Implementing Resettable UC-Functionalities with Untrusted Tamper-Proof Hardware-Tokens

Nico Döttling, Thilo Mie, Jörn Müller-Quade, and Tobias Nilges

Karlsruhe Institute of Technology, Karlsruhe, Germany
{doettling,mueller-quade,nilges}@kit.edu,mie@ira.uka.de

Abstract. Resettable hardware tokens, usually in the form of smart cards, are used for a variety of security-critical tasks in open environments. Many of these tasks require trusted hardware tokens. With the complexity of hardware, however, it is not feasible to check if the hardware contains an internal state or gives away information over side channels. This inspires the question of the cryptographic strength of untrusted resettable hardware tokens in the universal composability framework.

In this work, we consider the problem of realizing general UC-functionalities from untrusted resettable hardware-tokens, with the goal of minimizing both the amount of interaction and the number of tokens employed. Our main result consists of two protocols, realizing functionalities that are sufficient to UC-realize any resettable two-party functionality.

The first protocol requires two rounds of interaction in an initialization phase and only a single hardware-token. The second protocol is fully non-interactive and requires two tokens. One of these relaxations, allowing either communication with the issuer of the token or issuing two tokens, is necessary. We show that even a simple functionality cannot be realized non-interactively using a single token.

Keywords: Resettably secure computation, Tamper-Proof hardware, Universal Composability.

1 Introduction

In this paper we investigate the cryptographic strength of tamper-proof resettable hardware tokens in the universal composability model. This setting is motivated by smart cards. Smart cards are tamper-proof and resettable. Our aim is to obtain non-interactive protocols where a sender can issue tamper-proof hardware which is not necessarily trusted by the receiver. Non-interactive protocols allow communication only in one direction from the sender to the receiver. We will, however, differentiate two types of non-interactive protocols. In the first type we allow an exchange of hardware tokens and interactive communication between sender and receiver in an initialization phase before the input is given. After the input is given the only communication allowed is one message sent from the sender to the receiver. The second and more strict type allows (even in an initialization phase) communication and sending of hardware tokens in one direction only.

Of course, using resettable hardware and non-interactive protocols one cannot expect to realize any non-resettable ideal functionality. The real adversary could always reset the token and start the protocol all over again. Hence this unavoidable attack must be reflected in the ideal functionality. Note that this poses a limitation on the usefulness of some ideal functionalities. E.g. a coin-toss of a single bit becomes useless as an attacker could reset the protocol until his desired result occurs. Resettable functionalities resemble an ideal, i.e. black-box, code obfuscation which is impossible without secure hardware [1].

Our Contribution. In the following we prove that there exist simple resettable ideal functionalities, namely point-functions, which cannot securely be realized by a strictly non-interactive protocol using one single resettable hardware token. We prove the impossibility of realizing a point-function with a single resettable token in a strictly non-interactive protocol along the lines of [2]. Any successful simulator for a corrupted token would already yield a cheating strategy for the token in the real model.

If we were given a common-reference-string (CRS), however, non-interactive protocols can be realized via secure two party computations between the receiver and the hardware token. Note that the secure two-party computation must be adapted to be used with a resettable token. A general construction can be found in [3]. For each message m which is sent to the token in the secure two-party computation one sends the complete previous transcript plus the message m. The token then verifies the transcript and answers according to the protocol. Any UC-compiler for secure two-party computation in the CRS-hybrid-model (e.g. [4]) can be used to implement the underlying two-party protocol.

So the problem of non-interactive secure computation boils down to securely computing a CRS in an initialization phase of the protocol. As our main contribution we provide two constructions for securely realizing the CRS functionality with resettable tamper-proof hardware tokens. The first protocol-construction obtains a CRS using a single resettable hardware-token and a 4-move interactive initialization-phase. The second protocol-construction obtains a *resettable* CRS using two resettable hardware tokens and no further interaction with the sender.

Our Techniques. At the core of our first construction, which allows interaction with the sender in an initialization phase, is a Blum coin-toss protocol. In this protocol the hardware token is used like a UC-commitment which is opened via the possession of a secret which is sent by the sender in the unveil phase. This secret is not directly given to the token to avoid communication between the sender and the token, instead a zero-knowledge proof is used. Note that this ZK proof must be resettably-sound and therefore can only be achieved using non-black-box techniques.

