon receiving a message $(\texttt{commit}, x, c)$ from the receiver, run the verifier program V_1 with input (x, c). Forward the messages between V_1 and the receiver and output $\perp$ if V_1 rejects. Otherwise, compute $y \leftarrow \mathsf{PRF}(c)$ and $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, c)$. Set the flag *ready* $= 1$, set $\mathsf{CRS} = x \oplus y$ and output (y, σ).
     - Upon receiving input $(\texttt{run}, w)$ from the receiver, check if *ready* $= 1$. If not, abort; otherwise run M on input w starting from its most recent state, output whatever M outputs and store the new state of M.
   - T_2: Choose a random tape of appropriate length.
     - Upon receiving a message $(\texttt{verify}, c)$ from the receiver, run the verifier program V_2 with input (vk, c). Forward the messages between V_2 and the receiver and output $\perp$ if V_2 rejects. Otherwise, compute $y \leftarrow \mathsf{PRF}(c)$ and output y.

   Send $(\texttt{create}, \mathsf{T}_1)$ to $\mathcal{T}_1$ and $(\texttt{create}, \mathsf{T}_2)$ to $\mathcal{T}_2$. Additionally, send (vk, k) to the receiver.

2. Receiver:
   - Choose $x \leftarrow \{0,1\}^\ell$ uniformly at random and compute $(c, d) \leftarrow \mathsf{COM.Commit}(k, x)$.
   - Send $(\texttt{commit}, x, c)$ to $\mathcal{T}_1$. Run the prover program P_1 with input (x, c, d) and forward the messages between P_1 and $\mathcal{T}_1$.
   - Let (y, σ) be the answer from $\mathcal{T}_1$. Check if $\mathsf{SIG.Verify}(\mathsf{vk}, \sigma, c) = 1$, otherwise abort. Send $(\texttt{verify}, c)$ to $\mathcal{T}_2$ and run the prover P_2 with input (vk, c, σ).
   - Let y' be the output of $\mathcal{T}_2$. Check if $y = y'$, if not abort. Otherwise, set $\mathsf{CRS} = x \oplus y$.

Execution phase:

3. Receiver: Let w be the receiver's input. Send $(\texttt{run}, w)$ to $\mathcal{T}_1$ and output whatever $\mathcal{T}_1$ outputs.

Figure 5.17: Computationally UC-secure protocol realizing $\mathcal{F}_{\text{wrap}}^{\text{hybrid2}}$ in the $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$-hybrid model.

that COM *is computationally hiding and* $(\mathsf{P}_1, \mathsf{V}_1)$ *and* $(\mathsf{P}_2, \mathsf{V}_2)$ *are computational zero-knowledge.*

Proof. We will prove computational UC-security against a corrupted sender. Let therefor $\mathcal{A}_\mathsf{S}$ be the dummy-adversary and $\mathcal{Z}$ be a PPT-environment. We first state the sender simulator $\mathcal{S}_\mathsf{S}$ in Figure 5.18. Let Sim_1 and Sim_2 be the non-black-box simulators for the argument systems. We will show that for all PPT environments $\mathcal{Z}$, $\mathsf{Real}^{\mathcal{A}_\mathsf{S}}_{\Pi^{\text{hybrid2}}_{\text{wrap}}}(\mathcal{Z}) \underset{c}{\approx} \mathsf{Ideal}^{\mathcal{S}_\mathsf{S}}_{\mathcal{F}^{\text{hybrid2}}_{\text{wrap}}}(\mathcal{Z})$.

The simulator uses the tokens in reverse order compared to a real protocol execution. He uses the code of the second token from the sender to get the input y of the sender by faking a proof for the argument system $(\mathsf{P}_2, \mathsf{V}_2)$. Once he knows the input, he constructs from the code of the first token a new token that just adjusts the CRS as specified by the ideal functionality and then allows the receiver to run the actual token functionality on arbitrary inputs.

Simulator $\mathcal{S}_\mathsf{S}$

- Simulate $\mathcal{T}_1$ and $\mathcal{T}_2$ for $\mathcal{A}_\mathsf{S}$ and obtain T^*_1 and T^*_2.
- (Setup) Upon receiving (vk, k) from $\mathcal{A}_\mathsf{S}$ in Step 1, proceed as follows.
  - Compute $(\hat{c}, \hat{d}) \leftarrow \mathsf{COM.Commit}(k, 0)$ and use the code T^*_2 to construct a verifier program V^*_2 that runs the zero-knowledge step of T^*_2 when it is given input $\hat{c}$. The output of V^*_2 is the state of T_2 after the zero-knowledge step.
  - Start the non-black-box simulator Sim_2 with input $(\mathsf{V}^*_2, \mathsf{vk}, \hat{c})$ to obtain a new token state for T^*_2 and continue the execution of T^*_2 with this state until it outputs y^*.
  - Program a token $\mathsf{T}_\mathcal{S}$ that executes Step 2 with the receiver.
    - Read the common reference string CRS provided by $\mathcal{F}^{\text{hybrid2}}_{\text{wrap}}$ and set $\hat{x} = \mathsf{CRS} \oplus y^*$.
    - Use the code T^*_1 to construct a verifier program V^*_1 that runs the zero-knowledge step of T^*_1 when it is given input $(\hat{x}, \hat{c})$. The output of V^*_2 is the state of T_2 after the zero-knowledge step.
    - Start the non-black-box simulator Sim_2 with input $(\mathsf{V}^*_2, \hat{x}, \hat{c})$ to obtain a new token state for T^*_2 and continue the execution of T^*_2 with this state until it outputs (y'^*, σ^*). Abort if $y'^* = y^*$.
    - Upon receiving a message $(\mathtt{run}, w)$ from the receiver, run T^*_1 from its most recent state with input $(\mathtt{run}, w)$.
  - Send $(\mathtt{create}, \mathsf{T}_\mathcal{S})$ to $\mathcal{F}^{\text{hybrid2}}_{\text{wrap}}$.
- (Execution) Simulated straightforwardly according to $\Pi^{\text{hybrid2}}_{\text{wrap}}$.

Figure 5.18: Simulator for a corrupted sender in the protocol $\Pi^{\text{hybrid2}}_{\text{wrap}}$.

Consider the following series of hybrid experiments.

Experiment 0: This is the real model.

Experiment 1: Identical to Experiment 0, except that $\mathcal{S}_1$ does the following. Instead of running P_2 with input $(\mathsf{vk}, c, \sigma^*)$, it runs Sim_2 with input $(\mathsf{V}^*_2, \mathsf{vk}, c)$ (where V^*_2 is as described in Figure 5.18).

Experiment 2: Identical to Experiment 1, except that S_2 does the following. Instead of running P_1 with input (x, c, d), it runs Sim_1 with input (V_1^*, x, c) (where V_1^* is as described in Figure 5.18).

Experiment 3: Identical to Experiment 2, except that the commitment c is computed as $(\hat{c}, \hat{d}) \leftarrow \mathsf{COM.Commit}(k, 0)$ instead of $(c, d) \leftarrow \mathsf{COM.Commit}(k, x)$.

Experiment 4: Identical to Experiment 3, except that $\mathcal{S}_4$ first interacts with $\mathcal{T}_2$ and then with $\mathcal{T}_1$. Moreover, it sets $\hat{x} = \mathsf{CRS} \oplus y^*$ instead of choosing x uniformly at random. This is the ideal model.

Experiment 0 and Experiment 1 are indistinguishable, given that the argument system $(\mathsf{P}_1, \mathsf{V}_1)$ is zero-knowledge. By the same argumentation, Experiments 1 and 2 are indistinguishable given that $(\mathsf{P}_2, \mathsf{V}_2)$ is zero-knowledge. Both indistinguishability proofs are almost identical to the proof of Lemma 5.5.1 and thus omitted. The indistinguishability of Experiment 2 and Experiment 3 follows straightforwardly from the hiding property of the commitment scheme COM. Experiment 3 and Experiment 4 are identically distributed as in both experiments y^* is independently and uniformly distributed. □

Lemma 5.8. *The protocol* $\Pi_{\mathrm{wrap}}^{\mathrm{hybrid2}}$ *in Figure 5.17 computationally UC-realizes* $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{hybrid2}}$ *(cf. Figure 5.16) in the* $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$*-hybrid model against a corrupted receiver, given that* SIG *is EUF-CMA-secure,* PRF *is a pseudorandom function,* COM *is statistically binding and* $(\mathsf{P}_1, \mathsf{V}_1)$ *and* $(\mathsf{P}_2, \mathsf{V}_2)$ *satisfy the argument of knowledge property.*

Proof. We will prove computational UC-security against a corrupted receiver R. We therefore first provide the simulator $\mathcal{S}_{\mathsf{R}}$ in Figure 5.19.

We will prove computational UC-security against a corrupted receiver, i.e. we will show that for all PPT environments $\mathcal{Z}$, $\mathsf{Real}^{\mathcal{A}_{\mathsf{R}}}_{\Pi^{\mathrm{hybrid2}}_{\mathrm{wrap}}}(\mathcal{Z}) \approx_c \mathsf{Ideal}^{\mathcal{S}_{\mathsf{R}}}_{\mathcal{F}^{\mathrm{hybrid2}}_{\mathrm{wrap}}}(\mathcal{Z})$ where $\mathcal{A}_{\mathsf{R}}$ is the dummy-adversary. We therefore first provide the simulator $\mathcal{S}_{\mathsf{R}}$ in Figure 5.19.

The simulation strategy is straightforward. The simulator learns the queries to both tokens and can adapt their outputs to the CRS that is provided by the ideal functionality. To show indistinguishability of a real protocol run and a simulated protocol run, consider the following series of hybrid experiments.

Experiment 0: This is the real model.

Experiment 1: Identical to Experiment 0, except that $\mathcal{S}_1$ computes $\hat{y}$ not as $\hat{y} \leftarrow \mathsf{PRF}(c)$ but as follows. If a tuple $(x'^*, \hat{y}', c'^*, j')$ has been stored with $c'^* = c^*$, set $\hat{y} = \hat{y}'$. Otherwise, choose CRS uniformly at random and set $\hat{y} = x^* \oplus \mathsf{CRS}$. In addition, store the tuple $(x^*, \hat{y}, c^*)$.

Experiment 2: Identical to Experiment 1, except that $\mathcal{S}_2$ simulates $\mathcal{T}_1$ such that it outputs $\perp$ if V_1 accepts, even though a tuple $(x'^*, \hat{y}', c'^*, j')$ has been stored with $x'^* \neq x^*$ and $c'^* = c^*$.

Experiment 3: Identical to Experiment 2, except that $\mathcal{S}_3$ simulates $\mathcal{T}_2$ so that it outputs $\perp$ if V_2 accepts, even though no tuple $(x'^*, \hat{y}', c'^*, j')$ with $c'^* = c^*$ has been stored. This is the ideal experiment.

We first show in Lemma 5.8.1 that Experiments 0 and 1 are indistinguishable given that PRF is a pseudorandom function.

Simulator $\mathcal{S}_{\mathsf{R}}$

- Simulate $\mathcal{T}_1$ and $\mathcal{T}_2$ for $\mathcal{A}_{\mathsf{R}}$ and define a counter $j = 1$.
- (Setup) Simulate Step 1 straightforwardly according to $\Pi_{\text{wrap}}^{\text{hybrid2}}$, send the resulting verification key vk to $\mathcal{A}_{\mathsf{R}}$ and store sgk.
- (Token) Simulate the tokens in Step 2 as follows.
  - If $\mathcal{A}_{\mathsf{R}}$ sends a message $(\texttt{commit}, x^*, c^*)$ to $\mathcal{T}_1$, continue the simulation of $\mathcal{T}_1$ until the verifier program V_1 produces a result. If it rejects, abort; otherwise do the following. If V_1 accepts, even though a tuple $(x'^*, \hat{y}', c'^*, j')$ has already been stored with $x'^* \neq x^*$ and $c'^* = c^*$, output $\perp$. Else, compute $\hat{\sigma} \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, c^*)$, output $(\hat{y}, \hat{\sigma})$ and send $(\texttt{reset}, j')$ to $\mathcal{F}_{\text{wrap}}^{\text{hybrid2}}$. If no tuple has been stored at all, increment j by 1 and send $(\texttt{reset}, j)$ to $\mathcal{F}_{\text{wrap}}^{\text{hybrid2}}$ to obtain $(\texttt{crs}, \mathsf{CRS})$. Set $\hat{y} = \mathsf{CRS} \oplus x^*$, compute $\hat{\sigma} \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, c^*)$ and store $(x^*, \hat{y}, c^*, j)$.
  - If $\mathcal{A}_{\mathsf{R}}$ sends a message $(\texttt{verify}, c^*)$ to $\mathcal{T}_2$, continue the simulation of $\mathcal{T}_2$ until the verifier program V_2 produces a result. If V_2 rejects, abort; otherwise if V_2 accepts for c'^* and no tuple $(x'^*, \hat{y}', c'^*, j')$ with $c'^* = c^*$ has been stored, output $\perp$. Otherwise output $\hat{y}'$.
  - If $\mathcal{A}_{\mathsf{R}}$ sends a message $(\texttt{reset})$ to $\mathcal{T}_2$, reset $\mathcal{T}_2$.
- (Execution) If $\mathcal{A}_{\mathsf{R}}$ sends a message $(\texttt{run}, w)$ in Step 3, but no tuple $(x^*, \hat{y}, c^*, j)$ has been stored, abort. Otherwise, send $(\texttt{run}, w)$ to $\mathcal{F}_{\text{wrap}}^{\text{hybrid2}}$ and output whatever $\mathcal{F}_{\text{wrap}}^{\text{hybrid2}}$ outputs.

Figure 5.19: Simulator for a corrupted receiver in the protocol $\Pi_{\text{wrap}}^{\text{hybrid2}}$.

Lemma 5.8.1. *Given that* PRF *is a pseudorandom function, Experiment 0 and Experiment 1 are computationally indistinguishable from* $\mathcal{Z}$*'s view.*

Proof. Fix a PPT-environment $\mathcal{Z}$ that distinguishes between Experiment 0 and Experiment 1 with non-negligible advantage ϵ. We will construct a distinguisher $\mathcal{D}$ that distinguishes between a pseudorandom function PRF and a random oracle with advantage ϵ. $\mathcal{D}$ has oracle access to a function F which is either a random function or a pseudorandom function. $\mathcal{D}$ simulates to $\mathcal{Z}$ the protocol as specified, except that whenever the pseudorandom function PRF is evaluated, $\mathcal{D}$ makes an oracle call to F.

First assume that F is a pseudorandom function with uniformly chosen seed $s \leftarrow \{0,1\}^\kappa$. Then $\mathcal{Z}$'s view is distributed identically to Experiment 0. If, on the other hand, F is a random function, then $\mathcal{Z}$'s view is identically distributed to Experiment 1. Therefore, the distinguishing-advantage of $\mathcal{D}$ is exactly ϵ, which contradicts the pseudorandomness property of PRF. ■

The indistinguishability of Experiment 1 and Experiment 2 follows from the statistical binding property of the commitment scheme COM. The event that $\mathcal{T}_1$ outputs $\perp$ even though V_1 accepts happens only when $\mathcal{A}_{\mathsf{R}}$ manages to convince $\mathcal{T}_1$ that c^* is a commitment to two different values x^* and x'^*. Since COM is statistically binding, this happens only with negligible probability, and $\mathcal{A}_{\mathsf{R}}$ would thus break the soundness of the argument system $(\mathsf{P}_1, \mathsf{V}_1)$.

Lemma 5.8.2. *Given that* $(\mathsf{P}_1, \mathsf{V}_1)$ *suffices the computational soundness property and* COM *is a statistically binding commitment, Experiment 1 and Experiment 2 are computationally indistinguishable from* $\mathcal{Z}$*'s view.*

Proof. Fix a PPT-environment $\mathcal{Z}$ that distinguishes between Experiment 1 and Experiment 2 with non-negligible advantage ϵ. First note that Experiment 1 and Experiment 2 are identically distributed if the event that $\mathcal{T}_1$ outputs $\perp$, even though V_1 accepts, does not happen. Therefore, assume that $\mathcal{A}_\mathsf{R}$ succeeds in making $\mathcal{T}_1$ output $\perp$ even though V_1 accepts. Assume further that $\mathcal{A}_\mathsf{R}$ makes at most $m = \mathsf{poly}(\kappa)$ calls to the token $\mathcal{T}_1$. We will construct a malicious prover P^* that breaks the soundness property of the argument system $(\mathsf{P}_1, \mathsf{V}_1)$ with non-negligible probability. We use a variant of the soundness experiment where the malicious prover announces a false statement and then convinces the verifier that this statement is true. A simple averaging argument yields that this soundness notion immediately implies the regular soundness notion.

P^* first chooses an index $i \in \{1, \ldots, m\}$ uniformly at random. P^* then simulates the interaction of $\mathcal{Z}$ and $\mathcal{S}_2$ until $\mathcal{A}_\mathsf{R}$ makes the i'th call to $\mathcal{T}_1$. P^* now announces the statement (x^*, c^*) sent by $\mathcal{A}_\mathsf{R}$ to his own soundness experiment and then redirects the prover-messages sent to $\mathcal{T}_1$ to the verifier V it interacts with. P^* terminates after this argument.

Due to the statistical binding property of COM, c^* contains a unique value x^* except with negligible probability ν. This means that if $\mathcal{A}_\mathsf{R}$ manages to convince $\mathcal{T}_1$ in Experiment 2 that both $(x^*, c^*) \in \mathcal{L}_1$ and $(x'^*, c^*) \in \mathcal{L}_1$ for $x^* \neq x'^*$, at least one of the statements must be false with overwhelming probability.

We claim that P^* convinces V of a false statement with probability at least $\frac{\epsilon}{m} - \nu$. First note that the probability that such a false statement occurs in Experiment 2 is at least ϵ as $\mathcal{Z}$'s distinguishing advantage for Experiment 1 and Experiment 2 is at least ϵ and $\mathcal{Z}$ cannot (statistically) distinguish those two experiments if no false statement occurs. As there are at most m different statements proven in the protocol run, the probability that the index i coincides with the index of a false statement is at least $\frac{1}{m}$. Combined, P^* proves a false statement with probability at least $\frac{\epsilon}{m} - \nu$, which is non-negligible. This contradicts the soundness property of the argument system $(\mathsf{P}_1, \mathsf{V}_1)$. ■

Experiment 2 and Experiment 3 are indistinguishable given that SIG is an EUF-CMA-secure signature scheme. If $\mathcal{T}_2$ outputs $\perp$ even though the verifier V_2 accepts, then $\mathcal{A}_\mathsf{R}$ has convinced V_2 that it possesses a signature on a commitment c^* for which it never received a signature σ from $\mathcal{T}_1$. The proof of knowledge property of $(\mathsf{P}_2, \mathsf{V}_2)$ enables us to extract such a forged signature, contradicting the EUF-CMA-security of SIG.

Lemma 5.8.3. *Given that* $(\mathsf{P}_2, \mathsf{V}_2)$ *suffices the proof of knowledge property and* SIG *is EUF-CMA-secure, Experiment 2 and Experiment 3 are computationally indistinguishable from* $\mathcal{Z}$*'s view.*

Proof. From $\mathcal{Z}$'s view, Experiment 2 and Experiment 3 are identically distributed conditioned on the event that $\mathcal{A}_\mathsf{R}$ does not convince $\mathcal{T}_2$ that it possesses a valid signature σ for a commitment c^* before receiving such a σ from $\mathcal{T}_1$.

Thus, a $\mathcal{Z}$ distinguishing between Experiment 2 and Experiment 3 must convince $\mathcal{T}_2$ of this before receiving σ on c^* from $\mathcal{T}_1$. Assume that $\mathcal{Z}$ causes this event with

non-negligible probability ϵ. We will construct an adversary $\mathcal{B}$ that breaks the EUF-CMA-security of the signature scheme SIG with non-negligible probability. Assume that $\mathcal{A}_\mathsf{R}$ makes at most $m = \mathsf{poly}(\kappa)$ calls to the token $\mathcal{T}_2$.

Let vk be $\mathcal{B}$'s input from the EUF-CMA-experiment. $\mathcal{B}$ first chooses an index $i \in \{1, \ldots, m\}$ uniformly at random. Let $\mathcal{S}_3'$ be a simulator the behaves exactly like $\mathcal{S}_3$ except for two modifications. First, $\mathcal{S}_3'$ uses the vk provided by the EUF-CMA-experiment instead of generating a key pair $(\mathsf{vk}, \mathsf{sgk})$ by itself. Second, when $\mathcal{S}_3$ signs a message c^* by computing $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, c^*)$, $\mathcal{S}_3'$ replaces this by a call to the signing oracle that is provided by the EUF-CMA experiment. $\mathcal{B}$ obtains a signature σ for c^* and simulates the interaction between $\mathcal{Z}$ and $\mathcal{S}_3'$ until $\mathcal{A}_\mathsf{R}$ makes the i'th call to $\mathcal{T}_2$. Let c^* be the message sent by $\mathcal{A}_\mathsf{R}$. $\mathcal{B}$ now halts the computation of $\mathcal{Z}$. If the computation of $\mathcal{Z}$ would continue after this point, the subsequent messages passed by $\mathcal{A}_\mathsf{R}$ correspond to the messages of a malicious prover P^* for the argument system $(\mathsf{P}_2, \mathsf{V}_2)$. Thus, $\mathcal{B}$ can construct a malicious prover P^* from the state of the halted $\mathcal{Z}$ which basically continues the simulation of $\mathcal{Z}$ at its current state and forwards messages between $\mathcal{A}_\mathsf{R}$ and an external verifier V_2.

$\mathcal{B}$ can now take the code of P^* and run the extractor $\mathsf{Ext}_2(c^*, \mathsf{P}^*)$. Let $\hat{\sigma}$ be the output of Ext. $\mathcal{B}$ outputs $\hat{\sigma}$ to the EUF-CMA-experiment and terminates.

First notice that from $\mathcal{Z}$'s view, this simulation is identically distributed to Experiment 2. Thus, the event that $\mathcal{A}_\mathsf{R}$ succeeds in convincing $\mathcal{T}_2$ that it possesses a valid signature for a commitment c^* happens with probability at least ϵ. With probability at least $\frac{1}{m}$, the index i chosen by $\mathcal{B}$ matches the index of the proof where this event happens. Therefore, $\Pr[\langle \mathsf{P}^*, \mathsf{V} \rangle = 1] \geq \frac{\epsilon}{m}$. Due to the proof of knowledge property of the argument system $(\mathsf{P}_2, \mathsf{V}_2)$, $\Pr[\mathsf{Ext}(c^*, \mathsf{P}^*) \in w_{\mathcal{L}_2}(c^*)] > \Pr[\langle \mathsf{P}^*, \mathsf{V} \rangle = 1] - \nu \geq \frac{\epsilon}{m} - \nu$ for some negligible ν. Thus, with probability $\frac{\epsilon}{m} - \nu$ which is non-negligible, $\mathcal{B}$ outputs a valid signature σ on a commitment c^* for which it has not queried its signature oracle, thus breaking the EUF-CMA-security of the signature scheme SIG. ■

This concludes the proof of Lemma 5.8. □

5.4.4 Implications

One important and very powerful application of non-interactive two-party computation from resettable hardware that we want to mention here is virtual black-box (VBB) obfuscation. The concept of virtual black-box obfuscation was introduced by Barak et al. [BGI+01, BGI+12] and states that an adversary with direct access to a program cannot compute any predicate about the program that he could not have computed having just oracle access to the program. This means in particular that the VBB-obfuscated code of a program leaks no additional information other than the input/output behavior of the program. As it turns out, this definition of obfuscation is very strong, and Barak et al. [BGI+01, BGI+12] show that this notion of obfuscation is not achievable for all programs in the plain model.

In the model of tamper-proof hardware, however, this notion can be achieved trivially: just store the program inside a tamper-proof hardware token and send it to the receiver. Of course, this solution is very inefficient, requires a trusted token sender and a possibly stateful token. Thus, the challenge is to obtain a scheme where the token is untrusted, resettable and only has to perfom a small amount of work compared to the user of the obfuscated program. Solutions in the resettable tamper-proof

hardware model have been proposed by Goyal et al. [GIS+10] and, independently of our solution but with similar techniques, by Bitansky et al. [BCG+11].

We propose the following solution. Based on the resettable two-party computation from the previous sections, we implement a conditional decryption functionality for a fully homomorphic encryption (FHE) scheme. This conditional decryption functionality outputs the decryption of a value under the secret key of the FHE scheme if the encrypted input satisfies some predefined condition. In our case, this condition is a proof that the encrypted query was created by evaluating the obfuscated (i.e. encrypted) program with an input. Now, the sender uses the FHE scheme to encrypt a circuit representation of the program which can be homomorphically evaluated. Thus, most of the work is delegated to the receiver of the token, and the token only has to verify a proof and decrypt a value. In contrast to the solution presented by Goyal et al. [GIS+10], our protocol does not require the receiver to query the token for each gate of an obfuscated circuit. However, we need stronger computational assumptions.

A detailed description of our obfuscation scheme including a UC security proof is available via ePrint [DMMQN11], but this is not the focus of this thesis and hence we omit it here.

5.5 Relation to Two-Party Computation

In this chapter, we investigated the cryptographic strength of resettable tamper-proof hardware in this chapter. Our main focus was on reducing the computational assumptions and the number of tokens compared to previous results. To that end, we managed to find lower bounds for the number of tokens necessary and proved that non-black-box techniques are necessary to reach these lower bounds. Our protocols achieve these lower bounds and require only one-way functions. They are hence optimal both with respect to the number of tokens and the computational assumptions we made. Regarding the feasibility of (non-interactive) two-party computation, our results imply the following.

Computationally secure two-party computation: UC-secure two-party computation can be obtained e.g. by applying our compiler $\mathcal{C}_{ZK}$ from Section 5.3.1 to the statistically UC-secure protocol of Döttling et al. [DKMQ11] for oblivious transfer. This results in a computationally UC-secure OT protocol based on resettable hardware. By applying general feasibility results, e.g. [Kil88, IPS08], this implies UC-secure two-party computation. This is an improvement over Chandran et al. [CGS08] who need enhanced trapdoor permutations and 2 tokens, and also an improvement over [GIS+10, HPV15] who base OT on one-way functions and polynomially many tokens, even more so since they require that only one party must be able to produce tokens.

Computationally secure non-interactive two-party computation: Our constructions $\Pi_{\text{wrap}}^{\text{hybrid1}}$ and $\Pi_{\text{wrap}}^{\text{hybrid2}}$ shown in Sections 5.4.2 and 5.4.3 provide a CRS which can be used in conjunction with results for resettably-secure [GS09] and UC-secure [CLOS02] two-party computation to achieve UC-secure non-interactive two-party computation from resettable hardware tokens. This is an improvement over [GIS+10] who only achieved stand-alone security.

Model	computational 2PC		statistical 2PC	
	interactive	non-interactive	interactive	non-interactive
stateful	✓	✓	✓	✓
stateful (inc. com.)	✓	✓	✓	✓
stateful (outg. com.)	✓	✓	✗(✓)	✗
resettable	✓(1 Token)	✓(2 Tokens)	✗(✓) Commitment [DS13]	✗
bounded-resettable	✓	✓	✓	(✓)
reusable resettable	✓	✓	✗(✓)	✗

Table 5.20: Feasibility of interactive and non-interactive two-party computation from resettable tamper-proof hardware.

Due to the strong impossibilities concerning statistically secure two-party computation from resettable tokens (cf. Section 5.2) and already existing constructions for the remaining functionalities, statistically secure protocols are not further investigated in this chapter. Table 5.20 thus summarizes the state of the art.

6. Bounded-Resettable Tamper-Proof Hardware

From the results discussed in Chapter 5, we know that statistically secure two-party computation from resettable hardware is impossible for most interesting functionalities [GIMS10] (with the exception of commitments [GIMS10, DS13]). This stems from the fact that either the tokens completely hide their secret information, or leak all of the information. A malicious sender can thus cheat in the protocol, or the receiver can break the security. To circumvent this problem, it would be necessary to stop the information leakage at some point. If we stick to the previously defined model of resettable hardware, this is not possible. With each reset, the token receiver learns more information.

Our approach is to specify an upper bound on the number of resets that the token receiver can perform. We thus define a new notion of resettable tamper-proof hardware tokens that we call *bounded-resettable* hardware tokens, which can only be reset an a priori fixed number of times. One can imagine numerous scenarios where such a bound is naturally given.

Security for limited time. It is often the case that the security of a protocol is only needed for some limited amount of time (which is usually the case for commitments) and it therefore suffices to estimate an upper bound on the number of resets during the lifetime of the commitment.

Lifespan of hardware token. Physical hardware has a limited lifespan, e.g. typical smartcards have to be replaced after 5 years of use.[1] Thus security does not have to hold after that time.

Self-destructing hardware tokens. A token that self-destructs or makes its internal program inaccessible (i.e. an inherently stateful token) might be interrupted or delayed by an adversary, meaning the adversary might be able to

[1] See e.g. `http://www.datacard.com/knowledge-center/white-papers` for a comprehensive overview of current smartcard technologies and lifetimes. The paper on "Durability of Smart Cards for Government eID" specifically mentions lifespans between 3 and 10 years, depending on the used material.

query the token with several additional inputs before the token is actually useless. Thus a limited number of resets might be possible.

Finding a bound for all of the above mentioned scenarios seems particularly feasible because it can be derived from the response time of the hardware token and some reasonable safety margin. Interestingly, the bound only influences the token program while no additional assumption for the resettable token has to be made. Thus, in practice, standardized smart cards can be used as bounded-resettable hardware with no difference to the resettable token model.

Similar to the model of resettable hardware, where Goyal et al. [GIMS10] observe some interesting connections to the PCP model [FGL$^+$91, AS92] and the interactive PCP model [KR08], our model maintains the same connections for bounded-query PCPs [DFK$^+$93, KPT97]. In the PCP model, the prover sends the PCP to a PCP oracle that gives the verifier black-box access to the proof, which is very similar to storing the proof inside a tamper-proof token. The PCP cannot keep a state (it is usually defined to be a bit string), i.e. some type of resettable hardware provides an accurate representation. Depending on whether a considered protocol contains direct interaction between the token issuer and the token receiver or not, we are either in the PCP or in the IPCP model, respectively. The difference between the model we consider in the following and the (interactive) PCP model is that maliciously issued tokens/oracles can be stateful. This seems reasonable since it is hard to verify that a malicious token is stateless and executes a predefined functionality. We show that this weakened version of the PCP model still allows non-interactive zero-knowledge with $O(1)$ rounds of oracle queries and even general (interactive) secure computation. Moreover, our construction for zero-knowledge matches the result on ZK-PCPs by Kilian et al. [KPT97] in query/communication complexity, even with malicious stateful PCP oracles. The protocol of Kilian et al. [KPT97] can be made robust against malicious stateful oracles as well, but at the cost of issuing polynomially many oracles and having the verifier query each oracle only once. It is not clear, however, if their construction can be adapted to use only a constant number of oracles.

Our contribution. Our results in the new model cover most basic primitives. We construct two types of commitment schemes, one where the sender uses the token to commit himself at the receiver, and another commitment scheme where the sender sends a token to the receiver, who in turn uses the token to commit himself at the sender. Based on these commitment schemes we construct an OT protocol, where only one party must be able to create and program hardware tokens. We also present a zero-knowledge protocol that is bounded-resettable zero knowledge, i.e. the prover is resettable. This protocol can be made completely non-interactive.

All of our constructions have a constant number of rounds and require only a constant number of tokens. Our results thus demonstrate the first positive results regarding the feasibility of general statistically UC-secure two-party computation with any type of resettable hardware.

Our techniques. The main technical difficulty that we face is that we have to make the token functionality verifiable by the token receiver while at the same time allowing extraction by the simulator. At the heart of our constructions is what we call an *oracle validation scheme* that provides exactly the above mentioned features. This is a generalization of techniques that were previously employed in other protocols, e.g. in [DKMQ11].

Intuitively, the oracle validation scheme that we propose is based on polynomials of sufficiently large degree q, such that the receiver cannot learn the polynomial completely (via resets). The simulator on the other hand is not bound by this restriction, because he has the token code at his disposal, and can thus learn the stored polynomial by asking $q + 1$ queries and interpolating the polynomial. Care has to be taken to prevent the sender from storing an arbitrary maliciously created polynomial in the token which might not be extractable, e.g. because it is not a degree q polynomial.

Structure of this chapter. A preliminary version of this chapter is due to Döttling et al. [DKMN15]. The remaining part of the chapter is structured as follows. First, we formally define the UC-functionality for bounded-resettable hardware tokens in Section 6.1. We then introduce the concept of an oracle validation scheme in Section 6.2, which is then used in Section 6.3 to construct commitments and eventually OT. In Section 6.4 we also show that non-interactive zero-knowledge is possible in the bounded-resettable hardware model. A short summary of our results is given in Section 6.5.

6.1 Model

To formally argue about the security of our protocols, we define and discuss the ideal functionality for bounded-resettable tamper-proof hardware in Figure 6.1. It is a slightly modified version of the $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$-functionality described in Section 5.1. The token sender provides a (w.l.o.g. deterministic) Turing machine and the receiver can then run it once on an input word of his choice, staying oblivious of any internal secrets. A malicious receiver can reset the token and query it repeatedly, until some bound q is reached and the functionality does not respond any more. The query bound q models an estimation for how often an adversary can reset a token that is meant to shut down for good after the first query. All our protocols rely on q being polynomially bounded in the security parameter. A smaller bound q implies better efficiency.

