Universally Composable (Non-Interactive) Two-Party Computation from Untrusted Reusable Hardware Tokens*

Jeremias Mechler[1], Jörn Müller-Quade[1], and Tobias Nilges[2]

[1]Karlsruhe Institute of Technology, Germany
[2]Aarhus University, Denmark

Abstract

Universally composable protocols provide security even in highly complex environments like the Internet. Without setup assumptions, however, UC-secure realizations of cryptographic tasks are impossible. To achieve efficient protocols, practical setup assumptions are needed. Tamper-proof hardware tokens, e.g. smart cards and USB tokens, can be used for this purpose. Apart from the fact that they are widely available, they are also cheap to manufacture and well understood.

However, currently considered protocols based on tamper-proof hardware require a protocol-specific functionality of the hardware which cannot be reused for other protocols. For this to become possible, in addition to a versatile functionality, the hardware has to be modeled as a global setup.

We propose the first formalization of tamper-proof hardware as an *untrusted* global setup assumption. Based on this setup, we construct protocols for both UC-secure two-party computation and UC-secure non-interactive secure computation. The token functionality that we choose is a simple signature functionality, i.e. our protocols can be realized with currently available signature cards.

Keywords: universal composability, tamper-proof hardware, signatures

1 Introduction

In 2001 Canetti [Can01] proposed the *Universal Composability (UC)* framework. Protocols proven secure in this framework have strong security guarantees for protocol composition, i.e. the parallel or interleaved execution of protocols. Subsequently, it was shown that it is not possible to construct protocols in this strict framework without additional assumptions [CF01]. One of these so-called setup assumptions was put forward by Katz [Kat07], who proposed that protocol parties can exchange untrusted tamper-proof hardware tokens. All previously considered setups (e.g. a Public Key Infrastructure or a Common Reference String (CRS)) required a trusted setup, while in this scenario, the receiver of a hardware token does not trust its functionality. This spawned a line of research that

*The results presented in this paper previously appeared as part of [Nil15]

focuses mainly on the feasibility of UC-secure two-party computation. First, stateful tamper-proof hardware was considered [Kat07, MS08, GKR08, DNW09, GIS$^+$10, DKMQ11], then weaker models of tamper-proof hardware, where the hardware token cannot reliably keep a state, i.e. the receiver can reset the token [CGS08, Kol10, GIS$^+$10, GIMS10, DS13, DMMQN13, DKMN15, DKMQN15].

A recent line of research, e.g. [CJS14, BOV15, CSV16], has focused on efficient protocols based on a globally available setup. This stronger notion of UC security, called *Global UC (GUC)*, was introduced by Canetti et al. [CDPW07] and captures the fact that protocols are more efficient if they can use the same setup. Indeed, one drawback of all the aforementioned results is that the protocols require a specific token functionality for each protocol, i.e. the token can at best be used for several instances of the same protocol. Worse still, the security of most of these protocols would break down if the same tokens were used by other protocols. This also has several negative implications if one wanted 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 will be implemented in hardware. For currently available smart cards, however, the protocols are too complex to be realized efficiently enough in software on these cards to be applicable in practice.

To the best of our knowledge, reusable tamper-proof hardware was only considered by Hofheinz et al. [HMQU05], who introduce *catalysts* as a concept similar to GUC setups. In their model, they show that the card can be reused for multiple protocols, unlike a normal UC setup. However, they assume a *trusted* signature card as a UC setup.

A natural question that arises given the current state of the art is the following:

Is it possible to define tamper-proof hardware as a reusable setup?

Assuming such a setup, there is also the problem of protocol-specific token functionalities. If each protocol still needs a specific token program, reusability will not improve the state of affairs. Therefore, we also have to answer the following question:

Which functionality is versatile enough to be used in many different protocols?

In the following, we shall provide answers to both questions.

