# Reusing Tamper-Proof Hardware in UC-Secure Protocols

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

[1]Karlsruhe Institute of Technology, Germany
[2]ITK Engineering GmbH, Germany

**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. 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.

Currently considered protocols, however, suffer from two major drawbacks that impede their practical realization:

- The functionality of the tokens is protocol-specific, i.e. each protocol requires a token functionality tailored to its need.
- Different protocols cannot reuse the same token even if they require the same functionality from the token, because this would render the protocols insecure in current models of tamper-proof hardware.

In this paper we address these problems. First and foremost, we propose formalizations of tamper-proof hardware as an *untrusted* and *global* setup assumption. Modeling the token as a global setup naturally allows to reuse the tokens for arbitrary protocols. Concerning a versatile token functionality we choose a simple *signature* functionality, i.e. the tokens can be instantiated with currently available signature cards. Based on this we present solutions for a large class of cryptographic tasks.

**K**eywords: universal composability, tamper-proof hardware, unique signatures, global setup

## 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]. Typical setup assumptions like a common reference string or a public key infrastructure assume a *trusted* setup. Katz [Kat07] on the other hand put forward the idea that protocol parties create and exchange *untrusted* tamper-proof hardware tokens, i.e. the tokens may be programmed maliciously by the sending party.

Katz' proposal spawned a line of research that 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, CKS+14, DKMN15, DKMQN15, HPV16, HPV17].

Common to all of the aforementioned results is the fact that each protocol requires a token functionality that is tailored to the protocol. From a practical point of view it seems unlikely that these tokens will ever be produced by hardware vendors, and software implementations on standard smart cards are far too inefficient. Another negative side effect of protocol-specific tokens is that users need to keep at least one token for each application, which is prohibitive in practice.

We would therefore like to be able to use widely available standard hardware for our protocols. Examples are signature cards, where the token functionality is a simple signature functionality. The signing key is securely stored inside the tamper-proof hardware, while the verification key can be requested from the card. These cards are not required to keep an internal state (the keys can be hardwired). As an alternative several works in the literature 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 standardized and already deployed.

As it turns out, even if there were protocols that use a signature card as a setup assumption, it would not be possible to use the same token in a different protocol. This is due to the current definitions of tamper-proof hardware in the UC model. To the best of our knowledge, reusing tamper-proof hardware was only considered by Hofheinz et al. [HMQU05], who introduce the concept of *catalysts*. In their model, they show that the setup can be used for multiple protocols, unlike a normal UC setup, but they assume a *trusted* setup.

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 *Generalized UC (GUC)*, was introduced by Canetti et al. [CDPW07] and captures the fact that protocols are often more efficient if they can use the same setup. Indeed, a globally available token in the sense of GUC would naturally allow different protocols to use the same token. We note that the work of Chandran et al. [CGS08] and subsequent works following the approach of requiring only black-box access to the token during simulation (e.g. [GIS+10, CKS+14]) might in principle be suitable for reuse, however none of these works consider this scenario and the problem of highly protocol-specific token functionalities is prevalent in all of these works.

## 1.1 Our contribution.

We apply the GUC methodology to resettable tamper-proof hardware and define the first global setup that is *untrusted*, in contrast to *trusted and incorruptible* setups like a global random oracle [CJS14], key registration with knowledge [CDPW07] or trusted processors [PST17].

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

- The first model is inspired by the work of [HMQU05] and generalizes their approach from trusted signature cards to generic and untrusted resettable tokens. It is also designed to model real world restrictions regarding concurrent access to e.g. a smart card. A real world analogy is 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. We want to highlight that this only limits the access to the tokens for a short time, it is still possible to run several protocols requiring the same token in an interleaved manner.

- The second model is a GUC definition of a resettable tamper-proof hardware token following the approach of [CJS14], which is meant to give a GUC definition of reusable tamper-proof hardware. In particular, this means that there are no restrictions at all regarding access to the token.

We also consider a peculiarity of real world signature cards that is typically ignored in idealized descriptions. Most signature cards outsource some of the hashing of the message, which is usually needed in order to generate a signature, to the client. This is done to make the signature generation more efficient. We formally capture this in a new definition of signatures where the signing process is partitioned into a preprocessing and the actual signing. As we will show, cards that do outsource the hashing—even if only in part—cannot be used in all scenarios. Nevertheless, we show that a wide range of cryptographic functionalities can be realized, even if the card enforces preprocessing.

- UC-secure commitments in both models, even with outsourced hashing by the signature card. This means that all currently available signature cards can in principle be used with our protocols.
- UC-secure non-interactive secure computation (NISC) in the GUC model. Here it is essential that the hashing is performed on the card, i.e. not all signature cards are suitable for these protocols. This result establishes the minimal interaction required for (one-sided) two-party computation.

We show that the number of tokens sent is optimal, and that stateful tokens do not yield any advantage in the setting of globally available or reusable tokens.

## 1.2 Our techniques.

**Modelling reusable hardware tokens.** In the definition of the "realistic" model, a protocol is allowed to send a `seize` command to the token functionality, which will block all requests by other protocols to the token until it is released again via `release`. We have to make sure that messages cannot be exchanged between different protocols, thus the receiving party (of the signature, i.e. the sender of the signature card) has to choose a nonce. This nonce has to be included in the signature, thereby binding the message to a protocol instance. This obviously requires interaction, so non-interactive primitives cannot be realized in this model.

In order to explore the full feasibility of functionalities with reusable tokens and obtain more round-efficient protocols, we therefore propose a more idealized model following [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. Essentially, this turns the token into a full-blown GUC functionality and removes the additional interaction from the protocols.

**Commitments from signature cards.** Concerning our protocols in the above described models, 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 Choi et al. [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 tunnel messages through the signatures.

Based on tokens with this unique signature functionality, we construct a straight-line extractable commitment. 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 on current smart cards, however, the simulator might only learn a hash value, which makes extraction impossible. We thus modify this approach and make it work in our setting. Basically, we keep the intermediate values sent to the token in the execution and use them as a seed for a PRG, which can in turn be used to mask the actual message. Since the simulator observes this seed, it can extract the message. However, the token can still abort depending on the input, so we also have to use randomness extraction on the seed, otherwise the sender of the token might learn some bits of the seed.

Using the straight-line-extractable commitment as a building block, we modify the UC commitment of [CJS14] so that it works with signature cards.

**Non-interactive witness-extractable arguments.** A witness-extractable argument is basically a witness-indistinguishable argument of knowledge (WIAoK) with a straight-line extraction procedure. We construct such a non-interactive witness-extractable argument for later use in non-interactive secure computation (NISC). Our approach follows roughly the construction of Pass [Pas03], albeit not in the random oracle model. [Pas03] modify a traditional WIAoK by replacing the commitments with straight-line extractable ones. Further, they replace the application of a hash function to the transcript (i.e. the Fiat-Shamir heuristic) with queries to a random oracle. For our construction, we can basically use our previously constructed straight-line extractable commitments, but we also replace the queries to the random oracle by calls to the signature token, i.e. we can use the EUF-CMA security of the signature to ensure the soundness of the proof.

As hinted at above, this protocol requires the ideal model, since using a nonce would already require interaction. Also, there is a subtle technical issue when one tries to use signatures with preprocessing instead of the random oracle. In the reduction to the EUF-CMA security (where the reduction uses the signing oracle to simulate the token), it is essential that the commitments contain an (encoded) valid signature *before* they are sent to the token. However, if we use preprocessing, the preprocessed value does not provide the reduction with the commitments, which could in turn be extracted to reveal the valid signature and break the EUF-CMA security. Instead, it only obtains a useless preprocessed value, and once the reduction obtains the complete commitments via the non-interactive proof from the adversary, a valid call to the signature card on these commitments means that the adversary has a valid way to obtain a signature and the reduction does not go through. If the protocol were interactive, this would not be an issue, because we could force the adversary to first send the commitments and then provide a signature in a next step. But since the protocol is non-interactive, this does not work and we cannot use signature cards with preprocessing for this protocol. We believe this to be an interesting insight, since it highlights one of the differences in feasibility between idealized and practically available hardware.

| | # Tokens | Rounds | Assumption | Token Func. |
|---|---|---|---|---|
| [CGS08] | 2 (bidir.) | $\Theta(\kappa)$ | eTDP | specific for com. |
| [GIS$^+$10] | $\Theta(\kappa)$ (bidir.) | $\Theta(1)$ | CRHF[1] | specific for OT |
| [CKS$^+$14] | 2 (bidir.) | $\Theta(1)$ | VRF[2] | specific for OT |
| [HPV16] | $\Theta(\kappa^2)$ (bidir.) | $\Theta(1)$ | OWF | specific for OT |
| **Ours** (Section 3) | 2 (bidir.) | $\Theta(1)$ | Unique Sign.[3] | generic |
| **Ours** (Section 4) | 2 (bidir.) | $\Theta(1)$ | Unique Sign./DDH[4] | generic |

Table 1: Comparison of our result with existing solutions based on resettable hardware that technically allow reusing the tokens. All results mentioned above allow UC-secure 2PC, either directly or via generic completeness results.