We stress that tokens are not actually required to contain a state that counts the number of queries. Our definition of $\mathcal{F}_{\text{wrap}}^{\text{b-rs}}$ is just the most general way to model *any* kind of token for which an upper bound of resets can be derived. Intuitively, we just adapt the token program to the bound, while assuming a normal resettable hardware token.

Further note that our definition can be canonically extended to tokens that can be queried more than once by honest users as well. However, our approach has the advantage of being trivially secure against tokens that maliciously change their functionality depending on the input history.

Our model is weaker than the stateful-token model in the sense that no previously known protocol with stateful tokens can tolerate even a single reset. The security of all known protocols would completely break down. Therefore, none of the known positive results for stateful tokens do carry over to our model (unless $q = 1$). In turn, bounded-resettable protocols can be trivially implemented from unresettable stateful tokens. So, our results are strictly stronger than the corresponding results for stateful tokens. On the other hand, bounded-resettable tokens are strictly more powerful than arbitrarily resettable (i.e., standard stateless) tokens, since our constructions of non-interactive statistically secure commitments and statistically secure OT are

Functionality $\mathcal{F}_{\text{wrap}}^{\text{b-rs}}$

Implicitly parametrized by a query bound q. The variable *resets_left* is initialized by *resets_left* $= q - 1$.

Creation:

1. Await an input $(\texttt{create}, \mathsf{M}, t)$ from the token issuer, where M is a deterministic Turing machine and $t \in \mathbb{N}$. Store (M, t) and send $(\texttt{created})$ to the adversary. Ignore all further `create`-messages.

2. Await a message $(\texttt{delivery})$ from the adversary. Then, send $(\texttt{ready})$ to the token receiver.

Execution:

3. Await an input $(\texttt{run}, w)$ from the receiver. Run M on input w. When M halts without generating output or t steps have passed, send a special symbol $\perp$ to the receiver; otherwise send the output of M.

Reset (adversarial receiver only):

4. Upon receiving a message $(\texttt{reset})$ from a corrupted token receiver, verify that *resets_left* > 0. If so, decrease *resets_left* by 1 and go back to Step 3; otherwise ignore that message.

Figure 6.1: The wrapper functionality by which we model bounded-resettable tamper-proof hardware. The runtime bound t is merely needed to prevent malicious token senders from providing a perpetually running program code M; it will be omitted throughout the rest of the chapter.

known to be impossible with the fully resettable hardware tokens [GIMS10].

Bounded-resettable hardware combined with computationally secure primitives obviously yields no qualitative improvement, since with computational assumptions, general interactive and non-interactive UC-secure computation is already possible (cf. Sections 5.3 and 5.4).

6.2 Tools

In this section we introduce our main technical tool, which we call *oracle validation schemes* (OVS), that enforce honest oracle programming. This is usually necessary in the scenario where a (possibly malicious) sender can arbitrarily program an oracle and send it to the receiver. The honest receiver needs to be sure that the functionality stored in the oracle behaves as specified by the protocol.

We thus require the oracle functionality to be drawn from some predefined distribution of functionalities, and that the receiver will abort if the sender programs the oracle with a functionality outside of the allowed distribution. If the receiver needs only one query to the oracle to validate the functionality, we call the scheme a *query-once oracle validation scheme* (qo-OVS). All properties of an OVS hold with respect to unbounded parties.

We will later use an OVS in both of our commitment constructions based on

bounded-resettable tamper-proof hardware. Nonetheless, the notion of an oracle validation scheme is more general and can also be applied to other types of oracles. One can for instance consider the verification step in the OT protocol of Döttling et al. [DKMQ11] as an (ad-hoc) oracle validation scheme, albeit for a stateful hardware token. We thus believe that this tool finds application in other contexts as well and might therefore be of independent interest.

First, we formally define query-once oracle validation schemes in Section 6.2.1, and then present an instantiation of a qo-OVS for a q-bounded oracle in Section 6.2.2 that we later use with bounded-resettable hardware.

6.2.1 Query-Once Oracle Validation Scheme

We first have to define the security properties which we require from an oracle validation scheme. Apart from correctness, we want privacy, i.e. the oracle $\mathcal{O}$ must not leak any information about f apart from $f(x)$. Additionally, we need an efficient extractor with rewindable access to the oracle that extracts the oracle function f with overwhelming probability.

Definition 6.1. *We denote by $\mathcal{O}(\mathsf{O})$ an oracle $\mathcal{O}$ that executes a program O. A query-once oracle validation scheme qo-OVS consists of three algorithms* Create, $\mathsf{Validate}$ *and* $\mathsf{Query}^{\mathcal{O}}$ *such that*

- $\mathsf{Create}(\kappa)$ *is a PPT algorithm that takes as input a security parameter κ, creates an (efficiently evaluable) function f, programs an oracle functionality O that implements f and outputs (O, f).*
- $\mathsf{Validate}(c, f)$ *is a PPT algorithm that takes as input a challenge c and a function f and outputs a function $\tilde{f}$.*
- $\mathsf{Query}^{\mathcal{O}(\mathsf{O})}(x, \tilde{f}, c)$ *is a deterministic algorithm with oracle access to $\mathcal{O}$ that takes as input a random value x and the validation values c and $\tilde{f}$. It outputs $f(x)$ or $\perp$.*

and the following properties hold:

- ***Correctness:*** *If an oracle is created correctly, then the validation will always succeed, i.e.* $\forall f, c, x : \Pr[\mathsf{Query}^{\mathsf{Create}(\kappa)}(x, \mathsf{Validate}(c, f), c) = f(x)] = 1$.
- ***Privacy:*** *Let X denote the set of inputs that are sent to $\mathcal{O}$. Privacy holds if an adversary $\mathcal{A}$ has only negligible probability to guess the image of the oracle for any value not in X, i.e.* $\forall \mathcal{A}, \tilde{f}, c : \Pr[(x^*, f(x^*)) \leftarrow \mathcal{A}^{\mathcal{O}}(\tilde{f}, c) \wedge x^* \notin X] < \mathsf{negl}(\kappa)$.
- ***Extractability:*** *There exists an extractor $\mathsf{Ext}_{\mathsf{OVS}}$ that extracts from any (possibly corrupted) oracle program O^* and any validation values $(c, \tilde{f}^*)$ with some overwhelming probability ρ' a valid function f such that for uniformly random input x the outputs of $\mathsf{Query}^{f}(x, \tilde{f}^*, c)$ and $\mathsf{Query}^{\mathcal{O}(\mathsf{O}^*)}(x, \tilde{f}^*, c)$ are identical. The expected runtime of $\mathsf{Ext}_{\mathsf{OVS}}$ on input $(\mathsf{O}^*, \tilde{f}^*, c)$ has to be asymptotically bounded by $(\kappa \cdot |\mathsf{O}^*|)^{O(1)} \cdot \rho^{-1}$, where $|\mathsf{O}^*|$ is the size of the oracle program and ρ is the probability that $\mathsf{Query}^{\mathcal{O}(\mathsf{O}^*)}(x, \tilde{f}^*, c)$ outputs a value $f(x) \neq \perp$ for a uniformly random x.*

The general methodology of application of an oracle validation scheme is as follows. A sender party uses $\mathsf{Create}(\kappa)$ to generate an oracle. After this step, a receiver can draw a uniformly random challenge c and request a validation $\tilde{f}$ of the oracle functionality from the sender. We stress that this is allowed only once, i.e. the sender has to make sure that after the first request, all future requests are denied. These steps can be done in a setup phase. When the inputs are fixed, the receiver inputs x into the oracle and can validate that the oracle behaves as specified by the sender via the validation with $\tilde{f}$.

Remark. The definition can canonically be extended to allow any number k of queries to validate the functionality by exchanging x for $\mathbf{x} = (x_1, \ldots, x_k)$ and adapting the privacy requirement accordingly. This yields a general oracle validation scheme.

6.2.2 Oracle Validation Scheme for a q-Bounded Oracle

We now present a (query-once) oracle validation scheme based on any q-bounded oracle functionality, i.e. an oracle that admits only q queries. When using this protocol with bounded-resettable hardware, we only have to replace the oracle $\mathcal{O}$ by the wrapper functionality for the hardware token.

This section is divided into three parts. In Section 6.2.2.1 we present our protocol, we then move on to prove its security in Sections 6.2.2.2 and 6.2.2.3.

6.2.2.1 Protocol $\Pi_{\mathrm{OVS}}^{\text{q-bound}}$

The basic idea behind the protocol is to have the oracle implement a truly random polynomial p of degree q, such that the receiver, using only q queries, cannot learn p. While this directly yields privacy, the extraction of p turns out to be quite involved.

Oracle Validation Scheme $\Pi_{\mathrm{OVS}}^{\text{q-bound}}$

Let $\mathcal{O}_q$ be a q-bounded UC-hybrid functionality such that the simulator receives O.

- $\mathsf{Create}(\ell, q, n)$
  Sample two polynomials $p, p' \leftarrow \mathbb{F}_{2^\ell}^n[X]$ of degree at most q uniformly at random. Create a program O such that on input x it outputs $(p(x), p'(x))$. Output $(\mathsf{O}, (p, p'))$.
- $\mathsf{Validate}(\lambda, (p, p'))$
  Compute $\tilde{p} = p \cdot \lambda + p'$ and output $\tilde{p}$.
- $\mathsf{Query}^{\mathcal{O}_q(\mathsf{O})}(x, \tilde{p}, \lambda)$
  Send $x \in \mathbb{F}_{2^\ell}^n$ to $\mathcal{O}_q(\mathsf{O})$ and obtain (y, y'). Verify that $\deg(\tilde{p}) \leq q$ and $y \cdot \lambda + y' = \tilde{p}(x)$. If that check passes, output y, otherwise output $\perp$.

Figure 6.2: Construction of a query-once validation scheme for a q-bounded oracle.

Consider the protocol given in Figure 6.2. The Create-algorithm draws two random n-tuples of polynomials p, p' of a specified degree q and creates a functionality O that simply evaluates the polynomials on arbitrary inputs. If the oracle allows less than $q + 1$ queries, then the party querying the oracle cannot learn the polynomials and each answer of the oracle looks completely random. This guarantees the privacy property of the oracle validation scheme. During the $\mathsf{Validate}$-phase, a linear

combination $\tilde{p}$ of the two polynomials is computed, but due to the privacy of the scheme, this does not help to learn the two polynomials separately. Once the oracle is queried with a Query-call, any deviating behavior of the oracle from $\tilde{p}$ will be caught with overwhelming probability. This is guaranteed by the uniformly random input x, because a polynomial different from the one stored in the oracle will with overwhelming probability evaluate to a value differing from the oracle answer.

Our oracle validation scheme takes an additional parameter n during the oracle creation which specifies the dimension of the field. This will later allow for multiple commitments from a single oracle and has no influence on the security of the scheme.

6.2.2.2 Security Proof for $\Pi_{\mathrm{OVS}}^{\text{q-bound}}$

Next, we show that the oracle validation scheme $\Pi_{\mathrm{OVS}}^{\text{q-bound}}$ is indeed extractable—efficiency, correctness, and privacy are straightforward to see. The extractor construction is the main ingredient for our upcoming UC proofs, since our extractor is straight-line when given access to the oracle code.

Theorem 6.1. *The protocol $\Pi_{\mathrm{OVS}}^{\text{q-bound}}$ in Figure 6.2 describes an oracle validation scheme as per Definition 6.1. In particular, there exists an extractor $\mathsf{Ext}_{\mathsf{OVS}}$, such that on input $(\mathsf{O}^*, \tilde{p}^*, \lambda)$ with O^* and $\tilde{p}^*$ possibly corrupted, it holds:*

- *Provided arbitrarily rewindable access to O^* and given $(\tilde{p}^*, \lambda)$ with honestly chosen and uniformly random $\lambda \in \mathbb{F}_{2^\ell}$, $\mathsf{Ext}_{\mathsf{OVS}}$ computes a polynomial $\bar{p} \in \mathbb{F}_{2^\ell}^n[X]$ of degree at most q.*
- *If the value x is uniformly random, then with some overwhelming probability $1-\rho'$ (taken over the randomness of λ, x, and $\mathsf{Ext}_{\mathsf{OVS}}$'s random tape), $\mathsf{Query}^{\bar{p}}$ either rejects or outputs $p(x)$. In particular, we have a failure probability $\rho' \leq |2^\ell|^{-\Omega(1)}$.*
- *For all possible values $(\tilde{p}, \lambda)$, the expected number of queries from $\mathsf{Ext}_{\mathsf{OVS}}$ to O^* is $(q+1)\cdot\rho^{-1}$, where ρ is the probability of $\mathsf{Query}^{\mathcal{O}_q(\mathsf{O}^*)}$ outputting a value $p(x) \neq \perp$ conditioned on $(\tilde{p}, \lambda)$ and averaged over all inputs $x \in \mathbb{F}_{2^\ell}^n$. The rest of $\mathsf{Ext}_{\mathsf{OVS}}$'s calculations have an overall time complexity which is polynomial in (ℓ, q, n).*

Proof. We have to prove the three properties described in Definition 6.1.

Correctness. Validation yields a polynomial $\tilde{p} = p \cdot \lambda + p'$. Evaluating $\mathcal{O}(\mathsf{O})$ on x results in $(p(x), p'(x))$. Thus the check $\tilde{p}(x) = p(x)\cdot\lambda + p'(x)$ will always hold and $\mathsf{Query}^{\mathcal{O}(\mathsf{O})}$ will output $p(x)$ with probability 1.

Privacy. Privacy follows from the fact that for each query x_i, only the values $(p(x_i), p'(x_i))$ are returned by the oracle. Since $p, p' \in \mathbb{F}_{2^\ell}^n[X]$ are uniformly random polynomials of degree q, all of the tuples $(p(x_i), p'(x_i))$ are distributed uniformly at random from an adversarial point of view up to the point where $q+1$ such tuples are obtained. But by assumption our oracle is q-bounded so that it is impossible to obtain more than q tuples. Also, the polynomial $\tilde{p}$ does not yield any information about p and p', if only one validation query (i.e. one challenge λ) is answered. Combined, this ensures that an adversarial receiver $\mathcal{A}$ outputs $(x^*, f(x^*))$ for some value x^* that was not sent to the oracle at most with probability $2^{-\ell}$, which is negligible.

Extractability. Consider the extraction algorithm $\mathsf{Ext}_{\mathsf{OVS}}$ given in Figure 6.3. The extractor runs a simple trial-and-error approach. It samples uniformly random values x_i and evaluates the oracle on these values to obtain values y_i until it has a list of $q+1$ tuples that pass the validation check. Using these tuples (x_i, y_i), $\mathsf{Ext}_{\mathsf{OVS}}$ interpolates a polynomial $\bar{p}$ and outputs it. This is the approximation of the oracle functionality.

Extractor $\mathsf{Ext}_{\mathsf{OVS}}$

Upon input $(\mathsf{O}^*, \tilde{p}^*, \lambda)$, start the following algorithm.

1. Sample a uniformly random value $x_i \leftarrow \mathbb{F}_{2^\ell}^n$ and run O^* on x_i to obtain (y_i, y_i').
2. Check if $\lambda \cdot y_i + y_i' = \tilde{p}^*(x_i)$, if not go to Step 1. Store x_i in X. If $|X| \geq q+1$, go to Step 3, otherwise go to Step 1.
3. Compute the minimal-degree interpolation polynomial $\bar{p} \in \mathbb{F}_{2^\ell}^n[X]$ with $\bar{p}(x_i) = y_i$ for all $x_i \in X$.

Output $\bar{p}$.

Figure 6.3: The extraction algorithm for the oracle validation scheme $\Pi_{\mathrm{OVS}}^{\text{q-bound}}$.

First, we have to estimate the expected number of queries from $\mathsf{Ext}_{\mathsf{OVS}}$ to O^*. The sampling of each x_i is a stochastic process with geometric distribution of the number of oracle queries: Given that ρ is the probability that $\mathsf{Query}^{\mathcal{O}_q(\mathsf{O}^*)}$ returns $y \neq \perp$ conditioned on $(\lambda, \tilde{p}^*)$, the expected number of queries for sampling one x_i is $\sum_{j=1}^{\infty} j \cdot (1-\rho)^{j-1} \cdot \rho = \rho^{-1}$. The sampling of $x_1, \ldots, x_{q+1}$ hence requires $(q+1) \cdot \rho^{-1}$ queries to O^* on average.

Next, we turn to the question of how well the extracted polynomial $\bar{p}$ approximates the functionality of the oracle functionality O^*. We have to show that the minimal interpolation polynomial $\bar{p}$ based on the $q+1$ values in X represents the output of O^* on all inputs, i.e. the function stored in O^* is of degree q with overwhelming probability for a random challenge λ.

We do so by proving the following statement: for each $(\tilde{p}^*, \lambda)$, O^* implements a polynomial of degree at most q with some overwhelming probability $1 - \rho'$. To maintain the readability of this proof and a modular presentation, we defer the proof of the above mentioned statement to Lemma 6.2.

Combining Lemma 6.2 with the $q+1$ extracted tuples (x_i, y_i), we have that $\mathsf{Ext}_{\mathsf{OVS}}$ extracts the oracle functionality with overwhelming probability. This concludes our proof. □

6.2.2.3 $\mathsf{Ext}_{\mathsf{OVS}}$ Approximates O^* with Overwhelming Probability

We will now prove that the extractor as given in Figure 6.3 only requires $q+1$ values to correctly interpolate the oracle functionality. Before we start, we need to introduce some notation.

Interpolation polynomials. Let $\mathbb{F}$ be a finite field and let $n \in \mathbb{N}$. Given any mapping $f : \mathbb{F} \to \mathbb{F}^n$ and any non-empty set $M \subseteq \mathbb{F}$, let $\hat{\mathrm{p}}_M^{(f)} \in \mathbb{F}^n[X]$ denote

the unique polynomial of degree at most $|M|-1$, such that $\hat{\mathrm{p}}_M^{(f)}(x) = f(x)$ for all $x \in M$.

Generalization of the degree operator. Let $\mathbb{F}$ be a finite field and let $n \in \mathbb{N}$. For each polynomial $p := (p_1, \ldots, p_n) \in \mathbb{F}^n[X]$ we define its degree as $\deg(p) := \max_{i=1}^{n}(\deg(p_i))$. Further, given any mapping $f : \mathbb{F} \to \mathbb{F}^n$ and any non-empty set $M \subseteq \mathbb{F}$, let $\deg_M(f) := \deg(\hat{\mathrm{p}}_M^{(f)})$. For convenience we set $\deg_\emptyset(f) := -\infty$ and $\deg_M(f, f') := \max\Big(\deg_M(f), \deg_M(f')\Big)$.

Bipartite graphs and neighborhoods. We denote a bipartite graph G as a triple (V, U, E), where V and U are the vertex sets of the two parts of G and $E \subseteq V \times U$ is the set of edges of G. The neighborhood of any vertex $v \in V \cup U$ in G is denoted as $\mathcal{N}_G(v)$, or $\mathcal{N}(v)$ for short, and its degree as $\deg_G(v) := \big|\mathcal{N}_G(v)\big|$. Note that $\mathcal{N}_G(v) \subseteq U$ for all $v \in V$ and $\mathcal{N}_G(u) \subseteq V$ for all $u \in U$.

Lemma 6.2. *Let a mapping $s^* : \mathbb{F} \to \mathbb{F}^n[X]$, $\lambda \mapsto \tilde{p}_\lambda$ and a mapping $\mathsf{O}^* : \mathbb{F} \to \mathbb{F}^n \times \mathbb{F}^n$, $x \mapsto \Big(o(x), o'(x)\Big)$ be given. Then it holds for uniformly random $\lambda \in \mathbb{F}$ and $\tilde{p}^* \in \mathbb{F}^n$ with $\deg \tilde{p}^* \leq q$ that $\Pr[\deg(\mathsf{O}^*) > q] \leq \rho'$, where ρ' is negligible in $|\mathbb{F}|$.*

Proof. Our main idea is to define a relation between challenges λ and oracle queries x, such that x is related to λ if the validation succeeds. We then show that for each λ, the set of related values implies a polynomial of degree q. Let s^* denote the deterministic worst-case strategy of an adversary trying to prevent the extraction given a challenge λ. s^* maps each possible challenge $\lambda \in \mathbb{F}$ to a polynomial $\tilde{p}^*_\lambda \in \mathbb{F}^n[X]$ with $\deg(\tilde{p}^*_\lambda) \leq q$. Thus, for each combination of λ and x the output of O^* is fixed.

We represent the relation between λ and x as a bipartite graph, where a left-hand vertex λ is adjacent to a right-hand vertex x if O^* outputs (y, y') such that $y \cdot \lambda + y' = \tilde{p}^*(x)$. We want to show that after removing only a few "bad" edges, the graph decomposes into complete bipartite subgraphs, where the oracle functionality O^* behaves like a degree-q polynomial for all inputs x that pass the consistency check.

More formally, we will show that for a bipartite graph $G = (V, U, E)$ with V containing all λ and U containing all x, there exists a subset of edges $E_{\mathsf{bad}} \subset E \subseteq V \times U$ such that

1. uniformly random λ and x are adjacent via $e \in \mathsf{E}_{\mathsf{bad}}$ only with negligible probability, namely $|E_{\mathsf{bad}}|/|\mathbb{F}|^2 \leq |\mathbb{F}|^{-\Omega(1)}$, and

2. after removal of E_{bad} from G, O^* implements on each neighborhood of a possible challenge λ a polynomial function of degree at most q.

Since the graph $G' = (V, U, E \backslash E_{\mathsf{bad}})$ differs from G only in the negligible amount of removed edges, we know that the oracle O^* behaves on all but a negligible fraction of inputs like a degree-q polynomial, which $\mathsf{Ext}_{\mathsf{OVS}}$ can extract by making $q+1$ queries.

The proof is very technical and we modularize it by stating several separate lemmata, which we combine to obtain our claim. One of the key insights that we use is that for two distinct challenges λ_1 and λ_2, O^* implements a polynomial of degree at most q for all inputs x that are accepted for both λ_1 and λ_2.

Lemma 6.2.1. *Let any finite field $\mathbb{F}$, a dimension $n \in \mathbb{N}$ and two mappings $o, o' : \mathbb{F} \to \mathbb{F}^n$ be given. Further, let some family of polynomials $\{\tilde{p}_\lambda\}_{\lambda\in\mathbb{F}} \subseteq \mathbb{F}^n[X]$ be given, let $q := \max_{\lambda\in\mathbb{F}}(\deg(\tilde{p}_\lambda))$ and for each $\lambda \in \mathbb{F}$ let $M_\lambda := \{x \in \mathbb{F} \mid \lambda \cdot o(x) + o'(x) = \tilde{p}_\lambda(x)\}$. Then it holds for all distinct $\lambda_1, \lambda_2 \in \mathbb{F}$ that $\deg_{M_{\lambda_1}\cap M_{\lambda_2}}(o, o') \leq q$.*

Proof. We first show that the following claim holds.

Claim 1. *For each $M \subseteq \mathbb{F}$ there exists at most one $\lambda \in \mathbb{F}$ with $\deg_M(\lambda \cdot o + o') < \deg_M(o, o')$.*

Note that for arbitrary mappings $f, g : \mathbb{F} \to \mathbb{F}^n$ and any coefficients $\alpha, \beta \in \mathbb{F}$, $\hat{\mathrm{p}}_M^{(\alpha f+\beta g)} = \alpha \cdot \hat{\mathrm{p}}_M^{(f)} + \beta \cdot \hat{\mathrm{p}}_M^{(g)}$ and hence $\deg_M(\alpha f + \beta g) \leq \max\Big(\deg_M(f), \deg_M(g)\Big)$. Thus, if $\deg_M(\lambda_1 \cdot o + o') < \deg_M(o)$ and $\deg_M(\lambda_2 \cdot o + o') < \deg_M(o)$ for some distinct $\lambda_1, \lambda_2 \in \mathbb{F}$, we had the following contradiction:

$$\deg_M((\lambda_1 - \lambda_2) \cdot o) \;=\; \deg_M((\lambda_1 \cdot o + o') - (\lambda_2 \cdot o + o')) \;<\; \deg_M(o)$$

Analogously, if $\deg_M(\lambda_1 \cdot o + o') < \deg_M(o')$ and $\deg_M(\lambda_2 \cdot o + o') < \deg_M(o')$ for some distinct $\lambda_1, \lambda_2 \in \mathbb{F}$, we had the following contradiction:

$$\deg_M((\lambda_1 - \lambda_2) \cdot o') \;=\; \deg_M(\lambda_1 \cdot (\lambda_2 \cdot o + o') - \lambda_2 \cdot (\lambda_1 \cdot o + o')) \;<\; \deg_M(o')$$

Thus, Claim 1 must be true and we will use it to prove our lemma. Let two arbitrary but distinct $\lambda_1, \lambda_2 \in \mathbb{F}$ be given. Note that by definition of M_{λ_1},

$$\deg_{M_{\lambda_1}\cap M_{\lambda_2}}(\lambda_1 \cdot o + o') = \deg_{M_{\lambda_1}\cap M_{\lambda_2}}(\tilde{p}_{\lambda_1}) \leq q,$$

because for all $x \in M_{\lambda_1} \cap M_{\lambda_2}$, both $\lambda_1 \cdot o + o'$ and $\tilde{p}_{\lambda_1}$ behave identically. The same argument yields $\deg_{M_{\lambda_1}\cap M_{\lambda_2}}(\lambda_2 \cdot o + o') = \deg_{M_{\lambda_1}\cap M_{\lambda_2}}(\tilde{p}_{\lambda_2}) \leq q$ by definition of M_{λ_2}.

Thus, if $\deg_{M_{\lambda_1}\cap M_{\lambda_2}}(o, o') > q$, we had that $\deg_{M_{\lambda_1}\cap M_{\lambda_2}}(\lambda_1 \cdot o + o') < \deg_{M_{\lambda_1}\cap M_{\lambda_2}}(o, o')$ and also $\deg_{M_{\lambda_1}\cap M_{\lambda_2}}(\lambda_2 \cdot o + o') < \deg_{M_{\lambda_1}\cap M_{\lambda_2}}(o, o')$, which would contradict Claim 1. ■

Considering our graph G' as described above, Lemma 6.2.1 will be useful if G' does not contain any vertices x that have only one connection with a vertice λ. Then, for every challenge λ_1 with $\mathcal{N}_{G'}(\lambda_1) \neq \emptyset$, there exists a distinct second challenge λ_2 with $\mathcal{N}_{G'}(\lambda_1) = \mathcal{N}_{G'}(\lambda_2)$. Lemma 6.2.1 then implies that $\deg_{\mathcal{N}_{G'}(\lambda_1)}(o, o') = \deg_{\mathcal{N}_{G'}(\lambda_1)\cap\mathcal{N}_{G'}(\lambda_2)}(o, o') \leq q$.

Towards proving the existence of G', and thus a subset of edges E_{bad} as described above, we first need to find upper bounds, so that we can estimate the amount of edges that we have to remove. We start with a bound for intersection of the neighborhoods of challenges $\lambda_1, \ldots, \lambda_4$.

Lemma 6.2.2. *Let any finite field $\mathbb{F}$, a dimension $n \in \mathbb{N}$ and two mappings $o, o' : \mathbb{F} \to \mathbb{F}^n$ be given. Further, let some family of polynomials $\{\tilde{p}_\lambda\}_{\lambda\in\mathbb{F}} \subseteq \mathbb{F}^n[X]$ be given, let $q := \max_{\lambda\in\mathbb{F}}(\deg(\tilde{p}_\lambda))$ and for each $\lambda \in \mathbb{F}$ let $M_\lambda := \{x \in \mathbb{F} \mid \lambda \cdot o(x) + o'(x) = \tilde{p}_\lambda(x)\}$. Then for all $\lambda_1, \lambda_2, \lambda_3, \lambda_4 \in \mathbb{F}$ with $\lambda_1 \neq \lambda_2$ and $\lambda_3 \neq \lambda_4$ and $M_{\lambda_1} \cap M_{\lambda_2} \neq M_{\lambda_3} \cap M_{\lambda_4}$ we have that $q \geq |M_{\lambda_1} \cap M_{\lambda_2} \cap M_{\lambda_3} \cap M_{\lambda_4}|$.*

Proof. Let any $\lambda_1, \lambda_2, \lambda_3, \lambda_4 \in \mathbb{F}$ with $\lambda_1 \neq \lambda_2$ and $\lambda_3 \neq \lambda_4$ be given. Moreover, assume that $q < |M_{\lambda_1} \cap M_{\lambda_2} \cap M_{\lambda_3} \cap M_{\lambda_4}|$. We have to show that $M_{\lambda_1} \cap M_{\lambda_2} = M_{\lambda_3} \cap M_{\lambda_4}$.

Since any two polynomials $p, p' \in \mathbb{F}^n[X]$ are identical if they coincide on more than $\max\Big(\deg(p), \deg(p')\Big)$ nodes, we observe:

1. For all $M, M' \subseteq \mathbb{F}$ with $|M \cap M'| > \max\Big(\deg_M(o,o'), \deg_{M'}(o,o')\Big)$ it holds that $\hat{\mathrm{p}}_M^{(o)} = \hat{\mathrm{p}}_{M'}^{(o)}$ and $\hat{\mathrm{p}}_M^{(o')} = \hat{\mathrm{p}}_{M'}^{(o')}$.
2. For all $\lambda \in \mathbb{F}$ and $M \subseteq M_\lambda$ with $\deg_M(o,o') \leq q < |M|$ we have that $\tilde{p}_\lambda = \hat{\mathrm{p}}_M^{(o)} + \lambda \cdot \hat{\mathrm{p}}_M^{(o')}$.

Now, as $\deg_{M_{\lambda_1} \cap M_{\lambda_2}}(o,o') \leq q$ and $\deg_{M_{\lambda_3} \cap M_{\lambda_4}}(o,o') \leq q$ by Lemma 6.2.1, it follows by our Observation 1 that $\hat{\mathrm{p}}^{(o)}_{M_{\lambda_1} \cap M_{\lambda_2}} = \hat{\mathrm{p}}^{(o)}_{M_{\lambda_3} \cap M_{\lambda_4}}$ and $\hat{\mathrm{p}}^{(o')}_{M_{\lambda_1} \cap M_{\lambda_2}} = \hat{\mathrm{p}}^{(o')}_{M_{\lambda_3} \cap M_{\lambda_4}}$. Let $p := \hat{\mathrm{p}}^{(o)}_{M_{\lambda_1} \cap M_{\lambda_2}}$ and $p' := \hat{\mathrm{p}}^{(o')}_{M_{\lambda_1} \cap M_{\lambda_2}}$. Note that $\deg(p) \leq q$ and $\deg(p') \leq q$ by Lemma 6.2.1.

In conclusion, we get by our Observation 2 that $\tilde{p}_{\lambda_i} = p + \lambda_i \cdot p'$ for $i = 1, \ldots, 4$. However, as $p(x) = o(x)$ and $p'(x) = o'(x)$ for all $x \in (M_{\lambda_1} \cap M_{\lambda_2}) \cup (M_{\lambda_3} \cap M_{\lambda_4})$ by construction, it follows that $M_{\lambda_i} \supseteq (M_{\lambda_1} \cap M_{\lambda_2}) \cup (M_{\lambda_3} \cap M_{\lambda_4})$ for $i = 1, \ldots, 4$, and hence $M_{\lambda_1} \cap M_{\lambda_2} = M_{\lambda_3} \cap M_{\lambda_4}$. ■

Additionally, we also need a bound on the number of almost disjoint subsets of vertices to estimate the number of edges that we will remove later.

Lemma 6.2.3. *Let F be some finite universe and let $\varepsilon, \delta \in \mathbb{R}$, such that $|F|^\delta \leq |F|^\varepsilon/4$. Further, let $m \in \mathbb{N}_{>0}$ and $N_1, \ldots, N_m \subseteq F$, such that $|N_i| \geq |F|^{1-\delta}$ and $|N_i \cap N_j| \leq |F|^{1-\varepsilon-\delta}$ for all distinct i, j. Then $m < |F|^\varepsilon/2$.*

Proof. W.l.o.g. it suffices to show that $m \neq \Big\lceil |F|^\varepsilon/2 \Big\rceil$. We call $x \in F$ a *shared element*, if $x \in N_i \cap N_j$ for some distinct indices i, j. Note that each N_i contains at most $(m-1) \cdot |F|^{1-\varepsilon-\delta}$ shared elements, since by assumption $|N_i \cap N_j| \leq |F|^{1-\varepsilon-\delta}$ for all $j \neq i$. Hence, each N_i must contain at least $|F|^{1-\delta} - (m-1) \cdot |F|^{1-\varepsilon-\delta}$ non-shared elements. In other words, if $m = \Big\lceil |F|^\varepsilon/2 \Big\rceil$, we can estimate:

$$|F| \;\geq\; m \cdot \Big(|F|^{1-\delta} - (m-1) \cdot |F|^{1-\varepsilon-\delta}\Big) \;>\; \tfrac{|F|^\varepsilon}{2} \cdot \Big(|F|^{1-\delta} - \tfrac{|F|^\varepsilon}{2} \cdot |F|^{1-\varepsilon-\delta}\Big) \;=\; \tfrac{|F|^{1+\varepsilon-\delta}}{4}$$

However, since $|F|^{\varepsilon-\delta} \geq 4$ by assumption, this is a contradiction and thus concludes our proof. ■

Using the bounds of Lemma 6.2.2 and Lemma 6.2.3, we can now prove the existence of edges E_{bad} with the described properties. In particular, we show that the number of edges in E_{bad} can be bounded by $|V| \cdot |U|^{1-\delta} + |U|^{1+\varepsilon}$ for ε, δ with $16 \cdot |U|^{2\delta} \leq |U|^\varepsilon \leq \sqrt{|U|/q}$. Using a large enough field $\mathbb{F}$, we can choose $2\delta < \varepsilon < \frac{1}{2}$.