Our contribution. We apply the GUC methodology to resettable tamper-proof hardware, because a reusable tamper-proof hardware token formally translates into a "global" setup in the GUC setting. The token wrapper functionality that we define is the first reusable or global setup that is *untrusted*, in contrast to *trusted and incorruptible* setups like a global random oracle [CJS14], augmented CRS or key registration with knowledge [CDPW07].

In a little more detail, we present *two* models for reusable tamper-proof hardware:

- A very realistic model inspired by [HMQU05], which generalizes their approach to generic and untrusted resettable tokens. This model puts some restrictions on the feasibility of cryptographic protocols. Nevertheless, we show a UC-secure protocol for commitments, which implies general UC-secure two-party computation from untrusted reusable tokens.
- A more idealized model following the approach of [CJS14], which is meant to explore the general feasibility of protocols with reusable tamper-proof hardware. We show that untrusted reusable and resettable tokens allow for UC-secure non-interactive secure computation (NISC) in this model. Basically, we obtain the same result as [CJS14], but instead of using a global random oracle, our protocols use untrusted reusable hardware tokens.

The difference between these two models stems from the way in which we allow access to the tokens by the environment. In the realistic model, only a single protocol can have access to the token at once. A real world analogy could be an ATM that seizes the card for the duration of the cash withdrawal. During that time, the card cannot be used to sign a document. Further, we use a nonce to bind the messages from the protocol parties to the current protocol execution. This implies that each protocol must include communication between the parties (in both directions).

For the idealized model we take a different approach, taken from [CJS14]. The simulator is given access to all "illegitimate" queries that are made to the token, so that all queries concerning a protocol (identified by a process ID PID), even from other protocols, can be observed. This removes the additional interaction from the protocols and allows NISC.

For our results, we focus only on resettable hardware tokens. We show that this is imposed by the model, because stateful hardware tokens do not yield benefits regarding the number of tokens that have to be exchanged. Additionally, we show that it is inherent for UC-security that both parties have to exchange tokens, i.e. sending tokens in one direction only is not sufficient for UC-secure protocols with reusable hardware tokens.

We instantiate our tokens with a signature functionality, because we believe that such a functionality is very versatile and has applications in a wide range of cryptographic protocols. In the literature, several works discuss bit-oblivious transfer (OT) tokens as a very simple and cheap functionality [IPS08, GIS$^+$10, AAG$^+$14]. However, there are no standardized implementations of such tokens, while signature tokens are usually standardized and widely available. The problem with using these tokens for cryptographic protocols is that sometimes part of the computation of the signature (usually the hashing) is outsourced to the host. The formalization of signature cards assumed by [HMQU05] in contrast does not take this problem into account and cannot be instantiated with most of the currently available signature tokens, even if they are trusted (which they assume). In fact, we argue that certain types of tokens are not suitable for UC-secure protocols at all. To circumvent this problem, we present an exact definition of signature schemes that can be used for our protocols by generalizing the notion of a signature scheme to incorporate the preprocessing explicitly.

Our techniques. We adapt an approach used by Choi et al. [CKS$^+$14] for our signature tokens. One of the main difficulties when trying to achieve UC security with hardware tokens is to make sure that the tokens cannot behave maliciously. In our case, this would mean that we have to verify that the signature was created correctly. Usually, e.g. in [HMQU05, DMMQN13], this is done via zero-knowledge proofs of knowledge, but the generic constructions that are available are highly inefficient. Instead, similar to [CKS$^+$14], we use unique signatures. Unique signatures allow verification of the signature, but they also guarantee that the signature is subliminal-free, i.e. a malicious token cannot encode messages into the signature.

Based on tokens with this unique signature functionality, we construct a straight-line extractable commitment in the realistic model. The main idea is to send the message to the token and obtain a signature on it. The simulator can observe this message and extract it. Due to the aforementioned partitioning of the signature algorithm, however, the simulator might only learn a hash value, which makes extraction impossible. We thus have to modify this approach a bit to make it work in our setting. Based on this commitment, we can modify the UC commitment of [CJS14] to obtain UC-secure two-party computation from general feasibility results [CLOS02].