### 1.3 Related work.

In an independent and concurrent work, using an analogous approach based on [CJS14], Hazay et al. [HPV16] recently introduced a GUC-variant of tamper-proof hardware to deal with the problem of adversarial token transfers in the multi-party case. This problem is equivalent to the problem of allowing the parties to reuse the token in different protocols without compromising security. Apart from using completely different techniques, however, [HPV16] are only interested in the general feasibility of round-efficient protocols. In contrast, we would like to minimize the number of tokens that are sent. Additionally, [HPV16] only consider the somewhat idealized GUC token functionality, and do not investigate a more realistic approach (cf. Section 3). This is an important aspect, in particular since our results indicate that some of the protocols in the idealized model cannot be realized in our more natural token model that is compatible with existing signature cards. Thus, from a more practical point of view, even the feasibility of generic 2PC is not conclusively resolved from existing results.

Table 1 gives a concise overview of our result compared with previous solutions based on resettable hardware that make black-box use of the token program in the UC security proof. Other approaches as shown in e.g. [DKMQ11, DMMQN13, DKMQN15] are more efficient, but require the token code and therefore cannot be reused without losing security.

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 according to their definitions, so their impossibilities do not apply.

[1] A protocol based on OWF is also shown, but the round complexity increases to $\Theta(\kappa/\log(\kappa))$. Additionally, it was shown by Hazay et al. [HPV16] that there is a subtle error in the proof of the protocol.

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

[3] Unique signatures are only known from specific number-theoretic assumptions and closely related to VRFs. These are required for our protocols.

[4] DDH is necessary for the NISC protocol.

## 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 $\pi$ is shown by comparing it to an *ideal* functionality $\mathcal{F}$. The ideal functionality is incorruptible and secure by definition. The protocol $\pi$ 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.

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

Implicitly parametrized by a domain of secrets $S$.

**Commit phase:**

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

**Unveil phase:**

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

Figure 1: Ideal functionality for commitments.

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

### 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 $\mathsf{COM}$ *between a sender* $\mathsf{S}$ *and a receiver* $\mathsf{R}$ *consists of two PPT algorithms* $\mathsf{Commit}$ *and* $\mathsf{Open}$ *with the following functionality.*

- $\mathsf{Commit}$ *takes as input a message* $s$ *and computes a commitment* $c$ *and unveil information* $d$*.*
- $\mathsf{Open}$ *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* $s$*:*

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

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* $\mathcal{A}_{\mathsf{R}}$*:*

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

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

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

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 an (expected) PPT algorithm* $\mathsf{Ext}$ *that, given black-box access to any malicious PPT algorithm* $\mathcal{A}_{\mathsf{S}}$*, outputs a pair* $(\hat{s}, \tau)$ *such that*

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

Extractable commitments can be constructed from any commitment scheme via additional interaction, see e.g. [Gol01, PW09]. 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* $\mathsf{COM} = (\mathsf{Commit}, \mathsf{Open})$ *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 $\mathsf{TCOM}$ *between a sender* $\mathsf{S}$ *and a receiver* $\mathsf{R}$ *consists of five PPT algorithms* $\mathsf{KeyGen}$, $\mathsf{TVer}$, $\mathsf{Commit}$, $\mathsf{Equiv}$ *and* $\mathsf{Open}$ *with the following functionality.*

- $\mathsf{KeyGen}$ *takes as input the security parameter and creates a key pair* $(\mathsf{pk}, \mathsf{sk})$, *where* $\mathsf{sk}$ *serves as the trapdoor.*
- $\mathsf{TVer}$ *takes as input* $\mathsf{pk}$ *and* $\mathsf{sk}$ *and outputs* $1$ *iff* $\mathsf{sk}$ *is a valid trapdoor for* $\mathsf{pk}$.
- $\mathsf{Commit}$ *takes as input a message* $s$ *and computes a commitment* $c$ *and unveil information* $d$.
- $\mathsf{Equiv}$ *takes as input the trapdoor* $\mathsf{sk}$, *message* $s'$, *commitment* $c$, *unveil information* $d$ *and outputs an unveil information* $d'$ *for* $s'$.
- $\mathsf{Open}$ *takes as input a commitment* $c$, *unveil information* $d$ *and a message* $s$ *and outputs a bit* $b \in \{0, 1\}$.

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

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

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* $\mathcal{L} \in \mathcal{NP}$ *with witness relation* $\mathcal{R}_{\mathcal{L}}$ *consists of a pair of PPT algorithms* $(\mathsf{P}, \mathsf{V})$ *such that 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* $\mathsf{P}^*$,
$$\Pr[\langle \mathsf{P}^*, \mathsf{V} \rangle(x) = 1] \leq \mathsf{negl}(|x|).$$
- *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* $\mathsf{V}^*$, *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: There exists an (expected) PPT algorithm* $\mathsf{Ext}$ *such that for every* $x \in \mathcal{L}$ *and every PPT algorithm* $\mathsf{P}^*$, *there exists a negligible function* $\nu(\kappa)$ *such that* $\Pr[\mathsf{Ext}(x, \mathsf{P}^*) \in w_{\mathcal{L}}(x)] > \Pr[\langle \mathsf{P}^*, \mathsf{V} \rangle(x) = 1] - \nu(\kappa)$.

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 $\mathsf{sgk}$, and he can publish the verification key $\mathsf{vk}$ such that anyone can verify the correctness of a signature, given message and signature.

**Definition 10.** *A* digital signature scheme $\mathsf{SIG}$ *consists of three PPT algorithms* $\mathsf{KeyGen}$*,* $\mathsf{Sign}$ *and* $\mathsf{Verify}$*.*

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

*We require correctness, i.e. for all* $m$ *and* $(\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$*:*

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

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 $\mathsf{KeyGen}$ algorithm to create a key pair $(\mathsf{vk}, \mathsf{sgk})$. The adversary $\mathcal{A}$ is given the verification key $\mathsf{vk}$ 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 $\sigma^*$ for a message $m^*$ without having queried the signature oracle with $m^*$.

A signature scheme $\mathsf{SIG}$ 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 $\sigma$ does not depend on $m$). A 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* $\mathsf{SIG}$ *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, $\mathsf{vk}$, $\sigma_1$, and $\sigma_2$ need not be created honestly by the respective algorithms, but may be arbitrary strings.

## 3 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 (deterministic) 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 $\sigma'$ by $r^{-1}$ and obtain a valid signature $\sigma$ 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.

**Definition 12.** *(Digital signatures with preprocessing) A signature scheme* $\mathsf{SIG}$ *with preprocessing consists of five PPT algorithms* $\mathsf{KeyGen}$*,* $\mathsf{PreSg}$*,* $\mathsf{Sign}$*,* $\mathsf{Vfy}$ *and* $\mathsf{Verify}$*.*

- $\mathsf{KeyGen}(1^\kappa)$ *takes as input the security parameter* $\kappa$ *and generates a key pair consisting of a verification key* $\mathsf{vk}$ *and a signature key* $\mathsf{sgk}$*.*
- $\mathsf{PreSg}(\mathsf{vk}, m)$ *takes as input the verification key* $\mathsf{vk}$*, the message* $m$ *and outputs a deterministically preprocessed message* $p$ *with* $|p| = n$*.*
- $\mathsf{Sign}(\mathsf{sgk}, p)$ *takes as input a signing key* $\mathsf{sgk}$ *and a preprocessed message* $p$ *of fixed length* $n$*. It outputs a signature* $\sigma$ *on the preprocessed message* $p$*.*
- $\mathsf{Vfy}(\mathsf{vk}, p, \sigma)$ *takes as input a verification key* $\mathsf{vk}$*, a preprocessed message* $p$ *and a presumed signature* $\sigma$ *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* $\mathsf{vk}$*, a message* $m$ *and a presumed signature* $\sigma$ *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*

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

*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 $\mathsf{Vfy}$ and $\mathsf{KeyGen}, \mathsf{Sign}$ and $\mathsf{Verify}$. The $\mathsf{PreSg}$ algorithm is typically realized as a collision-resistant hash function.

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 $\mathsf{KeyGen}$, $\mathsf{Sign}$ and $\mathsf{Vfy}$ are not EUF-CMA secure).

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 $\mathsf{pad}$ 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.

Independent of the actual implementation of the preprocessing, the requirement that the signature scheme is EUF-CMA secure both with preprocessing and without implies that the preprocessing needs to be (at the very least) collision resistant. We will now argue that this property guarantees a certain pseudo min-entropy of this preprocessed value, which is nessary for the security analysis of our protocols.