Lemma 6.2.4. *Let a finite bipartite graph $G := (V, U, E)$ and some constant $q \in \mathbb{R}$ be given, such that $q \geq \Big|\mathcal{N}_G(v) \cap \mathcal{N}_G(\bar{v}) \cap \mathcal{N}_G(v') \cap \mathcal{N}_G(\bar{v}')\Big|$ for all vertices $v, \bar{v}, v', \bar{v}' \in V$ with $v \neq \bar{v}$ and $v' \neq \bar{v}'$ and $\mathcal{N}_G(v) \cap \mathcal{N}_G(\bar{v}) \neq \mathcal{N}_G(v') \cap \mathcal{N}_G(\bar{v}')$. Then, for any $\varepsilon, \delta \in \mathbb{R}$ with $16 \cdot |U|^{2\delta} \leq |U|^\varepsilon \leq \sqrt{|U|/q}$, there exists a subset of edges $E_{\mathsf{bad}} \subseteq E$ such that $|E_{\mathsf{bad}}| < |V| \cdot |U|^{1-\delta} + |U|^{1+\varepsilon}$ and each connected component of $G' := (V, U, E \backslash E_{\mathsf{bad}})$ is either a single vertex or a complete bipartite graph.*

Proof. W.l.o.g. we can choose ε such that $16 \cdot |U|^{2\delta} = |U|^\varepsilon$, and q such that $|U|^\varepsilon = \sqrt{|U|/q}$. We define the auxiliary constants ε' and δ' as follows: Let $\varepsilon' := \delta + \log_{|U|} 4$, i.e. $|U|^\delta = |U|^{\varepsilon'}/4$, and let $\delta' := \varepsilon' + \delta$. Note that $|U|^{\delta'} = 4 \cdot |U|^{2\delta} = |U|^\varepsilon/4$ and hence $q = |U|^{1-2\varepsilon} = |U|^{1-\varepsilon-\delta'}/4$ by construction. Now, by the following three steps, we

transform our graph G into G' by removing edges—w.l.o.g. we even remove vertices together with all their adjacent edges.

Step 1: We first remove all $v \in V$ with $|\mathcal{N}_G(v)| \leq |U|^{1-\delta}$, thus deleting at most $|V| \cdot |U|^{1-\delta}$ edges.

Step 2: Next, we remove all vertices $v \in V$ with $\max_{v' \in V \setminus \{v\}} |\mathcal{N}_G(v) \cap \mathcal{N}_G(v')| \leq |U|^{1-\varepsilon'-\delta}$, thus deleting less than $|U|^{1+\varepsilon'}/2$ edges according to Lemma 6.2.3. Note that $|U|^{1+\varepsilon'}/2 < |U|^{1+\varepsilon}/2$, since even $|U|^{\varepsilon'+\delta} = |U|^{\varepsilon}/4$ by construction.

Step 3: Finally, we find a mapping $\sigma : V \to \mathcal{P}(U)$, where $\mathcal{P}(U)$ denotes the power set of U, such that σ assigns to each vertex $v \in V$ a maximum possible set of neighbors $N \subseteq \mathcal{N}_G(v)$ that can be written as $N = \mathcal{N}_G(v) \cap \mathcal{N}_G(\bar{v})$ with $\bar{v} \neq v$. Note that $|\sigma(v)| > |U|^{1-\delta'}$ for all $v \in V$ due to the vertex removal in Step 2. Further note that by construction $\mathcal{N}_G(v) \cap \mathcal{N}_G(v') = \sigma(v)$ for all distinct $v, v' \in V$ with $\sigma(v) = \sigma(v')$. Hence, by Lemma 6.2.2 it must hold that $\left|\sigma(v) \cap \sigma(v')\right| \leq q$ for all $v, v' \in V$ with $\sigma(v) \neq \sigma(v')$. Since $q = |U|^{1-\varepsilon-\delta'}/4$ by construction, it follows by Lemma 6.2.3 that the image space of σ consists of less than $|U|^{\varepsilon}/2$ different vertex sets $N \subseteq U$.

Now we are going to remove all edges (v, u), where $u \notin \sigma(v)$. This obviously transforms G into a disjoint composition of complete bipartite graphs (plus some unconnected vertices). So, it is only left to show that there exist no more than $|U|^{1+\varepsilon}/2$ such edges.

Consider any $N \in \sigma(V)$. As already mentioned above, we have that $\mathcal{N}_G(v) \cap \mathcal{N}_G(v') = \sigma(v)$ for all distinct vertices $v, v' \in V$ with $\sigma(v) = \sigma(v')$, or in other words, $\mathcal{N}_G(v) \cap \mathcal{N}_G(v') = N$ for all distinct $v, v' \in \sigma^{-1}(N)$. Hence, no $u \in U \setminus N$ can be adjacent to distinct vertices $v, v' \in \sigma^{-1}(N)$. I.e., there cannot exist more than $|U \setminus N|$ edges $(v, u) \in E$ with $\sigma(v) = N$ and $u \notin N$. Thus, in this final step we are removing at most $\sum_{N \in \sigma(V)} |U \setminus N|$ edges. We can estimate this by $|U|^{1+\varepsilon}/2$ however, since we have already shown above that the image space of σ consists only of less than $|U|^{\varepsilon}/2$ different vertex sets $N \subseteq U$.

We remove at most $|V| \cdot |U|^{1-\delta}$ edges in Step 1 and less than $|U|^{1+\varepsilon}/2$ edges in each of Step 2 and Step 3, which sums up to less than $|V| \cdot |U|^{1-\delta} + |U|^{1+\varepsilon}$ removed edges as claimed. ■

It now remains to remove all the vertices x with $\deg_{G'}(x) = 1$, thus removing at most an additional $|U|$ edges from E_{bad}. We thus have $|E_{\mathsf{bad}}| < |\mathbb{F}|^{2\delta} + |\mathbb{F}|^{1+\varepsilon} + |\mathbb{F}|$.

Now we show that $\deg_{\mathcal{N}_{G'}(\lambda)}(\mathsf{O}^*) \leq q$ for all $\lambda \in V$. Let $\lambda_1 \in V$ be arbitrary but fixed. W.l.o.g., $\mathcal{N}_{G'}(\lambda_1) \neq \emptyset$. Since there are no vertices $x \in U$ with $\deg_{G'}(x) = 1$ any more, we find some $\lambda_2 \in V \setminus \{\lambda_1\}$ with $\mathcal{N}_{G'}(\lambda_1) \cap \mathcal{N}_{G'}(\lambda_2) \neq \emptyset$. Since each connected component of G' still is either a single vertex or a complete bipartite graph, it even holds that $\mathcal{N}_{G'}(\lambda_1) = \mathcal{N}_{G'}(\lambda_2)$ and by Lemma 6.2.1 it follows that $\deg_{\mathcal{N}_{G'}(\lambda_1)}(\mathsf{O}^*) = \deg_{\mathcal{N}_{G'}(\lambda_2)}(\mathsf{O}^*) = \deg_{\mathcal{N}_{G'}(\lambda_1) \cap \mathcal{N}_{G'}(\lambda_2)}(\mathsf{O}^*) \leq q$.

Given E_{bad} as described above, we finally need to argue that the following event has probability $|\mathbb{F}|^{-\Omega(1)}$:

- O^* outputs (y, y') such that $y \cdot \lambda + y' = \tilde{p}^*(x)$ and

- one of the oracle queries $x_1, \ldots, x_{q+1}$ is adjacent via $e \in E_{\mathsf{bad}}$ to the challenge λ given by $(\tilde{p}^*, \lambda)$, or $x_i = x_j$ for some $i \neq j$.

Since $|E_{\mathsf{bad}}|/|\mathbb{F}_{2^\ell}|^2 \leq |\mathbb{F}_{2^\ell}|^{-\Omega(1)}$, we already have with probability $1 - |\mathbb{F}_{2^\ell}|^{-\Omega(1)}$ (taken over the randomness of λ) that the given challenge λ is only adjacent to an $|\mathbb{F}_{2^\ell}|^{-\Omega(1)}$-fraction of all inputs $x \in \mathbb{F}_{2^\ell}$ or λ is adjacent to $|\mathbb{F}_{2^\ell}|^{\Omega(1)}$ edges of which only an $|\mathbb{F}_{2^\ell}|^{-\Omega(1)}$-fraction is contained in E_{bad}. This implies that $\rho' \leq |\mathbb{F}_{2^\ell}|^{-\Omega(1)}$.
This concludes the proof of Lemma 6.2. □

6.3 Statistically Secure Two-Party Computation

In the following, we use the oracle validation scheme from the previous section to construct commitments in the bounded-resettable hardware model. First, in Section 6.3.1 we construct a commitment where the sender of the token uses the token to commit to a value. For this, the oracle functionality from the OVS is enhanced to hide the commitment information. We move on to present a commitment from the token receiver to the token sender in Section 6.3.2, which is—again—based on the OVS from the previous section. This will allow us later to construct protocols where tokens have to be sent in one direction only. Based on these two commitment schemes we present a protocol for oblivious transfer in Section 6.3.3, thus providing all the building blocks necessary for two-party computation.

6.3.1 Commitments from Token Sender to Token Receiver

In this section, we present a protocol that realizes (non-interactive) commitments in the $\mathcal{F}_{\text{wrap}}^{\text{b-rs}}$-hybrid model. With regard to the future applications of this protocol, we do not UC-realize $\mathcal{F}_{\text{COM}}$ (cf. Section 2.7.2.1), but an enhanced functionality shown in Figure 6.4. The commitment functionality $\mathcal{F}_{\text{COM}}^{\text{s-o}}$ allows a sender to commit to several values simultaneously, and unveil some subset of the commitments at once during the unveil phase.

The basic idea how the token issuer can commit himself to some secret s is quite simple. He stores a random degree-q polynomial p on the token and sends the token together with $r := s + p(0)$ to the receiver. The token lets the receiver evaluate p on arbitrary challenges x, except for $x = 0$. To unveil s, the sender sends a description of p. The scheme is perfectly hiding, because even a corrupted receiver can query the token on at most q inputs, receiving only randomness that is statistically independent of $p(0)$. The scheme is statistically binding, because for any two distinct unveil messages p, p' and a uniformly random token input x it holds with overwhelming probability (namely at least $1 - \frac{q}{|\mathbb{F}|-1}$, where $\mathbb{F}$ is the finite field in which all computations take place) that $p(x) \neq p'(x)$ and thus at least one unveil message will be inconsistent with the receiver's view.

Unfortunately, the scheme as stated above is not UC-secure against a corrupted sender. The reason for this is that the sender simulator must be able to extract the secret s from the token program and the commit message r. If the token is issued honestly and thus implements a degree-q polynomial p, the simulator can evaluate the token code on $q + 1$ different inputs, then reconstruct p, and compute $s = r - p(0)$. However, a maliciously issued token can implement an arbitrarily complicated function, which behaves like a degree-q polynomial only on a vanishing

Functionality $\mathcal{F}_{\text{COM}}^{\text{s-o}}$

Implicitly parametrized by a domain of secrets S and the number n of commitments to be implemented.

Commit phase:

1. Await an input $(\texttt{commit}, s)$ with $s = (s_1, \ldots, s_n) \in S^n$ from the sender. Store s, send $(\texttt{committed})$ to the adversary and ignore any further `commit`-messages.
2. Await a message $(\texttt{notify})$ from the adversary. Then send $(\texttt{committed})$ to the receiver.

Unveil phase:

3. Await an input $(\texttt{unveil}, I, \hat{s})$ with $I \subseteq \{1, \ldots, n\}$ and $\hat{s} = (\hat{s}_i)_{i \in I} \in S^{|I|}$ from the sender. Then, store $(I, \hat{s})$ and send $(\texttt{opened})$ to the adversary.
4. Await a message $(\texttt{output})$ from the adversary. Then, if $\hat{s} = (s_i)_{i \in I}$, send $(\texttt{opened}, I, \hat{s})$ to the receiver; otherwise, send a special reject message $\perp$.

Figure 6.4: Ideal functionality for multiple commitments with selective opening.

but still non-negligible fraction of inputs. It is at the very least unclear if one can extract the correct polynomial from such a token efficiently. Therefore, we employ our query-once oracle validation scheme from Section 6.2.2 to make the token extractable. In a first step, the sender uses the OVS to create a token program and then enhances the functionality such that queries for $x = 0$ are not answered. Then the protocol proceeds as described above, except that the receiver makes a validation request and cross-checks this with the answer from the token. Since the OVS allows for an n-dimensional input space, we can create n commitments at once. A formal specification of the protocol is given in Figure 6.5.

Theorem 6.2. *The protocol $\Pi_{\text{COM}}^{\text{s-o}}$ in Figure 6.5 statistically UC-realizes $\mathcal{F}_{\text{COM}}^{\text{s-o}}$ (cf. Figure 6.4) in the $\mathcal{F}_{\text{wrap}}^{\text{b-rs}}$-hybrid model, given that OVS is an oracle validation scheme as per Definition 6.1.*

Proof. **Corrupted sender.** The simulator has to extract the commitments of the corrupted sender in the commit phase. For this, the simulator simply runs the straight-line extractor $\mathsf{Ext}_{\mathsf{OVS}}$ of OVS with the token code and obtains a polynomial $\hat{p}$. The extracted polynomial allows the simulator to reconstruct the committed secret s from the sender's commit message r as $s = r - \hat{p}(0)$. Note that $\mathsf{Ext}_{\mathsf{OVS}}$ may have exponential runtime if the acceptance probability is small, but only needs to be run in this case if by the end of the commit phase it is not already clear that the receiver will reject anyway. Therefore, the simulator must first check that $y \neq \perp$ and then run $\mathsf{Ext}_{\mathsf{OVS}}$ only if the check is passed. The description of the simulator is given in Figure 6.6, where $\mathcal{A}_{\mathsf{S}}$ is the dummy adversary.

The output of $\mathcal{S}_{\mathsf{S}}$ is statistically indistinguishable from a real protocol run because the extractor $\mathsf{Ext}_{\mathsf{OVS}}$ has a runtime complexity of $\left(\ell \cdot |\mathsf{T}^*|\right)^{O(1)} \cdot \rho^{-1}$ (cf. Definition 6.1), where ρ is just the probability that the check in the commit phase is passed. Thus, $\mathsf{Ext}_{\mathsf{OVS}}$ has an expected simulation complexity of $\left(\ell \cdot |\mathsf{T}^*|\right)^{O(1)}$ and will therefore with overwhelming probability extract the polynomial $\hat{p}$ and thus $\mathcal{A}_{\mathsf{S}}$'s input $\hat{s}$.

Protocol $\Pi^{\text{s-o}}_{\text{COM}}$

Implicitly parametrized by a token query bound q, a commitment number n, and a commitment length ℓ. Let OVS be the oracle validation scheme from Section 6.2.2.1 and $\mathcal{T}$ be an instance of $\mathcal{F}^{\text{b-rs}}_{\text{wrap}}$. The security parameter is ℓ. For any vector $v = (v_1, \ldots, v_n)$ and $I \subseteq \{1, \ldots, n\}$ let $v_I := (v_i)_{i \in I}$.

Setup phase:

1. Sender: Execute $\mathsf{OVS.Create}(\ell, n, q)$ and obtain $(\mathsf{T}', (p, p'))$. Construct a token program T that behaves like T', except that it returns $\perp$ if a query $x = 0$ is sent. Send $(\texttt{create}, \mathsf{T})$ to $\mathcal{T}$.
2. Receiver: Pick $\lambda \leftarrow \mathbb{F}_{2^\ell}$ uniformly at random and send it to the sender.
3. Sender: Compute $\mathsf{OVS.Validate}(\lambda, (p, p'))$ and let $\tilde{p}$ be the result. Send $\tilde{p}$ to the receiver.

Commit phase:

4. Sender: Let $s := (s_1, \ldots, s_n) \in \mathbb{F}^n_{2^\ell}$ be the sender's input. Send $r := s + p(0)$ to the receiver.
5. Receiver: Call $\mathsf{OVS.Query}^{\mathcal{T}}(x, \tilde{p}, \lambda)$ for a uniformly random $x \leftarrow \mathbb{F}^n_{2^\ell}$ and let y be the result.

Unveil phase:

6. Sender: Let $I \subseteq \{1, \ldots, n\}$ indicate the commitments to be opened. Send (I, p_I) to the receiver.
7. Receiver: If $p_I(x) = y_I \neq \perp$, output $\hat{s}_I := r_I - p_I(0)$; else reject.

Figure 6.5: Statistically UC-secure commitment scheme in the $\mathcal{F}^{\text{b-rs}}_{\text{wrap}}$-hybrid model where the sender is the token issuer.

Simulator $\mathcal{S}_{\mathsf{S}}$

- Simulate $\mathcal{T}$ for $\mathcal{A}_{\mathsf{S}}$ and let T^* denote the token functionality that $\mathcal{A}_{\mathsf{S}}$ inputs into the token.
- (Setup) Simulate the setup phase straightforwardly according to $\Pi^{\text{s-o}}_{\text{COM}}$. Draw a uniformly random λ and obtain a check polynomial $\tilde{p}^*$.
- (Commit) Upon receiving a value r from $\mathcal{A}_{\mathsf{S}}$, execute $\mathsf{Query}^{\mathsf{T}^*}$ on a random input x and let y be the result. If $y \neq \perp$, start the extractor $\mathsf{Ext}_{\mathsf{OVS}}$ of OVS with input $(\mathsf{T}^*, \tilde{p}^*, \lambda)$ and obtain $(\hat{p}, \hat{p}')$. Compute $\hat{s} = r - \hat{p}(0)$ and send $(\texttt{commit}, \hat{s})$ to $\mathcal{F}^{\text{s-o}}_{\text{COM}}$.
- (Unveil) Upon receiving a message (I, p^*_I) from $\mathcal{A}_{\mathsf{S}}$, check if $p^*_I = \hat{p}_I$, if not abort. Send $(\texttt{unveil}, I, \hat{s}_I)$ to $\mathcal{F}^{\text{s-o}}_{\text{COM}}$.

Figure 6.6: Simulator for a corrupted sender in the protocol $\Pi^{\text{s-o}}_{\text{COM}}$.

Corrupted receiver. The simulator against a corrupted receiver has to equivocate the commitment he made in the commitment phase. Since the simulator sim-

ulates the hybrid functionality, i.e. the token, to the receiver, he learns all queries that the receiver makes and can interpolate a polynomial $\hat{p}$ with the intended input at the y-intersect. As the receiver can make at most q queries, the polynomial will have degree at most q. The simulator is described in more detail in Figure 6.7, with $\mathcal{A}_{\mathsf{R}}$ being the dummy adversary.

Simulator $\mathcal{S}_{\mathsf{R}}$

- Simulate $\mathcal{T}$ for $\mathcal{A}_{\mathsf{R}}$. Let Q denote the set of token queries that $\mathcal{A}_{\mathsf{R}}$ inputs into the token.
- (Setup) Simulate the setup phase straightforwardly according to $\Pi^{\text{s-o}}_{\text{COM}}$ with two random polynomials p, p'. Answer all queries x to $\mathcal{T}$ with $(p(x), p'(x))$.
- (Commit) Upon receiving a message (`committed`) from $\mathcal{F}^{\text{s-o}}_{\text{COM}}$, send a uniformly random $r \leftarrow \mathbb{F}^n_{2^\ell}$ to $\mathcal{A}_{\mathsf{R}}$.
- (Unveil) Upon receiving a message $(\texttt{opened}, I, \hat{s}_I)$ from $\mathcal{F}^{\text{s-o}}_{\text{COM}}$, compute two polynomials $\hat{p}, \hat{p}' \in \mathbb{F}^n_{2^\ell}[X]$ such that:
  - $(\hat{p}(x_i), \hat{p}'(x_i)) = (p(x_i), p'(x_i))$ for all $x \in Q$
  - $\lambda \cdot \hat{p} + \hat{p}' = \tilde{p}$, $\deg(\hat{p}, \hat{p}') \leq q$, and
  - $\hat{p}_I(0) = r_I - \hat{s}_I$

  For any future query x to $\mathcal{T}$ answer with $(\hat{p}(x), \hat{p}'(x))$. Send $(I, \hat{p}_I)$ to $\mathcal{A}_{\mathsf{R}}$.

Figure 6.7: Simulator for a corrupted receiver in the protocol $\Pi^{\text{s-o}}_{\text{COM}}$.

The output of $\mathcal{S}_{\mathsf{R}}$ is statistically indistinguishable from a real protocol run due to the privacy of OVS, which guarantees that $\mathcal{A}_{\mathsf{R}}$ will not be able to learn any value $p(x)$ without querying the oracle, i.e. $\mathcal{S}_{\mathsf{R}}$. Thus $\mathcal{A}_{\mathsf{R}}$ will not notice that the polynomial in the unveil phase has changed. □

Remark. The commitment scheme $\Pi^{\text{s-o}}_{\text{COM}}$ is statistically binding, even if λ is fixed and known to the sender. This yields a statistically secure non-interactive commitment scheme in the bounded-resettable hardware model, which was proven impossible in the stateless-token model [GIMS10].

6.3.2 Commitments from Token Receiver to Token Sender

For a commitment from the token receiver to the token sender we need a slightly more sophisticated approach. As in our previous commitment scheme, we use the oracle validation scheme from Section 6.2.2 such that the token implements a random degree-q polynomial p. The token receiver can then commit to some secret s by executing the Query algorithm on a random x, thus learning $p(x)$, and announcing a commit message consisting of

- a fraction of bits of $p(x)$, say the first quarter of its bit-string representation,
- a 2-universal hash function h, and
- $m := s + h(x)$.

To unveil s, he just needs to announce the used token input x. We briefly sketch now why this scheme is hiding and binding. The latter follows directly from the privacy of the OVS: The token acts just like a perfectly random function. Thus, a corrupted commitment sender may only with negligible probability find two distinct unveil messages x, x' such that $p(x)$ and $p(x')$ agree on the first quarter of their bit-string representation. This establishes the binding property. The token issuer, however, learns only several bits of information about x during the commit phase, so that from his view x has still linear entropy afterwards. Since h is a 2-universal hash function, this means that he cannot predict $h(x)$ and thus the commitment is hiding. To obtain extractability of the commitment, we use the extractor of the OVS, as otherwise we have no UC-security against a corrupted commitment receiver. The formal protocol description is given in Figure 6.8.

Protocol $\Pi^{\text{rev}}_{\text{COM}}$

Let OVS be the oracle validation scheme from Section 6.2.2.1 and $\mathcal{T}$ be an instance of $\mathcal{F}^{\text{b-rs}}_{\text{wrap}}$. Implicitly parametrized by a token query bound q and a commitment length ℓ. The security parameter is ℓ. Let $\sigma : \mathbb{F}_{2^{4\ell}} \to \mathbb{F}^4_{2^\ell}$, $x \mapsto (\sigma_1(x), \ldots, \sigma_4(x))$ be the canonical $\mathbb{F}_{2^\ell}$-vector space isomorphism. For any vector $v = (v_1, \ldots, v_n)$ and $I \subseteq \{1, \ldots, n\}$ let $v_I := (v_i)_{i \in I}$.

Setup phase:

1. Receiver: Execute $\mathsf{OVS.Create}(4\ell, n, q)$ and obtain $(\mathsf{T}, (p, p'))$. Send $(\texttt{create}, \mathsf{T})$ to $\mathcal{T}$.
2. Sender: Pick $\lambda \leftarrow \mathbb{F}_{2^{4\ell}}$ uniformly at random and send it to the receiver.
3. Receiver: Compute $\mathsf{OVS.Validate}(\lambda, (p, p'))$ and let $\tilde{p}$ be the result. Send $\tilde{p}$ to the sender.

Commit phase:

4. Sender: Let $s = (s_1, \ldots, s_n) \in \mathbb{F}^n_{2^\ell}$ be the sender's input. Call $\mathsf{OVS.Query}^{\mathcal{T}}(x, \tilde{p}, \lambda)$ for a uniformly random $x \leftarrow \mathbb{F}^n_{2^{4\ell}}$ and let y denote the response. If $y \neq \perp$, pick n uniformly random $h_i \leftarrow \mathbb{F}^4_{2^\ell}$, compute $m_i := s_i + \langle h_i \,|\, \sigma(x_i) \rangle$ and $\tilde{y}_i := \sigma_1(y_i)$ and send $(m, h, \tilde{y})$ to the receiver; otherwise abort.

Unveil phase:

5. Sender: Let $I \subseteq \{1, \ldots, n\}$ indicate the commitments to be opened. Send (I, x_I) to the receiver.
6. Receiver: Verify that $\tilde{y}_I = \sigma_1(p_I(x_I))$. If so, output $\hat{s}_I := m_I - \langle h_I \,|\, \sigma(x_I) \rangle$; otherwise reject.

Figure 6.8: Statistically UC-secure commitment scheme in the $\mathcal{F}^{\text{b-rs}}_{\text{wrap}}$-hybrid model where the receiver is the token issuer.

Theorem 6.3. *The protocol $\Pi^{\text{rev}}_{\text{COM}}$ in Figure 6.8 statistically UC-realizes $\mathcal{F}^{\text{s-o}}_{\text{COM}}$ (cf. Figure 6.4) in the $\mathcal{F}^{\text{b-rs}}_{\text{wrap}}$-hybrid model, given that OVS is an oracle validation scheme as per Definition 6.1.*

We split the proof into several parts. First we show the security of the protocol against a corrupted commitment sender in Lemma 6.3 and then move on to prove security against a corrupted commitment receiver in Lemmas 6.4, 6.5 and 6.6. Combined, this proves Theorem 6.3.

Lemma 6.3. *The protocol $\Pi^{\text{rev}}_{\text{COM}}$ in Figure 6.8 statistically UC-realizes $\mathcal{F}^{\text{s-o}}_{\text{COM}}$ (cf. Figure 6.4) in the $\mathcal{F}^{\text{b-rs}}_{\text{wrap}}$-hybrid model against a corrupted commitment sender, given that* OVS *is an oracle validation scheme as per Definition 6.1.*

Proof. We just have to exploit that the simulator sees all token inputs. With overwhelming probability the commitment sender's announcement of $\tilde{y}$ in the commit phase either corresponds to a unique input x already sent to the token or he is caught cheating in the unveil phase. In the former case, the simulator can find x just by scanning through the token's input history and then compute the correct inputs. A formal description of the simulator follows (cf. Figure 6.9), with $\mathcal{A}_{\mathsf{S}}$ being the dummy adversary.

Simulator $\mathcal{S}_{\mathsf{S}}$

- Simulate $\mathcal{T}$ for $\mathcal{A}_{\mathsf{S}}$. Let Q denote the set of all token queries that $\mathcal{A}_{\mathsf{S}}$ inputs into the token.
- (Setup) Simulate the setup phase straightforwardly according to $\Pi^{\text{rev}}_{\text{COM}}$ with uniformly random polynomials $p, p' \leftarrow \mathbb{F}^n_{2^\ell}[X]$.
- (Commit) Upon receiving a message $(m^*, h^*, \tilde{y}^*)$ from $\mathcal{A}_{\mathsf{S}}$, check for all $x \in Q$ if $\tilde{y} = \sigma_1(p(x))$. If no x is found, draw $r \leftarrow \mathbb{F}^n_{2^\ell}$ uniformly at random and send $(\texttt{commit}, r)$ to $\mathcal{F}^{\text{s-o}}_{\text{COM}}$. Otherwise, compute $\hat{s}_i = m_i - \langle h_i \,|\, \sigma(x_i)\rangle$ for all $i \in \{1, \ldots, n\}$ and send $(\texttt{commit}, \hat{s})$ to $\mathcal{F}^{\text{s-o}}_{\text{COM}}$.
- (Unveil) Upon receiving a message x^* from $\mathcal{A}_{\mathsf{S}}$, check if $\tilde{y}^* = \sigma_1(p(x^*))$. If no, abort, otherwise send $(\texttt{unveil}, I, \hat{s}_I)$ to $\mathcal{F}^{\text{s-o}}_{\text{COM}}$.

Figure 6.9: Simulator for a corrupted commitment sender in the protocol $\Pi^{\text{rev}}_{\text{COM}}$.

If the unveil phase fails, the simulator will not have to present an opening to his commitments. Thus the simulation is indistinguishable from a real protocol run as long as the simulator finds a corresponding x to the value $\tilde{y}$ for an accepting unveil phase. The probability that the simulator fails while the unveil phase succeeds is negligible due to the privacy of OVS, which guarantees that it is infeasible to learn a value $p(x)$ without querying the oracle with x. □

Proving UC-security against a corrupted commitment receiver, i.e. providing a simulator that equivocates commitments, is more challenging. Note that even after extracting a polynomial p that approximates the token functionality, it is still non-trivial to find a token input $\hat{x}$ such that the first quarter of bits of $p(\hat{x})$ matches the given commit message $(m, h, \tilde{y})$ while $m - h(x) = \hat{s}$ for a new secret $\hat{s}$. This problem can be expressed as a polynomial equation system. Here the efficient algorithm of [CS09] for sampling random solutions comes into play. In addition, the simulator has to make sure that the sampled solution $\hat{x}$ is actually possible in the real model: He has to (re)sample $\hat{x}$ until $p(\hat{x})$ corresponds with the token functionality and the Query-call in Step 4 of the commit phase of $\Pi^{\text{rev}}_{\text{COM}}$ returns $\tilde{y} \neq \perp$. The resampling

of $\hat{x}$ imposes some extra difficulty for the runtime estimation. In a first step, we show that our scheme is statistically hiding. This is useful for the UC proof and has further application later on in our construction of a resettable zero-knowledge protocol (cf. Section 6.4.1).

Lemma 6.4. *The protocol $\Pi_{\mathrm{COM}}^{\mathrm{rev}}$ is statistically hiding, even if λ is fixed.*

Proof. Let λ and $\tilde{p}^*$ be arbitrary but fixed. Further let $y \leftarrow \mathsf{OVS.Query}^{\mathsf{T}^*}(x, \tilde{p}^*, \lambda)$, where $y = (t^*(x), t'^*(x)) \in (\mathbb{F}_{2^{4\ell}}^n \times \mathbb{F}_{2^{4\ell}}^n) \cup \perp$ and (t^*, t'^*) describes the (possibly) malicious token functionality T^*. Moreover, let $Z := \mathbb{F}_{2^\ell}^n \cup \{\perp\}$ and for each $z \in Z$ let M_z denote the set of all token inputs x that lead to a commit message $(m, h, \tilde{y})$ with $\tilde{y} = z$. I.e., $M_z = \{x \in \mathbb{F}_{2^{4\ell}}^n \,|\, y \neq \perp \wedge \sigma_1(t^*(x)) = z\}$ for $z \in \mathbb{F}_{2^\ell}^n$ and $M_\perp = \{x \in \mathbb{F}_{2^{4\ell}}^n \,|\, y = \perp\}$.

For each element x_i in uniformly random $x \leftarrow \mathbb{F}_{2^{4\ell}}^n$ and corresponding $\tilde{y}_i$ in $\tilde{y}$ (meaning that $\tilde{y} = \sigma_1(t^*(x))$ if $y \neq \perp$ and else $\tilde{y} = \perp$) it holds:

$$\begin{aligned}\max_{e:Z\to\mathbb{F}_{2^{4\ell}}} \Pr[x = e(\tilde{y})] = \mathrm{E}(|M_{\tilde{y}}|^{-1}) = \sum\nolimits_{z\in Z} \Pr[x \in M_z] \cdot |M_z|^{-1} = \sum\nolimits_{z\in Z} \tfrac{1}{|\mathbb{F}_{2^{4\ell}}|} \\ = 2^{-3\ell} + 2^{-4\ell}\end{aligned}$$

Hence, for uniformly random $u \in \mathbb{F}_{2^\ell}$ we can conclude by the Leftover hash lemma (Lemma 2.26):

$$\begin{aligned}\Delta\Big((\langle h \,|\, \sigma(x)\rangle, h, \tilde{y}), (u, h, \tilde{y})\Big) \leq \tfrac{1}{2}\sqrt{\max_{e:Z\to\mathbb{F}_{2^{4\ell}}} \Pr[x = e(\tilde{y})] \cdot |\mathbb{F}_{2^\ell}|} = \tfrac{1}{2}\sqrt{2^{-2\ell} + 2^{-3\ell}} \\ < 2^{-\ell}\end{aligned}$$

It directly follows that the statistical distance between a commitment on any secret s and a commitment on uniform randomness is also upper bounded by $2^{-\ell}$ for each component, thus $\Pi_{\mathrm{COM}}^{\mathrm{rev}}$ remains statistically hiding. □

Lemma 6.5. *The protocol $\Pi_{\mathrm{COM}}^{\mathrm{rev}}$ in Figure 6.8 statistically UC-realizes $\mathcal{F}_{\mathrm{COM}}^{\text{s-o}}$ (cf. Figure 6.4) in the $\mathcal{F}_{\mathrm{wrap}}^{\text{b-rs}}$-hybrid model against a corrupted receiver, given that* OVS *is an oracle validation scheme as per Definition 6.1.*

Proof. Consider the simulator in Figure 6.10 with the dummy adversary $\mathcal{A}_{\mathsf{R}}$.