In the idealized model, we show a simple modification of our straight-line extractable commitment into a non-interactive one. From this commitment we construct a non-interactive witness-

indistinguishable argument-of-knowledge that uses our signature token, similar to the protocol of Pass [Pas03] in the random oracle model. Here, we replace the Fiat-Shamir heuristic [FS87], i.e. the call to the random oracle, with a call to the signature token and show that a cheating prover must be able to forge a signature. Basically, we can just plug this primitive in the NISC construction of [CJS14] and obtain efficient and UC-secure NISC.

Related work. In an independent work, using an analogous approach based on [CJS14], Hazay et al. [HPV15] recently introduced a GUC-variant of tamper-proof hardware to deal with the problem of adversarial token transfers in the multi-party case, which contrary to us they explicitly model in the ideal functionality. An adversary playing man-in-the-middle might obtain a token and give the environment direct access to the token. This can render protocols based on tamper-proof hardware insecure, because the simulator can no longer observe all queries. Nevertheless, both of our token functionalities and the resulting protocols are also robust in their setting, because reusability of the tokens necessarily requires security against this type of attack.

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), 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 paper on the other hand is not a transferable setup, so their impossibilities do not apply.

2 Preliminaries

In this section, we introduce the security model and the basic primitives used throughout the paper.

2.1 UC framework

We show our results in the generalized UC framework (GUC) of Canetti et al. [CDPW07]. Let us first briefly describe the basic UC framework [Can01], and then highlight the changes required for GUC. In UC, the security of a *real* protocol π is shown by comparing it to an *ideal* functionality $\mathcal{F}$. The ideal functionality is incorruptible and secure by definition. The protocol π is said to realize $\mathcal{F}$, if for any adversary $\mathcal{A}$ in the real protocol, there exists a simulator $\mathcal{S}$ in the ideal model that mimics the behavior of $\mathcal{A}$ in such a way that any environment $\mathcal{Z}$, which is plugged either to the ideal or the real model, cannot distinguish both.

In UC, the environment $\mathcal{Z}$ cannot run several protocols that share a state, e.g. via the same setup. In GUC, this restriction is removed. In particular, $\mathcal{Z}$ can query the setup independently of the current protocol execution, i.e. the simulator will not observe this query.

We will realize a UC-secure commitment. The ideal functionality $\mathcal{F}_{\text{COM}}$ is defined in Figure 1.

2.2 Pseudorandom Functions

Definition 1. *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* $\{h : \{0,1\}^n \to \{0,1\}^m\}$.

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 $\texttt{commit}$-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 1: Ideal functionality for commitments.

2.3 Commitments

We need several types of commitment schemes. 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. *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*:*

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

The binding- and hiding-properties are defined as follows:

Definition 3. *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 4. *We say that* $\mathsf{COM} = (\mathsf{Commit}, \mathsf{Open})$ *is* statistically binding *if for every algorithm* $\mathcal{A}_{\mathsf{S}}$*:*

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

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 5. *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 6. *We say that* $\mathsf{COM} = (\mathsf{COM.Commit}, \mathsf{COM.Open})$ *is* straight-line extractable *if in addition to Definition 5, 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 7. *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 Witness-Indistinguishability

We construct a witness-indistiguishable argument of knowledge in this paper.

Definition 8. *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 9. *We say that a witness-indistinguishable argument/proof system is* straight-line witness-extractable *if in addition to Definition 8, the extractor does not use rewinding.*

2.5 Digital Signatures

Digital signatures allow to compute an unforgeable message digest. 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, given message and signature.

Definition 10. *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 message attack* (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).

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 11. *Let* SIG *be a digital signature scheme. A signature scheme is called* unique *if additionally to the properties of Definition 10 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.