Our second construction uses two hardware tokens which are both issued by the sender and works in the strictly non-interactive setting where nothing may ever be sent by the receiver. In the strictly non-interactive setting one cannot use one of the two tokens like a UC-commitment which is unveiled via the possession of a secret value, because both tokens are resettable and the receiver could learn the committed value, reset, and force the coin-toss to some malicious value. The basic idea behind our solution is a Blum coin-toss where the receiver commits to a random value x and sends x together with the commitment c to the first token. The token answers with a value y which is deterministically derived from the commitment by a pseudorandom function. The second token is then used to check if the value y is indeed deterministically derived. The second token is given the commitment c and must answer with the same y. In this case $x \oplus y$ is the result of the coin-toss. For the security proof in case of a corrupted receiver it is crucial that the communication with the two tokens takes place in this order. The second token must not reveal the value y too early, because the simulator must choose y depending on x which is not possible if he must choose y too early. To cope with that, the first token signs the commitment and the second token accepts a commitment only together with a proof of possession of a valid signature. For the simulation of a corrupted sender, however, it is necessary to obtain y before one is actually committed to x. To do so the simulator executes the protocol out of order. He is able to forge the proof of possession of a valid signature. To the best of our knowledge this simulation technique is novel and an interesting result on its own.

As an application of our non-interactive protocols we propose a *conditional decrypt* for a fully homomorphic encryption scheme. The condition for decryption being that a universal argument is provided to the token that a specific computation has been performed. This allows offloading computations from the token. The actual computation can be performed using fully homomorphic encryption and only the conditional decrypt has to be realized as a two-party computation. This construction can be used to achieve obfuscation.

Further Related and Concurrent Work. The notion of resettability has gained considerable attention, especially in the context of zero-knowledge. The results range from a resettable prover [3, 5, 6], to a resettable verifier [6, 7] and simultaneously resettable prover and verifier [8–10]. These works spawned a line of work realizing compilers for resettable and stateless secure multi-party computation. Goyal and Sahai [11] presented a compiler which enables multi-party computation of arbitrary PPT-functionalities either with honest majority where any party can be resettable, or no honest majority is required and only one predetermined party may be resettable. Surpassing the honest majority requirement, Goyal and Maji [12] introduced a compiler that transforms most PPT-functionalities (except a certain class of pseudorandom generators) into a stateless variant. Goyal et al. [13] present an unconditionally secure zero-knowledge protocol for $\mathcal{NP}$. They show that statistically secure commitments with one stateless token are only possible when interaction between the parties is allowed, but impossible in the non-interactive setting. This is due to the fact that in the non-interactive case the token needs to contain superpolynomial entropy since both sender and receiver are unbounded. In contrast, we consider computational UC-security [14], so their impossibility result does not apply to us.

There are several results concerning multi-party computation in the UC-framework. [14] showed that under the assumption of an honest majority, any multi-party protocol is realizable in the UC-framework. Since it is known to be impossible to construct large classes of UC-secure two-party protocols in the plain model [14–16], the result of Canetti et al. [4] was a breakthrough. In their work, a common reference string is needed as a setup assumption to overcome the impossibility results and realize arbitrary multi-party protocols in the UC-framework. Further work based on tamper-proof hardware includes [17, 18].

In a work independent and concurrent to ours, [19] investigated how stateless tamper-proof hardware tokens can serve as a minimal UC-setup assumption. They present a black-box protocol realizing OT with two stateless tokens and show that OT from one stateless token is not possible if only black-box techniques are used. This is similar to our impossibility result, but we cover any amount of resettable tokens in the non-interactive setting. Further, they construct a coin-toss protocol with a single hardware token using similar techniques to our first protocol. However, they do not consider a non-interactive coin-toss protocol.

2 Preliminaries

Let in the following k denote a security parameter. We use the cryptographic standard notions of negligible functions, as well as computational/statistical/perfect indistinguishability.