As a first step, we show that the preconditions of [CS09, Theorem 1.1] are met. For completeness, we restate their main theorem.

Theorem ([CS09, Theorem 1.1])**.** *Let $k > 0$ be a constant integer, $n > k$ and $d > 0$ be integers, let p^ℓ be a sufficiently large prime power and $\epsilon > 0$ be an arbitrarily small constant. Suppose that $f_1, \ldots, f_k \in \mathbb{F}_{p^\ell}[x_1, \ldots, x_n]$ are polynomials, each of total degree at most d, and let*

$$V \;=\; V(f_1, \ldots, f_k) \;=\; \{\xi \in \mathbb{F}_{p^\ell}^n \mid f_1(\xi) = \ldots = f_k(\xi) = 0\}$$

be the variety defined by $f_1, \ldots, f_k$. There exists a randomized algorithm that, given the description of $f_1, \ldots, f_k$ as a list of their nonzero monomials, outputs a random point $v \in \mathbb{F}_{p^\ell}^n$ such that the distribution of v is $\frac{6}{p^{\ell(1-\epsilon)}}$-close to the uniform distribution on V. The worst-case runtime complexity of this algorithm is polynomial in $n, d, \ell \log(p)$ and the description of $f_1, \ldots, f_k$.

Simulator $\mathcal{S}_{\mathsf{R}}$

- Simulate $\mathcal{T}$ for $\mathcal{A}_{\mathsf{R}}$. Let T^* denote the token functionality that $\mathcal{A}_{\mathsf{R}}$ inputs into the token.
- (Setup) Simulate the setup phase straightforwardly according to $\Pi^{\text{rev}}_{\text{COM}}$. Draw a uniformly random λ and obtain a check polynomial $\tilde{p}^*$.
- (Commit) Upon receiving a message (`committed`) from $\mathcal{F}^{\text{s-o}}_{\text{COM}}$, simulate the commit phase of $\Pi^{\text{rev}}_{\text{COM}}$ with a random input s and let $(m, h, \tilde{y})$ be the message to $\mathcal{A}_{\mathsf{R}}$.
- (Unveil) Upon receiving a message $(\texttt{opened}, I, \hat{s}_I)$ from $\mathcal{F}^{\text{s-o}}_{\text{COM}}$, if the simulated commitment sender has already aborted, do nothing. Otherwise, compute a new unveil message $\hat{x}_I$ using the following equivocation program for each $\hat{x}_i \in \hat{x}_I$.
  1. Start the extractor $\mathsf{Ext}_{\mathsf{OVS}}$ of OVS with input $(\mathsf{T}^*, \tilde{p}^*, \lambda)$. If $\mathsf{Ext}_{\mathsf{OVS}}$ queries T^* more than 2^ℓ times, abort; otherwise let $\hat{p}$ denote $\mathsf{Ext}_{\mathsf{OVS}}$'s output.
  2. Compute the unique polynomial $\hat{p}_1 \in \mathbb{F}_{2^\ell}[X_1, \ldots, X_4]$ such that $\deg(\hat{p}_1) \leq \deg(\hat{p})$ and $\sigma_1 \circ \hat{p} = \hat{p}_1 \circ \sigma$, where "$\circ$" denotes the function composition operator. Then pick a uniformly random solution $\hat{x} \in \mathbb{F}_{2^{4\ell}}$ of the following polynomial equation system, using the efficient algorithm of [CS09]:
  $$\hat{p}_1(\sigma(\hat{x})) = \tilde{y}$$
  $$\langle h \,|\, \sigma(\hat{x}) \rangle = m - \hat{s}$$
  Resample $\hat{x}$ until $\hat{p}(\hat{x}) = \mathsf{OVS.Query}^{\mathsf{T}^*}(\hat{x}, \tilde{p}^*, \lambda) \neq \perp$. Give up, if more than $2^{\sqrt{\ell}}$ iterations are required.
  3. Output $\hat{x}$

  Send $(I, \hat{x}_I)$ to $\mathcal{A}_{\mathsf{R}}$.

Figure 6.10: Simulator for a corrupted commitment receiver in the protocol $\Pi^{\text{rev}}_{\text{COM}}$.

Concretely, in our case the field is $\mathbb{F}_{2^\ell}$ and $n = 4$, as elements of $\mathbb{F}_{2^{4\ell}}$ are interpreted as 4-dimensional vectors over $\mathbb{F}_{2^\ell}$. The variety V is given by the polynomials (in $x = (x_1, \ldots, x_4)$)

$$\hat{p}_1(x) - \tilde{y} = 0$$
$$\langle h \,|\, x \rangle - m + \hat{s} = 0$$

where $\hat{p}_1(x) - \tilde{y} \in \mathbb{F}_{2^\ell}[x_1, \ldots, x_4]$ is a polynomial of degree q and $\langle h \,|\, x \rangle - m + \hat{s} \in \mathbb{F}_{2^\ell}[x_1, \ldots, x_4]$ is trivially a polynomial of degree 1. Thus the parameters are $k = 2$, $n = 4$, $p = 2$, and $d = q$. We can set $\epsilon = \frac{1}{2}$ and the above theorem yields an efficient algorithm that samples $\frac{6}{2^{\ell/2}}$-close to uniform from V.

Given the simulator, we now have to show that the simulation is indistinguishable from a real protocol execution, and in particular, the expected runtime of the simulator is polynomial. We show this via a series of hybrid experiments.

Experiment 0: This is the real model.

Experiment 1: Identical to Experiment 0, except that the commitment sender commits to pure randomness in the commit phase and runs in the unveil phase

a complete search over all token inputs to equivocate the commitment to his real input (which requires to reset the token exponentially many times).

Experiment 2: Identical to Experiment 1, except that the complete search in the equivocation step is only over token inputs x on which the token functionality $x \mapsto \Big(t^*(x), t'^*(x)\Big)$ coincides with the mapping $x \mapsto \Big(\hat{p(x)}, t'^*(x)\Big)$, where $\hat{p}$ denotes the polynomial computed by $\mathsf{Ext}_{\mathsf{OVS}}$ from the token program and the transcript of the setup phase.

Experiment 3: The ideal model, conditioned on the event that the simulator does give up.

Experiment 4: This is the ideal model.

Experiment 0 and Experiment 1 are indistinguishable, because the commitment is statistically hiding (cf. Lemma 6.4). Indistinguishability between Experiment 1 and Experiment 2 follows from the negligibility of $\mathsf{Ext}_{\mathsf{OVS}}$'s failure probability ρ' (cf. Definition 6.1). Experiment 2 and Experiment 3 are indistinguishable by construction of the simulator—here we need the above observation that by [CS09] one finds solutions for a polynomial equation system that are statistically close to random solutions. Experiment 3 and Experiment 4 are indistinguishable, given that the simulator has expected polynomial runtime complexity and thus gives up only with negligible probability.

We will prove this next. The expected polynomial runtime of $\mathcal{S}_{\mathsf{R}}$ depends on two factors, for which we will show polynomial runtime step by step.

1. The runtime of the extractor $\mathsf{Ext}_{\mathsf{OVS}}$ of OVS.
2. The number of resamplings of $\hat{x}$ by $\mathcal{S}_{\mathsf{R}}$.

Concerning Step 1, given that OVS is an oracle validation scheme, we have that the runtime of $\mathsf{Ext}_{\mathsf{OVS}}$ has to be polynomially bounded in $\Big(\ell \cdot |\mathsf{T}^*|\Big)^{O(1)} \cdot \rho^{-1}$, where ρ is the probability that $\mathsf{OVS.Query}^{\mathsf{T}^*}$ outputs $y \neq \perp$. Since we already know that $y \neq \perp$, we thus have an expected runtime of $\Big(\ell \cdot |\mathsf{T}^*|\Big)^{O(1)}$.

For Step 2, we first estimate the expected number of resamplings of one $\hat{x}_i \in \hat{x}_I$. In the following let $y \leftarrow \mathsf{OVS.Query}^{\mathsf{T}^*}(\hat{x}_i, \tilde{p}^*, \lambda)$, where $y = (t^*(\hat{x}_i), t'^*(\hat{x}_i)) \in (\mathbb{F}_{2^{4\ell}} \times \mathbb{F}_{2^{4\ell}}) \cup \perp$ and (t^*, t'^*) describes the malicious token functionality T^*. Further let M denote the set of token inputs that pass the consistency check and let N denote the subset of token inputs for which the extracted polynomial $\hat{p}$ is correct, i.e. $M = \{x \in \mathbb{F}_{2^{4\ell}} \,|\, y \neq \perp\}$ and $N = \{x \in M \,|\, y_1 = \hat{p}(x)\}$. We have to show that a randomly sampled $\hat{x}_i$ lies in N with overwhelming probability.

Let ρ' be the failure probability from Definition 6.1. Since $\rho' \leq 2^{-\Omega(\ell)}$, we find some $\varepsilon \in \mathbb{R}_{>0}$ with $\rho' \leq 2^{-\varepsilon\ell}$, assuming a sufficiently large security parameter ℓ. Note that $\rho' = \mathrm{E}\Big(|M \setminus N| \cdot |\mathbb{F}_{2^{4\ell}}|^{-1}\Big)$ just by construction. We distinguish the following three cases.

1. It holds that $|M| \leq 2^{(4-\varepsilon/4)\ell}$, i.e. the simulator finds no value $\hat{x}_i$ which passes the validation. However, conditioned on any choice of λ the probability of a non-aborted commit phase is $|M| \cdot |\mathbb{F}_{2^{4\ell}}|^{-1}$, which is negligible in this case. Thus, the probability that $\mathcal{S}_{\mathsf{R}}$ aborts beforehand is overwhelming probability

and even if the iteration bound of $2^{\sqrt{\ell}}$ is always reached, this case contributes only $2^{\sqrt{\ell}-\varepsilon\ell/4}$ to the expected number of resamplings of $\hat{x}_i$, which is negligible.

2. It holds that $|M \setminus N| \geq 2^{(4-3\varepsilon/4)\ell}$, i.e. the simulator cannot find a value $\hat{x}_i$ for which the extracted polynomial matches the token functionality. Since $E\big(|M \setminus N|\big) = \rho' \cdot |\mathbb{F}_{2^{4\ell}}| \leq 2^{(4-\varepsilon)\ell}$, this may be the case at most with probability $2^{-\varepsilon\ell/4}$. Thus, even if the iteration bound of $2^{\sqrt{\ell}}$ is always reached, this case also contributes only $2^{\sqrt{\ell}-\varepsilon\ell/4}$ to the average number of resamplings of $\hat{x}_i$.

3. It holds that $|M| > 2^{(4-\varepsilon/4)\ell}$ and $|M \setminus N| < 2^{(4-3\varepsilon/4)\ell}$. We may consider the simulated commitment sender's token input x_i as uniformly random over M, because $x \notin M$ leads to an abort in the commit phase anyway. Each sampling of $\hat{x}_i$ is uniformly random (up to a negligible statistical distance δ) over the set $(\sigma_1 \circ \hat{p})^{-1}((\sigma_1 \circ t^*)(x)) \cap \{\hat{x}_i \in \mathbb{F}_{2^{4\ell}} \mid \langle h_i \mid \sigma(\hat{x}_i) \rangle = m_i - \hat{s}_i\}$. The distance δ stems from the fact that we only know how to sample *almost* uniformly from large varieties. As discussed above, we have $\delta \leq \frac{6}{2^{\ell/2}}$. Note that m_i and hence also $m_i - \hat{s}_i$ is uniformly random and independent of (λ, x_i, h_i). If we instantiate the technical Lemma 6.6 with $g := \sigma_1 \circ \hat{p}$ and $f := \sigma_1 \circ t^*$, it follows that each sampling of $\hat{x}_i$ has the following success probability:

$$\begin{aligned}\Pr[\hat{x}_i \in N] &> \frac{|N|}{|\mathbb{F}_{2^{4\ell}}|} - \frac{|M \setminus N|}{|M|} - \frac{|\mathbb{F}_{2^{\ell}}|}{\sqrt{|M|}} - \delta \\ &= \frac{|M|}{|\mathbb{F}_{2^{4\ell}}|} - \frac{|M \setminus N|}{|\mathbb{F}_{2^{4\ell}}|} - \frac{|M \setminus N|}{|M|} - \frac{|\mathbb{F}_{2^{\ell}}|}{\sqrt{|M|}} - \delta\end{aligned}$$

Since we assumed $|M| > 2^{(4-\varepsilon/4)\ell}$ and $|M \setminus N| < 2^{(4-3\varepsilon/4)\ell}$, we can estimate (w.l.o.g. for $\varepsilon < 1$):

$$\begin{aligned}\frac{|M \setminus N|}{|\mathbb{F}_{2^{4\ell}}|} &< 2^{-3\varepsilon\ell/4} < \frac{2^{-\varepsilon\ell/2} \cdot |M|}{|\mathbb{F}_{2^{4\ell}}|} \\ \frac{|M \setminus N|}{|M|} &< 2^{-\varepsilon\ell/2} < \frac{2^{-\varepsilon\ell/4} \cdot |M|}{|\mathbb{F}_{2^{4\ell}}|} \\ \frac{|\mathbb{F}_{2^{\ell}}|}{\sqrt{|M|}} &< 2^{(\varepsilon/8-1)\ell} < \frac{2^{(3\varepsilon/8-1)\ell} \cdot |M|}{|\mathbb{F}_{2^{4\ell}}|} < \frac{2^{-5\ell/8} \cdot |M|}{|\mathbb{F}_{2^{4\ell}}|} \\ \delta \leq 6 \cdot 2^{-\ell/2} &< \frac{6 \cdot 2^{(\varepsilon/4-1/2)\ell} \cdot |M|}{|\mathbb{F}_{2^{4\ell}}|} < \frac{6 \cdot 2^{-\ell/4} \cdot |M|}{|\mathbb{F}_{2^{4\ell}}|}\end{aligned}$$

Thus, the success probability for each sampling of $\hat{x}_i$ is lower bounded by:

$$\frac{(1 - 2^{-\varepsilon\ell/2} - 2^{-\varepsilon\ell/4} - 2^{-5\ell/8} - 6 \cdot 2^{-\ell/4}) \cdot |M|}{|\mathbb{F}_{2^{4\ell}}|}$$

In other words, for large enough ℓ the success probability is arbitrarily close to $|M| \cdot |\mathbb{F}_{2^{4\ell}}|^{-1}$. However, this is also exactly the success probability for the consistency check in the protocol's commit phase. Let $\tilde{\rho} := |M| \cdot |\mathbb{F}_{2^{4\ell}}|^{-1}$ and

$\vartheta := 1 - 2^{-\varepsilon\ell/2} - 2^{-\varepsilon\ell/4} - 2^{-5\ell/8} - 6 \cdot 2^{-\ell/4}$. Hence we can upper bound the expected number of sampling steps for $\hat{x}_i$ by the following term:

$$\tilde{\rho} \cdot \left(\sum_{j=1}^{\infty} j \cdot (1 - \vartheta \cdot \tilde{\rho})^{i-1} \cdot \vartheta \cdot \tilde{\rho} \right) = \tilde{\rho} \cdot \frac{1}{\vartheta \cdot \tilde{\rho}} = \frac{1}{\vartheta}$$

Note that by assumption $|M| > 2^{(4-\varepsilon/4)\ell}$ and therefore $\tilde{\rho} \neq 0$.

To summarize, we have shown that the expected number of sampling steps for $\hat{x}_i$ is upper bounded by $1 + o(1)$. Since $n = \mathsf{poly}(\ell)$, this can be done for all $\hat{x}_i$ to obtain $\hat{x}_I$ in polynomial time. We can thus conclude that $\mathcal{S}_{\mathsf{R}}$ has expected polynomial runtime. □

The following lemma provides a bound for the probability that a randomly sampled $\hat{x}$, which passes the consistency check, evaluates to the same value when given to the oracle and when evaluated by the extracted polynomial. In this lemma, $\mathbf{x}$ denotes a random variable.

Lemma 6.6. *Let any finite sets U, M, N, Z with $\emptyset \neq N \subseteq M \subseteq U$ and two mappings $f, g : U \to Z$ be given with $f(x) = g(x)$ for all $x \in N$. Further, let $\mathcal{H}$ be a family of 2-universal hash functions $h : U \to Z$. Finally, let $\mathbf{x} \xleftarrow{\text{r}} M$, $\mathbf{h} \xleftarrow{\text{r}} \mathcal{H}$, $\mathbf{m} \xleftarrow{\text{r}} Z$, $\mathbf{z} := f(\mathbf{x})$, and $\hat{\mathbf{x}} \xleftarrow{\text{r}} g^{-1}(\mathbf{z}) \cap \mathbf{h}^{-1}(\mathbf{m})$ with the convention that $\hat{\mathbf{x}} = \perp \notin U$ in case of $g^{-1}(\mathbf{z}) \cap \mathbf{h}^{-1}(\mathbf{m}) = \emptyset$. Then, $\Pr[\hat{\mathbf{x}} \in N] > \frac{|N|}{|U|} - \frac{|M \setminus N|}{|M|} - \frac{|Z|}{\sqrt{|M|}}$.*

Proof. We define the following auxiliary random variables:

$$\begin{array}{ll} \mathbf{z}' := g(\mathbf{x}) & \hat{\mathbf{x}}' \xleftarrow{\text{r}} g^{-1}(\mathbf{z}') \cap \mathbf{h}^{-1}(\mathbf{m}) \quad \text{with } \hat{\mathbf{x}}' := \perp, \text{ if } g^{-1}(\mathbf{z}') \cap \mathbf{h}^{-1}(\mathbf{m}) = \emptyset \\ \mathbf{m}' := \mathbf{h}(\mathbf{x}) & \hat{\mathbf{x}}'' \xleftarrow{\text{r}} g^{-1}(\mathbf{z}') \cap \mathbf{h}^{-1}(\mathbf{m}') \end{array}$$

Note that $\Delta(\mathbf{z}, \mathbf{z}') \leq \Delta\Big((\mathbf{z}, \mathbf{x}), (\mathbf{z}', \mathbf{x})\Big) = \Pr[f(\mathbf{x}) \neq g(\mathbf{x})] \leq \Pr[\mathbf{x} \notin N]$ and therefore also $\Delta\Big((\mathbf{m}, \mathbf{h}, \mathbf{z}), (\mathbf{m}, \mathbf{h}, \mathbf{z}')\Big) \leq \Pr[\mathbf{x} \notin N]$, as $\mathbf{h}$ and $\mathbf{m}$ are just additional independent randomness. Thus we have:

$$\Pr[\hat{\mathbf{x}} \in N] \geq \Pr[\hat{\mathbf{x}}' \in N] - \Pr[\mathbf{x} \notin N] = \Pr[\hat{\mathbf{x}}' \in N] - \frac{|M \setminus N|}{|M|}$$

Now, for any mapping $e : Z \to M$ we can estimate $\Pr[\mathbf{x} = e(\mathbf{z}')]$ as follows:

$$\Pr[\mathbf{x} = e(\mathbf{z}')] = \sum_{z \in Z} \underbrace{\Pr[\mathbf{z}' = z]}_{\leq \frac{|M \cap g^{-1}(z)|}{|M|}} \cdot \underbrace{\Pr[\mathbf{x} = e(z) \,|\, \mathbf{x} \in g^{-1}(z)]}_{\leq \frac{1}{|M \cap g^{-1}(z)|}} \leq \frac{|Z|}{|M|}$$

Hence, using the leftover hash lemma (cf. Lemma 2.26) it holds that:

$$\Delta\Big((\mathbf{m}, \mathbf{h}, \mathbf{z}'), (\mathbf{m}', \mathbf{h}, \mathbf{z}')\Big) \leq \tfrac{1}{2}\sqrt{\max_{e:Z \to M} \Pr[\hat{\mathbf{x}} = e(\mathbf{z})] \cdot |Z|} \leq \tfrac{1}{2}\sqrt{\frac{|Z|}{|M|} \cdot |Z|} < \frac{|Z|}{\sqrt{|M|}}$$

This yields:

$$\Pr[\hat{\mathbf{x}}' \in N] > \Pr[\hat{\mathbf{x}}'' \in N] - \frac{|Z|}{\sqrt{|M|}} \quad \text{and therefore}$$
$$\Pr[\hat{\mathbf{x}} \in N] > \Pr[\hat{\mathbf{x}}'' \in N] - \frac{|M \setminus N|}{|M|} - \frac{|Z|}{\sqrt{|M|}}$$

So, we finally need to estimate $\Pr[\hat{\mathbf{x}}'' \in N]$. Note that $\hat{\mathbf{x}}'' \xleftarrow{r} g^{-1}(g(\mathbf{x})) \cap \mathbf{h}^{-1}(\mathbf{h}(\mathbf{x}))$ by construction, i.e., $\hat{\mathbf{x}}''$ is a uniformly random preimage of $\big(g(\mathbf{x}), \mathbf{h}(\mathbf{x})\big)$ under the mapping $x \mapsto \big(g(x), \mathbf{h}(x)\big)$. In a first step we will estimate the probability, that $\mathbf{x}''$ lies in M.

First note that for any $\beta, A, B \in \mathbb{R}$ with $B > \beta > 0$ the function $\alpha \mapsto \frac{\alpha^2}{\beta} + \frac{(A-\alpha)^2}{B-\beta}$ has a global minimum at $\alpha = \frac{\beta A}{B}$. Thus, given any finite sequences $(\alpha_z)_{z\in Z} \subset \mathbb{R}$ and $(\beta_z)_{z\in Z} \subset \mathbb{R}_{>0}$, it follows by induction on $|Z|$ that $\sum_{z\in Z} \frac{\alpha_z^2}{\beta_z} \geq \sum_{z\in Z} \frac{\alpha_z A}{B} = \frac{A^2}{B}$, where $A := \sum_{z\in Z} \alpha_z$ and $B := \sum_{z\in Z} \beta_z$.

Now for each $z \in Z$, let $U_z := g^{-1}(z) \cap \mathbf{h}^{-1}(z)$ and $M_z := U_z \cap M$. Let $Z' := g(U) \cap \mathbf{h}(U)$. By our considerations above we have:

$$\sum_{x\in M} \frac{|M_{g(x)\cap\mathbf{h}(x)}|}{|U_{g(x)\cap\mathbf{h}(x)}|} = \sum_{z\in Z'} \sum_{x\in M_z} \frac{|M_z|}{|U_z|} = \sum_{z\in Z'} \frac{|M_z|^2}{|U_z|} \geq \frac{|M|^2}{|U|}$$

Hence the following holds:

$$\Pr[\hat{\mathbf{x}} \in M] = \sum_{x\in M} \underbrace{\Pr[\mathbf{x} = x]}_{=|M|^{-1}} \cdot \underbrace{\Pr[\hat{\mathbf{x}} \in M \,|\, \mathbf{x} = x]}_{=|M_{g(x)\cap\mathbf{h}(x)}|\cdot|U_{g(x)\cap\mathbf{h}(x)}|^{-1}} \geq \tfrac{|M|}{|U|}$$

Conditioned on the event that $\hat{\mathbf{x}}'' \in M$, we can consider $\hat{\mathbf{x}}''$ just as a resampled version of $\mathbf{x}$, which is uniformly distributed. Thus, $\Pr[\hat{\mathbf{x}}'' \in N \,|\, \hat{\mathbf{x}}'' \in M] = \frac{|N|}{|M|}$ and therefore $\Pr[\hat{\mathbf{x}}'' \in N] = \Pr[\hat{\mathbf{x}}'' \in M] \cdot \Pr[\hat{\mathbf{x}}'' \in N \,|\, \hat{\mathbf{x}}'' \in M] \geq \frac{|N|}{|U|}$. Combined, we get $\Pr[\hat{\mathbf{x}} \in N] > \frac{|N|}{|U|} - \frac{|M\setminus N|}{|M|} - \frac{|Z|}{\sqrt{|M|}}$ as claimed. □

6.3.3 Multiple OT from a Constant Number of Tokens

We present an OT protocol in Section 6.3.3.1 that is an adapted version of a protocol by Goyal et al. [GIMS10] for a single OT based on stateless tamper-proof hardware. While their protocol needs token encapsulation, we can enhance the protocol such that this is no longer necessary. The structure of the protocol is as follows. In a first step, the receiver of the OT has to create a commitment on his choice bit c and send it to the sender. This is done using a commitment based on a hardware token. The sender creates a token T with the following functionality. The protocol transcript of the commitment phase and all random coins of the sender are stored in the token. When it receives the unveil message for the commitment, it checks the correctness and provides the OT output s_c if the check succeeds. Then the sender sends T to the receiver, who can unveil his choice bit and obtain the corresponding output.

Our protocol has the drawback that, when used for multiple parallel OTs at once, the token can abort depending on all choice bits. Thus, in Section 6.3.3.2 we show how to obtain a protocol that allows multiple parallel OTs without the combined abort flaw.

6.3.3.1 Multiple OT with Combined Abort

In the original protocol of [GIMS10] the unveil phase of the commitment required the commitment receiver (i.e. the OT sender in this case) to query a token, such that T must encapsulate the token of the commitment of the OT receiver. Our

commitment scheme $\Pi_{\text{COM}}^{\text{rev}}$ (cf. Section 6.3.2), however, does not need any interaction with a token by the commitment receiver. Thus we can circumvent the token encapsulation and obtain an OT protocol in the $\mathcal{F}_{\text{wrap}}^{\text{b-rs}}$-hybrid model. If we want to implement several OT instances in parallel, we encounter the following problem: Each of the OT outputs can depend on all choice bits. To counter this, we have the sender commit to his inputs in advance. This still leaves a small flaw in the construction: The token can abort depending on all choice bits. We thus only realize a weaker ideal OT functionality, where a corrupted sender can specify an abort predicate depending on the choice bits of the receiver (cf. Figure 6.11). A similar level of security was achieved by [IKO+11] in the context of non-interactive secure computation.

Functionality $\mathcal{F}_{\text{MOT}}^{\text{c-ab}}$

Implicitly parametrized by a sender input domain S and the number n of single OTs to be implemented.

- Upon input $(\texttt{create}, (s_0^{(1)}, s_1^{(1)}), \ldots, (s_0^{(n)}, s_1^{(n)}))$ with $(s_0^{(i)}, s_1^{(i)}) \in S \times S$ from the sender, verify that the sender is uncorrupted; otherwise ignore that input. Next, store $(s_0^{(0)}, s_1^{(0)}), \ldots, (s_0^{(n)}, s_1^{(n)})$, send $(\texttt{sent})$ to the adversary, and henceforth ignore any further input from the sender.
- Upon input $(\texttt{mal\_create}, (s_0^{(1)}, s_1^{(1)}), \ldots, (s_0^{(n)}, s_1^{(n)}), Q)$ with $(s_0^{(i)}, s_1^{(i)}) \in S \times S$ and a predicate $Q : \{0,1\}^n \to \{\texttt{accept}, \texttt{reject}\}$ from the sender, verify that the sender is corrupted; otherwise ignore that input. Next, store $(s_0^{(0)}, s_1^{(0)}), \ldots, (s_0^{(n)}, s_1^{(n)})$ and Q, send $(\texttt{sent})$ to the adversary, and henceforth ignore any further input from the sender.
- Upon input $(\texttt{choice}, c)$ with $c = (c_1, \ldots, c_n) \in \{0,1\}^n$ from the receiver, store c, send $(\texttt{chosen})$ to the adversary, and henceforth ignore any further input from the receiver.
- Upon receiving a message $(\texttt{output})$ from the adversary, check that there are stored inputs $(s_0^{(1)}, s_1^{(1)}), \ldots, (s_0^{(n)}, s_1^{(n)})$ from the sender and c from the receiver; else ignore this message. If the sender is corrupted, compute $Q(c)$ and abort if $Q(c) = \texttt{reject}$. Next, send $(s_{c_1}^{(1)}, \ldots, s_{c_n}^{(n)})$ to the receiver and ignore any further $(\texttt{output})$-messages from the adversary.
- Upon receiving a message $(\texttt{notify})$ from the adversary, check that there are stored inputs $(s_0^{(1)}, s_1^{(1)}), \ldots, (s_0^{(n)}, s_1^{(n)})$ from the sender and c from the receiver; else ignore this message. Next, send an empty output to the sender and ignore any further $(\texttt{notify})$-messages from the adversary.

Figure 6.11: Ideal functionality for multiple oblivious transfer with a combined abort property.

Our final protocol proceeds as follows. In a first step, the sender commits to his inputs $(s_0^{(1)}, s_1^{(1)}), \ldots, (s_0^{(n)}, s_1^{(n)})$ using the commitment protocol $\Pi_{\text{COM}}^{\text{s-o}}$ from Section 6.3.1 and samples a random key for an information-theoretic MAC (cf. Section 2.4.1). This MAC allows us to make the token T and thus all inputs independent of the actual commitment message from $\Pi_{\text{COM}}^{\text{rev}}$ by having the sender authenticate

the message for the token. The sender then programs T such that it unveils the corresponding commitments on $s_{c_1}^{(1)}, \ldots, s_{c_n}^{(n)}$ if it receives an authenticated unveil message for the choice bits $c_1, \ldots, c_n$. Now the receiver commits to his choice bits and obtains a MAC σ on the transcript τ of the commitment phase from the sender. The receiver then sends the unveil information together with τ and σ to T to obtain the corresponding OT inputs. The formal description of the protocol is given in Figure 6.12

Protocol $\Pi_{\text{MOT}}^{\text{c-ab}}$

Implicitly parametrized by the number n of single OTs to be implemented. Based upon our commitment schemes $\Pi_{\text{COM}}^{\text{s-o}}$ and $\Pi_{\text{COM}}^{\text{rev}}$ and a statistically secure message authentication scheme MAC. Let $\mathcal{T}$ be an instance of $\mathcal{F}_{\text{wrap}}^{\text{b-rs}}$.

1. Sender: Let $(s_0^{(1)}, s_1^{(1)}), \ldots, (s_0^{(n)}, s_1^{(n)})$ be the sender's n OT-inputs. Sample a key k for the message authentication scheme MAC. Commit to the $2n$ values $s_0^{(i)}, s_1^{(i)}$ via $\Pi_{\text{COM}}^{\text{s-o}}$ and prepare a hardware token T with the following functionality:
   - On input (c, w, τ, σ), verify that $c \in \{0,1\}^n$, $\sigma \leftarrow \mathsf{MAC}_k(\tau)$, and w is a correct $\Pi_{\text{COM}}^{\text{rev}}$-unveil of c with commit phase transcript τ. If so, return the $\Pi_{\text{COM}}^{\text{s-o}}$-unveil messages for $s_{c_1}^{(1)}, \ldots, s_{c_n}^{(n)}$.

   Send $(\texttt{create}, \mathsf{T})$ to $\mathcal{T}$.
2. Receiver: Let $c = (c_1, \ldots, c_n)$ be the receiver's choice bits. Commit to c via $\Pi_{\text{COM}}^{\text{rev}}$.
3. Sender: Take the message transcript τ of Step 2, compute $\sigma = \mathsf{MAC}_k(\tau)$, and send σ to the receiver.
4. Receiver: Let w be the $\Pi_{\text{COM}}^{\text{rev}}$-unveil message for c. Input (c, w, τ, σ) into $\mathcal{T}$; let $(r_1, \ldots, r_n)$ denote the response. Verify that $r_1, \ldots, r_n$ are correct unveil messages for the corresponding $\Pi_{\text{COM}}^{\text{s-o}}$-commitments from Step 1 indexed by c. If so, output the unveiled values; otherwise abort.

Figure 6.12: Reduction of multiple OT with combined abort in the $\mathcal{F}_{\text{wrap}}^{\text{b-rs}}$-hybrid model to the commitment protocols $\Pi_{\text{COM}}^{\text{s-o}}$ and $\Pi_{\text{COM}}^{\text{rev}}$.

Theorem 6.4. *The protocol $\Pi_{\text{MOT}}^{\text{c-ab}}$ in Figure 6.12 statistically UC-realizes $\mathcal{F}_{\text{MOT}}^{\text{c-ab}}$ (cf. Figure 6.11) in the $\mathcal{F}_{\text{wrap}}^{\text{b-rs}}$-hybrid model, given that MAC is a statistically secure MAC and $\Pi_{\text{COM}}^{\text{s-o}}$ and $\Pi_{\text{COM}}^{\text{rev}}$ are statistically UC-secure.*

Proof. **Corrupted Sender.** We first consider UC-security against a corrupted OT sender. The simulator is depicted in Figure 6.13. The simulator first extracts the inputs of the malicious sender and then computes a predicate Q based on the token functionality T^*. This predicate is based on one protocol run between the simulator and the sender based on random choice bits and therefore does not behave like the token for all receiver inputs. To make it reusable, the simulator programs Q such that it uses the receiver simulator of $\Pi_{\text{COM}}^{\text{rev}}$ to equivocate the one protocol run to any vector of choice bits from the receiver. He then inputs the sender inputs with the predicate into the ideal functionality.