3 Limitations

It is known that there exist limitations regarding the feasibility of UC-secure protocols based on resettable tamper-proof hardware, both with computational and with statistical security. Concerning statistical security, Goyal et al. [GIMS10] show that non-interactive commitments and OT cannot be realized from resettable tamper-proof hardware tokens, even with standalone security. In the computational setting, Döttling et al. [DMMQN13] and Choi et al. [CKS$^+$14] show that if (any number of) tokens are sent only in one direction, i.e. are not exchanged by both parties, it is impossible to realize UC-secure protocols without using non-black-box techniques. Intuitively, this follows from the fact that the simulator does not have any additional leverage over a malicious receiver of such a token. Thus, a successful simulator strategy could be applied by a malicious receiver as well. The above mentioned results apply to our scenario as well.

Jumping ahead, the impossibilities stated next 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 12. *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* $\mathcal{F}_{\mathrm{PF}}$.

Sketch. 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 [DMMQN13] 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 13. *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 $\mathcal{F}_{\mathrm{OT}}$.*

Sketch. 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, which is the one advantage that he has over the adversary. The simulator falls back to observing the input/output behavior of the token, exactly as in the case of standard resettable hardware. Due to the impossibility of statistically secure OT based on resettable hardware [GIMS10], the claim follows. □

4 Real Signature Tokens

It is our objective to instantiate the token functionality with a signature scheme. In order to allow currently available signature tokens to be used with our protocol, our formalization of a generalized signature scheme must take the peculiarities of real tokens into account.

One of the most important aspects regarding signature tokens is the fact that most tokens split the actual signing process into two parts: the first step is a preprocessing that usually computes a digest of the message. To improve efficiency, some tokens require this step to be done on the host system, at least in part. In a second step, this digest is signed on the token using the encapsulated signing key. In our case, this means that the *adversary contributes to computing the signature*. This has severe implications regarding the extraction in UC-secure protocols, because it is usually assumed that the simulator can extract the input from observing the query to the token.

To illustrate the problem, imagine a signature token that executes textbook RSA, and requires the host to compute the hash. A malicious host can *blind* his real input due to the homomorphic properties of RSA. Let (e, N) be the verification key and d the signature key for the RSA function. The adversary chooses a message m and computes the hash value $h(m)$ under the hash function h. Instead of sending $h(m)$ directly to the signature token, he chooses a random r, computes $h(m)' = h(m) \cdot r^e \mod N$ and sends $h(m)'$ to the token. The signature token computes $\sigma' = (h(m) \cdot r^e)^d = h(m)^d \cdot r \mod N$ and sends it to the adversary, who can multiply σ' by r^{-1} and obtain a valid signature σ on m. Obviously, demanding EUF-CMA for the signature scheme is not enough, because the signature is valid and the simulator is not able to extract m.

The protocols of [HMQU05] will be rendered insecure if the tokens perform *any kind* of preprocessing outside of the token, so the protocols cannot be realized with most of the currently available signature tokens (even if they are trusted). We aim to find an exact definition of the requirements, so that tokens which outsource part of the preprocessing can still be used in protocols. The following definition of a signature scheme with preprocessing thus covers a large class of currently available signature tokens and corresponding standards.

All protocols in the following sections can be instantiated with currently available signature tokens that adhere the definition above. Tokens that completely outsource the computation of the

message digest to the host do not satisfy this definition (because KeyGen, Sign and Vfy are not EUF-CMA secure). However, there is one additional caveat: current signature tokens usually do not implement unique signature schemes. Therefore, currently available tokens can only be used with our protocols if the receiver trusts the token.

Commonly used algorithms for message signing make are based on cryptographic assumptions 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.
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 maliciously specified mode.

4.1 Model

Our definition of reusable resettable tamper-proof hardware is defined analogously to normal resettable tamper-proof hardware tokens as in [GIS+10, DMMQN13], but we add a mechanism that allows a protocol party to seize the hardware token. This approach is inspired by the work of Hofheinz et al. [HMQU05]. While the token is seized, no other (sub-)protocol can use it. 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 2.

We assume that the token receiver can verify that it obtained the correct token, e.g. by requesting some token identifier from the sender.

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 2: 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 paper.