2.1 Framework

We state and prove our results in the UniversalComposability (UC)-framework of [14]. In this framework security is defined by comparison of a *real model* and an *ideal model*. The protocol of interest Π is running in the latter, where an adversary $\mathcal{A}$ coordinates the behavior of all corrupted parties. We assume static corruption, i.e. the adversary $\mathcal{A}$ cannot adaptively change corruption during a protocol-run. In the ideal model, which is secure by definition, an ideal functionality $\mathcal{F}$ implements the desired protocol task and a simulator $\mathcal{S}$ tries to mimic the actions of $\mathcal{A}$. An environment $\mathcal{Z}$ is plugged either to the ideal or the real model and has to guess which model it is actually plugged to. Denote the random variable representing the output of $\mathcal{Z}$ when interacting with the real model by $\mathsf{Real}^{\mathcal{A}}_{\Pi}(\mathcal{Z})$ and when interacting with the ideal model by $\mathsf{Ideal}^{\mathcal{S}}_{\mathcal{F}}(\mathcal{Z})$. Protocol Π is said to be UC-secure if for any environment $\mathcal{Z}$ the distributions $\mathsf{Real}^{\mathcal{A}}_{\Pi}(\mathcal{Z})$ and $\mathsf{Ideal}^{\mathcal{S}}_{\mathcal{F}}(\mathcal{Z})$ are (computationally, statistically or perfectly) indistinguishable.

2.2 Strongly Unforgeable Signatures

As our scheme requires the use of signatures, we shall briefly review the standard notion of strongly unforgeable signature schemes. A signature scheme SIG consists of three PPT-algorithms KeyGen, Sign and Verify.

- $\mathsf{KeyGen}(1^k)$ generates a public verification key vk and a private signature key sgk.
- $\mathsf{Sign}_{sgk}(m)$ takes a signature key sgk and a message $m \in \{0,1\}^*$ and returns a signature σ.
- $\mathsf{Verify}_{vk}(m, \sigma)$ takes as input a verification key vk, a message $m \in \sigma^*$ and a signature σ and outputs 1 if σ is a valid signature for m and 0 otherwise.

In the EUF-CMA-experiment an adversary $\mathcal{A}$ is given a verification key vk and access to a signature-oracle. $\mathcal{A}$ wins the experiment if it manages to forge a valid signature σ for a message of its choice m, without having queried its signature-oracle with m. A signature scheme SIG is called EUF-CMA-secure, if no PPT-adversary $\mathcal{A}$ wins the EUF-CMA-experiment better than with negligible probability. For the sake of simplicity, we require signature schemes with a deterministic verification procedure and succinct signature length (i.e. the length of σ does not depend on m). Standard hash-and-sign [20, 21] constructions suffice these requirements.

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

2.3 Resettably-Sound Zero-Knowledge Arguments of Knowledge

For the construction of our protocols we use resettably-sound zero-knowledge (rsZK) arguments of knowledge [7]. We briefly define the notions.

Definition 1. *A resettably-sound zero-knowledge argument of knowledge system for a language $L \in \mathcal{NP}$ (with witness-relation $\mathcal{R}_L$ and witness-set $w_L(x) = \{w : (x, w) \in R_L\}$) consists of a pair of PPT-machines (P, V), where the verifier V is stateless, such that there exist two PPT-machines Sim and Ext and the following conditions hold.*

- ***Completeness.*** *For every $(x, w) \in \mathcal{R}_L$ it holds that $\Pr[\langle \mathsf{P}(w), \mathsf{V}\rangle(x) = 1] = 1$.*
- ***Computational Soundness.*** *For every $x \notin L$ and every PPT-machine P^* it holds that $\Pr[\langle \mathsf{P}^*, \mathsf{V}\rangle(x) = 1] < \mathsf{negl}(|x|)$.*
- ***Computational Zero-Knowledge.*** *For every $(x, w) \in \mathcal{R}_L$ and every stateful or stateless PPT V^* it holds the distributions $\mathsf{Real} = \{\langle \mathsf{P}(w), \mathsf{V}^*\rangle(x)\}$ and $\mathsf{Ideal} = \{\mathsf{Sim}(x, \mathsf{V}^*)\}$ are computationally indistinguishable.*
- ***Proof of Knowledge.*** *For every $x \in L$ and every PPT-machine P^* there exists a negligible ν such that $\Pr[\mathsf{Ext}(x, \mathsf{P}^*)] \in w_L(x)] > \Pr[\langle \mathsf{P}^*, \mathsf{V}\rangle(x) = 1] - \nu$.*