We show by a series of hybrid experiments that the simulator produces an output indistinguishable from a real protocol run.

Simulator $\mathcal{S}_{\mathsf{S}}$

- Simulate $\mathcal{T}$ for the $\mathcal{A}_{\mathsf{S}}$ and obtain T^* in Step 1 of $\Pi^{\text{c-ab}}_{\text{MOT}}$. Use the sender-simulator $\mathcal{S}_{\mathsf{S}}^{\Pi^{\text{s-o}}_{\text{COM}}}$ of $\Pi^{\text{s-o}}_{\text{COM}}$ to extract the OT inputs $(\hat{s}_0^{(1)}, \hat{s}_1^{(1)}), \ldots, (\hat{s}_0^{(n)}, \hat{s}_1^{(n)})$ of $\mathcal{A}_{\mathsf{S}}$.
- To simulate Step 2, choose $c = (c_1, \ldots, c_n)$ uniformly at random and proceed as described in the protocol.
- In Step 4, construct an abort predicate Q from T^*, σ and τ, where σ was obtained in Step 2 and τ in Step 3, which behaves as follows.
  1. Upon input $\bar{c} \in \{0,1\}^n$, use the receiver-simulator $\mathcal{S}_{\mathsf{R}}^{\Pi^{\text{rev}}_{\text{COM}}}$ for $\Pi^{\text{rev}}_{\text{COM}}$ to obtain an unveil message $\bar{w}$ that equivocates τ to $\bar{c}$.
  2. Run T^* on input $(\bar{c}, \bar{w}, \tau, \sigma)$; let $(r_1, \ldots, r_n)$ denote the response.
  3. Simulate the check in Step 4 of $\Pi^{\text{c-ab}}_{\text{MOT}}$, i.e. verify that $r_1, \ldots, r_n$ are correct unveil messages for the corresponding $\Pi^{\text{s-o}}_{\text{COM}}$ commitments indexed by $\bar{c}$. If so, accept; otherwise reject.

  Send $(\texttt{mal\_create}, (\hat{s}_0^{(1)}, \hat{s}_1^{(1)}), \ldots, (\hat{s}_0^{(n)}, \hat{s}_1^{(n)}), Q)$ to $\mathcal{F}^{\text{c-ab}}_{\text{MOT}}$.

Figure 6.13: Simulator for a corrupted OT sender in the protocol $\Pi^{\text{c-ab}}_{\text{MOT}}$.

Experiment 0: This is the real model.

Experiment 1: Identical to Experiment 0, except that $\mathcal{S}_1$ commits to uniformly random inputs and uses the receiver-simulator $\mathcal{S}_{\mathsf{R}}^{\Pi^{\text{rev}}_{\text{COM}}}$ of $\Pi^{\text{rev}}_{\text{COM}}$ to equivocate his commitment to the token.

Experiment 2: Identical to Experiment 1, except that $\mathcal{S}_2$ uses the sender-simulator $\mathcal{S}_{\mathsf{S}}^{\Pi^{\text{s-o}}_{\text{COM}}}$ of $\Pi^{\text{s-o}}_{\text{COM}}$ and constructs a predicate Q as described in Figure 6.13. This is the ideal model.

Experiment 0 and Experiment 1 are indistinguishable since the protocol $\Pi^{\text{rev}}_{\text{COM}}$ is UC-secure. This guarantees that the equivocation step succeeds with overwhelming probability. Experiments 1 and 2 are indistinguishable due to the UC-security of $\Pi^{\text{s-o}}_{\text{COM}}$. In particular, the predicate Q performs exactly the check that the receiver performs in Experiment 1, except that the extracted values are used for the check. UC-security ensures that the extraction will succeed with overwhelming probability, thus this check is passed with the same probability (up to a negligible factor) as in Experiment 1.

Corrupted Receiver. Next, we show security against a corrupted receiver $\mathcal{A}_{\mathsf{R}}$. The simulator against a corrupted receiver is shown in Figure 6.14. The main idea is that the simulator will commit to random OT inputs, extract the choice bits of the receiver, and then retrieve the real outputs from the ideal functionality. From there, the simulator uses the receiver simulator of $\Pi^{\text{s-o}}_{\text{COM}}$ to equivocate his inputs according to the real outputs.

We now show the indistinguishability of a simulated protocol run from a real protocol run in a series of hybrid experiments.

Experiment 0: This is the real model.

Experiment 1: Identical to Experiment 0, except that $\mathcal{S}_1$ aborts, if $\tau^* \neq \hat{\tau}$ and σ^* is a correct MAC on τ^*.

Simulator $\mathcal{S}_{\mathsf{R}}$

- Simulate $\mathcal{T}$ for $\mathcal{A}_{\mathsf{R}}$. Let Q denote the set of token queries that $\mathcal{A}_{\mathsf{R}}$ inputs into the token.
- Choose $(s_0^{(1)}, s_1^{(1)}), \ldots, (s_0^{(n)}, s_1^{(n)})$ uniformly at random, and commit to them via $\Pi_{\mathrm{COM}}^{\text{s-o}}$. Simulate the rest according to Step 1 of the protocol.
- Use the sender-simulator $\mathcal{S}_{\mathsf{S}}^{\Pi_{\mathrm{COM}}^{\mathrm{rev}}}$ for $\Pi_{\mathrm{COM}}^{\mathrm{rev}}$ in Step 2 to extract the committed choice bits $\hat{c} = \hat{c}_1, \ldots, \hat{c}_n$ and send $(\texttt{choice}, \hat{c})$ to $\mathcal{F}_{\mathrm{MOT}}^{\text{c-ab}}$.
- Simulate Step 3 according to the protocol with $\hat{\tau}$ being the message transcript of Step 2.
- Send $(\texttt{output})$ to $\mathcal{F}_{\mathrm{MOT}}^{\text{c-ab}}$ and let $\hat{s}_{c_1}^{(1)}, \ldots, \hat{s}_{c_n}^{(n)}$ be the answer. Use the receiver-simulator $\mathcal{S}_{\mathsf{R}}^{\Pi_{\mathrm{COM}}^{\text{s-o}}}$ of $\Pi_{\mathrm{COM}}^{\text{s-o}}$ to produce unveil messages $r'_1, \ldots, r'_n$ that equivocate the commitments to the correct values. Let $(c^*, w^*, \tau^*, \sigma^*)$ be the input of $\mathcal{A}_{\mathsf{R}}$ into the token. Abort if $c^* \neq \hat{c}$ and the unveil message w^* unveils correctly to c^*. If that check if passed, abort if $\tau^* \neq \tau$ and σ^* is a correct MAC on τ^*. Otherwise, output $r'_1, \ldots, r'_n$ for the corresponding commitments indexed by $\hat{c}$.

Figure 6.14: Simulator for a corrupted OT receiver in the protocol $\Pi_{\mathrm{MOT}}^{\text{c-ab}}$.

Experiment 2: Identical to Experiment 1, except that $\mathcal{S}_2$ aborts, if $c^* \neq \hat{c}$, although w^* is a correct unveil for the $\Pi_{\mathrm{COM}}^{\mathrm{rev}}$ commitment.

Experiment 3: Identical to Experiment 2, except that $\mathcal{S}_3$ equivokes the commitments at the positions specified by $\hat{c}$ to the output of $\mathcal{F}_{\mathrm{MOT}}^{\text{c-ab}}$. This is the ideal model.

Indistinguishability between Experiment 0 and Experiment 1 follows directly from the unforgeability of MAC, i.e. the event that $\mathcal{S}_2$ aborts happens only with negligible probability. Experiment 1 and Experiment 2 are indistinguishable since $\Pi_{\mathrm{COM}}^{\mathrm{rev}}$ is UC-secure and thus the extraction will succeed with overwhelming probability. Experiment 2 and Experiment 3 are indistinguishable due to the UC-security of $\Pi_{\mathrm{COM}}^{\text{s-o}}$, i.e. the equivocation will succeed with overwhelming probability.

□

Remark. It is possible to use only two tokens in the protocol $\Pi_{\mathrm{MOT}}^{\text{c-ab}}$ instead of three by combining the token T with the token of $\Pi_{\mathrm{COM}}^{\mathrm{rev}}$. Note that the token T already receives a complete message transcript of the messages of $\Pi_{\mathrm{COM}}^{\mathrm{rev}}$. Thus the security of the protocol remains intact even if a malicious token can keep a complex state. Although the token has to be queried twice now, the token functionality can straighforwardly be adapted to allow this, as pointed out in Section 6.1.

Remark. Note that the above protocol combined with the impossibility of OT in the stateless token model [GIMS10] directly implies an impossibility result for commitments in the stateless token model, where the unveil phase consists only of a single message from the sender to the receiver and local computations (without accessing any tokens) by the receiver. Otherwise, the commitments in our OT construction could be replaced by these commitments, yielding an OT protocol in the stateless token model without encapsulation. This contradicts the impossibility result proven in [GIMS10].

6.3.3.2 How to Get Rid of the Combined-Abort Flaw

We now sketch how to obtain an ideal oblivious transfer as specified in Section 2.7.2.3 from $\mathcal{F}_{\text{MOT}}^{\text{c-ab}}$. Although this problem seems to be related to OT combiners [HKN+05, MPW07], i.e. the extraction of an OT from several possibly corrupted OTs, OT combiners cannot be applied in our case. An OT combiner uses a set of *independent* OTs from which some may be corrupted, while our protocol gives us uncorrupted OTs, but it leaks a *joint* predicate over all receiver inputs.

Using a classical OT combiner based on 2-universal hashing in our scenario allows the following attack. Standard OT combiners are built such that the receiver's ideal OT choice bits are basically 2-universal hash values of the flawed choice bits, which are uniformly random, and similarly for the outputs. Applying this construction to our scenario, the sender can just pick two input pairs $\left(\tilde{s}_0^{(i)}, \tilde{s}_1^{(i)}\right)$ and $\left(\tilde{s}_0^{(j)}, \tilde{s}_1^{(j)}\right)$, flip the bits of $\tilde{s}_1^{(i)}$ and $\tilde{s}_1^{(j)}$, and otherwise execute the protocol honestly. He then specifies a predicate Q such that the ideal functionality aborts upon input $\tilde{c} = (\tilde{c}_1, \ldots, \tilde{c}_n)$ if and only if $\tilde{c}_i = \tilde{c}_j = 1$. With non-negligible probability, the 2-universal hash functions are chosen such that

- the receiver's i-th flawed OT input-output tuple $(\tilde{c}_i, \tilde{r}_i)$ influences the calculation of an ideal OT input-output tuple (c_k, r_k), but not (c_l, r_l), where l is an index such that
- the receiver's j-th flawed OT input-output tuple $(\tilde{c}_j, \tilde{r}_j)$ influences (c_l, r_l), but not (c_k, r_k).

If the 2-universal hash functions are chosen like that, if $\tilde{c}_i = 1$ (c_k, r_k) is affected by the bit-flip of $\tilde{s}_1^{(i)}$, and similarly (c_l, r_l) is affected by the bit-flip of $\tilde{s}_1^{(j)}$ if $\tilde{c}_j = 1$. The definition of the predicate Q now ensures that—although both events are statistically independent—it will never happen that both values (c_k, r_k) and (c_l, r_l) are affected by the attack. We now have a correlation between the joint distribution of the receiver's inputs and outputs of the ideal OT and the event that the flawed OT aborts. This is not simulatable with an ideal OT.

Thus, to prevent this kind of attack, we need an OT extractor as defined by Ishai et al. [IKOS09] that extracts independent OTs from correlated instances. But we cannot simply apply the solution of [IKOS09], because their OT extractor only achieves semi-honest security for the unconditional setting with a maliciously chosen leakage function, albeit with a constant rate, i.e. they only need $O(n)$ bits of communication to achieve n OTs. Our scenario focuses on malicious parties that try to cheat in the extraction protocol. We make no restriction on the rate, i.e. we implement n ideal OT instances from $n^{O(1)}$ $\mathcal{F}_{\text{MOT}}^{\text{c-ab}}$ instances.

Our solution relies on the MPC-in-the-head paradigm [IKOS07, IPS08] and is thus conceptually similar to the OT combiner of [HIKN08]. While it might be possible to prove the protocol of [HIKN08] secure based on $\mathcal{F}_{\text{MOT}}^{\text{c-ab}}$, we will just use a combination of existing works from the literature to obtain our extractor. In particular, we use:

1. Our implementation of $\mathcal{F}_{\text{MOT}}^{\text{c-ab}}$.
2. A statistically UC-secure 2PC protocol in the OT-hybrid model, e.g. [IPS08].
3. A statistically UC-secure OT protocol for multiple OTs, based on untrusted tamper-proof hardware, e.g. [DKMQ11].

First note that it is possible to precompute OT with random inputs and later use a simple interactive and information-theoretic protocol to adjust the precomputed values to the real inputs [Bea95]. Also, a precomputed OT can be reversed [WW06], i.e. the OT receiver of a random OT can be transformed into an OT sender and vice versa. We will use the protocol of [IPS08] to implement the token functionality of [DKMQ11] based on $\mathcal{F}_{\text{MOT}}^{\text{c-ab}}$. [IPS08] allows to use precomputation, so that we have to use $\mathcal{F}_{\text{MOT}}^{\text{c-ab}}$ only once in the setup with random inputs. The parties just execute the protocol of [DKMQ11], where each token query is replaced by the 2PC of the emulated token. The security of the described construction follows from the fact the specification of an abort predicate Q for $\mathcal{F}_{\text{MOT}}^{\text{c-ab}}$ directly implies a maliciously programmed token that aborts depending on the inputs. The UC-security of the construction of [DKMQ11], however, guarantees robustness against such a token functionality. Thus we obtain UC-secure OT from $\mathcal{F}_{\text{MOT}}^{\text{c-ab}}$.

6.4 Statistically Secure Non-Interactive Two-Party Computation

For the case of non-interactive two-party computation, we cannot give a general positive result, but only a partial solution. On the one hand, the commitment that is given in Section 6.3.1 can be made non-interactive (as remarked), but this comes at the cost of losing UC-security. It seems unlikely that the protocol can be made UC-secure with the techniques that we use. Nevertheless, in the following we show a statistically UC-secure non-interactive zero-knowledge argument of knowledge, which might indicate that some interesting functionalities can still be realized non-interactively.

6.4.1 Bounded-Resettable Zero-Knowledge Proofs of Knowledge

We construct a bounded-resettable zero-knowledge proof system for the $\mathcal{NP}$-complete problem `3-COLOR`. The goal of the prover is to convince the verifier that he knows a 3-coloring of a given graph $G = (V, E)$, i.e. he knows a map $\varphi : V \to \{1, 2, 3\}$, without revealing it to the verifier. For this, the prover commits himself on a coloring of the vertices and the verifier can query the prover with an edge of the graph. The prover has to open the commitments to the vertices that are adjacent to the challenge. If the colors for both vertices are different, the verifier accepts. While the sketched protocol only has soundness $\frac{1}{|E|}$, i.e. a malicious prover can convince the verifier of a false statement with probability $1 - \frac{1}{|E|}$, repetition of the protocol leads to soundness arbitrary close to 1.

The main problem imposed by a *resettable* prover is that a malicious verifier could try to run the same protocol several times, each time with different challenges, and hence step by step learn the prover's witness. This is essentially the same strategy that is usually applied to construct an extractor for a proof of knowledge. The standard technique to deal with a resetting verifier is to have the verifier commit to his challenge in advance and let the color permutations that are chosen by the prover depend on the commitment of the verifier in a pseudorandom way. Our construction follows this approach.

We modify the constant-round zero-knowledge protocol of [GK96a] for 3-COLOR such that the prover becomes resettable and only two tokens have to be sent to the verifier. In the protocol of [GK96a], the verifier first commits to his challenge (the edges determining the vertices that are to be revealed), then the prover commits to permutations of the colored vertices. The verifier then reveals the challenge and the prover opens the specified commitments.

For our construction, we replace the computational commitments in [GK96a] with the statistical commitments from Sections 6.3.1 and 6.3.2. Therefore the prover has to send 2 tokens to the verifier. At a first glance this will render the protocol insecure if the token issuer is resettable, since our commitment schemes have an interactive setup phase, and if the token receiver can query the sender with two distinct values λ_1, λ_2 with $\lambda_1 \neq \lambda_2$, he can directly derive the token functionality. However, if we fix λ in $\Pi_{\mathrm{COM}}^{\mathrm{rev}}$, the resulting commitment scheme $\widetilde{\Pi}_{COM}^{rev}$ remains statistically hiding (cf. Lemma 6.4).

As mentioned above, to achieve resettable security we have to make the prover's behavior dependent on the commitment of the verifier. Since we aim for information-theoretic security, instead of using a pseudorandom function, we use a random polynomial f of sufficient degree. Thus we also have to change the token generation for $\Pi_{\mathrm{COM}}^{\text{s-o}}$ such that the input domain is $X \times C$, where X is the message domain, and C is the set of all possible commitments of $\widetilde{\Pi}_{COM}^{rev}$. Upon input a message (x, c), the token generation phase of $\Pi_{\mathrm{COM}}^{\text{s-o}}$ is executed with randomness $f(c)$, and the resulting token program is then executed with input x. The formal description of the protocol is given in Figure 6.15.

Theorem 6.5. *The protocol $\Pi_{\mathrm{SZK}}^{\text{b-r}}$ in Figure 6.15 statistically UC-realizes $\mathcal{F}_{\mathrm{ZK}}$ (cf. Section 2.7.2.2) in the $\mathcal{F}_{\mathrm{wrap}}^{\text{b-rs}}$-hybrid model, given that $\Pi_{\mathrm{COM}}^{\text{s-o}}$ and $\Pi_{\mathrm{COM}}^{\mathrm{rev}}$ are statistically UC-secure.*

Proof. **Corrupted Prover.** To prove security against a corrupted prover, the simulator has to extract the witness of the prover. This is done by having the simulator execute the real protocol with the prover. If the prover manages to convince the simulator of the validity of a statement, the simulator will use the sender simulator of $\Pi_{\mathrm{COM}}^{\text{s-o}}$ to extract all commitments and thus a witness. A formal description of the simulator is depicted in Figure 6.16. Let $\mathcal{A}_{\mathsf{P}}$ denote the dummy adversary.

The only possibility to distinguish between the real model and the ideal model is the abort of the simulator if no correct coloring was extracted. By the UC-security of $\Pi_{\mathrm{COM}}^{\text{s-o}}$, however, the sender simulator $\mathcal{S}_{\mathsf{S}}$ will extract the correct values of the commitment with overwhelming probability. Thus, it remains to show that the verifier in the real protocol will not accept a false proof except with negligible probability. In a single proof, the verifier will detect a cheating prover only with probability $\frac{1}{|E|}$, i.e. the winning probability of a prover is $1 - \frac{1}{|E|}$. For $t = n \cdot |E|$ repetitions, we get:

$$\begin{aligned}\Pr[\langle \mathsf{P}^*, \mathsf{V}\rangle(x) = 1 \wedge x \notin \mathcal{L}] &= \left(1 - \tfrac{1}{|E|}\right)^t = \left(1 - \tfrac{1}{|E|}\right)^{n\cdot|E|} = \exp\left(n \cdot |E| \cdot \log\left(1 - \tfrac{1}{|E|}\right)\right) \\ &\leq \exp\left(n \cdot |E| \cdot \left(-\tfrac{1}{|E|}\right)\right) = \exp(-n)\end{aligned}$$

We can conclude that the real model and the ideal model are indistinguishable except with negligible probability.

Corrupted Verifier. We construct a simulator against a corrupted verifier as follows (cf. Figure 6.17). In Step 1 of $\Pi_{\mathrm{SZK}}^{\text{b-r}}$, we exploit that $\widetilde{\Pi}_{COM}^{rev}$ is still UC-secure against a corrupted commitment sender and thus the challenge $\bar{E}$ can be

Protocol $\Pi_{\mathrm{SZK}}^{\text{b-r}}$

Implicitly parametrized by a simple 3-colorable graph $G = (V, E)$ and a query bound q in the sense that a malicious verifier can reset the prover at most $q - 1$ times. Let $n := |V|$, $t := n \cdot |E|$, $V := \{1, \ldots, n\}$ and let $\mathcal{T}_{\mathsf{com}}^{\mathsf{V}}$ and $\mathcal{T}_{\mathsf{com}}^{\mathsf{P}}$ be two instances of $\mathcal{F}_{\text{wrap}}^{\text{b-rs}}$.

Auxiliary input for prover:

A 3-coloring of G, denoted $\varphi : V \to \{1, 2, 3\}$.

Setup phase:

Prover: Select a random degree-q polynomial $f \leftarrow \mathbb{F}_{2^l}[X]$, where l is the number of random bits needed for token generation in $\Pi_{\mathrm{COM}}^{\text{s-o}}$ for $n \cdot t$ commitments. Further, select a random degree-q polynomial $g \leftarrow \mathbb{F}_{2^k}[X]$, where k is the number of random bits needed to generate t random permutations over $\{1, 2, 3\}$. W.l.o.g., l and k are larger than the commit message length in $\widetilde{\Pi}_{COM}^{rev}$. Create two tokens $\mathsf{T}_{\mathsf{com}}^{\mathsf{V}}$ and $\mathsf{T}_{\mathsf{com}}^{\mathsf{P}}$ with the following functionalities:

- $\mathsf{T}_{\mathsf{com}}^{\mathsf{V}}$: Implement the token functionality of $\widetilde{\Pi}_{COM}^{rev}$.
- $\mathsf{T}_{\mathsf{com}}^{\mathsf{P}}$: Upon input $(x, c_{\mathsf{com}}^{\mathsf{V}})$, simulate the token generation procedure of $\Pi_{\mathrm{COM}}^{\text{s-o}}$ with randomness $f(c_{\mathsf{com}}^{\mathsf{V}} \| 0 \ldots 0)$, evaluate the generated token program on input x, and output the result.

Send $(\texttt{create}, \mathsf{T}_{\mathsf{com}}^{\mathsf{V}})$ to $\mathcal{T}_{\mathsf{com}}^{\mathsf{V}}$ and $(\texttt{create}, \mathsf{T}_{\mathsf{com}}^{\mathsf{P}})$ to $\mathcal{T}_{\mathsf{com}}^{\mathsf{P}}$.

Proof phase:

1. Verifier: Uniformly and independently select a random value $\lambda_{\mathsf{com}}^{\mathsf{P}}$ according to the setup phase of $\Pi_{\mathrm{COM}}^{\text{s-o}}$ and a t-tuple of edges $\bar{E} = (\{u_1, v_1\}, \ldots, \{u_t, v_t\})$ as a challenge for the zero-knowledge proof. Use $\widetilde{\Pi}_{COM}^{rev}$ to commit to $(\bar{E}, \lambda_{\mathsf{com}}^{\mathsf{P}})$ and send the corresponding commit message $c_{\mathsf{com}}^{\mathsf{V}}$ to the prover.

2. Prover: Compute $r = f(c_{\mathsf{com}}^{\mathsf{V}} \| 0 \ldots 0)$ and $r' = g(c_{\mathsf{com}}^{\mathsf{V}} \| 0 \ldots 0)$. Use r' to select t random permutations $\pi_1, \ldots, \pi_t$ over $\{1, 2, 3\}$ and set $\phi_i(v) = \pi_i(\varphi(v))$ for each $v \in V$ and $i \in \{1, \ldots, t\}$. Use r to simulate the token generation of $\Pi_{\mathrm{COM}}^{\text{s-o}}$ and compute the corresponding $\Pi_{\mathrm{COM}}^{\text{s-o}}$-commit message $c_{\mathsf{com}}^{\mathsf{P}}$ to commit to $\phi_i(v)$ for all $v \in V$ and $i \in \{1, \ldots, t\}$. Send $c_{\mathsf{com}}^{\mathsf{P}}$ to the verifier.

3. Verifier: Send $c_{\mathsf{com}}^{\mathsf{V}}$ and the corresponding $\widetilde{\Pi}_{COM}^{rev}$-unveil message to the prover, thus unveiling $(\bar{E}, \lambda_{\mathsf{com}}^{\mathsf{P}})$.

4. Prover: If the unveil was not correct, abort. Else, compute $r = f(c_{\mathsf{com}}^{\mathsf{V}} \| 0 \ldots 0)$ and simulate the token generation of $\Pi_{\mathrm{COM}}^{\text{s-o}}$ as in Step 2. Compute the response $\tilde{p}_{\mathsf{com}}^{\mathsf{P}}$ for $\lambda_{\mathsf{com}}^{\mathsf{P}}$ according to the setup phase of $\Pi_{\mathrm{COM}}^{\text{s-o}}$. Let $w_{\mathsf{com}}^{\mathsf{P}}$ be the $\Pi_{\mathrm{COM}}^{\text{s-o}}$-unveil message for the commitments indexed by $\bar{E}$. Send $(\tilde{p}_{\mathsf{com}}^{\mathsf{P}}, w_{\mathsf{com}}^{\mathsf{P}})$ to the verifier.

5. Verifier: Check the unveiled commitments according to the unveil phase of $\Pi_{\mathrm{COM}}^{\text{s-o}}$. Also verify for each edge $\{u_i, v_i\} \in \bar{E}$ that $\phi_i(u_i) \neq \phi_i(v_i)$. If all checks are passed, accept the proof; if not, reject.

Figure 6.15: Statistically UC-secure bounded-resettable statistical zero-knowledge proof of knowledge in the $\mathcal{F}_{\text{wrap}}^{\text{b-rs}}$-hybrid model.

Simulator $\mathcal{S}_{\mathsf{P}}$

- Simulate $\mathcal{T}^{\mathsf{V}}_{\mathsf{com}}$ and $\mathcal{T}^{\mathsf{P}}_{\mathsf{com}}$ for $\mathcal{A}_{\mathsf{P}}$ and obtain ${\mathsf{T}^{\mathsf{V}}_{\mathsf{com}}}^*$ and ${\mathsf{T}^{\mathsf{P}}_{\mathsf{com}}}^*$.
- (Setup) Simulate the setup straightforwardly according to $\Pi^{\text{b-r}}_{\text{SZK}}$.
- (Proof) Simulate the verifier honestly in the protocol $\Pi^{\text{b-r}}_{\text{SZK}}$, i.e. use a uniformly random challenge as an input. If the verifier would accept, start the sender-simulator $\mathcal{S}_{\mathsf{S}}^{\Pi^{\text{s-o}}_{\text{COM}}}$ of $\Pi^{\text{s-o}}_{\text{COM}}$ with input ${\mathsf{T}^{\mathsf{P}}_{\mathsf{com}}}^*$ and the message transcript from Step 2 to extract the commitments and obtain t colorings $\varphi^*_1, \ldots, \varphi^*_t$. If none of the colorings $\varphi^*_1, \ldots, \varphi^*_t$ is valid, abort. Otherwise select one valid coloring $\hat{\varphi} = \varphi^*_i$ and send $(\texttt{witness}, \hat{\varphi})$ to $\mathcal{F}_{\text{ZK}}$. Simulate the rest according to protocol.

Figure 6.16: Simulator for a corrupted prover in the protocol $\Pi^{\text{b-r}}_{\text{SZK}}$.

extracted. Then, in Step 2, the simulated prover can commit to different colorings for each challenged vertex pair $\{u_i, v_i\} \in \bar{E}$ and to arbitrary colorings otherwise. The remaining protocol is just simulated straightforwardly. Let $\mathcal{A}_{\mathsf{V}}$ be the dummy adversary.

Simulator $\mathcal{S}_{\mathsf{V}}$

- Simulate $\mathcal{T}^{\mathsf{V}}_{\mathsf{com}}$ and $\mathcal{T}^{\mathsf{P}}_{\mathsf{com}}$ for $\mathcal{A}_{\mathsf{V}}$. Let $Q = (Q^{\mathsf{V}}_{\mathsf{com}}, Q^{\mathsf{P}}_{\mathsf{com}})$ denote the set of token queries that $\mathcal{A}_{\mathsf{V}}$ inputs into the tokens.
- (Setup) Simulate the setup phase straightforwardly according to $\Pi^{\text{b-r}}_{\text{SZK}}$.
- (Proof) Upon receiving a message (`accept`) from $\mathcal{F}_{\text{ZK}}$, start the sender-simulator $\mathcal{S}_{\mathsf{S}}^{\widetilde{\Pi}^{rev}_{COM}}$ of $\widetilde{\Pi}^{rev}_{COM}$ with the inputs from $Q^{\mathsf{V}}_{\mathsf{com}}$ and obtain a message $\hat{E}$. Create a pseudo-coloring $\hat{\varphi}$ for V. For each $\{u_i, v_i\} \in \bar{E}$, assign the vertices u_i and u_j different colors $\hat{\varphi}(u_i) \neq \hat{\varphi}(u_j)$. For all other vertices $v \in V$ set $\hat{\varphi}(v) \xleftarrow{\$} \{1, 2, 3\}$. Proceed as in the real protocol using the coloring $\hat{\varphi}$, but abort if $\hat{E} \neq \bar{E}$ although the unveil of $\widetilde{\Pi}^{rev}_{COM}$ succeeds.

Figure 6.17: Simulator for a corrupted verifier in the protocol $\Pi^{\text{b-r}}_{\text{SZK}}$.

Consider the following hybrid experiments.

Experiment 0: This is the real model.

Experiment 1: Identical to Experiment 0, except that $\mathcal{S}_1$ extracts the challenges and aborts if the extracted challenge does not match the real challenge.

Experiment 2: Identical to Experiment 1, except that $\mathcal{S}_2$ computes the pseudo-coloring $\hat{\varphi}$ and uses it for the rest of the protocol. This is the ideal model.

Indistinguishability of Experiment 0 and Experiment 1 follows from the UC-security of $\widetilde{\Pi}^{rev}_{COM}$, which guarantees that the extraction will succeed with overwhelming probability. Experiment 1 and Experiment 2 are indistinguishable because $\Pi^{\text{s-o}}_{\text{COM}}$ is statistically hiding. □

It remains to show that the proof system is actually bounded-resettable zero-knowledge.

Corollary 6.7. *The protocol* $\Pi_{\text{SZK}}^{\text{b-r}}$ *is bounded-resettable zero-knowledge.*

Proof. The resettable security of the prover follows from two facts. First, the prover's randomness (r, r') depends deterministically on his first message $c_{\mathsf{com}}^{\mathsf{V}}$, but remains completely unpredictable for the verifier because he is not allowed to make more than q queries. Second, by the binding property of $\widetilde{\Pi}_{COM}^{rev}$, a corrupted verifier cannot cheat in Step 3 of $\Pi_{\text{SZK}}^{\text{b-r}}$ other than switch to another instance of the zero-knowledge protocol with an unrelated prover randomness (r, r'). □

Remark. Our construction $\Pi_{\text{SZK}}^{\text{b-r}}$ can directly be used to obtain a *non-interactive* zero-knowledge proof of knowledge scheme in the bounded-resettable hardware model by storing the prover functionality in an additional token (or two other tokens, if each token should be queried only once).

6.5 Relation to Two-Party Computation

In light of the impossibility results for standard resettable hardware [GIMS10], a weaker model of resettable hardware is necessary to obtain positive results for general two-party computation with resettable hardware. The model that we proposed makes a natural physical assumption and allows efficient constructions of many cryptographic primitives such as commitments, OT and ZK proof systems. In the larger context, our results imply the following.

Statistically secure two-party computation: In the previous sections, we presented statistically UC-secure constructions for all the building blocks necessary (in particular OT) to perform general UC-secure two-party computation from existing general solutions [Kil88, IPS08]. Our result is the first construction of general statistically secure two-party computation in any resettable hardware token model.

Statistically secure non-interactive two-party computation: While we don't show a general feasibility result for statistically UC-secure non-interactive two-party computation, both the commitment scheme $\Pi_{\text{COM}}^{\text{s-o}}$ and the zero-knowledge proof system $\Pi_{\text{SZK}}^{\text{b-r}}$ can be made non-interactive. Thus we showed a partial positive answer for the feasibility of non-interactive two-party functionalities.

Our results are somewhat surprising, since they show that by simply fixing an upper bound on the number of resets for resettable hardware one obtains the same cryptographic strength as completely stateful tokens. While the bound imposes a severe constraint in theory, for practical applications in the real world such a bound occurs naturally. We want to stress again that the bound is part of the token program, i.e. there is *no* state that the token has to keep, and standard stateless tokens (e.g. smartcards) can be used to realize our functionalities.

Our results are summarized in Table 6.18. Computational results are already implied by resettable tokens.

Model	computational 2PC		statistical 2PC	
	interactive	non-interactive	interactive	non-interactive
stateful	✓	✓	✓	✓
stateful (inc. com.)	✓	✓	✓	✓
stateful (outg. com.)	✓	✓	✗(✓)	✗
resettable	✓	✓	✗(✓)	✗
bounded-res.	✓	✓	✓(2 Tokens)	(✓) Commitment,ZK
reusable resettable	✓	✓	✗(✓)	✗

Table 6.18: Feasibility of interactive and non-interactive two-party computation from bounded-resettable tamper-proof hardware.