Definition 14. *(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 preprocessed message* p *of fixed length. It outputs a signature* σ *on the message* p.
- $\mathsf{Vfy}(\mathsf{vk}, p, \sigma)$ *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.*
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$ *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.*

We assume that the scheme is correct, i.e. it must hold for all messages m *that*

$$\forall \kappa \in \mathbb{N}\ \forall (\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathsf{KeyGen}(\kappa) : \mathsf{Verify}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sgk}, \mathsf{PreSg}(\mathsf{vk}, m))) = 1.$$

Additionally, we require uniqueness according to Definition 11.

Existential unforgeability can be defined analogously to the definition for normal signature schemes. However, the EUF-CMA property has to hold for both $\mathsf{KeyGen}, \mathsf{Sign}$ and Vfy and $\mathsf{KeyGen}, \mathsf{Sign}$ and Verify. The PreSg algorithm is typically realized as a hash function.

4.2 UC-Secure Two-Party Computation

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 scheme in Section 4.2.1. We then use this commitment in an adaption of a UC-secure commitment by Canetti et al. [CJS14], as shown in Section 4.2.2.

4.2.1 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 due to Hofheinz et al. [HMQU05] which assumes trusted signature tokens as a setup such that it remains secure even with maliciously created signature tokens. Towards this goal, we adapt the idea of Choi et al. [CKS$^+$14] to use unique signatures to our scenario. This is necessary, because verifying the functionality of an untrusted token is difficult. A unique signature scheme allows this verification very efficiently (compared to other measures such as e.g. resettably-sound ZK proofs). 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. Both values are concatenated with the nonce, preprocessed, 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 (PRG) 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 3. We abuse the notation in that we define $(c, d) \leftarrow \mathsf{COM.Commit}(x)$ to denote that the commitment c was created with randomness d.

Theorem 1. *The protocol $\Pi_{\text{COM}}^{\text{se}}$ in Figure 3 is a straight-line extractable commitment scheme as per Definition 6 in the $\mathcal{F}_{\text{wrap}}^{\text{ru-strict}}$-hybrid model, given that unique signatures exist.*

Very briefly, extractability follows from the fact that the extractor can see all messages that were sent to the token, including the randomness that allows to extract the commitments c_s and c_x. Therefore, the extractor can just search through all messages that were sent until it finds the inputs that match the commitment values. Hiding follows from the hiding property of the commitments and the pseudorandomness of the PRG. The randomness extraction with the 2-universal hash function prevents the token from leaking any information that might allow a receiver to learn some parts of the randomness of the commitments.

We split the proof into two lemmata, showing the computational hiding property of $\Pi_{\text{COM}}^{\text{se}}$ in Lemma 15 and the straight-line extraction in Lemma 16.

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 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 linear 2-universal hash function f from the family of linear 2-universal hash functions $\{f_h : \{0,1\}^{3\kappa} \to \{0,1\}^{\kappa}\}_{h \leftarrow \mathcal{H}}$.
   - 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 $\mathsf{SIG.Verify}(\mathsf{vk}, \sigma_x, (x, N)) = \mathsf{SIG.Verify}(\mathsf{vk}, \sigma_r, (r, N)) = 1$. Additionally, check if $\mathsf{COM.Open}(c_x, f(p_r), x) = 1$, $\mathsf{COM.Open}(c_\sigma, d_\sigma, (\sigma_x, \sigma_r, x, r, N)) = 1$ and $c_s \leftarrow \mathsf{PRG}(f(p_x)) \oplus s$. If not, abort; otherwise accept.

Figure 3: Computationally secure straight-line extractable commitment scheme in the $\mathcal{F}^{\text{ru-strict}}_{\text{wrap}}$-hybrid model.