2.4 Perfectly Binding Commitments

Another tool we need is a non-interactive perfectly binding commitment scheme. Generally, a commitment scheme consists of two phases: the *commit phase* in which a sender commits to a value v without revealing it, and the *reveal phase* where the sender reveals his private coins r together with v such that a receiver can verify the correctness of the commitment. The value v has to be hidden from the receiver while the commitment has to be binding for the sender. For our application, a standard computationally hiding and perfectly binding commitment scheme is sufficient, e.g. a construction based on a one-way permutation [22]. Perfectly binding means that there is exactly one randomness to unveil correctly.

3 Resettable Hardware in the UC-Framework

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

Functionality $\mathcal{F}_{\mathsf{M}}$ (parametrized by a security parameter k).

- $\mathsf{Sender\ Input}$ Upon receiving $(\texttt{sender}, \mathit{init})$ from S, store init, write init on M's input tape and run M until it halts. Store the state of M. Accept no further inputs by $\mathcal{S}$
- $\mathsf{Receiver\ Input}$ Upon receiving $(\texttt{receiver}, \mathit{msg})$ from R, write msg on M's input tape and run M starting from most recent state until it halts. Store the state of M. Read a message out from M's output tape and send out to R.
- Reset (Adversarial receiver only) Upon receiving $\texttt{reset}$ from R, reset the Turing machine M to its initial state. Write init on M's input tape and run M until it halts. Store the state of M.

We use a definition of wrapper-functionalities very similar to [17, 18]. For simplicity, we state the functionality in the two-party case where only a sender-machine S and a receiver-machine R are present. This definition allows the sender S to wrap a program T in a hardware token, and send this token to the receiver R who can query it an arbitrary (polynomial) number of times. Additionally, we allow an adversarial receiver to reset the program T to its initial state.

Functionality $\mathcal{F}_{wrap}$ (parametrized by a security parameter k and a polynomial runtime bound $p(\cdot)$).

- Create Upon receiving $(\texttt{create}, \mathsf{T}, p(\cdot))$ from S, where T is a Turing machine, send $\texttt{create}$ to R and store T.
- $\mathsf{Execute}$ Upon receiving $(\texttt{run}, w)$ from R, check if a $\texttt{create}$-message has already been sent by S, if not output $\perp$. Run $\mathsf{T}(w)$ for at most $p(k)$ steps, and let m be the output. Save the current state of T. Output m to R
- Reset (Adversarial Receiver only) Upon receiving $\texttt{reset}$ from R, reset the Turing machine T to its initial state.

The messages between $\mathcal{F}_{wrap}$ and R are delivered immediately without scheduling by the adversary. In the sequel, we will use the notation T for programs (given as code, Turing-machine etc.) and $\mathcal{T}$ for the instance of the wrapper-functionality $\mathcal{F}_{wrap}$ in which T runs.

We will introduce two new hybrid functionalities as an intermediate building block between $\mathcal{F}_{wrap}$ and general resettable UC-functionalities. Both functionalities are enhanced wrapper-functionalities where both the receiver of the token and the wrapped program are given access to a trusted common reference string. The two different flavors of this functionality we consider here differ in that the common reference string is either resettable by a corrupted receiver or non-resettable.

Functionality $\mathcal{F}_{wrap}^{hybrid1}$ (parametrized by a security parameter k and a polynomial runtime bound $p(\cdot)$).