7. Reusable Resettable Tamper-Proof Hardware

All protocols based on tamper-proof hardware that we discussed in the previous chapters require a specific token functionality for each protocol (e.g. the token stores a polynomial or it stores an input for a CRS and returns it when receiving a ZK proof), i.e. the token can at best be used for several instances of the same protocol. This has several negative implications if one wants to use tokens in real applications. Most notably, it might be necessary to keep a lot of different tokens. Additionally, the tokens cannot be mass-produced, making it seem unlikely that the functionality is implemented in hardware. For currently available smart cards, however, the protocols are too complex to be realized efficiently enough via software to be applicable in practice.

In this chapter, we focus on hardware tokens that can be used for different protocols while providing a predefined functionality. Reusable tamper-proof hardware was first considered by Hofheinz et al. [HMQU05], who introduce catalysts as a concept similar to generalized UC (GUC) setups [CDPW07]. They show that using a *trusted* signature card as a UC setup, which behaves as a catalyst, the card can be reused for multiple protocols (unlike a normal UC setup). Later, [CGS08] mention that in their security proof for UC-secure commitments from resettable hardware tokens, the simulator does not need to learn the token code, because he only needs black-box access to the functionality. This turns out to be the main property that is required to allow reusing the setup. Upon inspection of existing protocols, especially in the context of resettable hardware, many protocols actually use the tokens in a black-box manner in the proof [CGS08, GIS$^+$10, DS13, CKS$^+$14]. Nevertheless, all of these protocols require specific functionalities for their tokens.

Generally, physically uncloneable functions (PUFs) also provide a fixed functionality, which has (assumed) statistical security. One could thus imagine using PUFs to realize reusable tokens. However, in the context of transferable setups (i.e. setups that do not disclose whether they have been passed on, like PUFs), Boureanu et al. [BOV15] show that neither OT nor key exchange can be realized, and PUFs fall into the category of transferable setups. Tamper-proof hardware as defined in this thesis on the other hand is not a transferable setup, so their impossibilities do not apply to us.

One goal of this chapter is to find a meaningful functionality that allows a wide range of cryptographic protocols. In the literature, several works discuss bit-OT tokens as a very simple and cheap functionality [IPS08, GIS+10, AAG+14]. We propose to use a signature functionality as a general token functionality, because we want to minimize the number of tokens that has to be sent and we also believe that the efficiency of protocols based on bit-OT can at least be matched by signature cards that are actually available today. Due to the fact that we consider *untrusted* hardware tokens (i.e. a malicious protocol party can arbitrarily program malicious tokens), we require a unique signature scheme that is stored on the token.

Finding a correct formalization for reusable (resettable) tamper-proof hardware turns out to be very challenging. This is mainly due to the fact that depending on the formalization, the cryptographic strength of the model changes significantly. Let us elaborate on this. If a setup, i.e. in our case a resettable hardware token, is used for several protocols, standard UC simulation techniques do not work in general. Usually, it is assumed that the simulator simulates the token wrapper to the parties. This allows the simulator to obtain the token code and also to observe all queries that are made to the token. If we want to reuse the token, the simulator cannot simulate (and thereby manipulate) the token wrapper functionality, because for two different protocols the simulator will have to manipulate the token functionality in different ways, and an environment could distinguish this from a real execution. Instead, we have to find a way to simulate while having only black-box access to the token wrapper. The generalized UC (GUC) framework of Canetti et al. [CDPW07] considers exactly this scenario. In GUC, the main problem with the simulation is that the environment has direct access to the setup as well, while the simulator can only observe the messages that the corrupted party sends to the setup. In a related setting, Canetti et al. [CJS14] consider a global random oracle and solve the problem by giving the simulator access to all "illegitimate" queries that are made to the setup, so that all queries concerning a protocol (identified by an process-id PID), even from other protocols, can be observed.

Alternatively, one could use the technique of [HMQU05], where the setup is seized by a party at the beginning of the protocol and released at the end of it. During that time, no other party has access to the setup. Thus this model is skew to GUC, because the environment does not have access to the setup at all times. To keep the environment from using messages from other protocols, at the beginning of each protocol a nonce is chosen by the other party which has to be included in the communication.

Just from the above description, it becomes obvious that the latter formalization inherently requires some form of interaction between the parties. If the nonce is not chosen during the protocol runtime by the other party, messages from other protocol executions can be used during the current protocol run. Then the simulator is no longer able to observe all queries and the simulation will fail. If we use the technique described in [CJS14], it seems plausible that non-interactive two-party computation is possible (as in their case with a global random oracle), while the technique of [HMQU05] can only allow interactive two-party computation.

Our contribution. We provide the first reusable setup that is *untrusted*, compared to *trusted and incorruptible* setups like a global random oracle [CJS14], augmented CRS or key-registration with knowledge [CDPW07].

Towards this, we present two models of reusable and resettable tamper-proof hardware, one inspired by [HMQU05] but with the focus of real untrusted signa-

ture cards, which makes the model somewhat restrictive. Nevertheless, we show a UC-secure protocol for commitments, which implies general UC-secure two-party computation from reusable resettable tokens. The second model follows the more GUC-like approach of [CJS14], but makes some restrictions on the types of signature cards that can be used. In turn, this allows more powerful protocols, and we show that reusable signature cards allow for UC-secure non-interactive computation. We basically obtain the same result as [CJS14], but instead of using a global random oracle, our protocols can actually be instantiated with real signature cards.

In all our protocols, tokens have to be sent in both directions. Intuitively, this is necessary because we cannot use non-black-box techniques, and thus Lemma 5.1 applies.

Our techniques. Our aim is to design very efficient protocols. Thus we have to avoid expensive tools like generic zero-knowledge proofs. Typically, these proofs are necessary to avoid a direct channel from the token to the token sender via subliminal channels in the signatures. We adapt an approach that was proposed by Choi et al. [CKS+14], who use unique signatures. This allows us to directly send the messages to the sender without having to worry about subliminal channels.

In the restrictive model, a possible precomputation step of the party before sending the value to the token leads to problems with the extraction. Unlike the models of [HMQU05, CJS14] we have to find a way to extract the input of the party although it is not directly sent to the hardware token. Our approach is to use the precomputed value in combination with a pseudorandom generator to allow commitments to messages of arbitrary length. Additionally, we have to use randomness extraction to prevent a malicious token from tunneling information to its sender by aborting depending on the inputs.

In the less restrictive model, we construct non-interactive straight-line extractable commitments and non-interactive straight-line witness extractable arguments. For our construction of non-interactive straight-line witness extractable arguments, we adapt the Fiat-Shamir transformation [FS87] to the setting of signature tokens. Thus we are no longer dependent on the random oracle heuristic.

Structure of this chapter. The chapter is structured as follows. In Section 7.1, we formalize some restrictions and bounds that hold with respect to reusable tokens. In Section 7.2, we define a model of reusable resettable tamper-proof hardware that is compatible with real signature cards and present UC-secure commitments in this model. Then, we show in Section 7.3 that a less strict model allows even non-interactive secure computation with reusable tamper-proof hardware.

7.1 Limitations

Jumping ahead, the impossibilities stated here hold for both specifications of reusable tamper-proof hardware that we present in the following. In particular, UC and UC-like frameworks usually impose the restriction that the simulator only has black-box access to the reusable setup. Thus, compared to the standard definition of resettable tamper-proof hardware, the model of resettable reusable tamper-proof hardware has some limitations concerning non-interactive two-party computation. The degree of non-interactivity that can be achieved with resettable hardware, i.e. just sending tokens (and possibly an additional message) to the receiver, is impossible to obtain in the model of resettable reusable hardware.

Corollary 7.1. *There exists no protocol Π_{PF} using any number of reusable and resettable hardware tokens $\mathcal{T}_1, \ldots, \mathcal{T}_n$ issued from the sender to the receiver that computationally UC-realizes the ideal point function $\mathcal{F}_{\mathrm{PF}}$.*

Proof. This follows directly from the observation that the simulator for protocols based on reusable hardware is only allowed to have black-box access to the token, i.e. the simulator does not have access to the code of the token(s). Applying the same argumentation as in Lemma 5.1 and [CKS+14] yields the claim. □

The best we can hope for is thus a protocol for non-interactive two-party computation where the parties exchange two messages (including hardware tokens) to obtain a (somewhat) non-interactive protocol. Indeed, in the following sections we will present a protocol that achieves exactly this result based on reusable and resettable hardware tokens. Maybe even more interesting, even stateful reusable hardware tokens will not yield any advantage compared to resettable tokens.

Corollary 7.2. *There exists no protocol Π_{OT} using any number of reusable and stateful hardware tokens $\mathcal{T}_1, \ldots, \mathcal{T}_n$ issued from the sender to the receiver that statistically UC-realizes $\mathcal{F}_{\mathrm{OT}}$.*

Proof. First note, as above, that the simulator of a protocol against a token sender will not get the token code because he only has black-box access to the token. Thus the simulator cannot use rewinding during the simulation, and falls back to the input/output behavior of the token, i.e. Corollary 7.1 applies. Further, due to the impossibility of statistically secure OT based on stateless hardware [GIMS10], the claim follows. □

7.2 Real Signature Cards

A common way to obtain digital signatures in practice is the use of dedicated hardware tokens where the secret key is stored on tamper-proof hardware and is never exposed to a potentially compromised host system. Smart cards, which are common examples of such tokens, are standardized with regard to their physical properties, interface and command format in ISO/IEC 7816. RSA-based signature and encryption schemes are defined in PKCS#1 [JK03]. Client applications usually do not interface directly with the smart card on raw data. More commonly, they use the PKCS#11 interface [Lab04] which is then implemented by a library that has a driver for a specific token (e.g. OpenSC[1])

Commonly used algorithms for message signing make use of one-way functions that are based on assumed intractable problems such as the Discrete Logarithm Problem or the RSA problem. For reasons of efficiency *and* security, the signature is usually not computed directly on the message. Most algorithms have the following structure:

1. Compute a hash value $h(m)$ on the message m, where h is a cryptographic hash function. This is done for two reasons: first, it allows the signing of messages of arbitrary length. Second, it prevents an attacker from performing homomorphic operations on the message which could possibly render the signature scheme insecure.

[1] `https://github.com/OpenSC/OpenSC/wiki`

2. (Optional) Apply a padding pad to $h(m)$. This can either be done to store some information such as the algorithm used for message hashing (such as with the ASN.1 padding used in RSASSA-PKCS1-v1_5) or for improved security as with RSASSA-PSS. Care has to be taken not to introduce a subliminal channel into the signature.

3. Perform the actual cryptographic operation on the last step's result.

The preprocessing, i.e. hashing and padding, does not require the signing key. To increase performance, these steps are sometimes (in part) performed on the host before sending the result to the signature token. For the RSASSA-PKCS1-v1_5 signature scheme, PKCS#1 does not mandate where each step is performed. In contrast, RSASSA-PSS was specifically designed to allow computation of the message's hash on the host. As a consequence, signature tokens may support a variety of operation modes with different input data:

1. The whole message m is supplied to the token.

2. Hashing algorithms based on the Merkle-Damgård construction [Dam90, Mer79] allow for the the message to be processed block-wise. Thus all hashing steps but the last one are performed by the host, and the result is processed with the last block on the token.

3. The (optionally padded) hash is supplied to the token.

The supported operation modes depend on the token and are in some cases negotiable. While some signature cards support all aforementioned modes, others only supports Mode 2. If the client application does not interface with the token directly but e.g. uses a PKCS#11 library, mode selection is done by the library's token driver according to the token's capabilities and the requested signature scheme. In particular, it cannot be ruled out that some tokens can be forced to operate in a specified mode.

Our strict definition of reusable tamper-proof hardware in Section 7.2.1 thus makes no assumption on the underlying token and allows all three modes. We show in Section 7.2.2 that this still allows UC-secure commitments.

7.2.1 Model

Our definition of reusable resettable tamper-proof hardware is defined analogously to normal resettable tamper-proof hardware (cf. Chapter 5), but we add a mechanism that allows a protocol party to seize the signature card. This approach is inspired by the work of Hofheinz et al. [HMQU05]. As long as the card is seized, no other (sub-)protocol can use the signature card. An adversarial sender can still store a malicious functionality in the wrapper, and an adversarial receiver is allowed to reset the program. The formal description of the wrapper $\mathcal{F}_{\text{wrap}}^{\text{ru-strict}}$ is given in Figure 7.1.

We assume that the token receiver can verify that it obtained the correct token, e.g. by requesting some token identifier from the sender.

We want to use a functionality that is applicable to different scenarios, otherwise a reusable hardware token is meaningless. Due to its current availability, we focus on signature cards. In our protocols, we assume that each party has access to a hardware token containing a signature functionality. Since it is our goal to make

Functionality $\mathcal{F}_{\text{wrap}}^{\text{ru-strict}}$

Implicitly parametrized by a security parameter κ.

Creation:

1. Await an input $(\texttt{create}, \mathsf{M}, t)$ from the token issuer, where M is a deterministic Turing machine and $t \in \mathbb{N}$. Store (M, t) and send $(\texttt{created})$ to the adversary. Ignore all further `create`-messages.

2. Await a message $(\texttt{delivery})$ from the adversary. Then, send $(\texttt{ready})$ to the token receiver.

Execution:

3. Await an input $(\texttt{run}, w, sid)$ from the receiver. If no `create`-message has been sent, return a special symbol $\perp$. Otherwise, if $seized = sid$, run M on w from its most recent state. When M halts without generating output or t steps have passed, send $\perp$ to the receiver; otherwise store the current state of M and send the output of M to the receiver.

4. Await an input $(\texttt{seize}, sid)$ from the receiver. If $seized = \perp$, set $seized = sid$.

5. Await an input $(\texttt{release})$ from the receiver. Set $seized = \perp$.

Reset (adversarial receiver only):

6. Upon receiving a message $(\texttt{reset})$ from a corrupted token receiver, reset M to its initial state.

Figure 7.1: The wrapper functionality by which we model reusable resettable tamper-proof hardware. The runtime bound t is merely needed to prevent malicious token senders from providing a perpetually running program code M; it will be omitted throughout the rest of the chapter.

realistic assumptions about existing signature cards, we will define the signature scheme that the token is supposed to contain in a very general way. Thus, a large class of currently available signature cards and the corresponding standards fall into this definition. Basically, we make an explicit distinction between the preprocessing that is done on the host, and the actual signing process that is performed on the token.

Definition 7.3. *(Digital signatures with preprocessing) A signature scheme* SIG *with preprocessing consists of five PPT algorithms* KeyGen, PreSg, Sign, Vfy *and* Verify.

- $\mathsf{KeyGen}(\kappa)$ *takes as input the security parameter* κ *and generates a key pair consisting of a verification key* vk *and a signature key* sgk.

- $\mathsf{PreSg}(\mathsf{vk}, m)$ *takes as input the verification key* vk, *the message* m *and outputs a preprocessed message* p.

- $\mathsf{Sign}(\mathsf{sgk}, p)$ *takes as input a signing key* sgk *and a possibly preprocessed message* p *of fixed length. It outputs a signature* σ *on the message* p.

- $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$ *takes as input a verification key* vk*, a message* m *and a presumed signature* σ *on this message. It computes* $p \leftarrow \mathsf{PreSg}(\mathsf{vk}, m)$ *and then checks if* $\mathsf{Vfy}(\mathsf{vk}, p, \sigma) = 1$*. It outputs 1 if the check is passed and 0 otherwise.*
- $\mathsf{Vfy}(\mathsf{vk}, p, \sigma)$ *takes as input a verification key* vk*, a preprocessed message* p *and a presumed signature* σ *on this message. It outputs 1 if the signature is correct and 0 otherwise.*

We assume that the scheme is correct, i.e. it must hold for all messages m *that*

$$\forall \kappa \in \mathbb{N}\ \forall (\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathsf{KeyGen}(\kappa) : \mathsf{Vfy}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sgk}, \mathsf{PreSg}(m))) = 1.$$

Existential unforgeability can be defined analogously to the definition for normal signature schemes. However, the EUF-CMA property has to hold for $\mathsf{KeyGen}, \mathsf{Sign}$ and Vfy. The PreSg-algorithm is typically realized as a hash-function.

7.2.2 Protocols

In this section, we present the building blocks that are necessary for UC-secure two-party computation. First, we present a straight-line extractable commitment in Section 7.2.2.1. We then use this commitment in an adaption of a UC-secure commitment by Canetti et al. [CJS14], as shown in Section 7.2.2.2.

7.2.2.1 Straight-line Extractable Commitment

We need a straight-line extractable commitment scheme in the $\mathcal{F}_{\text{wrap}}^{\text{ru-strict}}$-hybrid model to achieve two-party computation. We enhance a protocol of Hofheinz et al. [HMQU05] which assumes trusted signature cards as a setup such that it is secure even with maliciously created signature cards. Towards this goal, we adapt a technique from Choi et al. [CKS+14] using unique signatures to our scenario. This is necessary because with untrusted tokens, it is difficult to verify the functionality that is stored on the token. A unique signature scheme allows this verification very efficiently. Additionally, it prevents the token from channeling information to the receiver of the signatures via subliminal channels.

Our protocol proceeds as follows. As a global setup, we assume that the commitment receiver created a token containing a digital signature functionality, i.e. basically serving as a signature oracle. In a first step, the commitment receiver sends a nonce N to the sender such that the sender cannot use messages from other protocols involving the hardware token. The sender then randomly draws two values x, r. Each value is concatenated with the nonce, preprocessed if the value is too large for the signature token, and sent to the token, which returns signatures on these values. The sender then commits to x using r as the randomness. This will allow the extractor to verify if he correctly extracted the commitment. The sender also commits to both signatures and x, r and N in a separate extractable commitment. To commit to the actual input s, the sender uses x as input for a pseudorandom generator and generates a pseudorandom one-time-pad of length $|s|$. Care has to be taken because a maliciously programmed signature card might leak some information about x and r to the receiver. Thus, the sender applies a 2-universal hash function to the values before using them and sends all commitments and the blinded message to the receiver. To unveil, the sender has to send its inputs and random coins to the receiver, who can then validate the correctness of the commitments. A

formal description of the protocol is shown in Figure 7.2. We abuse the notation in that we define $(c, p) \leftarrow \mathsf{COM.Commit}(x)$ to denote that the commitment c was created with randomness p.

Protocol $\Pi^{\text{se}}_{\text{COM}}$

Let $\mathcal{T}$ be an instance of $\mathcal{F}^{\text{ru-strict}}_{\text{wrap}}$ and PRG be a pseudorandom generator. Further let COM be a computationally hiding and extractable commitment scheme. Let SIG be a unique signature scheme according to Definition 2.14.

Global setup phase:

- Receiver: Compute $(\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathsf{SIG.KeyGen}(\kappa)$. Program a stateless token T with the following functionality.
  - Upon receiving a message $(\mathtt{vk})$, return vk.
  - Upon receiving a message $(\mathtt{sign}, m)$, compute $\sigma_m \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, m)$ and output σ_m.

  Send $(\mathtt{create}, \mathsf{T})$ to $\mathcal{T}$.
- Sender: Query $\mathcal{T}$ with $(\mathtt{vk})$ to obtain the verification key vk and check if it is a valid verification key for SIG.

Commit phase:

1. Receiver: Choose a nonce $N \leftarrow \{0,1\}^\kappa$ uniformly at random and send it to the sender.
2. Sender: Let s be the sender's input.
   - Draw $x, r \leftarrow \{0,1\}^{3\kappa}$ uniformly at random and choose a 2-universal hash function f from the family of 2-universal hash functions $\{f_h : \{0,1\}^{3\kappa} \rightarrow \{0,1\}^\kappa\}_{h \leftarrow \mathcal{H}}$.
   - Send $(\mathtt{seize})$ to $\mathcal{T}$. Compute $p_x \leftarrow \mathsf{SIG.PreSg}(\mathsf{vk}, (x, N))$, $p_r \leftarrow \mathsf{SIG.PreSg}(\mathsf{vk}, (r, N))$ and send $(\mathtt{sign}, p_x)$ and $(\mathtt{sign}, p_r)$ to $\mathcal{T}$ to obtain σ_x and σ_r.
   - Compute $c_s \leftarrow \mathsf{PRG}(f(p_x)) \oplus s$, $(c_x, f(p_r)) \leftarrow \mathsf{COM.Commit}(x)$ and $(c_\sigma, d_\sigma) \leftarrow \mathsf{COM.Commit}\big((\sigma_x, \sigma_r, x, r, N)\big)$.
   - Send (c_s, c_x, c_σ, f) to the receiver. Release $\mathcal{T}$ by sending $(\mathtt{release})$.

Unveil phase:

3. Sender: Send $(s, x, r, \sigma_x, \sigma_r, d_\sigma)$ to the receiver.
4. Receiver: Check if $\mathsf{SIG.Verify}(\sigma_x, (x, N)) = \mathsf{SIG.Verify}(\sigma_r, (r, N)) = 1$. Additionally, check if $\mathsf{COM.Open}(c_x, x, f(p_r)) = 1$, $\mathsf{COM.Open}(c_\sigma, (\sigma_x, \sigma_r, x, r, N), d_\sigma) = 1$ and $c_s \leftarrow \mathsf{PRG} f(p_x) \oplus s$. If not, abort; otherwise accept.

Figure 7.2: Computationally secure straight-line extractable commitment scheme in the $\mathcal{F}^{\text{ru-strict}}_{\text{wrap}}$-hybrid model.

Theorem 7.1. *The protocol $\Pi^{\text{se}}_{\text{COM}}$ in Figure 7.2 is a straight-line extractable commitment scheme as per Definition 2.10 in the $\mathcal{F}^{\text{ru-strict}}_{\text{wrap}}$-hybrid model, given that verifiable random functions exist.*

We split the proof into two lemmata, showing the computational hiding property of $\Pi^{\text{se}}_{\text{COM}}$ in Lemma 7.4 and the straight-line extraction in Lemma 7.5.

Lemma 7.4. *The protocol $\Pi^{\text{se}}_{\text{COM}}$ in Figure 7.2 is computationally hiding, given that* COM *is an extractable computationally hiding commitment scheme, f is a 2-universal hash function,* PRG *is a pseudorandom generator and* SIG *is an EUF-CMA-secure unique signature scheme.*

Proof. Let us consider a modified commit phase of the protocol $\Pi^{\text{se}}_{\text{COM}}$: instead of committing to the values $s, x, r, N, \sigma_x, \sigma_r$, the sender S inputs random values in the commitments and replaces the pseudorandom string which is used as a one-time-pad with a completely random string. Thus no information about the actual input remains. In the following, we will show that from the receiver's point of view, the real protocol and the modified protocol as described above are computationally indistinguishable. This implies that the commit phase of the protocol $\Pi^{\text{se}}_{\text{COM}}$ is hiding. Consider the following series of hybrid experiments.

Experiment 0: The real protocol $\Pi^{\text{se}}_{\text{COM}}$.

Experiment 1: Identical to Experiment 0, except that instead of computing PRG on $f(x)$ and $(c_x, f(r)) \leftarrow \mathsf{COM.Commit}(x)$, draw two values a and b uniformly at random and compute $\mathsf{PRG}(a)$ and $(c_x, b) \leftarrow \mathsf{COM.Commit}(x)$.

Experiment 2: Identical to Experiment 1, except that instead of using $\mathsf{PRG}(a)$ to obtain a one-time pad for s, S draws a value v uniformly at random and computes $v \oplus s$.

Experiment 3: Identical to Experiment 2, except that instead of using COM to commit to $(\sigma_x, \sigma_r, x, r, N)$, S commits to a random string of the same length.

Experiment 4: Identical to Experiment 3, except that instead of using COM to commit to x with randomness b, S commits to a random string of the same length. This is the ideal protocol.

Experiments 0 and 1 are statistically close, given that f is a 2-universal hash function and SIG is unique. A malicious receiver $\mathcal{A}_{\mathsf{R}}$ provides a maliciously programmed token $\mathcal{T}^*$ which might help distinguish the two experiments. In particular, the token might hold a state and it could try to communicate with $\mathcal{A}_{\mathsf{R}}$ via two communication channels:

1. $\mathcal{T}^*$ can try to hide messages in the signatures.
2. $\mathcal{T}^*$ can abort depending on the input of S.

The first case is prevented by using a unique signature scheme. The sender S asks $\mathcal{T}^*$ for a verification key vk^* and can verify that this key has the correct form for the assumed signature scheme. Then the unique signature property of the signature scheme states that each message has a unique signature. Furthermore, there exist no other verification keys such that a message has two different signatures. It was shown in [BVS06] that unique signatures imply subliminal free signatures. Summarized, given an adversary $\mathcal{A}_{\mathsf{R}}$ that can hide messages in the signatures, we can use this adversary to construct another adversary that can break the unique signature property of the signature scheme.

The second case is a bit more involved. The main idea is to show that applying a 2-universal hash function to x and r generates uniformly distributed values, even if R has some information about x and r. Since x and r are still drawn uniformly at random from $\{0,1\}^{3\kappa}$, $\mathcal{T}^*$ can only abort depending on a logarithmic part of the input. Otherwise, the probability for the event that $\mathcal{T}^*$ aborts becomes negligible in κ. Let X be the random variable describing inputs into the signature token and let Y describe the random variable representing the leakage. Y thus has at most 2 possible values. Thus, Lemma 2.23 gives a lower bound for the average min-entropy of X under Y, namely

$$\tilde{H}_\infty(X|Y) \geq H_\infty(X) - H_\infty(Y) = 3k - 1.$$

In the protocol, we apply $f \in \{f_h : \{0,1\}^{3\kappa} \to \{0,1\}^\kappa\}_{h \leftarrow \mathcal{H}}$ to X. Note that f is chosen *after* R^* sent the token. This means that we can apply the Generalized Leftover Hash Lemma (cf. Lemma 2.25):

$$\Delta((f_{\mathcal{H}}(X), H, Y); (U_k, H, Y)) \leq \frac{1}{2}\sqrt{2^{\tilde{H}_\infty(X|Y)}2^\kappa} \leq \frac{1}{2}\sqrt{2^{-(3\kappa-1)+\kappa}} \leq 2^{-\kappa}$$

We conclude that from $\mathcal{A}_\mathsf{R}$'s view, $f(x)$ and $f(r)$ are distributed uniformly over $\{0,1\}^\kappa$ and thus Experiment 0 and Experiment 1 are statistically indistinguishable. We will only sketch the rest of the proof.

Computational indistinguishability of Experiments 1 and 2 follows directly from the pseudorandomness of PRG, i.e. given a receiver R^* that distinguishes both experiments, we can use this receiver to construct an adversary that distinguishes random from pseudorandom values. Experiment 2 and Experiment 3 are computationally indistinguishable given that COM is computationally hiding. From a distinguishing receiver R^* we can directly construct an adversary that breaks the hiding property of the commitment scheme. And by the exact same argumentation, Experiments 3 and 4 are computationally indistinguishable. □

We now show the straight-line extractability of $\Pi^{\text{se}}_{\text{COM}}$.

Lemma 7.5. *The protocol $\Pi^{\text{se}}_{\text{COM}}$ in Figure 7.2 is straight-line extractable, given that COM is an extractable computationally hiding commitment scheme and SIG is an EUF-CMA-secure unique signature scheme.*

Proof. Consider the extraction algorithm in Figure 7.3. It searches the inputs of $\mathcal{A}_\mathsf{S}$ into the hybrid functionality $\mathcal{F}^{\text{ru-strict}}_{\text{wrap}}$ for the combination of input and randomness for the commitment that is to be extracted.

Let Q denote the set of inputs that $\mathcal{A}_\mathsf{S}$ sent to $\mathcal{F}^{\text{ru-strict}}_{\text{wrap}}$. Extraction will fail only in the event that values (x^*, r^*) are unveiled that have never been sent to $\mathcal{T}$, i.e. $x^*, r^* \notin Q$. We have to show that $\mathsf{Ext}_\mathsf{COM}$ extracts c^* with overwhelming probability, i.e. if the receiver accepts the commitment, an abort in Step 1 happens only with negligible probability.

Assume for the sake of contradiction that $\mathcal{A}_\mathsf{S}$ causes this event with non-negligible probability ε. We will use $\mathcal{A}_\mathsf{S}$ to construct an adversary $\mathcal{B}$ that breaks the EUF-CMA security of the signature scheme SIG with non-negligible probability. Let vk be the verification key that $\mathcal{B}$ receives from the EUF-CMA experiment. $\mathcal{B}$ simulates

Extractor $\mathsf{Ext}_{\mathsf{COM}}$

Upon input $c^* = ((c_s^*, c_x^*, c_\sigma^*, f^*), Q)$, where Q is the set of all queries that $\mathcal{A}_\mathsf{S}$ sent to $\mathcal{F}_{\text{wrap}}^{\text{ru-strict}}$, start the following algorithm.

1. Test for all $\alpha_i, \alpha_j \in Q$ if $\mathsf{COM.Open}(c_x^*, \alpha_i, f^*(\alpha_j)) = 1$. Otherwise, abort.
2. Let $(\hat{x}, \hat{r}) = (\alpha_i, \alpha_j)$ be the values obtained in the previous step. Output $\hat{s} \leftarrow \mathsf{PRG}(f^*(\hat{x})) \oplus c_s^*$.

Figure 7.3: The extraction algorithm for the straight-line extractable commitment protocol $\Pi_{\text{COM}}^{\text{se}}$.

$\mathcal{F}_{\text{wrap}}^{\text{ru-strict}}$ for $\mathcal{A}_\mathsf{S}$ by returning vk upon receiving a query $(\mathtt{vk})$; further let Q be the set of queries that $\mathcal{A}_\mathsf{S}$ sends to $\mathcal{F}_{\text{wrap}}^{\text{ru-strict}}$. For each query $(\mathtt{sign}, m)$, $\mathcal{B}$ forwards the message to the signature oracle of the EUF-CMA game and returns the resulting signature σ to $\mathcal{A}_\mathsf{S}$.

$\mathcal{B}$ now simulates the interaction between $\mathcal{A}_\mathsf{S}$ and R up to the point when $\mathcal{A}_\mathsf{S}$ sends the message c_σ. The next messages between $\mathcal{A}_\mathsf{S}$ and R represent the interaction between an honest receiver and a malicious commitment sender $\mathcal{A}'_\mathsf{S}$ for the extractable commitment scheme COM. Thus, $\mathcal{B}$ constructs a malicious $\mathcal{A}'_\mathsf{S}$ from the state of $\mathcal{A}_\mathsf{S}$, which interacts with an external commitment receiver.

Due to the extractability of COM, there exists an extractor Ext that on input $(c_\sigma, \mathcal{A}'_\mathsf{S})$ outputs a message $(\hat{\sigma}_x, \hat{\sigma}_r, \hat{x}, \hat{r}, \hat{N})$ except with negligible probability ν. $\mathcal{B}$ runs Ext, outputs $(\hat{\sigma}_x, (\hat{x}, \hat{N}))$ to the EUF-CMA experiment and terminates.

From $\mathcal{A}_\mathsf{S}$'s view, the above simulation is distributed identically to the real protocol conditioned on the event that the unveil of the commitment c_σ succeeds. By assumption, $\mathcal{A}_\mathsf{S}$ succeeds in committing to a signature with non-negligible probability ε in this case. It follows that the extractor Ext of COM will output a message $(\hat{\sigma}_x, \hat{\sigma}_r, \hat{x}, \hat{r}, \hat{N})$ with non-negligible probability $\varepsilon - \nu$. Thus $\mathcal{B}$ will output a valid signature $\hat{\sigma}_x$ for a value $(\hat{x}, \hat{N})$ with non-negligible probability. However, it did not query the signature oracle on this value, which implies breaking the EUF-CMA security of the signature scheme SIG.

Thus, the extractor will correctly output the value s with overwhelming probability. □

7.2.2.2 Obtaining UC-Secure Commitments

In order to achieve computationally secure two-party computation, we want to transform the straight-line extractable commitment from Section 7.2.2.1 into a UC-secure commitment. A UC-secure commitment can be used to create a UC-secure CRS via a coin-toss (cf. Section 5.4). General feasibility results, e.g. [CLOS02], then imply two-party computation from a UC-CRS.

One possibility to obtain a UC-secure commitment from our straight-line extractable commitment is to use the compiler of Damgård and Scafuro [DS13], which transforms any straight-line extractable commitment into a UC-secure commitment. The compiler provides an information-theoretic transformation, but this comes at the cost of requiring $O(\kappa)$ straight-line extractable commitments to commit to one bit only. If we use signature cards, this translates to a lot of calls to the signature cards and makes the protocol rather inefficient.