**Lemma 13.** *Let $s$ be a statistical security parameter, $\kappa \geq s$ a computational security parameter normalized to $s$ and* $(\mathsf{sgk}, \mathsf{vk}) \leftarrow \mathsf{SIG.KeyGen}(\kappa)$*. Then*

$$\tilde{H}_\infty(\mathsf{SIG.PreSg}(\mathsf{vk}, U_{3\kappa})) \geq \frac{s}{2}.$$

*Proof.* Assume that $\tilde{H}_\infty(\mathsf{SIG.PreSg}(U_{3\kappa})) = t < \frac{s}{2}$. Then a simple brute force search finds a collision in $2^{2t} < 2^s$ steps. This contradicts the assumption that the probability of finding a collision is $2^{-s}$. □

We thus get that if the collision probability is (statistically) lower bounded by $2^{-s}$, the output of $\mathsf{PreSg}$ on random inputs of sufficient length has at least an (average) min-entropy of $\frac{s}{2}$ bits. In order to achieve a meaningful guarantee in our protocols, we have to choose $s$ (and therefore the output length of $\mathsf{PreSg}$) appropriately. Assuming a typical cryptographic hash function is used for $\mathsf{PreSg}$, the min-entropy is therefore at least $\frac{n}{4}$, where $n$ is the output length of $\mathsf{PreSg}$.

### 3.1 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], with the difference that we consider untrusted tokens instead of a trusted signature token. 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.

**Functionality $\mathcal{F}_{\text{wrap}}^{\text{ru-strict}}$**

Implicitly parametrized by a security parameter $\kappa$.

**Creation:**

1. Await an input $(\texttt{create}, \mathsf{M}, t)$ from the token issuer, where $\mathsf{M}$ is a deterministic Turing machine and $t \in \mathbb{N}$. Store $(\mathsf{M}, t)$ 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 $\mathsf{M}$ on $w$ from its most recent state. When $\mathsf{M}$ halts without generating output or $t$ steps have passed, send $\perp$ to the receiver; otherwise store the current state of $\mathsf{M}$ and send the output of $\mathsf{M}$ 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 $\mathsf{M}$ 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 $\mathsf{M}$; it will be omitted throughout the rest of the paper.

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

For completeness, we add the definition of a catalyst introduced by Hofheinz et al. [HMQU05].

**Definition 14.** *Let* $\Pi$ *be a protocol realising the functionalities* $\mathcal{F}$ *and* $\mathcal{C}$ *in the* $\mathcal{C}$*-hybrid model. We say that* $\mathcal{C}$ *is used as a* catalyst *if* $\Pi$ *realises* $\mathcal{C}$ *by simply relaying all requests and the respective answers directly to the ideal functionality* $\mathcal{C}$*.*

In other words, the environment (and therefore other protocols) have access to the catalyst $\mathcal{C}$ while it is used in the protocol $\Pi$. In particular, this implies that the catalyst $\mathcal{C}$ cannot be simulated for a protocol. All in all, this notion is very similar to Definition 17.

## 3.2 UC-Secure Commitments

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 3.2.1. We then use this commitment in an adaption of a UC-secure commitment by Canetti et al. [CJS14], as shown in Section 3.2.2.

### 3.2.1 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 typically inefficient 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 has 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 draws a random value $x$. It ignores the precomputation step and sets the result $p_x$ of this step to be $x$ concatenated with the nonce $N$. The value $p_x$ is sent to the token, which returns a signature. From the value $p_x$ the sender derives the randomness for a one-time pad $\mathsf{otp}$ and randomness $r$ for a commitment. Using $r$ the sender commits to $x$, which will allow the extractor to verify if he correctly extracted the commitment. The sender also commits to the signature, $x$ and $N$ in a separate extractable commitment. To commit to the actual input $s$, the sender uses $\mathsf{otp}$. Care has to be taken because a maliciously programmed signature card might leak some information about $p_x$ to the receiver. Thus, the sender applies a 2-universal hash function before using it 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, using* $\mathcal{F}_{\text{wrap}}^{\text{ru-strict}}$ *as a catalyst.*

**Protocol $\Pi^{\text{se}}_{\text{COM}}$**

Let $\mathcal{T}$ be an instance of $\mathcal{F}^{\text{ru-strict}}_{\text{wrap}}$ and $\mathsf{PRG}$ be a pseudorandom generator. Further let $\mathsf{COM}$ be a computationally hiding and extractable commitment scheme. Let $\mathsf{SIG}$ be a unique signature scheme according to Definition 12.

**Global setup phase:**

- Receiver: Compute $(\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathsf{SIG.KeyGen}(1^\kappa)$. Program a stateless token $\mathsf{T}$ with the following functionality.
  - Upon receiving a message $(\mathtt{vk})$, return $\mathsf{vk}$.
  - Upon receiving a message $(\mathtt{sign}, m)$, compute $\sigma_m \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, m)$ and output $\sigma_m$.

  Send $(\mathtt{create}, \mathsf{T})$ to $\mathcal{T}$.
- Sender: Query $\mathcal{T}$ with $(\mathtt{vk})$ to obtain the verification key $\mathsf{vk}$ and check if it is a valid verification key for $\mathsf{SIG}$.

**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 $x \leftarrow \{0,1\}^{3\kappa}$ uniformly at random and choose a linear 2-universal hash function $f$ from the family of linear 2-universal hash functions $\{f_h : \{0,1\}^{4\kappa} \to \{0,1\}^\kappa\}_{h \leftarrow \mathcal{H}}$.
   - Send $(\mathtt{seize})$ to $\mathcal{T}$. Set $p_x = x||N$ and send $(\mathtt{sign}, p_x)$ to $\mathcal{T}$ to obtain $\sigma_x$. Abort if $\mathsf{SIG.Vfy}(\mathsf{vk}, p_x, \sigma_x) \neq 1$.
   - Derive $(\mathsf{otp}, r) \leftarrow \mathsf{PRG}(f(p_x))$ with $|\mathsf{otp}| = |s|$ and compute $c_s = s \oplus \mathsf{otp}$, $(c_x, r) \leftarrow \mathsf{COM.Commit}(p_x)$ and $(c_\sigma, d_\sigma) \leftarrow \mathsf{COM.Commit}(\sigma_x, x, N)$.
   - Send $(c_s, c_x, c_\sigma, f)$ to the receiver. Release $\mathcal{T}$ by sending $(\mathtt{release})$.

**Unveil phase:**

3. Sender: Send $(s, x, \sigma_x, d_\sigma)$ to the receiver.
4. Receiver: Set $p_x = x||N$ and compute $(\mathsf{otp}, r) \leftarrow \mathsf{PRG}(f(p_x))$. Check if $\mathsf{SIG.Vfy}(\mathsf{vk}, p_x, \sigma_x) = 1$, $\mathsf{COM.Open}(c_x, r, x) = 1$, $\mathsf{COM.Open}(c_\sigma, d_\sigma, (\sigma_x, x, N)) = 1$ and $c_s = s \oplus \mathsf{otp}$. 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.

Very briefly, extractability follows from the fact that the extractor can see all messages that were sent to the token, including the seed for the PRG 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 input that matches 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 $\Pi^{\text{se}}_{\text{COM}}$ in Lemma 15 and the straight-line extraction in Lemma 16.

**Lemma 15.** *The protocol $\Pi^{\text{se}}_{\text{COM}}$ in Figure 3 is computationally hiding, given that* $\mathsf{COM}$ *is an extractable computationally hiding commitment scheme, $f$ is a linear 2-universal hash function,* $\mathsf{PRG}$ *is a pseudorandom generator and* $\mathsf{SIG}$ *is an EUF-CMA-secure unique signature scheme.*

*Proof.* Let us consider a modified commit phase of the protocol $\Pi^{\mathrm{se}}_{\mathrm{COM}}$: instead of committing to the values $s, x, N, \sigma_x$, the sender $\mathsf{S}$ inputs random values in the commitments and replaces the generated pseudorandom string by 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 $\Pi^{\mathrm{se}}_{\mathrm{COM}}$ is computationally hiding. Consider the following series of hybrid experiments.

**Experiment 0:** The real protocol $\Pi^{\mathrm{se}}_{\mathrm{COM}}$.

**Experiment 1:** Identical to Experiment 0, except that instead of computing $(\mathsf{otp}, r) \leftarrow \mathsf{PRG}(f(p_x))$, draw $a$ uniformly at random and compute $(\mathsf{otp}, r) \leftarrow \mathsf{PRG}(a)$.

**Experiment 2:** Identical to Experiment 1, except that instead of using $\mathsf{PRG}(a)$ to obtain $\mathsf{otp}$ and $r$, $\mathsf{S}$ draws $\mathsf{otp}$ and $r$ uniformly at random.