Lemma 15. *The protocol $\Pi^{\text{se}}_{\text{COM}}$ in Figure 3 is computationally hiding, given that COM is an extractable computationally hiding commitment scheme, f is a linear 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 linear 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, [DORS08] gives a lower bound for the average min-entropy of X under Y, namely

$$\tilde{H}_\infty(X|Y) \geq H_\infty(X) - H_\infty(Y) = 3\kappa - 1.$$

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. [DORS08]):

$$\Delta((f_{\mathcal{H}}(X), H, Y); (U_k, H, Y)) \leq \frac{1}{2}\sqrt{2^{\tilde{H}_\infty(X|Y)} 2^{\kappa}} \leq \frac{1}{2}\sqrt{2^{-(3\kappa-1)+\kappa}} \leq 2^{-\kappa}$$

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 16. *The protocol $\Pi^{\text{se}}_{\text{COM}}$ in Figure 3 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 4. It searches the inputs of $\mathcal{A}_\mathsf{S}$ into the hybrid functionality $\mathcal{F}_{\text{wrap}}^{\text{ru-strict}}$ for the combination of input and randomness for the commitment that is to be extracted.

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

Upon input $c^* = \big((c_s^*, c_x^*, c_\sigma^*, f^*), Q\big)$, 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 $\mathsf{COM.Open}(c_x^*, f^*(\alpha_j), \alpha_i) = 1$. 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 4: The extraction algorithm for the straight-line extractable commitment protocol $\Pi^{\text{se}}_{\text{COM}}$.

Let Q denote the set of inputs that $\mathcal{A}_\mathsf{S}$ sent to $\mathcal{F}_{\text{wrap}}^{\text{ru-strict}}$. 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 $\mathcal{F}_{\text{wrap}}^{\text{ru-strict}}$ 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. □

4.2.2 Obtaining UC-Secure Commitments

In order to achieve computationally secure two-party computation, we want to transform the straight-line extractable commitment from Section 4.2.1 into a UC-secure commitment. A UC-secure commitment can be used to create a UC-secure CRS via a coin-toss (e.g. [DMMQN13]). General feasibility results, e.g. [CLOS02], then imply two-party computation from this 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 a signature token, this translates to many calls to the signature token 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 wanted to replace their commitments with our construction, we would encounter 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. Without this value, however, the extraction procedure will not work. 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[1]. 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

[1] Note that any commitment scheme can be made extractable (with rewinding) via an interactive protocol, e.g. [Gol01, MOSV06].

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 to 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 5.

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 4.2.1.

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 linear 2-universal hash function f from the family of linear 2-universal hash functions $\{f_h : \{0,1\}^{3\kappa} \to \{0,1\}^{\kappa}\}_{h \leftarrow \mathcal{H}}$.
   - 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 $(c_\sigma, d_\sigma) \leftarrow \mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Commit}\big((\sigma_x, \sigma_r, p_x, p_r, N)\big)$.
   - Send (c_s, c_x, c_σ, f) to the receiver. Release $\mathcal{T}$ by sending $(\mathtt{release})$.

Unveil phase:

3. Sender: 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 $\mathsf{SIG.Vfy}(\mathsf{vk}, \sigma_s, p_s) = \mathsf{SIG.Vfy}(\mathsf{vk}, \sigma_r, p_r) = 1$. Additionally, check if $\mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Open}(c_s, f(p_r), s) = 1$, $\mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Open}(c_\sigma, d_\sigma, (\sigma_s, \sigma_r, p_x, p_r, N)) = 1$ and $c_s \oplus p_x = s$. If not, abort; otherwise accept.

Figure 5: Computationally UC-secure protocol realizing $\mathcal{F}_{\mathrm{COM}}$ in the $\mathcal{F}_{\mathrm{wrap}}^{\text{ru-strict}}$-hybrid model.

Theorem 2. *The protocol Π_{COM} in Figure 5 computationally UC-realizes $\mathcal{F}_{\mathrm{COM}}$ (cf. Section 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 6. 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.

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 $\mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Open}(c_x^*, f^*(\alpha_j), \alpha_i) = 1$, 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 6: Simulator against a corrupted sender in the protocol Π_{COM}

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 16 with two 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 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.

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 $\mathsf{SECOM}.\mathsf{Open}(c_{\mathsf{sk}}^*, d_{\mathsf{sk}}^*, \mathsf{sk}^*) = 1$.