Instead, we adapt the UC commitment protocol of [CJS14] to our model. The key insight in their protocol is that trapdoor extraction is sufficient to realize a UC-secure commitment. They propose to use a trapdoor commitment in conjunction with straight-line extractable commitments via a global random oracle to realize a UC-secure commitment. If we want to replace their commitments with our construction, there is a subtle problem that we want to discuss here. In their compiler, the commitment sender first commits to his input via the trapdoor commitment scheme. Then, he queries the random oracle with his input (which is more or less equivalent to a straight-line extractable commitment) and the unveil information for the trapdoor commitment. In the security proof against a corrupted sender, the simulator has to extract the trapdoor commitment. Thus, in their case, the simulator just searches all queries to the random oracle for the correct unveil information. In our very strict model, if we replace the oracle call with our straight-line extractable commitments, this approach fails. At first sight, it seems possible to just use the extractor for the straight-line extractable commitment to learn the value. However, it is crucial for the proof of security against a corrupted receiver that the commitment value is never published. Further, while we can still see all queries that are made to the hardware token, the simulator does not (necessarily) learn the complete input, but rather a precomputed value for the signature. Therefore, a little more work is necessary in order to realize a UC-secure commitment in our model.

In essence, we can use our straight-line extractable commitment in a non-black-box way, although we have to enhance it by *extractable trapdoor* commitments[2]. The protocol proceeds as follows: First, the receiver chooses a trapdoor for the trapdoor commitment $\mathsf{TCOM}_{\mathsf{ext}}$ and commits to it via a straight-line extractable commitment. This ensures that the simulator against a corrupted receiver can extract the trapdoor and then equivocate the commitments of $\mathsf{TCOM}_{\mathsf{ext}}$. The sender then commits with $\mathsf{TCOM}_{\mathsf{ext}}$ to his input (in a similar fashion as in our straight-line extractable commitment) and uses the token to sign the unveil information. Against a corrupted sender, the simulator can thus extract the unveil information and thus extract the commitment. The commitment is sent the receiver, which concludes the commit phase. To unveil, the sender first commits to the unveil information of $\mathsf{TCOM}_{\mathsf{ext}}$ such that he cannot change his commitment when the receiver unveils the trapdoor in the next step. From there, the commitments are checked for validity and if everything checks out, the commitment is accepted. The formal description of our protocol is given in Figure 7.4.

Theorem 7.2. *The protocol* Π_{COM} *in Figure 7.4 computationally UC-realizes* $\mathcal{F}_{\mathrm{COM}}$ *(cf. Section 2.7.2.1) in the* $\mathcal{F}_{\mathrm{wrap}}^{\text{ru-strict}}$*-hybrid model, given that* $\mathsf{TCOM}_{\mathsf{ext}}$ *is an extractable trapdoor commitment,* SECOM *is a straight-line extractable commitment and* SIG *is an EUF-CMA secure unique signature scheme.*

Proof. **Corrupted sender.** Consider the simulator in Figure 7.5. It is basically a wrapper around the extraction algorithm for our straight-line extractable commitment. Against a corrupted sender, we only have to extract the input of the sender and input it into the ideal functionality.

The only possibility for an environment $\mathcal{Z}$ to distinguish $\mathsf{Real}_{\mathcal{A}_{\mathsf{S}}}^{\Pi_{\mathrm{COM}}}$ and $\mathsf{Ideal}_{\mathcal{S}_{\mathsf{S}}}^{\mathcal{F}_{\mathrm{COM}}}$ is the case of an abort by the simulator. However, we can apply Lemma 7.5 with two

[2]Note, that any commitment scheme can be made extractable (with rewinding) via an interactive protocol (cf. Section 2.3).

Protocol Π_{COM}

Let $\mathsf{TCOM}_{\mathsf{ext}}$ be an extractable trapdoor commitment scheme and let SECOM be the straight-line extractable commitment from Section 7.2.2.1.

Global Setup phase:

- Sender and receiver: Compute $(\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathsf{SIG.KeyGen}(\kappa)$. Program a stateless token $\mathsf{T_S}$ and $\mathsf{T_R}$, respectively, with the following functionality.
  - Upon receiving a message $(\mathtt{vk})$, return vk.
  - Upon receiving a message $(\mathtt{sign}, m)$, compute $\sigma_m \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, m)$ and output σ_m.

  Send $(\mathtt{create}, \mathsf{T_S})$ to $\mathcal{T}_{\mathsf{S}}$ and $(\mathtt{create}, \mathsf{T_R})$ to $\mathcal{T}_{\mathsf{R}}$, respectively.
- Sender and receiver: Query $\mathcal{T}_{\mathsf{S}}$ and $\mathcal{T}_{\mathsf{R}}$, respectively, with $(\mathtt{vk})$ to obtain the verification key vk and check if it is a valid verification key for SIG.

Commit phase:

1. Receiver: Compute $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{TCOM}_{\mathsf{ext}}.\mathsf{KeyGen}(\kappa)$ and draw a nonce $N \leftarrow \{0,1\}^{\kappa}$. Further compute $(c_{\mathsf{sk}}, d_{\mathsf{sk}}) \leftarrow \mathsf{SECOM.Commit}(\mathsf{sk})$ and send $(\mathsf{pk}, c_{\mathsf{sk}}, N)$ to the sender.
2. Sender: Let s be the sender's input.
   - Draw $x, r \leftarrow \{0,1\}^{3\kappa}$ uniformly at random and choose a 2-universal hash function f from the family of 2-universal hash functions $\{f_h : \{0,1\}^{3\kappa} \rightarrow \{0,1\}^{\kappa}\}_{h \leftarrow \mathcal{H}}$.
   - Send $(\mathtt{seize})$ to $\mathcal{T}$. Compute $p_x \leftarrow \mathsf{SIG.PreSg}(\mathsf{vk}, (x, N))$ and send $(\mathtt{sign}, p_x)$ to $\mathcal{T}_{\mathsf{S}}$ to obtain σ_x. Then, compute $p_r \leftarrow \mathsf{SIG.PreSg}(\mathsf{vk}, (r, N))$ and send $(\mathtt{sign}, p_r)$ to $\mathcal{T}_{\mathsf{S}}$ to obtain σ_r.
   - Compute $c_s = f(p_x) \oplus s$, $(c_x, f(p_r)) \leftarrow \mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Commit}(p_x)$ and $(c_\sigma, d_\sigma) \leftarrow \mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Commit}((\sigma_x, \sigma_r, p_x, p_r, N))$.
   - Send (c_s, c_x, c_σ, f) to the receiver. Release $\mathcal{T}$ by sending $(\mathtt{release})$.

Unveil phase:

3. Sender: Compute $(c_d, d_d) \leftarrow \mathsf{SECOM.Commit}(f(p_r), d_\sigma)$ and send c_d to the receiver.
4. Receiver: Send $(\mathsf{sk}, d_{\mathsf{sk}})$ to the sender.
5. Sender: Check if $\mathsf{TCOM}_{\mathsf{ext}}.\mathsf{TVer}(\mathsf{pk}, \mathsf{sk}) = 1$ and $\mathsf{SECOM.Open}(c_{\mathsf{sk}}, d_{\mathsf{sk}}, \mathsf{sk}) = 1$. Send $(s, p_x, p_r, \sigma_x, \sigma_r, d_\sigma)$ to the receiver.
6. Receiver: Check if $\mathsf{SIG.Vfy}(\sigma_s, p_s) = \mathsf{SIG.Vfy}(\sigma_r, p_r) = 1$. Additionally, check if $\mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Open}(c_s, s, f(p_r)) = 1$, $\mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Open}(c_\sigma, (\sigma_s, \sigma_r, p_x, p_r, N), d_\sigma) = 1$ and $c_s \oplus p_x = s$. If not, abort; otherwise accept.

Figure 7.4: Computationally UC-secure protocol realizing $\mathcal{F}_{\mathrm{COM}}$ in the $\mathcal{F}_{\mathrm{wrap}}^{\text{ru-strict}}$-hybrid model.

small modifications: first, we use an extractable trapdoor commitment instead of an extractable commitment, and second we do not use the PRG in our construction, hence this proof step can be omitted. It follows that the extraction is successful

Simulator $\mathcal{S}_{\mathsf{S}}$

- Upon receiving a message $(\texttt{sign}, m)$ from $\mathcal{A}_{\mathsf{S}}$, relay this message to $\mathcal{T}_{\mathsf{S}}$ and store m in a list Q. Forward the reply from $\mathcal{T}_{\mathsf{S}}$ to $\mathcal{A}_{\mathsf{S}}$.
- Upon receiving a message $(\texttt{vk})$, relay this message to $\mathcal{T}_{\mathsf{S}}$ and forward the reply to $\mathcal{A}_{\mathsf{S}}$.
- (Commit) Simulate Step 1 of Π_{COM} and let (c_s^*, c_σ^*, f^*) be the answer from $\mathcal{A}_{\mathsf{S}}$. Test for all $\alpha_i, \alpha_j \in Q$ if $\mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Open}(c_x^*, \alpha_i, f^*(\alpha_j)) = 1$, otherwise abort. Let $(\hat{x}, \hat{r}) = (\alpha_i, \alpha_j)$ be the values obtained in the previous step. Compute $\hat{s} = c_s^* \oplus \hat{x}$ and send $(\texttt{commit}, \hat{s})$ to $\mathcal{F}_{\mathrm{COM}}$.
- (Unveil) Simulate the behavior of an honest receiver and obtain s^*. If $s^* = \hat{s}$, send $(\texttt{unveil})$ to $\mathcal{F}_{\mathrm{COM}}$, otherwise abort.

Figure 7.5: Simulator against a corrupted sender in the protocol Π_{COM}

with overwhelming probability and the simulation is thus indistinguishable from a real protocol run.

Corrupted receiver. The case of a corrupted receiver is more complicated. The simulator proceeds as follows. In the commit phase, he just commits to the all zero string and sends the rest of the messages according to the protocol. To equivocate the commitment, the simulator first extracts the trapdoor $\hat{\mathsf{sk}}$ from the commitment that the receiver sent in the commit phase. He computes the image t under the 2-universal hash-function f that equivocates c_s to the value $\hat{s}$ obtained from the ideal functionality. Then, he samples a preimage $\hat{p}_x$ of t, and uses the trapdoor $\hat{\mathsf{sk}}$ to equivocate the commitment c_x to $\hat{p}_x$. Let $\hat{p}_r$ be the new unveil information. The simulator sends both $\hat{p}_x$ and $\hat{p}_r$ to the token $\mathcal{T}_{\mathsf{R}}$ to obtain σ_x and σ_r. Now, the second commitment c_σ has to be equivocated to the new signatures and inputs. From there, the simulator just executes a normal protocol run with the newly generated values.

Let $\mathcal{A}_{\mathsf{R}}$ be the dummy adversary. The formal description of the simulator is given in Figure 7.6.

Experiment 0: This is the real model.

Experiment 1: Identical to Experiment 0, except that $\mathcal{S}_1$ aborts if the extraction of $\hat{\mathsf{sk}}$ from c_{sk}^* fails, although $\mathsf{SECOM}.\mathsf{Open}(c_{\mathsf{sk}}^*, d_{\mathsf{sk}}^*, \mathsf{sk}^*)$.

Experiment 2: Identical to Experiment 1, except that $\mathcal{S}_2$ uses a uniformly random value t_x instead of applying f to p_x, and computes the preimage $\hat{p}_x$ of t_x under the 2-universal hash function f.

Experiment 3: Identical to Experiment 2, except that $\mathcal{S}_3$ computes $(c_\sigma, d_\sigma) \leftarrow \mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Commit}(0)$ in the commit phase. In the unveil phase, he sends $(\texttt{sign}, \hat{p}_x), (\texttt{sign}, \hat{p}_r)$ to $\mathcal{T}_{\mathsf{R}}$. As an unveil information, he computes $\hat{d}_\sigma \leftarrow \mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Equiv}(\hat{\mathsf{sk}}, c_\sigma, (\hat{\sigma}_x, \hat{\sigma}_r, \hat{p}_x, \hat{p}_r, N))$.

Experiment 4: Identical to Experiment 3, except that $\mathcal{S}_4$ computes $(c_x, d_x) \leftarrow \mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Commit}(0)$ in the commit phase and then computes the unveil information $\hat{p}_r \leftarrow \mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Equiv}(\hat{\mathsf{sk}}, c_x, \hat{p}_x)$. This is the ideal model.

Simulator $\mathcal{S}_{\mathsf{R}}$

- Upon receiving a message $(\texttt{sign}, m)$ from $\mathcal{A}_{\mathsf{R}}$, relay this message to $\mathcal{T}_{\mathsf{R}}$ and store m in a list Q. Forward the reply from $\mathcal{T}_{\mathsf{R}}$ to $\mathcal{A}_{\mathsf{R}}$.
- Upon receiving a message $(\texttt{vk})$, relay this message to $\mathcal{T}_{\mathsf{R}}$ and forward the reply to $\mathcal{A}_{\mathsf{R}}$.
- (Commit) Upon receiving a message $(\texttt{committed})$ from $\mathcal{F}_{\mathrm{COM}}$ and a message $(\mathsf{pk}, c_{\mathsf{sk}}, N)$, simulate Step 2 of Π_{COM} with input $s = 0$.
- (Unveil) Upon receiving a message $(\texttt{opened}, \hat{s})$, proceed as follows:
  - Start the straight-line extractor $\mathsf{Ext}_{\mathsf{COM}}$ from SECOM to extract the commitment c^*_{sk} to obtain $\hat{\mathsf{sk}}$ and check if $\mathsf{TCOM}_{\mathsf{ext}}.\mathsf{TVer}(\mathsf{pk}^*, \hat{\mathsf{sk}}) = 1$, if not abort.
  - Compute $t = \hat{c}_s \oplus \hat{s}$ and choose a preimage $\hat{p}_x \in \{x \mid f(x) = t\}$ of this value under f.
  - Compute $\hat{p}_r \leftarrow \mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Equiv}(\hat{\mathsf{sk}}, \hat{c}_x, \hat{p}_x)$, send $(\texttt{sign}, \hat{p}_x)$ and $(\texttt{sign}, \hat{p}_r)$ to $\mathcal{T}_{\mathsf{R}}$ and obtain $\hat{\sigma}_x$ and $\hat{\sigma}_r$, respectively.
  - Compute $\hat{d}_\sigma \leftarrow \mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Equiv}(\hat{\mathsf{sk}}, \hat{c}_\sigma, (\hat{\sigma}_x, \hat{\sigma}_r, \hat{p}_x, \hat{p}_r, N))$.
  - Execute the unveil phase according to Π_{COM}: Commit to $\hat{d}_\sigma$ and $f(\hat{p}_r)$ in Step 3, and abort if $\mathsf{sk}^* \neq \hat{\mathsf{sk}}$ in Step 5. Otherwise, send $(\hat{s}, \hat{p}_x, \hat{p}_r, \hat{\sigma}_x, \hat{\sigma}_r, \hat{d}_\sigma, N)$ to $\mathcal{A}_{\mathsf{R}}$.

Figure 7.6: Simulator against a corrupted receiver in the protocol Π_{COM}

Experiment 0 and Experiment 1 are computationally indistinguishable given that SECOM is a straight-line extractable commitment. A distinguishing environment can directly be transformed into an adversary that breaks the straight-line extraction property. Experiments 1 and 2 are statistically indistinguishable, given that f is a 2-universal hash function (the same argumentation as in Lemma 7.4 applies). Additionally, it is obvious that a preimage is efficiently sampleable. Experiment 2 and Experiment 3 are computationally indistinguishable, given that $\mathsf{TCOM}_{\mathsf{ext}}$ is a trapdoor commitment scheme. A distinguishing environment $\mathcal{Z}$ can straightforwardly be used to break the equivocation property of the commitment scheme. The same argumentation holds for Experiment 3 and Experiment 4. □

Remark. The commitment length of our protocol is bounded by the length of the input into the token. Thus, for longer messages, the protocol has to be applied piecewise for each part of the message.

Our protocol is less efficient than the one by Hofheinz et al. [HMQU05] for long messages because their definition of a signature card does not accurately model real signature cards. In particular, a real signature token only learns a limited amount of information with each signature, making it impossible to commit to large messages as efficiently as in their protocol. Moreover, since they only assume trusted hardware tokens (and we consider untrusted ones) it is necessary to extract good randomness from the values that are sent to the signature card, meaning that a malicious card cannot compromise the security of the protocol via aborts. This leads to a further loss in efficiency. It is interesting to note that there is a trade-off between the asymptotically less efficient protocol of applying [DS13] to our straight-line extractable

commitment $\Pi^{\text{se}}_{\text{COM}}$ and our direct construction of a UC-secure commitment Π_{COM}. If the commitment message is relatively short compared to the security parameter, then Π_{COM} is much more efficient than the compiled version of $\Pi^{\text{se}}_{\text{COM}}$. However, if the message length of the commitment by far exceeds the length of the security parameter (i.e. if ℓ is the commitment length and $\ell > O(\kappa^2)$), [DS13] becomes more efficient.

Remark. If we instantiate the above protocol in the $\mathcal{F}^{\text{ru}}_{\text{wrap}}$-hybrid model (cf. Section 7.3.1), the practical efficiency is even superior to the protocol of [HMQU05] because we do not use zero-knowledge proofs, which require an rather inefficient $\mathcal{NP}$ reduction.

7.3 Ideal Signature Cards

The model considered in the previous section allows a broad class of signature algorithms that can be placed on the token. This comes with the drawback that a lot of functions cannot be realized securely. In particular, non-interactive protocols are directly ruled out by the model. In this section, we want to explore what is theoretically feasible with reusable hardware tokens, at the cost of limiting the types of signatures that are suitable for our scenario. In particular, we require that the complete message that is to be signed is given to the signature card. Nevertheless, there are currently available signature cards that can be used for the protocols that are presented in this section, e.g. FinID[3].

7.3.1 Model

In contrast to $\mathcal{F}^{\text{ru-strict}}_{\text{wrap}}$, we now adapt the simulation trapdoor of Canetti et al. [CJS14] from a global random oracle to the scenario of reusable tamper-proof hardware. To overcome the problem that the simulator cannot read queries to the setup functionality outside of the current protocol, the authors require parties that query the setup to include the current session id SID of the protocol. If a malicious party queries the setup in another protocol, using the SID of the first protocol, the setup will store this query in a list and give the simulator access to this list (via the ideal functionality with which the simulator communicates). This mechanism ensures that the simulator only learns illegitimate queries, since honest parties will always use the correct SID.

We thus enhance the standard resettable wrapper functionality $\mathcal{F}^{\text{resettable}}_{\text{wrap}}$ by the query list, and parse inputs as a concatenation of actual input and the session id (cf. Figure 7.7).

Compared to our previous reusable token specification $\mathcal{F}^{\text{ru-strict}}_{\text{wrap}}$, it is no longer necessary to use a nonce to bind the messages to one specific protocol instance. Thus, the inherent interaction of the $\mathcal{F}^{\text{ru-strict}}_{\text{wrap}}$-hybrid model is removed in the $\mathcal{F}^{\text{ru}}_{\text{wrap}}$-hybrid model. This will allow a much broader class of functionalities to be realized. For our purposes, however, we have to assume that the token learns the complete input, in contrast to the previous model (cf. Section 7.2.1). This is similar to the model assumed in [HMQU05], but in contrast to their work, we focus on untrusted tokens.

[3] `https://eevertti.vrk.fi/Default.aspx?id=377`

Functionality $\mathcal{F}_{\text{wrap}}^{\text{ru}}$

Implicitly parametrized by a security parameter κ and a list $\bar{\mathcal{F}}$ of ideal functionality programs.

Creation:

1. Await an input $(\texttt{create}, \mathsf{M}, t)$ from the token issuer, where M is a deterministic Turing machine and $t \in \mathbb{N}$. Store (M, t) and send $(\texttt{created})$ to the adversary. Ignore all further `create`-messages.

2. Await a message $(\texttt{delivery})$ from the adversary. Then, send $(\texttt{ready})$ to the token receiver.

Execution:

3. Await an input $(\texttt{run}, w)$ from the receiver with party id PID and session id SID. Parse w as (w', sid). If no `create`-message has been sent, return a special symbol $\perp$. Otherwise, run M on w from its most recent state and add $(sid, w', \mathsf{M}(w'))$ to the list of illegitimate queries Q_{sid} if $\mathsf{SID} \neq sid$. When M halts without generating output or t steps have passed, send $\perp$ to the receiver; otherwise store the current state of M and send the output of M to the receiver.

Reset (adversarial receiver only):

4. Upon receiving a message $(\texttt{reset})$ from a corrupted token receiver, reset M to its initial state.

5. Upon receiving a message $(\texttt{list})$ from an ideal functionality in the list $\bar{\mathcal{F}}$ with SID sid, return Q_{sid}.

Figure 7.7: The wrapper functionality by which we model reusable resettable tamper-proof hardware. The runtime bound t is merely needed to prevent malicious token senders from providing a perpetually running program code M; it will be omitted throughout the rest of the chapter.

Let us briefly state why we believe that this model is still useful. On the one hand, there are signature cards that support that the user inputs the complete message without any preprocessing. On the other hand, the messages that we input are typically rather short (linear in the security parameter), implying that the efficiency of the token is not reduced by much. Even to the contrary, this allows us to construct more round- and communication-efficient protocols, such that the overall efficiency increases.

7.3.2 Protocols

In this section, we show how to realize UC-secure non-interactive computation and the required tools. First, we construct a non-interactive straight-line extractable commitment scheme in Section 7.3.2.1, which is a simple adaptation of the straight-line extractable commitment from Section 7.2.2.1 to the weaker model $\mathcal{F}_{\text{wrap}}^{\text{ru}}$. We use this non-interactive commitment to create a non-interactive straight-line witness-extractable argument in Section 7.3.2.2. Then we sketch how our non-interactive straight-line witness-extractable argument can be used to obtain a one-sided simu-

latable OT protocol (cf. Section 7.3.2.3) which is the essential building block necessary to realize UC-secure non-interactive computation in the $\mathcal{F}_{\text{wrap}}^{\text{ru}}$-hybrid model in Section 7.3.2.4.

7.3.2.1 Non-Interactive Straight-Line Extractable Commitment

Since the ideal token functionality takes care of messages from other protocols that might maliciously be used in this protocol, it is no longer necessary to send a nonce from the receiver to the sender during the commitment. Additionally, the simulator now learns the actual inputs into the signature functionality, which enables us to extract messages directly without having to work with preprocessed values. We slightly modify the commitment from Section 7.2.2.1 to fit into the $\mathcal{F}_{\text{wrap}}^{\text{ru}}$-hybrid model.

Lemma 7.6. *The protocol $\Pi_{\text{COM}}^{\text{ni-se}}$ in Figure 7.8 is a straight-line extractable commitment scheme as per Definition 2.10 in the $\mathcal{F}_{\text{wrap}}^{\text{ru}}$-hybrid model, given that* COM *is an extractable computationally hiding commitment scheme and* SIG *is an EUF-CMA-secure unique signature scheme.*

Proof. We show that $\Pi_{\text{COM}}^{\text{ni-se}}$ satisfies Definition 2.9. The security proof essentially follows the proof of Theorem 7.1 with some minor modifications. The proof for the hiding property can be adopted completely, except that the PRG is not necessary in the protocol and therefore the hybrid step can be omitted.

The extraction step is technically the same, but the analysis is even simpler than in the proof of Theorem 7.1. Consider the extraction algorithm in Figure 7.9. It searches the inputs into the hybrid functionality $\mathcal{F}_{\text{wrap}}^{\text{ru}}$ for the combination of input and randomness for the commitment that is to be extracted. The only difference to the extractor of $\Pi_{\text{COM}}^{\text{se}}$ is that the input is directly extracted from the queries to the token.

The rest of the proof is identical to the extractability proof of Theorem 7.1. □

7.3.2.2 Non-Interactive Straight-line Witness-Extractable Arguments

Our protocol is based on the construction of Pass [Pas03] who presented a protocol for a non-interactive straight-line witness-extractable proof in the random oracle model. Let $\Pi = (\alpha, \beta, \gamma)$ be a Σ-protocol, i.e. a three message zero-knowledge proof system. We also assume that Π has special soundness, i.e. from answers γ_1, γ_2 to two distinct challenges β_1, β_2, it is possible to reconstruct the witness that the prover used.

The main idea of his construction is as follows. Instead of performing a Σ-protocol interactively, a Fiat-Shamir transformation [FS87] is used to make the protocol non-interactive. The prover computes the first message α of the Σ-protocol, selects two possible challenges β_1 and β_2, computes the resulting answers γ_1 and γ_2 based on the witness w according to the Σ-protocol for both challenges and computes commitments c_i to the challenge/response pairs. Instead of having the verifier choose one challenge, in [FS87], a hash-function is applied to the commitment to determine which challenge is to be used. The prover then sends (α, c) and the unveil information of the c_i to the verifier. The verifier only has to check if the unveil is correct under the hash function and if the resulting Σ-protocol transcript $(\alpha, \beta_i, \gamma_i)$ is correct. The resulting protocol only has soundness $\frac{1}{2}$ and thus has to be executed several times in

Protocol $\Pi_{\mathrm{COM}}^{\text{ni-se}}$

Let $\mathcal{T}$ be an instance of $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{ru}}$. Further let COM be a computationally hiding and extractable commitment scheme. Let SIG be a unique signature scheme according to Definition 2.14.

Global setup phase:

- Receiver: Compute $(\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathsf{SIG.KeyGen}(\kappa)$. Program a stateless token T with the following functionality.
  - Upon receiving a message $(\mathtt{vk})$, return vk.
  - Upon receiving a message $(\mathtt{sign}, m)$, compute $\sigma_m \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, m)$ and output σ_m.

  Send $(\mathtt{create}, \mathsf{T})$ to $\mathcal{T}$.
- Sender: Query $\mathcal{T}$ with $(\mathtt{vk})$ to obtain the verification key vk and check if it is a valid verification key for SIG.

Commit phase:

1. Sender: Let s be the sender's input.
   - Draw $r \leftarrow \{0,1\}^{3\kappa}$ uniformly at random and choose a 2-universal hash function f from the family of 2-universal hash functions $\{f_h : \{0,1\}^{3\kappa} \rightarrow \{0,1\}^{\kappa}\}_{h \leftarrow \mathcal{H}}$.
   - Send $(\mathtt{sign}, s)$ and $(\mathtt{sign}, r)$ to $\mathcal{T}$ and obtain σ_s and σ_r.
   - Compute both $(c_\sigma, d_\sigma) \leftarrow \mathsf{COM.Commit}(\sigma_s, \sigma_r, s, r)$ and $(c_s, f(r)) \leftarrow \mathsf{COM.Commit}(s)$.
   - Send (c_s, c_σ, f) to the receiver.

Unveil phase:

2. Sender: Send $(s, r, \sigma_s, \sigma_r, d_\sigma)$ to the receiver.
3. Receiver: Check if $\mathsf{SIG.Verify}(\sigma_s, s) = \mathsf{SIG.Verify}(\sigma_r, r) = 1$. Additionally, check if $\mathsf{COM.Open}(c_s, f(r), s) = 1$ and $\mathsf{COM.Open}(c_\sigma, d_\sigma, (\sigma_s, \sigma_r, s, r)) = 1$. If not, abort; otherwise accept.

Figure 7.8: Computationally secure non-interactive straight-line extractable commitment scheme in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{ru}}$-hybrid model.

Extractor $\mathsf{Ext}_{\mathsf{COM}}$

Upon input $((c_s^*, c_\sigma^*, f^*), Q)$, where Q is the set of all query/answer pairs that $\mathcal{A}_{\mathsf{S}}$ sent to and received from $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{ru}}$, start the following algorithm.

1. Test for all $\alpha_i, \alpha_j \in Q$ if $\mathsf{COM.Open}(c_s^*, \alpha_i, f^*(\alpha_j)) = 1$. Otherwise, abort.
2. Let $(\hat{s}, \hat{r}) = (\alpha_i, \alpha_j)$ be the values obtained in the previous step. Output $\hat{s}$.

Figure 7.9: The extraction algorithm for the non-interactive straight-line extractable commitment protocol $\Pi_{\mathrm{COM}}^{\text{ni-se}}$.

parallel. [Pas03] replaces the hash function by a random oracle and thus obtains a proof system. If the commitment is straight-line extractable, so is the proof system.

We replace the random oracle by the token functionality defined in Section 7.3.1. While we can only achieve computational security against a malicious prover, this allows us to use the straight-line extractable commitment scheme from Section 7.3.2.1 to obtain an straight-line witness-extractable argument. A formal description of the protocol is given in Figure 7.10.

Protocol Π_{NIWI}

Let (α, β, γ) denote the three messages of a Σ-protocol Π for a language $\mathcal{L}$. Further let SECOM be an instance of $\Pi_{\text{COM}}^{\text{ni-se}}$ from Section 7.3.2.1.

Global setup phase:

- Verifier: Compute $(\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathsf{SIG.KeyGen}(\kappa)$. Program a stateless token T with the following functionality.
  - Upon receiving a message $(\mathtt{vk})$, return vk.
  - Upon receiving a message $(\mathtt{sign}, m)$, compute $\sigma_m \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, m)$ and output σ_m.

  Send $(\mathtt{create}, \mathsf{T})$ to $\mathcal{T}$.
- Sender: Query $\mathcal{T}$ with $(\mathtt{vk})$ to obtain the verification key vk and check if it is a valid verification key for SIG.

Proof phase:

1. Prover: Let a statement x and a witness w be the prover's input, such that $(x, w) \in \mathcal{R}_{\mathcal{L}}$. Let $l = \mathsf{poly}(\kappa)$ be the length of the signature.
   - Compute l first messages $\alpha = (\alpha^{(1)}, \ldots, \alpha^{(l)})$ of the Σ-protocol. Pick $2l$ random challenges $(\beta_0^{(1)}, \beta_1^{(1)}), \ldots, (\beta_0^{(l)}, \beta_1^{(l)})$ for the Σ-protocol with $\beta_0^{(i)} \neq \beta_1^{(i)} \forall i \in \{1, \ldots, l\}$. Compute the corresponding answers $(\gamma_0^{(1)}, \gamma_1^{(1)}), \ldots, (\gamma_0^{(l)}, \gamma_1^{(l)})$ for the Σ-protocol from w.
   - Commit to the challenge/response pairs $(\beta_b^{(i)}, \gamma_b^{(i)})$ via $(c_b^{(i)}, d_b^{(i)}) \leftarrow \mathsf{SECOM.Commit}(\beta_b^{(i)}, \gamma_b^{(i)})$ for all $i \in \{1, \ldots, l\}$ and $b \in \{0, 1\}$. Let $c = ((c_0^{(1)}, c_1^{(1)}), \ldots, (c_0^{(l)}, c_1^{(l)}))$.
   - Send $(\mathtt{sign}, (\alpha, c))$ to $\mathcal{T}$ and let σ denote the result.

   Send $\pi = (\alpha, c, \sigma, (d_{\sigma_1}^{(1)}, \ldots, d_{\sigma_l}^{(l)})$ to the verifier where σ_i signifies the i-th bit of σ.
2. Verifier: First, check if $\mathsf{SIG.Verify}(\mathsf{vk}, \sigma, (a, c)) = 1$. If that is not the case, abort; otherwise check for all i that $(\alpha^{(i)}, \beta_{\sigma_i}^{(i)}, \gamma_{\sigma_i}^{(i)})$ is an accepting transcript for $x \in \mathcal{L}$ and $\mathsf{SECOM.Open}(c_{\sigma_i}^{(i)}, d_{\sigma_i}^{(i)}, (\beta_{\sigma_i}^{(i)}, \gamma_{\sigma_i}^{(i)})) = 1$. If that check is passed, accept; otherwise abort.

Figure 7.10: Computationally secure non-interactive straight-line witness-extractable argument in the $\mathcal{F}_{\text{wrap}}^{\text{ru}}$-hybdrid model.

Theorem 7.3. *The protocol Π_{NIWI} in Figure 7.10 is a straight-line witness-extractable argument as per Definition 2.20 in the $\mathcal{F}_{\text{wrap}}^{\text{ru}}$-hybrid model, given that* SECOM *is a*

straight-line extractable commitment scheme and SIG *is an EUF-CMA secure unique signature scheme.*

Proof. Let Π be a public-coin special-sound honest-verifier zero-knowledge (SHVZK) protocol.

Completeness: Completeness of Π_{NIWI} follows directly from the completeness of the Σ-protocol Π.

Witness-Indistinguishability: Cramer et al. [CDS94, Pas03] show that a SHVZK protocol directly implies a public-coin witness-indistinguishable protocol. Since witness-indistinguishable protocols are closed under parallel composition as shown be Feige et al. [FS90], Π_{NIWI} is witness-indistinguishable.

Extractablility: Let $\mathsf{Ext}_{\mathsf{COM}}$ be the straight-line extractor of SECOM. We will construct a straight-line extractor for Π_{NIWI} (cf. Figure 7.11).

Extractor $\mathsf{Ext}_{\mathsf{NIWI}}$

Let $\mathsf{Ext}_{\mathsf{COM}}$ be the extraction algorithm for SECOM. Upon input $(\pi^* = (\alpha^*, c^*, \sigma^*, (d^{*(1)}_{\sigma_1}, \ldots, d^{*(l)}_{\sigma_l})), Q)$, where Q is the set of queries to $\mathcal{F}^{\text{ru}}_{\text{wrap}}$, start the following algorithm.