**Experiment 3:** Identical to Experiment 2, except that instead of using $\mathsf{COM}$ to commit to $(\sigma_x, x, N)$, $\mathsf{S}$ commits to a random string of the same length.

**Experiment 4:** Identical to Experiment 3, except that instead of using $\mathsf{COM}$ to commit to $p_x$ with randomness $r$, $\mathsf{S}$ 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 $\mathsf{SIG}$ 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 $\mathsf{S}$.

The first case is prevented by using a unique signature scheme. The sender $\mathsf{S}$ asks $\mathcal{T}^*$ for a verification key $\mathsf{vk}^*$ and can verify that this key has the correct form for the assumed signature scheme. Then the uniqueness 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 uniqueness 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 $p_x$ generates a uniformly distributed value, even if $\mathsf{R}$ has some information about $p_x$. Since $x$ is 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 $\kappa$ (because the leakage function is fixed once the token is sent). Let $X$ be the random variable describing inputs into the signature token and let $Y$ describe the random variable representing the leakage. In the protocol, we apply $f \in \{f_h : \{0,1\}^{4\kappa} \to \{0,1\}^{\kappa}\}_{h \leftarrow \mathcal{H}}$ to $X$, which has at least min-entropy $3\kappa$, ignoring the nonce $N$. $Y$ has at most 2 possible outcomes, abort or proceed. Thus, [DORS08] gives a lower bound for the average min-entropy of $X$ given $Y$, namely

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

Note that $f$ is chosen *after* $\mathsf{R}^*$ 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, $f(x)$ is 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 $\mathsf{PRG}$, i.e. given a receiver $\mathsf{R}^*$ 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 $\mathsf{COM}$ is computationally hiding. From a distinguishing receiver $\mathsf{R}^*$ 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 $\Pi^{\mathrm{se}}_{\mathrm{COM}}$.

**Lemma 16.** *The protocol $\Pi^{\mathrm{se}}_{\mathrm{COM}}$ in Figure 3 is straight-line extractable, given that $\mathsf{COM}$ is an extractable computationally hiding commitment scheme and $\mathsf{SIG}$ 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 $\mathcal{F}^{\text{ru-strict}}_{\text{wrap}}$ for the combination of input and randomness for the commitment that is to be extracted.

**Extractor $\mathsf{Ext}_{\mathsf{SEC}}$**

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 $\mathcal{F}^{\text{ru-strict}}_{\text{wrap}}$, start the following algorithm.

1. For all $\alpha \in Q$, compute $(\hat{\mathsf{otp}}, \hat{r}) \leftarrow \mathsf{PRG}(f^*(\alpha))$ and test if $\mathsf{COM.Open}(c_x^*, \hat{r}, \alpha) = 1$. Otherwise, abort.
2. Let $(\hat{\mathsf{otp}}, \hat{r})$ be the values obtained in the previous step. Output $\hat{s} = c_s^* \oplus \hat{\mathsf{otp}}$.

Figure 4: The extraction algorithm for the straight-line extractable commitment protocol $\Pi^{\mathrm{se}}_{\mathrm{COM}}$.

Let $Q$ denote the set of inputs that $\mathcal{A}_\mathsf{S}$ sent to $\mathcal{F}^{\text{ru-strict}}_{\text{wrap}}$. Extraction will fail only in the event that a value $x^*$ is unveiled that has never been sent to $\mathcal{T}$, i.e. $p_x^* \notin Q$. We have to show that $\mathsf{Ext}_{\mathsf{SEC}}$ extracts $c_s^*$ 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 $\varepsilon(\kappa)$. We will use $\mathcal{A}_\mathsf{S}$ to construct an adversary $\mathcal{B}$ that breaks the EUF-CMA security of the signature scheme $\mathsf{SIG}$ with non-negligible probability. Let $\mathsf{vk}$ be the verification key that $\mathcal{B}$ receives from the EUF-CMA experiment. $\mathcal{B}$ simulates $\mathcal{F}^{\text{ru-strict}}_{\text{wrap}}$ for $\mathcal{A}_\mathsf{S}$ by returning $\mathsf{vk}$ upon receiving a query ($\texttt{vk}$); 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 $(\texttt{sign}, m)$, $\mathcal{B}$ forwards the message to the signature oracle of the EUF-CMA game and returns the resulting signature $\sigma$ to $\mathcal{A}_{\mathsf{S}}$.

$\mathcal{B}$ now simulates the interaction between $\mathcal{A}_{\mathsf{S}}$ and $\mathsf{R}$ up to the point when $\mathcal{A}_{\mathsf{S}}$ sends the message $c_{\sigma}^*$. The next messages between $\mathcal{A}_{\mathsf{S}}$ and $\mathsf{R}$ represent the interaction between an honest receiver and a malicious commitment sender $\mathcal{A}'_{\mathsf{S}}$ for the extractable commitment scheme $\mathsf{COM}$. 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 $\mathsf{COM}$, there exists an extractor $\mathsf{Ext}$ that on input $(c_{\sigma}^*, \mathcal{A}'_{\mathsf{S}})$ outputs a message $(\hat{\sigma}_x, \hat{x}, \hat{N})$ except with negligible probability $\nu(\kappa)$. $\mathcal{B}$ runs $\mathsf{Ext}$, sets $\hat{p}_x = \hat{x}||\hat{N}$ and outputs $(\hat{\sigma}_x, \hat{p}_x)$ 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 $c_{\sigma}$ succeeds. By assumption, $\mathcal{A}_{\mathsf{S}}$ succeeds in committing to a signature with non-negligible probability $\varepsilon(\kappa)$ in this case. It follows that the extractor $\mathsf{Ext}$ of $\mathsf{COM}$ will output a message $(\hat{\sigma}_x, \hat{x}, \hat{N})$ with non-negligible probability $\varepsilon(\kappa) - \nu(\kappa)$. Thus $\mathcal{B}$ will output a valid signature $\hat{\sigma}_x$ for a value $\hat{p}_x$ 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 $\mathsf{SIG}$.

Thus, the extractor $\mathsf{Ext}_{\mathsf{SEC}}$ will correctly output the value $s$ with overwhelming probability. □

### 3.2.2 Obtaining UC-Secure Commitments

In order to achieve computationally secure two-party computation, we want to transform the straight-line extractable commitment from Section 3.2.1 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 the techniques of the straight-line extractable commitment from the previous section, although we have to enhance it at several points. First, we need to query the signature token twice, for both $x$ and $r$, instead of deriving $r$ from $x$ via a PRG. This is necessary because all protocol steps have to be invertible in order to equivocate the commitment, and finding a preimage for a PRG is not efficiently possible. Second, we have to replace the extractable commitments 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 the straight-line extractable commitment) and uses the token to sign the unveil information. Against a corrupted sender, the simulator can thus extract the unveil information and thereby 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.

**Theorem 2.** *The protocol* $\Pi_{\mathrm{COM}}$ *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, using* $\mathcal{F}_{\mathrm{wrap}}^{\text{ru-strict}}$ *as a catalyst, given that* $\mathsf{TCOM}_{\mathsf{ext}}$ *is an extractable trapdoor commitment,* $\mathsf{SECOM}$ *is a straight-line extractable commitment and* $\mathsf{SIG}$ *is an EUF-CMA-secure unique signature scheme.*

*Proof.* **Corrupted sender.** Consider the simulator in Figure 6. It is basically a modified version of the extraction algorithm for the straight-line extractable commitment. Against a corrupted sender, we only have to extract the input of the sender and input it into the ideal functionality.

The only possibility for an environment $\mathcal{Z}$ to distinguish $\mathsf{Real}_{\mathcal{A}_{\mathsf{S}}}^{\Pi_{\mathrm{COM}}}$ and $\mathsf{Ideal}_{\mathcal{S}_{\mathsf{S}}}^{\mathcal{F}_{\mathrm{COM}}}$ is the case of an abort by the simulator. However, we can adapt Lemma 16 to this scenario.

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 $\sigma_x$ and $\sigma_r$. Now, the second commitment $c_\sigma$ 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.

[1]Note that any commitment scheme can be made extractable (with rewinding) via an interactive protocol, e.g. [Gol01, PW09].

**Protocol $\Pi_{\mathrm{COM}}$**

Let $\mathsf{TCOM}_{\mathsf{ext}}$ be an extractable trapdoor commitment scheme and let $\mathsf{SECOM}$ be the straight-line extractable commitment from Section 3.2.1.

**Global Setup phase:**

- Sender and receiver: Compute $(\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathsf{SIG.KeyGen}(1^\kappa)$. Program a stateless token $\mathsf{T_S}$ and $\mathsf{T_R}$, respectively, with the following functionality.
  - Upon receiving a message $(\mathtt{vk})$, return $\mathsf{vk}$.
  - Upon receiving a message $(\mathtt{sign}, m)$, compute $\sigma_m \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, m)$ and output $\sigma_m$.