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 a preimage $\hat{p}_x$ of t_x under the linear 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 $\hat{d}_\sigma \leftarrow \mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Equiv}(\hat{\mathsf{sk}}, c_\sigma, (\hat{\sigma}_x, \hat{\sigma}_r, \hat{p_x}, \hat{p}_r, N))$.

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

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.

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 15 applies). Additionally, it is obvious that a preimage is efficiently sampleable due to the linearity of f. 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. For longer messages, the protocol has to be applied piecewise for each part of the message.

5 Ideal Signature Tokens

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 some UC functionalities cannot be realized. 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 signature tokens that are suitable for our scenario. Therefore, we require that the complete message that is to be signed is given to the signature token. Nevertheless, there are currently available signature cards that can be used for the protocols that are presented in this section, e.g. FinID[2].

[2] `https://eevertti.vrk.fi/Default.aspx?id=377`

5.1 Model

In contrast to $\mathcal{F}_{\text{wrap}}^{\text{ru-strict}}$, 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 $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$ by the query list, and parse inputs as a concatenation of actual input and the session id (cf. Figure 8).

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 8: 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.

Compared to our previous reusable token specification $\mathcal{F}_{\text{wrap}}^{\text{ru-strict}}$, it is no longer necessary to use a nonce to bind the messages to one specific protocol instance. Thus, the inherent interaction of the $\mathcal{F}_{\text{wrap}}^{\text{ru-strict}}$-hybrid model is removed in the $\mathcal{F}_{\text{wrap}}^{\text{ru}}$-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 strict model. This is similar to the model assumed in [HMQU05], but in contrast to their work, we focus on untrusted tokens.

Let us briefly state why we believe that this model is still useful. On the one hand, there are signature tokens 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.

5.2 UC-Secure Non-Interactive Two-Party Computation

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 5.2.1, which is a straight-forward modification of our construction of the straight-line extractable commitment from Section 4.2.1 to the weaker model $\mathcal{F}_{\text{wrap}}^{\text{ru}}$. Since the idealized model takes care of illegitimate queries to the setup, a nonce is no longer required and the construction becomes non-interactive. We use this commitment in the following construction of a non-interactive straight-line witness-extractable argument in Section 5.2.2. Then we sketch how this non-interactive straight-line witness-extractable argument can be used to realize UC-secure non-interactive computation in the $\mathcal{F}_{\text{wrap}}^{\text{ru}}$-hybrid model in Section 5.2.3.

5.2.1 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 4.2.1 to fit into the $\mathcal{F}_{\text{wrap}}^{\text{ru}}$-hybrid model.

Lemma 17. *The protocol $\Pi_{\text{COM}}^{\text{ni-se}}$ in Figure 9 is a straight-line extractable commitment scheme as per Definition 6 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 5. The security proof essentially follows the proof of Theorem 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 1. Consider the extraction algorithm in Figure 10. 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 1. □

5.2.2 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 (NIWIAoK) 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

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 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 $\{f_h : \{0,1\}^{3\kappa} \to \{0,1\}^{\kappa}\}_{h \leftarrow \mathcal{H}}$.
   - 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 $\mathsf{SIG.Verify}(\mathsf{vk}, \sigma_s, s) = \mathsf{SIG.Verify}(\mathsf{vk}, \sigma_r, r) = 1$. 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; otherweise accept.

Figure 9: 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 $\big((c_s^*, c_\sigma^*, f^*), Q\big)$, 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 10: The extraction algorithm for the non-interactive straight-line extractable commitment protocol $\Pi_{\mathrm{COM}}^{\text{ni-se}}$.

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 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 5.1. While we can only achieve computational security against a malicious prover, this allows us to use the straight-line extractable commitment scheme from Section 5.2.1 to obtain a straight-line witness-extractable argument. A formal description of the protocol is given in Figure 11.

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 5.2.1. Let SIG be a unique signature scheme according to Definition 14.