1. Run the verifier algorithm on π^*, if it aborts, abort. Run $\mathsf{Ext}_{\mathsf{COM}}$ with input (c^*, Q) to extract all commitments and obtain $(\hat{\beta}^{(i)}_b, \hat{\gamma}^{(i)}_b)\, \forall i \in \{1, \ldots, l\}$.
2. Select the first correct witness $\hat{w}$ derived from $(\hat{\gamma}^{(i)}_0, \hat{\gamma}^{(i)}_1)$ and output $\hat{w}$.

Figure 7.11: The extraction algorithm for the non-interactive straight-line witness-extractable argument Π_{NIWI}.

It remains to show that if the verifier accepts, $\mathsf{Ext}_{\mathsf{NIWI}}$ outputs a correct witness with overwhelming probability. First, note that $\mathsf{Ext}_{\mathsf{COM}}$ extracts the inputs of c^* with overwhelming probability, and by the special soundness of Π, we know that if both challenges in the commitment are extracted, $\mathsf{Ext}_{\mathsf{NIWI}}$ will obtain a witness. Thus, the only possibility for $\mathsf{Ext}_{\mathsf{NIWI}}$ to fail with the extraction is if a malicious PPT prover $\mathcal{A}_{\mathsf{P}}$ manages to convince the verifier with a witness w^* such that $(x, w^*) \notin \mathcal{R}_{\mathcal{L}}$.

Each of the l instances of Π has soundness $\frac{1}{2}$, since a malicious $\mathcal{A}_{\mathsf{P}}$ can only answer at most one challenge correctly, and otherwise a witness is obtained. Thus, $\mathcal{A}_{\mathsf{P}}$ has to make sure that in all l instances, the correctly answered challenge is selected. Assume for the sake of contradiction that $\mathcal{A}_{\mathsf{P}}$ manages to convince the verifier with some non-negligible probability ϵ of a witness w^* such that $(x, w^*) \notin \mathcal{R}_{\mathcal{L}}$. We will construct an adversary $\mathcal{B}$ from $\mathcal{A}_{\mathsf{P}}$ that breaks the EUF-CMA property of SIG with probability ϵ.

Let $\mathcal{B}$ be the adversary for the EUF-CMA game. Let vk be the verification key that $\mathcal{B}$ receives from the EUF-CMA game. $\mathcal{B}$ simulates $\mathcal{F}^{\text{ru}}_{\text{wrap}}$ to $\mathcal{A}_{\mathsf{P}}$ by returning vk upon receiving a query $(\mathtt{vk})$; further let Q be the set of queries that $\mathcal{A}_{\mathsf{P}}$ sends to $\mathcal{F}^{\text{ru}}_{\text{wrap}}$. For each query $(\mathtt{sign}, m)$, $\mathcal{B}$ forwards the message to the signature oracle of the EUF-CMA game and returns the resulting signature σ to $\mathcal{A}_{\mathsf{P}}$.

If $\mathcal{B}$ receives a signature query of the form $(\texttt{sign}, m^*)$ with $m^* = (\alpha^*, c^*)$, start the extractor $\mathsf{Ext}_{\mathsf{COM}}$ with input $(c^*,)$ to extract the commitments c^* using Q. Create a signature σ^* by selecting σ_i^* as the index of the correctly evaluating challenge. The verifier will only accept if that is the case. If $\mathsf{SIG.Verify}(\mathsf{vk}, \sigma^*, (\alpha^*, c^*)) = 1$, send (m^*, σ^*) to the EUF-CMA game, otherwise abort. We thus have that $\mathcal{A}_{\mathsf{P}}$ wins the EUF-CMA game with probability ϵ, which contradicts the EUF-CMA security of SIG. □

7.3.2.3 One-Sided Simulatable OT

To obtain our main result, we need to introduce a variant of OT called one-sided simulatable OT. One-sided simulatable OT requires that only the security against a corrupted receiver is simulation-based. The security against a corrupted sender is indistinguishability-based.

Definition 7.7 ([CJS14])**.** *Let $\mathcal{F}_{\mathrm{OT}}$ be the ideal functionality for oblivious transfer. We say that a protocol Π securely realizes $\mathcal{F}_{\mathrm{OT}}$ with* one-sided simulation *in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{ru}}$-hybrid model if the following holds:*

1. *For every non-uniform PPT adversary $\mathcal{A}_{\mathsf{R}}$ controlling the receiver in the real model, there exists a non-uniform PPT adversary $\mathcal{S}$ for the ideal model, such that for any environment $\mathcal{Z}$,*

$$\mathsf{Real}_{\Pi}^{\mathcal{A}_{\mathsf{R}}}(\mathcal{Z}) \underset{c}{\approx} \mathsf{Ideal}_{\mathcal{F}_{\mathrm{OT}}}^{\mathcal{S}}(\mathcal{Z})$$

2. *For every non-uniform PPT adversary $\mathcal{A}_{\mathsf{S}}$ controlling the sender,*

$$\{\mathsf{view}_{\Pi,\mathcal{A}_{\mathsf{S}}(z)}^{\mathsf{R}}(s_0, s_1, 0)\}_{z\in\{0,1\}^*} \approx \{\mathsf{view}_{\Pi,\mathcal{A}_{\mathsf{S}}(z)}^{\mathsf{R}}(s_0, s_1, 1)\}_{z\in\{0,1\}^*}$$

where $\mathsf{view}_{\Pi,\mathcal{A}_{\mathsf{S}}(z)}^{\mathsf{R}}$ denotes the view of the adversary $\mathcal{A}_{\mathsf{S}}$ after a real execution of Π with the honest receiver R.

It is possible to instantiate the construction of one-sided simulateable OT from [CJS14] with our straight-line witness-extractable argument Π_{NIWI} to obtain a one-sided simulateable OT in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{ru}}$-hybrid model. The argument system is used as a black-box in their construction and no modification of the protocol or proof is necessary. The construction of [CJS14] is based on the efficient cut-and-choose OT protocol of Lindell and Pinkas [LP11] which provides a plain model instantiation of the two-message CRS-based OT protocol of Peikert et al. [PVW08]. This is achieved by replacing the CRS with an interactive protocol in which the receiver generates the parameters and provides a proof of correctness. The main difference between [LP11] and [CJS14] is the replacement of the zero-knowledge proof with a non-interactive witness-indistinguishable proof because this step is only needed for the simulation, i.e. the extraction, of the OT against a corrupted sender. The protocol thus retains its standalone receiver privacy while it is still simulateable against a corrupted receiver.

7.3.2.4 UC-secure NISC

In this section, we briefly describe how UC-secure non-interactive secure computation (NISC) can be achieved in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{ru}}$-hybrid model. Our solution is very

similar to the one of Canetti et al. [CJS14], hence we start with a description of their solution and highlight our changes. Due to the complexity of the protocol, we will not give a formal description and proof of the protocol and refer the interested reader to [CJS14].

Canetti et al. [CJS14] modify a UC-secure NISC protocol in the CRS-hybrid model by Ashfar et al. [AMPR14] such that it can be used with a global random oracle. In the following, we provide a very high-level description of their protocol. The basic approach is to squash a cut-and-choose garbled circuit protocol down to two messages, i.e. the sender provides many garbled circuits and the receiver can then verify that the sender garbled the correct circuit by examining approximately half of them. The rest of the circuits can be used to execute the actual computation.

In more detail, the receiver first specifies via OT (we call this instance the circuit-OT) which circuits he wants to check. Additionally, he creates another OT (called input-OT) to specify the labels he needs for his input. Here, [CJS14] use a two message one-sided simulatable OT based on their global random oracle (as compared to an OT in the CRS model like in [AMPR14]). Then, the sender starts to garble t circuits. All randomness for the garbling of each circuit C_i as well as for the respective input-OT is derived from a separate seed $seed_i$. In the original protocol of [AMPR14], this seed is chosen by the sender, while [CJS14] require the sender to query the random oracle with a message q_i to obtain a seed. The seed can be extracted separately and a full-fledged OT protocol is no longer necessary. The sender computes a set of commitments on his input, and then uses a key k_i to encrypt a message for each circuit that enables the receiver to check that the inputs in each of the circuits is consistent with the previously sent commitment. If the receiver were to learn both $seed_i$ and k_i, he could reconstruct the input of the sender and thus break the security of the protocol. Thus the sender inputs pairs $(k_i, seed_i)$ into the circuit-OT and that the receiver can either learn the inputs for the corresponding circuit or check its correctness, but not both. In addition, he inputs the input labels for the circuits into the input-OT. Once all this is done, he sends all OT messages, the garbled circuits and the commitments to the receiver, who can then check for correctness and evaluate the garbled circuit with his input. There are a lot of important details that we omit here, our intention is to focus only on the parts that are relevant for the changes we have to make to the protocol.

Our modifications to the above protocol are minor. First, we use our variant of one-sided simulatable OT in the $\mathcal{F}_{\text{wrap}}^{\text{ru}}$-hybrid model. Then, since there is no global random oracle available, we use a token programmed with a signature function to sign the query $seed_i$. However, the signature σ_i is not necessarily uniformly random, hence we do not require the sender to use the signature as a seed (as is done with the answer from the random oracle in [CJS14]). Instead, to ensure that the simulator can extract the seed, we require the sender to input $(k_i, (seed_i, \sigma_i))$ into the circuit-OT. The rest of the protocol is identical to [CJS14], and the proof has to be modified only marginally: the simulator against a corrupted sender obtains the seed $seed_i$ from $\mathcal{F}_{\text{wrap}}^{\text{ru}}$ if the protocol succeeds. In the proof of [CJS14], in the hybrid game $\mathcal{H}_1$, it is no longer necessary for a cheating sender to guess the answer of a random oracle to the query q_i, but instead he has to forge a signature on $seed_i$.

Model	computational		statistical	
	2PC	ni-2PC	2PC	ni-2PC
stateful	✓	✓	✓	✓
stateful (inc. com.)	✓	✓	✓	✓
stateful (outg. com.)	✓	✓	✗(✓)	✗
resettable	✓	✓	✗(✓)	✗
bounded-resettable	✓	✓	✓	(✓)
reusable resettable	✓(2 tokens in $\mathcal{F}_{\text{wrap}}^{\text{ru-strict}}$)	✓(2 tokens in $\mathcal{F}_{\text{wrap}}^{\text{ru}}$)	✗(✓) Commitments [DS13]	✗

Table 7.12: Feasibility of interactive and non-interactive two-party computation from reusable resettable tamper-proof hardware.

7.4 Relation to Two-Party Computation

In this chapter, we showed that real untrusted signature cards can be used to UC-realize very efficient protocols. Our results are achieved by adapting the protocols of [CJS14] from a global random oracle to the scenario of reusable resettable hardware. We basically achieved the same efficiency in our protocols as they do, but our setup assumption is untrusted, in contrast to the incorruptible global random oracle they used.

Computationally secure two-party computation: With our computationally UC-secure commitment from Section 7.2.2.2, a UC-secure CRS can be constructed via a Blum coin toss [Blu81], which in turn allows to UC-realize any two-party functionality [CLOS02]. This result is given in the strict model that allows for any signature card in practice.

Computationally secure non-interactive two-party computation: It is impossible to achieve non-interactive UC-secure computation with reusable tamper-proof hardware in the sense that only one message is sent in one direction, as we showed in Section 7.1. The notion of non-interactive secure computation (NISC) can be achieved, and we sketch how this protocol can be UC-realized with reusable resettable tokens in a less strict model that only allows certain types of signature tokens.

We did not consider statistically secure protocols in this chapter. It seems difficult to find a statistically secure functionality that is useful for many protocols, especially considering that apart from UC-secure commitments, most interesting functionalities are impossible to realize. Also, statistically secure protocols inherently allow only an a priori bounded number of uses, so the applicability is limited. Table 7.12 thus summarizes the state of the art.

8. Conclusion and Prospects

In this thesis, we investigated the cryptographic strength of tamper-proof hardware. We defined and studied several models of tamper-proof hardware, with the goal of investigating the feasibility of cryptographic primitives in these models. As Table 8.1 shows, we were able to give a full characterization of the cryptographic strength of the different hardware models concerning UC-secure (non-)interactive two-party computation.

We will shortly discuss our results and describe new research questions that arise from them.

Partially Isolated Stateful Tamper-Proof Hardware

In Chapter 4, we defined two weaker models of stateful hardware where the isolation assumption is suspended in one direction, i.e. either the sender can send messages to the token, or vice versa. We provided a full characterization of these models with respect to the feasibility of (non-)interactive two-party computation, and showed that our constructions are optimal with respect to the number of tokens that we need in the protocols.

A natural question is whether our results can also be obtained from one-way

	computational 2PC		statistical 2PC	
Model	interactive	non-interactive	interactive	non-interactive
stateful	✓	✓	✓	✓
stateful (inc. com.)	✓	✓	✓	✓
stateful (outg. com.)	✓	✓	✗(✓)	✗
resettable	✓	✓	✗(✓)	✗
bounded-resettable	✓	✓	✓	(✓)
reusable resettable	✓	✓	✗(✓)	✗

Table 8.1: Overview of our results.

functions. A construction of OTM based on one-way functions would have to be direct without using a CRS because all CRS-based UC-secure protocols for OT in the literature require number-theoretic assumptions or enhanced trapdoor permutations.

Also, it might be interesting to investigate the scenario of partial isolation with regard to resettable hardware tokens (cf. Chapter 5). We believe that non-interactive protocols should remain secure in the case of incoming communication, thus our result from Section 5.4 would directly yield non-interactive secure computation. However, our solution does not solve the case of outgoing communication.

Resettable Tamper-Proof Hardware

In Chapter 5, we improved upon previous work in the area of resettable tokens and provided optimal constructions for (non-)interactive two-party computation, both with respect to the number of tokens and the computational assumptions used.

While our constructions only assume one-way functions, which is optimal, the underlying two-party computation protocol requires OT. Currently, it seems as if OT in the plain model is a necessary assumption to obtain non-interactive two-party computation, mainly because OT is not a resettable functionality and thus cannot be realized non-interactively based on resettable hardware alone. One possible approach to resolve this question could be to find a construction for resettable two-party computation based on random OT only, because random OT is still realizable in the non-interactive setting with resettable hardware.

Bounded-Resettable Tamper-Proof Hardware

In the context of bounded-resettable tamper-proof hardware (cf. Chapter 6), we showed the general feasibility of two-party computation. In addition, we provided positive results for non-interactive protocols.

We use our commitments as building blocks for the more complex protocols, so there might be the possibility to obtain more efficient protocols by creating a direct construction of the corresponding functionalities. Efficiency improvements might both be obtained for the complexity of the protocols and for the number of tokens used. Likewise, it would be interesting to find lower bounds on the number of tokens necessary for a given functionality, because this would either show the optimality of our solutions or hint at more efficient constructions.

An interesting challenge is to find a solution for general non-interactive UC-secure two-party computation. We believe that the question can be resolved in the affirmative. In principle, OTM can be achieved by storing two values on a bounded-resettable token, which would imply non-interactive two-party computation. The receiver has to query the token at least $\left\lceil \frac{q}{2} \right\rceil$ times to obtain one value, and the bounded number of resets prevents the receiver from learning both values. Obviously, this construction is very inefficient because even an honest user has to "reset" the token very often, but it shows that non-interactive two-party computation cannot be ruled out in general.

Reusable Resettable Tamper-Proof Hardware

In Chapter 7 we introduced untrusted and resettable signature cards as a reusable setup assumption and showed that they even allow non-interactive two-party computation. One interesting research direction is to further analyze reusable stateful

tokens, because our results in Section 7.1 seem to indicate that reusable stateful and resettable tokens might be equal in their cryptographic strength.

A more technical question that arises by our work is whether it is possible to achieve non-interactive secure computation even with the more realistic definition of signatures with preprocessing on the host. In our current non-interactive protocols, the proofs of security will no longer hold with this definition because the token has to learn the complete view of the intermediate steps of the sender party of the non-interactive protocol.

Bibliography

[AAG+14] Shashank Agrawal, Prabhanjan Ananth, Vipul Goyal, Manoj Prabhakaran, and Alon Rosen. Lower bounds in the hardware token model. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 663–687. Springer, February 2014. (Cited on pages 66 and 124.)

[ADW09] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 36–54. Springer, August 2009. (Cited on page 25.)

[AL07] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 137–156. Springer, February 2007. (Cited on page 50.)

[AMPR14] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 387–404. Springer, May 2014. (Cited on page 145.)

[AS92] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs; A new characterization of NP. In *33rd FOCS*, pages 2–13. IEEE Computer Society Press, October 1992. (Cited on page 88.)

[BCG+11] Nir Bitansky, Ran Canetti, Shafi Goldwasser, Shai Halevi, Yael Tauman Kalai, and Guy N. Rothblum. Program obfuscation with leaky hardware. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 722–739. Springer, December 2011. (Cited on pages 25 and 85.)

[Bea95] Donald Beaver. Precomputing oblivious transfer. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 97–109. Springer, August 1995. (Cited on pages 32, 67, and 116.)

[BFSK11] Christina Brzuska, Marc Fischlin, Heike Schröder, and Stefan Katzenbeisser. Physically uncloneable functions in the universal composition framework. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 51–70. Springer, August 2011. (Cited on page 3.)

[BGGL01] Boaz Barak, Oded Goldreich, Shafi Goldwasser, and Yehuda Lindell. Resettably-sound zero-knowledge and its applications. In *42nd FOCS*,

pages 116–125. IEEE Computer Society Press, October 2001. (Cited on page 51.)

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, August 2001. (Cited on page 84.)

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):1–48, May 2012. (Cited on page 84.)

[Blu81] Manuel Blum. Coin flipping by telephone. In Allen Gersho, editor, *CRYPTO'81*, volume ECE Report 82-04, pages 11–15. U.C. Santa Barbara, Dept. of Elec. and Computer Eng., 1981. (Cited on pages 71 and 146.)

[BOGKW88] Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *20th ACM STOC*, pages 113–131. ACM Press, May 1988. (Cited on page 25.)

[BOV15] Ioana Boureanu, Miyako Ohkubo, and Serge Vaudenay. The limits of composable crypto with transferable setup devices. In Feng Bao, Steven Miller, Jianying Zhou, and Gail-Joon Ahn, editors, *ASIACCS 15*, pages 381–392. ACM Press, April 2015. (Cited on page 123.)

[BP13] Nir Bitansky and Omer Paneth. On the impossibility of approximate obfuscation and applications to resettable cryptography. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 241–250. ACM Press, June 2013. (Cited on page 15.)

[Bra94] Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 302–318. Springer, August 1994. (Cited on page 2.)

[BVS06] Jens-Matthias Bohli, Maria Isabel Gonzalez Vasco, and Rainer Steinwandt. A subliminal-free variant of ECDSA. In Jan Camenisch, Christian S. Collberg, Neil F. Johnson, and Phil Sallee, editors, *Information Hiding IH 2006*, volume 4437 of *LNCS*, pages 375–387. Springer, July 2006. (Cited on page 131.)

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001. (Cited on pages 2, 17, and 18.)

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 61–85. Springer, February 2007. (Cited on pages 123 and 124.)

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, August 1994. (Cited on page 143.)

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, August 2001. (Cited on pages 2, 26, 27, 39, and 40.)

[CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *32nd ACM STOC*, pages 235–244. ACM Press, May 2000. (Cited on pages 1, 69, and 71.)

[CGS08] Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for UC secure computation using tamper-proof hardware. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 545–562. Springer, April 2008. (Cited on pages 2, 49, 50, 51, 52, 67, 85, and 123.)

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 597–608. ACM Press, November 2014. (Cited on pages 10, 124, 125, 129, 134, 138, 144, 145, and 146.)

[CKM11] Seung Geol Choi, Aggelos Kiayias, and Tal Malkin. BiTR: Built-in tamper resilience. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 740–758. Springer, December 2011. (Cited on page 25.)

[CKS+14] Seung Geol Choi, Jonathan Katz, Dominique Schröder, Arkady Yerukhimovich, and Hong-Sheng Zhou. (Efficient) universally composable oblivious transfer using a minimal number of stateless tokens. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 638–662. Springer, February 2014. (Cited on pages 50, 51, 55, 63, 67, 123, 125, 126, and 129.)

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002. (Cited on pages 2, 21, 67, 69, 85, 133, and 146.)

[CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, August 1993. (Cited on pages 1 and 2.)

[CP94] Ronald Cramer and Torben P. Pedersen. Improved privacy in wallets with observers (extended abstract). In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 329–343. Springer, May 1994. (Cited on page 2.)

[CPS13] Kai-Min Chung, Rafael Pass, and Karn Seth. Non-black-box simulation from one-way functions and applications to resettable security. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 231–240. ACM Press, June 2013. (Cited on pages 13, 15, and 66.)

[CS09] Mahdi Cheraghchi and Amin Shokrollahi. Almost-uniform sampling of points on high-dimensional algebraic varieties. In Susanne Albers and Jean-Yves Marion, editors, *Symposium on Theoretical Aspects of Computer Science - Proceedings of STACS 2009*, volume 3 of *LIPIcs*, pages 277–288. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2009. (Cited on pages 104, 105, 106, and 107.)

[Dam90] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 416–427. Springer, August 1990. (Cited on page 127.)

[DFK+93] Cynthia Dwork, Uriel Feige, Joe Kilian, Moni Naor, and Shmuel Safra. Low communication 2-prover zero-knowledge proofs for NP. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 215–227. Springer, August 1993. (Cited on page 88.)

[DFK+14] Dana Dachman-Soled, Nils Fleischhacker, Jonathan Katz, Anna Lysyanskaya, and Dominique Schröder. Feasibility and infeasibility of secure computation with malicious PUFs. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 405–420. Springer, August 2014. (Cited on page 3.)

[DKMN15] Nico Döttling, Daniel Kraschewski, Jörn Müller-Quade, and Tobias Nilges. General statistically secure computation with bounded-resettable hardware tokens. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 319–344. Springer, March 2015. (Cited on pages 5 and 89.)

[DKMQ11] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 164–181. Springer, March 2011. (Cited on pages 2, 23, 27, 28, 29, 30, 31, 32, 37, 47, 49, 50, 67, 85, 88, 91, 115, and 116.)

[DKMQ12] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. David & Goliath oblivious affine function evaluation - asymptotically optimal building blocks for universally composable two-party computation from a single untrusted stateful tamper-proof hardware token. Cryptology ePrint Archive, Report 2012/135, 2012. `http://eprint.iacr.org/2012/135`. (Cited on pages 25 and 67.)

[DKMQN15] Nico Döttling, Daniel Kraschewski, Jörn Müller-Quade, and Tobias Nilges. From stateful hardware to resettable hardware using symmetric assumptions. In *ProvSec 2015*, LNCS. Springer, November 2015. To appear. (Cited on pages 4 and 50.)

[DMMQN11] Nico Döttling, Thilo Mie, Jörn Müller-Quade, and Tobias Nilges. Basing obfuscation on simple tamper-proof hardware assumptions. Cryptology ePrint Archive, Report 2011/675, 2011. `http://eprint.iacr.org/2011/675`. (Cited on page 85.)

[DMMQN13] Nico Döttling, Thilo Mie, Jörn Müller-Quade, and Tobias Nilges. Implementing resettable UC-functionalities with untrusted tamper-proof hardware-tokens. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 642–661. Springer, March 2013. (Cited on pages 4 and 50.)

[DMQN15] Rafael Dowsley, Jörn Müller-Quade, and Tobias Nilges. Weakening the isolation assumption of tamper-proof hardware tokens. In Anja Lehmann and Stefan Wolf, editors, *ICITS 15*, volume 9063 of *LNCS*, pages 197–213. Springer, May 2015. (Cited on pages 4 and 26.)

[DNW08] Ivan Damgård, Jesper Buus Nielsen, and Daniel Wichs. Isolated proofs of knowledge and isolated zero knowledge. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 509–526. Springer, April 2008. (Cited on page 25.)

[DNW09] Ivan Damgård, Jesper Buus Nielsen, and Daniel Wichs. Universally composable multiparty computation with partially isolated parties. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 315–331. Springer, March 2009. (Cited on pages 22, 25, and 26.)

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008. (Cited on pages 16 and 17.)

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *49th FOCS*, pages 293–302. IEEE Computer Society Press, October 2008. (Cited on pages 2 and 25.)

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, August 2012. (Cited on page 3.)

[DS13] Ivan Damgård and Alessandra Scafuro. Unconditionally secure and universally composable commitments from physical assumptions. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 100–119. Springer, December 2013. (Cited on pages 49, 50, 54, 86, 87, 123, 133, 137, 138, and 146.)

[Dwo06] Cynthia Dwork. Differential privacy (invited paper). In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP 2006, Part II*, volume 4052 of *LNCS*, pages 1–12. Springer, July 2006. (Cited on page 69.)

[FGL+91] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Approximating clique is almost NP-complete (preliminary version). In *32nd FOCS*, pages 2–12. IEEE Computer Society Press, October 1991. (Cited on page 88.)

[FJN+13] Tore Kasper Frederiksen, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. MiniLEGO: Efficient secure two-party computation from general assumptions. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 537–556. Springer, May 2013. (Cited on page 3.)

[FPS+11] Marc Fischlin, Benny Pinkas, Ahmad-Reza Sadeghi, Thomas Schneider, and Ivan Visconti. Secure set intersection with untrusted hardware tokens. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 1–16. Springer, February 2011. (Cited on page 22.)

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, August 1987. (Cited on pages 125 and 140.)

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *22nd ACM STOC*, pages 416–426. ACM Press, May 1990. (Cited on page 143.)

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, August 1986. (Cited on page 9.)

[GIMS10] Vipul Goyal, Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. Interactive locking, zero-knowledge pcps, and unconditional cryptography. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 173–190. Springer, August 2010. (Cited on pages 23, 49, 50, 53, 54, 67, 87, 88, 90, 102, 110, 114, 120, and 126.)

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 308–326. Springer, February 2010. (Cited on pages 2, 4, 23, 50, 51, 52, 63, 67, 85, 123, and 124.)

[GK96a] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–190, 1996. (Cited on page 117.)

[GK96b] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25(1):169–192, 1996. (Cited on pages 2 and 17.)

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In David Wagner, editor, *CRYPTO 2008*, volume 5157

of *LNCS*, pages 39–56. Springer, August 2008. (Cited on pages 3, 22, and 26.)

[GLM+04] Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 258–277. Springer, February 2004. (Cited on page 2.)

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 251–261. Springer, May 2001. (Cited on pages 2 and 25.)

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987. (Cited on pages 3, 63, and 69.)

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, May 1996. (Cited on page 1.)

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001. (Cited on page 10.)

[GS09] Vipul Goyal and Amit Sahai. Resettably secure computation. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 54–71. Springer, April 2009. (Cited on pages 68, 69, 71, and 85.)

[HIKN08] Danny Harnik, Yuval Ishai, Eyal Kushilevitz, and Jesper Buus Nielsen. OT-combiners via secure computation. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 393–411. Springer, March 2008. (Cited on pages 67 and 115.)

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999. (Cited on page 8.)

[HKN+05] Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. On robust combiners for oblivious transfer and other primitives. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 96–113. Springer, May 2005. (Cited on page 115.)

[HL08] Carmit Hazay and Yehuda Lindell. Constructions of truly practical secure protocols using standardsmartcards. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 08*, pages 491–500. ACM Press, October 2008. (Cited on pages 3 and 22.)

[HMQU05] Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. Universally composable zero-knowledge arguments and commitments from

signature cards. In *Proceedings of the 5th Central European Conference on Cryptology MoraviaCrypt 2005*, 2005. (Cited on pages 2, 123, 124, 125, 127, 129, 137, and 138.)

[HPV15] Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Round-optimal token-based secure computation. Cryptology ePrint Archive, Report 2015/887, 2015. `http://eprint.iacr.org/2015/887`. (Cited on pages 50, 51, and 85.)

[HRVW09] Iftach Haitner, Omer Reingold, Salil P. Vadhan, and Hoeteck Wee. Inaccessible entropy. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 611–620. ACM Press, May / June 2009. (Cited on page 54.)

[IKO+11] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 406–425. Springer, May 2011. (Cited on page 111.)

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007. (Cited on page 115.)

[IKOS09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Extracting correlations. In *50th FOCS*, pages 261–270. IEEE Computer Society Press, October 2009. (Cited on page 115.)

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, August 2008. (Cited on pages 37, 63, 67, 85, 115, 116, 120, and 124.)

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, August 2003. (Cited on page 2.)

[Jag15] Tibor Jager. Verifiable random functions from weaker assumptions. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 121–143. Springer, March 2015. (Cited on page 51.)

[JK03] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447 (Informational), February 2003. (Cited on page 126.)

[JKSS10] Kimmo Järvinen, Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Embedded SFE: Offloading server and network using hardware tokens. In Radu Sion, editor, *FC 2010*, volume 6052 of *LNCS*, pages 207–221. Springer, January 2010. (Cited on page 22.)

[Kat07] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 115–128. Springer, May 2007. (Cited on pages 2, 3, 21, and 22.)

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31. ACM Press, May 1988. (Cited on pages 37, 85, and 120.)

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, August 1999. (Cited on pages 2 and 25.)

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, August 1996. (Cited on pages 2 and 25.)

[Kol10] Vladimir Kolesnikov. Truly efficient string oblivious transfer using resettable tamper-proof tokens. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 327–342. Springer, February 2010. (Cited on pages 50 and 51.)

[KPT97] Joe Kilian, Erez Petrank, and Gábor Tardos. Probabilistically checkable proofs with zero knowledge. In *29th ACM STOC*, pages 496–505. ACM Press, May 1997. (Cited on page 88.)

[KR08] Yael Tauman Kalai and Ran Raz. Interactive PCP. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 536–547. Springer, July 2008. (Cited on page 88.)

[Lab04] RSA Laboratories. PKCS #11 v2.20: Cryptographic Token Interface Standard, 2004. (Cited on page 126.)

[LP11] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 329–346. Springer, March 2011. (Cited on page 144.)

[LPSY15] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, LNCS, pages 319–338. Springer, August 2015. (Cited on page 3.)

[LR14] Yehuda Lindell and Ben Riva. Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 476–494. Springer, August 2014. (Cited on page 3.)

[Lys02] Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 597–612. Springer, August 2002. (Cited on pages 12 and 51.)

[Mer79] Ralph Charles Merkle. *Secrecy, Authentication, and Public Key Systems.* PhD thesis, Stanford University, Stanford, CA, USA, 1979. (Cited on page 127.)

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 369–378. Springer, August 1988. (Cited on page 66.)

[MOSV06] Daniele Micciancio, Shien Jin Ong, Amit Sahai, and Salil P. Vadhan. Concurrent zero knowledge without complexity assumptions. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 1–20. Springer, March 2006. (Cited on page 10.)

[MPW07] Remo Meier, Bartosz Przydatek, and Jürg Wullschleger. Robuster combiners for oblivious transfer. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 404–418. Springer, February 2007. (Cited on page 115.)

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 278–296. Springer, February 2004. (Cited on pages 2 and 25.)

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th FOCS*, pages 120–130. IEEE Computer Society Press, October 1999. (Cited on page 51.)

[MS08] Tal Moran and Gil Segev. David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 527–544. Springer, April 2008. (Cited on pages 2, 22, and 23.)

[Nao90] Moni Naor. Bit commitment using pseudo-randomness. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 128–136. Springer, August 1990. (Cited on pages 10 and 13.)

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, August 2012. (Cited on page 67.)

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *21st ACM STOC*, pages 33–43. ACM Press, May 1989. (Cited on page 12.)

[OSVW13] Rafail Ostrovsky, Alessandra Scafuro, Ivan Visconti, and Akshay Wadia. Universally composable secure computation with (malicious) physically uncloneable functions. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 702–718. Springer, May 2013. (Cited on page 3.)

[Pap01] Ravikanth Srinivasa Pappu. *Physical One-way Functions*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2001. (Cited on page 1.)

[Pas03] Rafael Pass. On deniability in the common reference string and random oracle model. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 316–337. Springer, August 2003. (Cited on pages 140, 142, and 143.)

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, August 1992. (Cited on page 11.)

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *43rd FOCS*, pages 366–375. IEEE Computer Society Press, November 2002. (Cited on page 69.)

[PSW14] Manoj Prabhakaran, Amit Sahai, and Akshay Wadia. Secure computation using leaky tokens. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP 2014, Part I*, volume 8572 of *LNCS*, pages 907–918. Springer, July 2014. (Cited on page 25.)

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, August 2008. (Cited on pages 27, 34, 67, and 144.)

[PW09] Rafael Pass and Hoeteck Wee. Black-box constructions of two-party protocols from one-way functions. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 403–418. Springer, March 2009. (Cited on page 10.)

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *22nd ACM STOC*, pages 387–394. ACM Press, May 1990. (Cited on page 12.)

[RvD13] Ulrich Rührmair and Marten van Dijk. PUFs in security protocols: Attack models and security evaluations. In *2013 IEEE Symposium on Security and Privacy*, pages 286–300. IEEE Computer Society Press, May 2013. (Cited on page 25.)

[SA03] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar,

editors, *CHES 2002*, volume 2523 of *LNCS*, pages 2–12. Springer, August 2003. (Cited on pages 2 and 25.)

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979. (Cited on pages 9, 41, and 42.)

[WC81] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22:265–279, 1981. (Cited on page 11.)

[WW06] Stefan Wolf and Jürg Wullschleger. Oblivious transfer is symmetric. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 222–232. Springer, May / June 2006. (Cited on page 116.)

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982. (Cited on pages 3 and 63.)