  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 $\mathsf{vk}$ and check if it is a valid verification key for $\mathsf{SIG}$.

**Commit phase:**

1. Receiver: Compute $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{TCOM_{ext}.KeyGen}(1^\kappa)$ 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 $\{f_h : \{0,1\}^{4\kappa} \to \{0,1\}^\kappa\}_{h \leftarrow \mathcal{H}}$.
   - Send $(\mathtt{seize})$ to $\mathcal{T}_\mathsf{S}$. Compute $p_x = x||N$ and send $(\mathtt{sign}, p_x)$ to $\mathcal{T}_\mathsf{S}$ to obtain $\sigma_x$. Then, compute $p_r = r||N$ and send $(\mathtt{sign}, p_r)$ to $\mathcal{T}_\mathsf{S}$ to obtain $\sigma_r$. Abort if $\mathsf{SIG.Vfy}(\mathsf{vk}, p_x, \sigma_x) \neq 1$ or $\mathsf{SIG.Vfy}(\mathsf{vk}, p_r, \sigma_r) \neq 1$.
   - Compute $c_s = f(p_x) \oplus s$, $(c_x, f(p_r)) \leftarrow \mathsf{TCOM_{ext}.Commit}(p_x)$ and $(c_\sigma, d_\sigma) \leftarrow \mathsf{TCOM_{ext}.Commit}(\sigma_x, \sigma_r, p_x, p_r, N)$.
   - Send $(c_s, c_x, c_\sigma, f)$ to the receiver. Release $\mathcal{T}_\mathsf{S}$ 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 $\mathsf{TCOM_{ext}.TVer}(\mathsf{pk}, \mathsf{sk}) = 1$ 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 $\mathsf{SIG.Vfy}(\mathsf{vk}, p_s, \sigma_s) = \mathsf{SIG.Vfy}(\mathsf{vk}, p_r, \sigma_r) = 1$. Additionally, check if $\mathsf{TCOM_{ext}.Open}(c_x, f(p_r), p_x) = 1$, $\mathsf{TCOM_{ext}.Open}(c_\sigma, d_\sigma, (\sigma_s, \sigma_r, p_x, p_r, N)) = 1$ and $c_s \oplus f(p_x) = s$. 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.

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 $c^*_\mathsf{sk}$ fails, although $\mathsf{SECOM.Open}(c^*_\mathsf{sk}, d^*_\mathsf{sk}, \mathsf{sk}^*) = 1$.

**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 $f$.

**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 $\Pi_{\text{COM}}$ and let $(c_s^*, c_\sigma^*, f^*)$ 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 $(\hat{p}_x, \hat{p}_r) = (\alpha_i, \alpha_j)$ be the values obtained in the previous step. Compute $\hat{s} = c_s^* \oplus f^*(\hat{p}_x)$ and send $(\texttt{commit}, \hat{s})$ to $\mathcal{F}_{\text{COM}}$.
- (Unveil) Simulate the behavior of an honest receiver and obtain $s^*$. If $s^* = \hat{s}$, send $(\texttt{unveil})$ to $\mathcal{F}_{\text{COM}}$, otherwise abort.

Figure 6: Simulator against a corrupted sender in the protocol $\Pi_{\text{COM}}$

**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 $\mathcal{F}_{\text{COM}}$ and a message $(\mathsf{pk}^*, c_{\mathsf{sk}}^*, N)$ from $\mathcal{A}_{\mathsf{R}}$, simulate Step 2 of $\Pi_{\text{COM}}$ with input $s = 0$.
- (Unveil) Upon receiving a message $(\texttt{opened}, \hat{s})$, proceed as follows:
  - Start the straight-line extractor $\mathsf{Ext}_{\mathsf{SEC}}$ from $\mathsf{SECOM}$ to extract the commitment $c_{\mathsf{sk}}^*$ 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 $\hat{p}_r \leftarrow \mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Equiv}(\hat{\mathsf{sk}}, \hat{p}_x, \hat{c}_x, \hat{d}_x)$, 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 $\hat{d'}_\sigma \leftarrow \mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Equiv}(\hat{\mathsf{sk}}, (\hat{\sigma}_x, \hat{\sigma}_r, \hat{p}_x, \hat{p}_r, N), \hat{c}_\sigma, \hat{d}_\sigma)$.
  - Execute the unveil phase according to $\Pi_{\text{COM}}$: 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 $\Pi_{\text{COM}}$

**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 $\hat{d}_\sigma \leftarrow \mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Equiv}(\hat{\mathsf{sk}}, (\hat{\sigma}_x, \hat{\sigma}_r, \hat{p_x}, \hat{p}_r, N), c_\sigma, d_\sigma)$.

**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 $\hat{p}_r \leftarrow \mathsf{TCOM}_{\mathsf{ext}}.\mathsf{Equiv}(\hat{\mathsf{sk}}, \hat{p_x}, c_x, d_x)$. This is the ideal model.

Experiment 0 and Experiment 1 are computationally indistinguishable given that $\mathsf{SECOM}$ 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.

# 4 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.

## 4.1 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).

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.

Our security notion is as follows.

**Definition 17.** *Let $\mathcal{F}$ be an ideal functionality and let $\Pi$ be a protocol. We say that $\Pi$ UC-realizes $\mathcal{F}$ in the* global tamper-proof hardware model *if for any real PPT adversary $\mathcal{A}$, there exists an ideal PPT adversary $\mathcal{S}$ such that for every PPT enviroment $\mathcal{Z}$, it holds that*

$$\mathsf{Ideal}_{\mathcal{F},\mathcal{S}}^{\mathcal{F}_{\text{wrap}}^{\text{ru}}}(\mathcal{Z}) \approx \mathsf{Real}_{\Pi,\mathcal{A}}^{\mathcal{F}_{\text{wrap}}^{\text{ru}}}(\mathcal{Z})$$

**Functionality $\mathcal{F}_{\text{wrap}}^{\text{ru}}$**

Implicitly parametrized by a security parameter $\kappa$ 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 $\mathsf{M}$ is a deterministic Turing machine and $t \in \mathbb{N}$. Store $(\mathsf{M}, t)$ 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 $\mathsf{PID}$ and session id $\mathsf{SID}$. Parse $w$ as $(w', sid)$. If no `create`-message has been sent, return a special symbol $\perp$. Otherwise, run $\mathsf{M}$ 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 $\mathsf{M}$ halts without generating output or $t$ steps have passed, send $\perp$ to the receiver; otherwise store the current state of $\mathsf{M}$ and send the output of $\mathsf{M}$ to the receiver.

**Reset (adversarial receiver only):**

4. Upon receiving a message $(\texttt{reset})$ from a corrupted token receiver, reset $\mathsf{M}$ 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 $\mathsf{M}$; it will be omitted throughout the rest of the chapter.

Compared to the standard UC security, the setup is now available both in the real and the ideal settings.

## 4.2 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 4.2.1, which is a straight-forward modification of our construction of the straight-line extractable commitment from Section 3.2.1 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 4.2.2. 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 4.2.3.

### 4.2.1 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 3.2.1 to fit into the $\mathcal{F}_{\text{wrap}}^{\text{ru}}$-hybrid model.

**Protocol $\Pi_{\text{COM}}^{\text{ni-se}}$**

Let $\mathcal{T}$ be an instance of $\mathcal{F}_{\text{wrap}}^{\text{ru}}$ and $\mathsf{PRG}$ be a pseudorandom generator. Further let $\mathsf{COM}$ be a computationally hiding and extractable commitment scheme. Let $\mathsf{SIG}$ be a unique signature scheme according to Definition 12.

**Global setup phase:**

- Receiver: Compute $(\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathsf{SIG.KeyGen}(1^\kappa)$. Program a stateless token $\mathsf{T}$ with the following functionality.
  - Upon receiving a message $(\mathtt{vk})$, return $\mathsf{vk}$.
  - Upon receiving a message $(\mathtt{sign}, m)$, compute $\sigma_m \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, m)$ and output $\sigma_m$.

  Send $(\mathtt{create}, \mathsf{T})$ to $\mathcal{T}$.
- Sender: Query $\mathcal{T}$ with $(\mathtt{vk})$ to obtain the verification key $\mathsf{vk}$ and check if it is a valid verification key for $\mathsf{SIG}$.

**Commit phase:**

1. Sender: Let $s$ be the sender's input.
   - Draw $r \leftarrow \{0,1\}^{3\kappa}$ uniformly at random and choose a linear 2-universal hash function $f$ from the family of linear 2-universal hash functions $\{f_h : \{0,1\}^{3\kappa} \rightarrow \{0,1\}^\kappa\}_{h \leftarrow \mathcal{H}}$.
   - Send $(\mathtt{sign}, s)$ and $(\mathtt{sign}, r)$ to $\mathcal{T}$ and obtain $\sigma_s$ and $\sigma_r$. Abort if $\mathsf{SIG.Verify}(\mathsf{vk}, s, \sigma_s) \neq 1$ or $\mathsf{SIG.Verify}(\mathsf{vk}, r, \sigma_r) \neq 1$.
   - 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 $(c_s, c_\sigma, f)$ to the receiver.