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 $c = \big((c_0^{(1)}, c_1^{(1)}), \ldots, (c_0^{(l)}, c_1^{(l)})\big)$.
   - 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 11: Computationally secure non-interactive straight-line witness-extractable argument in the $\mathcal{F}_{\text{wrap}}^{\text{ru}}$-hybdrid model.

Theorem 3. *The protocol Π_{NIWI} in Figure 11 is a straight-line witness-extractable argument as per Definition 9 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 12).

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

Let $\mathsf{Ext}_{\mathsf{COM}}$ be the extraction algorithm for SECOM. Upon input $\big(\pi^* = (\alpha^*, c^*, \sigma^*, (d^{*(1)}_{\sigma_1}, \ldots, d^{*(l)}_{\sigma_l})), Q\big)$, 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 $\big(c^*, Q\big)$ 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 12: 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 $(\mathtt{sign}, m^*)$ 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.

□

5.2.3 UC-secure NISC

To obtain our main result, we adapt the construction of Canetti et al. [CJS14] to the case of reusable tamper-proof hardware. In order to realize NISC, they present a construction of one-sided simulatable OT, which is basically an OT protocol that is UC-secure against the receiver and stand-alone secure against the sender. They require a NIWIAoK based on a global random oracle to achieve this notion. The construction uses the NIWIAoK in a black-box fashion, and we can replace this NIWIAoK with our construction from the previous section and obtain the same result.

We move on to briefly describe how UC-secure non-interactive secure computation (NISC) can be achieved in the $\mathcal{F}_{\text{wrap}}^{\text{ru}}$-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 $seed_i$.

Acknowledgement The authors would like to thank Dirk Achenbach and Björn Kaidel for helpful discussions on an earlier version of this paper.

References

[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, Heidelberg, February 2014.

[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, Heidelberg, May 2014.

[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.

[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.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[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, Heidelberg, February 2007.

[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, Heidelberg, August 1994.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001.

[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, Heidelberg, April 2008.

[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.

[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, Heidelberg, February 2014.

[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.

[CSV16] Ran Canetti, Daniel Shahaf, and Margarita Vald. Universally composable authentication and key-exchange with global PKI. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 265–296. Springer, Heidelberg, March 2016.

[Dam90] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 416–427. Springer, Heidelberg, August 1990.

[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, Heidelberg, March 2015.

[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, Heidelberg, March 2011.

[DKMQN15] Nico Döttling, Daniel Kraschewski, Jörn Müller-Quade, and Tobias Nilges. From stateful hardware to resettable hardware using symmetric assumptions. In Man Ho Au and Atsuko Miyaji, editors, *ProvSec 2015*, volume 9451 of *LNCS*, pages 23–42. Springer, Heidelberg, November 2015.

[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, Heidelberg, March 2013.

[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, Heidelberg, March 2009.

[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.

[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, Heidelberg, December 2013.

[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, Heidelberg, August 1987.

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *22nd ACM STOC*, pages 416–426. ACM Press, May 1990.

[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, Heidelberg, August 2010.

[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, Heidelberg, February 2010.

[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, Heidelberg, August 2008.

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[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.

[HPV15] Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Composable security in the tamper proof model under minimal complexity. Cryptology ePrint Archive, Report 2015/887, 2015. `http://eprint.iacr.org/2015/887`.

[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, Heidelberg, August 2008.

[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, Heidelberg, May 2007.

[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, Heidelberg, February 2010.

[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, Heidelberg, August 2002.

[Mer79] Ralph Charles Merkle. *Secrecy, Authentication, and Public Key Systems.* PhD thesis, Stanford University, Stanford, CA, USA, 1979.

[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, Heidelberg, March 2006.

[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, Heidelberg, April 2008.

[Nil15] Tobias Nilges. *The Cryptographic Strength of Tamper-Proof Hardware.* PhD thesis, Karlsruhe Institute of Technology, 2015.

[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, Heidelberg, August 2003.

[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, Heidelberg, August 1992.

[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, Heidelberg, March 2009.