**Unveil phase:**

2. Sender: Send $(s, r, \sigma_s, \sigma_r, d_\sigma)$ to the receiver.
3. Receiver: Check if $\mathsf{SIG.Verify}(\mathsf{vk}, s, \sigma_s) = \mathsf{SIG.Verify}(\mathsf{vk}, r, \sigma_r) = 1$. 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 $\mathcal{F}_{\text{wrap}}^{\text{ru}}$-hybrid model.

**Lemma 18.** *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* $\mathsf{COM}$ *is an extractable computationally hiding commitment scheme and* $\mathsf{SIG}$ *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. □

**Extractor $\mathsf{Ext}_{\mathsf{NIC}}$**

Upon input $((c_s^*, c_\sigma^*, f^*), Q)$, where $Q$ is the set of all query/answer pairs that $\mathcal{A}_{\mathsf{S}}$ sent to and received from $\mathcal{F}_{\text{wrap}}^{\text{ru}}$, start the following algorithm.

1. Test for all $\alpha_i, \alpha_j \in Q$ if $\mathsf{COM.Open}(c_s^*, f^*(\alpha_j), \alpha_i) = 1$. 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 $\Pi_{\text{COM}}^{\text{ni-se}}$.

### 4.2.2 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 $\Sigma$-protocol, i.e. a three message zero-knowledge proof system. We also assume that $\Pi$ has special soundness, i.e. from answers $\gamma_1, \gamma_2$ to two distinct challenges $\beta_1, \beta_2$, it is possible to reconstruct the witness that the prover used.

The main idea of his construction is as follows. Instead of performing a $\Sigma$-protocol interactively, a Fiat-Shamir transformation [FS87] is used to make the protocol non-interactive. The prover computes the first message $\alpha$ of the $\Sigma$-protocol, selects two possible challenges $\beta_1$ and $\beta_2$, computes the resulting answers $\gamma_1$ and $\gamma_2$ based on the witness $w$ according to the $\Sigma$-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 $(\alpha, c)$ 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 $\Sigma$-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. Further, if the commitments to $(\beta_i, \gamma_i)$ are straight-line extractable, the resulting argument system will be witness-extractable, i.e. an argument of knowledge.

The straight-line extractable commitment $\Pi_{\text{COM}}^{\text{se}}$ from Section 3.2.1 requires interaction, so we cannot directly plug this into the protocol without losing the non-interactive nature of the argument system. But note that the first message of $\Pi_{\text{COM}}^{\text{se}}$ is simply sending a nonce, which is no longer necessary in the $\mathcal{F}_{\text{wrap}}^{\text{ru}}$-hybrid model. Thus, by omitting this message, $\Pi_{\text{COM}}^{\text{se}}$ becomes a valid non-interactive straight-line extractable commitment.

A formal description of the protocol complete NIWIAoK is given in Figure 11.

**Theorem 3.** *The protocol $\Pi_{\text{NIWI}}$ 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* $\mathsf{NICOM}$ *is a straight-line extractable commitment scheme and* $\mathsf{SIG}$ *is an EUF-CMA-secure unique signature scheme.*

*Proof.* Let $\Pi$ be a public-coin special-sound honest-verifier zero-knowledge (SHVZK) protocol.

**Completeness:** Completeness of $\Pi_{\text{NIWI}}$ follows directly from the completeness of the $\Sigma$-protocol $\Pi$.

**Witness-Indistinguishability:** Cramer et al. [CDS94, Pas03] show that a SHVZK protocol directly implies a public-coin witness-indistinguishable protocol. Since witness-indistinguishable

**Protocol $\Pi_{\text{NIWI}}$**

Let $(\alpha, \beta, \gamma)$ denote the three messages of a $\Sigma$-protocol $\Pi$ for a language $\mathcal{L}$. Further let $\mathsf{NICOM}$ be an instance of $\Pi^{\text{se}}_{\text{COM}}$ from Section 3.2.1 without the first message. Let $\mathsf{SIG}$ be a unique signature scheme according to Definition 12.

**Global setup phase:**

- Verifier: Compute $(\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathsf{SIG.KeyGen}(1^\kappa)$. Program a stateless token $\mathsf{T}$ with the following functionality.
  - Upon receiving a message $(\mathtt{vk})$, return $\mathsf{vk}$.
  - Upon receiving a message $(\mathtt{sign}, m)$, compute $\sigma_m \leftarrow \mathsf{SIG.Sign}(\mathsf{sgk}, m)$ and output $\sigma_m$.

  Send $(\mathtt{create}, \mathsf{T})$ to $\mathcal{T}$.
- Sender: Query $\mathcal{T}$ with $(\mathtt{vk})$ to obtain the verification key $\mathsf{vk}$ and check if it is a valid verification key for $\mathsf{SIG}$.

**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 $\Sigma$-protocol. Pick $2l$ random challenges $(\beta_0^{(1)}, \beta_1^{(1)}), \ldots, (\beta_0^{(l)}, \beta_1^{(l)})$ for the $\Sigma$-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 $\Sigma$-protocol from $w$.
   - Commit to the challenge/response pairs $(\beta_b^{(i)}, \gamma_b^{(i)})$ via $(c_b^{(i)}, d_b^{(i)}) \leftarrow \mathsf{NICOM.Commit}(\beta_b^{(i)}, \gamma_b^{(i)})$ 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 $\sigma$ denote the result. Abort if $\mathsf{SIG.Verify}(\mathsf{vk}, (\alpha, c), \sigma) \neq 1$.

   Send $\pi = (\alpha, c, \sigma, (d_{\sigma_1}^{(1)}, \ldots, d_{\sigma_l}^{(l)}), (\beta_{\sigma_1}^{(1)}, \gamma_{\sigma_1}^{(1)}), \ldots, (\beta_{\sigma_l}^{(l)}, \gamma_{\sigma_l}^{(l)}))$ to the verifier where $\sigma_i$ signifies the $i$-th bit of $\sigma$.
2. Verifier: First, check if $\mathsf{SIG.Verify}(\mathsf{vk}, (a, c), \sigma) = 1$. 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 $\mathsf{NICOM.Open}(c_{\sigma_i}^{(i)}, d_{\sigma_i}^{(i)}, (\beta_{\sigma_i}^{(i)}, \gamma_{\sigma_i}^{(i)})) = 1$. If that check is passed, accept; otherwise abort.

Figure 11: Computationally secure non-interactive straight-line witness-extractable argument in the $\mathcal{F}^{\text{ru}}_{\text{wrap}}$-hybdrid model.

protocols are closed under parallel composition as shown be Feige et al. [FS90], $\Pi_{\text{NIWI}}$ is witness-indistinguishable.

**Extractablility:** Let $\mathsf{Ext}_{\mathsf{NIC}}$ be the straight-line extractor of $\mathsf{NICOM}$. We will construct a straight-line extractor for $\Pi_{\text{NIWI}}$ (cf. Figure 12).

It remains to show that if the verifier accepts, $\mathsf{Ext}_{\mathsf{NIWI}}$ outputs a correct witness with overwhelming probability. First, note that $\mathsf{Ext}_{\mathsf{NIC}}$ extracts the inputs of $c^*$ with overwhelming probability, and by the special soundness of $\Pi$, 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 $\Pi$ has soundness $\frac{1}{2}$, since a malicious $\mathcal{A}_{\mathsf{P}}$ can only answer at most

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

Let $\mathsf{Ext}_{\mathsf{SEC}}$ be the extraction algorithm for $\mathsf{SECOM}$. 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 $\mathcal{F}^{\mathrm{ru}}_{\mathrm{wrap}}$, start the following algorithm.

1. Run the verifier algorithm on $\pi^*$, if it aborts, abort. Run $\mathsf{Ext}_{\mathsf{NIC}}$ 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 $\Pi_{\mathrm{NIWI}}$.

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 $\varepsilon(\kappa)$ 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 $\mathsf{SIG}$ with probability $\varepsilon(\kappa)$.

Let $\mathcal{B}$ be the adversary for the EUF-CMA game. Let $\mathsf{vk}$ be the verification key that $\mathcal{B}$ receives from the EUF-CMA game. $\mathcal{B}$ simulates $\mathcal{F}^{\mathrm{ru}}_{\mathrm{wrap}}$ to $\mathcal{A}_{\mathsf{P}}$ by returning $\mathsf{vk}$ upon receiving a query $(\mathtt{vk})$; further let $Q$ be the set of queries that $\mathcal{A}_{\mathsf{P}}$ sends to $\mathcal{F}^{\mathrm{ru}}_{\mathrm{wrap}}$. 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 $\sigma$ 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 $\mathsf{Ext}_{\mathsf{NIC}}$ with input $(c^*, Q)$ to extract the commitments $c^*$ using $Q$. Create a signature $\sigma^*$ by selecting $\sigma^*_i$ as the index of the correctly evaluating challenge. The verifier will only accept if that is the case. If $\mathsf{SIG.Verify}(\mathsf{vk}, (\alpha^*, c^*), \sigma^*) = 1$, send $(m^*, \sigma^*)$ to the EUF-CMA game, otherwise abort. We thus have that $\mathcal{A}_{\mathsf{P}}$ wins the EUF-CMA game with probability $\varepsilon(\kappa)$, which contradicts the EUF-CMA security of $\mathsf{SIG}$.

□

### 4.2.3 UC-secure NISC

**One-Sided Simulatable OT** We recap the construction of one-sided simulated oblivious transfer from Canetti et al. [CJS14] for completeness. Their construction is based on the efficient plain model cut-and-choose OT protocol of Lindell and Pinkas [LP11], which in turn is based on the protocol of Peikert et al. [PVW08]. The main difference in [CJS14] is to eliminate one message from the original protocol by replacing a zero-knowledge proof of the correctness of selected parameters with a witness-indistinguishable proof, because this step is only needed for the simulation, i.e. the extraction, of the OT against a corrupted sender. The protocol thus retains its sender privacy, while it is still simulatable against a corrupted receiver. Since we can achieve a non-interactive witness-indistinguishable argument of knowledge in the $\mathcal{F}^{\mathrm{ru}}_{\mathrm{wrap}}$-hybrid model, the complete protocol only requires 2 rounds.

For more details we refer the interested reader to [CJS14].

We will not provide a formal security proof for this protocol, because the proof of [CJS14] uses the NIWI in a black-box fashion and we simply replace the NIWI in their protocol by our

**Protocol $\Pi_{\text{OT}}^{\text{os-s}}$**

Let $\mathsf{NIWI}$ be the non-interactive argument of knowledge from Section 4.2.2. $\mathsf{RAND}$ is the following randomization algorithm of [PVW08] that outputs two uniformly and independently distributed group elements: Upon input $(g, h, g', h')$, choose $s, t \in \mathbb{Z}_q$ uniformly at random and output $(u = g^s h^t, v = g'^s, h'^s)$.

**Common Parameters:** $(\mathbb{G}, q, g_{0,0}, g_{0,1})$

**Setup Phase:**

1. Receiver: (Input a bit $c$)
   - Choose $y_0, y_1, \alpha_{0,0}, \alpha_{0,1} \in \mathbb{G}$ uniformly at random and set $\alpha_{1,0} = \alpha_{0,0} + 1$ and $\alpha_{1,1} = \alpha_{0,1} + 1$. Compute $g_{1,b} = (g_{0,b})^{y_b}$,$h_{0,b} = (g_0)^{\alpha_{0,b}}$ and $h_{1,b} = (g_1)^{\alpha_{1,b}}$ for $b \in \{0,1\}$.
   - Run $\mathsf{NIWI}$ for the language $\mathcal{L} = \{(\{g_{i,j}\}_{i,j=0}^1, \{h_{i,j}\}\}_{i,j=0}^1) | \exists \alpha : h_{0,0} = (g_{0,0})^\alpha \wedge h_{1,0} g_{1,0}^{-1} = (g_{1,0})^\alpha \text{ OR } h_{0,1} = (g_{0,1})^\alpha \wedge h_{1,1} g_{1,1}^{-1} = (g_{1,1})^\alpha\}$ with witness $\alpha_{0,b}$ for a randomly chosen $b$ and let $\pi$ be the result.
   - Choose $r_0, r_1 \in \mathbb{Z}_q$ uniformly at random and compute $g^0 = (g_{c,0})^{r_0}$,$g^1 = (g_{c,1})^{r_1}$,$h^0 = (h_{c,0})^{r_0}$,$h^1 = (h_{c,1})^{r_1}$ and set $\mathsf{pk}_0 = (g^0, h^0)$ and $\mathsf{pk}_1 = (g^1, h^1)$.
   - Send $\mathsf{par}_0 = (g_0, h_{0,0}, g_{1,0}, h_{1,0})$, $\mathsf{par}_1 = (g_1, h_{0,1}, g_{1,1}, h_{1,1})$, $\pi$ and $\mathsf{pk}_0, \mathsf{pk}_1$ to the sender.

**Transfer**

2. Sender: (Input two strings $s_0, s_1$)
   - Check that the proof $\pi$ is correct, if not abort.
   - Choose $s_{0,0}, s_{0,1}$ uniformly at random and set $s_{1,0} = s_{0,0} + s_0$ and $s_{1,1} = s_{0,1} + s_1$.
   - Compute $(u_{b,0}, v_{b,0}) \leftarrow \mathsf{RAND}(g_{b,0}, h_{b,0}, \mathsf{pk}_0)$ for $b \in \{0,1\}$ and $w_{0,0} = v_{0,0}s_{0,0}$,$w_{1,0} = v_{1,0}s_{1,0}$. Compute $(u_{b,1}, v_{b,1}) \leftarrow \mathsf{RAND}(g_{b,1}, h_{b,1}, \mathsf{pk}_1)$ for $b \in \{0,1\}$ and $w_{0,1} = v_{0,1}s_{0,1}$,$w_{1,1} = v_{1,1}s_{1,1}$.
   - Send $(u_{0,0}, w_{0,0}), (u_{1,0}, w_{1,0})$ and $(u_{0,1}, w_{0,1}), (u_{1,1}, w_{1,1})$ to the receiver.
3. Receiver: Obtain $s_{c,0} = w_{c,0}(u_{c,0})^{-r_0}$ and $s_{c,1} = w_{c,1}(u_{c,1})^{-r_1}$, and output $s_c = s_{c,0} + s_{c,1}$.

Figure 13: One-sided simulatable OT in the $\mathcal{F}_{\text{wrap}}^{\text{ru}}$-hybrid model.

non-interactive witness-indistinguishable argument from Section 4.2.2.

**Universally Composable NISC** Given the one-sided simulateable OT from the previous section, we show how to UC-realize the non-interactive secure computation protocol from [CJS14] based on reusable resettable hardware. 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 security proof of the protocol and refer the interested reader to [CJS14]. A detailed description of the protocol is given in Figures 14 and 15

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 as in [AMPR14]). Then, the sender starts to garble $t$ circuits. All randomness for the garbling of each circuit $gc_i$ 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 so 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 $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{ru}}$-hybrid model. Then, since there is no global random oracle available, we use a token programmed with a signature function to sign the query $q_i$. However, the signature $\sigma_i$ 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, we apply a 2-universal hash function to $q_i$ and let this be the seed $seed_i$. To ensure that the simulator can extract the seed and the receiver can verify the correctness of the garbling, we require the sender to input $(k_i, (q_i, \sigma_i))$ 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 $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{ru}}$ 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 $q_i$.

Let $f : \{0,1\}^{|x|} \times \{0,1\}^{|y|} \to \{0,1\}^{|z|}$ and $C$ be the circuit computing $f$. Let $\mathsf{EGCom}$ be an ElGamal commitment $(g^r, h^r g^b) = msEGCom(h, b, r)$. Let $\mathsf{Enc}$ denote an encryption scheme, $\mathsf{COM}$ a commitment scheme, $\mathsf{h}$ a collision-resistant hash function that is a suitable randomness extractor and $\mathsf{PRF}$ a pseudorandom function. Further let $t$ denote the number of circuits and $\mathsf{sid}$ the session identifier.

We denote by $\mathsf{OT}_0(b), \mathsf{OT}_1(k_0, k_1)$ the two messages from the OT protocol $\Pi_{\mathrm{OT}}^{\text{os-s}}$ from Section 4.2.3. Changes to the original protocol are highlighted in red.

**Protocol $\Pi_{\mathrm{NISC}}$**

1. $\mathsf{P}_1$ (Input $x$): Let $\mathsf{IN}_x$ denote the input wires for $C$.
   - Pick a random $t$-bit string $c_1, \ldots, c_t$. Let $T$ be the set of $i$ such that $c_i = 1$.
   - For each index $i \in [t]$, publish $\mathsf{OT}_0(c_i)$.
   - For each input wire $j \in \mathsf{IN}_x$ publish $\mathsf{OT}_0(x_j)$.
2. $\mathsf{P}_2$ (Input $y$): Let $\mathsf{IN}_y$ denote the input wires for $C$.

   **Commitments to input, output and trapdoor:**
   - Pick a uniformly random trapdoor $w \in \mathbb{Z}_q$ and send $h = g^w$.
   - Send $\mathsf{EGCom}(h, y_j, r_j)$ for $j \in \mathsf{IN}_y$, where $y_j$ is the input for wire $j$ and $r_j$ uniformly random.
   - Send $h_{j,0} = g^{w_{j,0}}$ and $h_{j,1} = g^{w_{j,1}}$ for each output wire $j \in \mathsf{OUT}$, where $w_{j,0} \in \mathbb{Z}_q$ and $w_{j,1} = w - w_{j,0}$.
   - ▷ Choose a 2-universal hash function $f_h$ from the family of 2-universal hash functions $\{f_h : \{0,1\}^{3\kappa} \to \{0,1\}^{\kappa}\}_{h \leftarrow \mathcal{H}}$. Query $\mathcal{T}$ with $(\mathtt{vk})$ and obtain $\mathsf{vk}$.

   **Generate garbled circuits:** For each circuit $i \in [t]$
   - ▷ Randomly choose $q_i \in \{0,1\}^{3\kappa}$ and send $(\mathtt{sign}, q_i)$ to $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{ru}}$ to obtain $\sigma_i$. Abort if $\mathsf{SIG.Verify}(\mathsf{vk}, q_i, \sigma_i) \neq 1$. Set $seed_i = f_h(q_i)$.
   - Compute $u_{i,j,b} = \mathsf{ECCom}(h, b, r_{i,j,b})$ for all wires $j \in \mathsf{IN}_y, b \in \{0,1\}$ and $r_{i,j,b} = \mathsf{PRF}_{seed_i}(\text{``}EGCom'' \circ j \circ b)$.
   - Compute garbled circuit $gc_i$:
     - For $j \in \mathsf{IN}_y$ and $b \in \{0,1\}$, set $\mathsf{label}(gc_i, j, b) = \mathsf{h}(\mathsf{P}_2 \circ \mathsf{sid}, u_{i,j,b})$.
     - Any other label for wire $j$ and $b$ is constructed as usual using randomness generated by $seed_i$, i.e. $\mathsf{PRF}_{seed_i}(\text{``}label'' \circ j \circ b)$.
   - Send commitments $\{c_{i,j,\delta}, c_{i,j,1-\delta}\}_{j \in \mathsf{IN}_y}$, where $\delta_{ij}$ is uniformly random and $(c_{i,j,\delta}, d_{i,j,\delta}) \leftarrow \mathsf{COM.Commit}(u_{i,j,\delta})$. The randomness for the commitments is derived from $seed_i$ via $\mathsf{PRF}$.

   **Cheating recovery box:** For $j \in \mathsf{OUT}$ send
   (a) $h_{j,0} \cdot g^{K_{i,j,0}}$ and $h_{j,1} \cdot g^{K_{i,j,1}}$, where $K_{i,j,b}$ are uniformly random.
   (b) $\mathsf{Enc}(\mathsf{label}(gc_i, j, 0), K_{i,j,0})$, $\mathsf{Enc}(\mathsf{label}(gc_i, j, 1), K_{i,j,1})$

   **Proofs of input/output consistency:**
   - Let $\mathsf{inputs}_i$ be the set $\{u_{i,j,y_j}, d_{i,j,y_j}\}_{j \in \mathsf{IN}_y}$. Let $\mathsf{inputsEq}$ be the set $\{r - j - r_{i,j,y_j}\}_{j \in \mathsf{IN}_y}$.
   - Let $\mathsf{outputsDC}$ be the set $\{w_{j,0} + K_{i,j,0}, w_{j,1} + K_{i,j,1}\}_{j \in \mathsf{OUT}}$.
   - Pick a random $k_i$ and send $\mathsf{Enc}(k_i, \mathsf{inputs}_i \circ \mathsf{inputsEq}_i \circ \mathsf{outputsDC}_i)$.

   **OT answers:**
   - (Input OT) Send $\{\mathsf{OT}_1(\mathsf{label}(gc_i, j, 0), \mathsf{label}(gc_i, j, 1))\}_{j \in \mathsf{IN}_y, i \in [t]}$. Derive the randomness $r_{i,j}$ for wire $j$ and circuit $i$ via $\mathsf{PRF}_{seed_i}(\text{``}OT'' \circ 1 \circ \text{``}r'' \circ i \circ j)$.
   - ▷ (Circuit OT) Send $\{\mathsf{OT}_1((q_i, \sigma_i), k_i)\}_{i \in [t]}$.

Figure 14: UC-secure NISC protocol in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{ru}}$-hybrid model.

## 5 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

**Protocol $\Pi_{\text{NISC}}$ cont'd**

3. $\mathsf{P}_1$:

**Circuit consistency:**

- ▷ For each $i \in T$, check if $\mathsf{SIG.Verify}(\mathsf{vk}, q_i, \sigma_i) = 1$. Set $seed_i = f_h(x_i)$.
- Check that $seed_i$ correctly generated $gc_i$ and the answers of the $i$-th execution of the input OT, abort if not.

**Input/output consistency:** For all $i \in [t] \setminus T$:

- Verify that $h_{j,0} \cdot h_{j,1} = h$ for all $j \in \mathsf{OUT}$.
- Check that $\mathsf{outputDC}_i$ are correct discrete logs of the values in the set $\{h_{j,b} g^{K_{i,j,b}}\}_{j \in \mathsf{OUT}, b \in \{0,1\}}$.
- Check that $\mathsf{inputsEq}_i$ are consistent with the input commitments: check if $u_{i,j,y_j}(g^{r_j - r_{i,j,y_j}}, h^{r_j - r_{i,j,y_j}}) = \mathsf{EGCom}(h, y_j, r_j)$, otherwise abort.
- Evaluate circuit $gc_i$: Let $\{\mathsf{l}_{i,j}\}_{j \in \mathsf{OUT}}$ be the set of labels that are obtained. Decrypt the corresponding values $\mathsf{Enc}(\mathsf{l}_{i,j}, K_{i,j,b})$ from the cheating recovery box. Check if the result is a correct decommitment of the output recovery commitment $h_{j,b} g^{K_{i,j,b}}$ where $b$ are the inputs received from $gc_i$. If all steps pass, label $gc_i$ as *semi-trusted*.

**Compute output:** If all semi-trusted $gc_i$ output the same value, output that value. Otherwise:

- Let $gc_i, gc_i'$ be two semi-trusted circuits with different outputs in the $j$-th output wire, and let $\mathsf{l}_{i,j}$ and $\mathsf{l}_{i,j}'$ be the output labels. Learn $w_{j,0}$ from one of the labels and $w_{j,1}$ from the other (since $K_{i,j,b}$ and $K_{i,j,1-b}$ can be obtained from the cheating recovery boxes, and $w_{j,b} + K_{i,j,b}$,$w_{j,1-b} + K_{i,j,1-b}$ from $\mathsf{outputsDC}_i$ and $\mathsf{outputsDC}_i'$).
- Compute $w = w_{j,0} + w_{j,1}$ and decrypt the input commitments. Let $y$ be the hereby obtained value. Output $f(x, y)$.

Figure 15: UC-secure NISC protocol in the $\mathcal{F}_{\text{wrap}}^{\text{ru}}$-hybrid model.

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, GUC and GUC-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 19.** *There exists no protocol $\Pi_{\text{PF}}$ 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 $\mathcal{F}_{\text{PF}}$.*

*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 a protocol for non-interactive two-party computation where the parties exchange two messages (including hardware tokens) to obtain a (somewhat) non-interactive protocol. Maybe even more interesting, even stateful reusable hardware tokens will not yield any advantage compared to resettable tokens, if the tokens are only sent in one direction.

**Corollary 20.** *There exists no protocol* $\Pi_{\mathrm{OT}}$ *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. □

**Acknowledgement** The authors would like to thank Dirk Achenbach and Björn Kaidel for helpful discussions on an earlier version of this paper, and the anonymous reviewers for their comments. We further thank Maciej Obremski for disussions on the entropy estimation of the signatures.

# 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.

[HPV16] Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Composable security in the tamper-proof hardware model under minimal complexity. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 367–399. Springer, Heidelberg, October / November 2016.

[HPV17] Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Constant round adaptively secure protocols in the tamper-proof hardware model. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 428–460. Springer, Heidelberg, March 2017.

[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.

[Jag15] Tibor Jager. Verifiable random functions from weaker assumptions. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 121–143. Springer, Heidelberg, March 2015.

[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.

[LP11] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 329–346. Springer, Heidelberg, March 2011.

[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.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th FOCS*, pages 120–130. IEEE Computer Society Press, October 1999.

[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.

[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.

[PST17] Rafael Pass, Elaine Shi, and Florian Tramèr. Formal abstractions for attested execution secure processors. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 260–289. Springer, Heidelberg, May 2017.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.

[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.