- Create Upon receiving $(\mathtt{create}, \mathsf{T}, p(\cdot))$ from S, where T is a Turing machine, store T, choose the common reference string crs uniformly at random of length ℓ and give T read-access to crs. Send $(\mathtt{create}, \mathsf{crs})$ to R and S.
- Execute Upon receiving $(\mathtt{run}, w)$ from R, check if a $\mathtt{create}$-message has already been sent by S, if not output $\perp$. Run $\mathsf{T}(w)$ for at most $p(k)$ steps, and let m be the output. Save the current state of T. Output m to R
- Reset (Adversarial Receiver only) Upon receiving $\mathtt{reset}$ from R, reset the Turing machine T to its initial state.

Remark. By having the functionality $\mathcal{F}_{wrap}^{hybrid1}$ send the common reference string crs to the sender S we model an artifact that arises in our protocol.

Functionality $\mathcal{F}_{wrap}^{hybrid2}$ (parametrized by a security parameter k and a polynomial runtime bound $p(\cdot)$). Let H be a random oracle that maps to strings of length ℓ.

- Create Upon receiving $(\mathtt{create}, \mathsf{T}, p(\cdot))$ from S, where T is a Turing machine, store T, set the common reference string to be $\mathsf{crs} = H(1)$ and give T read-access to crs. Send $(\mathtt{create}, \mathsf{crs})$ to R.
- Execute Upon receiving $(\mathtt{run}, w)$ from R, check if a $\mathtt{create}$-message has already been sent by S, if not output $\perp$. Run $\mathsf{T}(w)$ for at most $p(k)$ steps, and let m be the output. Save the current state of T. Output m to R
- Reset (Adversarial Receiver only) Upon receiving $(\mathtt{reset}, j)$ from R, reset Turing machine T to its initial state and set the common reference string to $\mathsf{crs} = H(j)$.

We will briefly sketch how general-purpose UC-compilers in the CRS-hybrid-model, like for instance the compiler of [4], can be used to implement arbitrary resettable two-party UC-functionalities in the $\mathcal{F}_{wrap}^{hybrid1}$ and $\mathcal{F}_{wrap}^{hybrid2}$ hybrid model. Recall that these functionalities provide both the receiver and the encapsulated program access to an encapsulated common reference string. The basic idea is to assign the receiver the role of one protocol-party and the encapsulated program the role of the other protocol-party. The case of a malicious sender is trivial, as any UC-simulator against a corrupted sender can also serve as a simulator against a malicious token. The case of a malicious receiver needs to take reset-attacks against the token into account. However, this can be dealt with by applying a transformation due to [3]. This transformation replaces random coins used by the token with pseudorandom coins, which deterministically depend on all the messages received by the token at each point in time. This transformation merely requires a pseudorandom function.

4 Limitations

In this section, we will sketch two limitations regarding the implementation of resettable UC-functionalities using untrusted resettable hardware tokens. For simplicity, we consider protocols realizing the point-function functionality $\mathcal{F}_{\mathsf{PF}}$. This functionality is initialized by an input $\hat{x} \in \{0,1\}^n$ from the sender. The receiver can query $\mathcal{F}_{\mathsf{PF}}$ an arbitrary (polynomial) amount of times with inputs x, receiving output $\mathsf{PF}_{\hat{x}}(x)$, where $\mathsf{PF}_{\hat{x}}(x) = 1$ if $x = \hat{x}$ and $\mathsf{PF}_{\hat{x}}(x) = 0$ otherwise.

Lemma 1. *There exists no protocol which (computationally) UC-realizes the $\mathcal{F}_{\mathsf{PF}}$-functionality using only a single hardware token and no further communication. Moreover, any protocol UC-realizing the $\mathcal{F}_{\mathsf{PF}}$-functionality using any amount of resettable hardware tokens (issued from* S *to* R*) must make use of non-black-box techniques in its security proof.*

Proof. First assume there exists a protocol Π_{PF} that UC-implements $\mathcal{F}_{\mathsf{PF}}$ using a single (resettable) hardware token and no interaction (w.l.o.g. we can assume that messages from S to R are sent together with the token). Let $\tilde{\mathcal{A}}_{\mathsf{R}}$ be the dummy-adversary for the receiver R. Since Π_{PF} is UC-secure, there exists a simulator $\mathcal{S}_{\mathsf{R}}$ such that it holds for any PPT-environment $\mathcal{Z}$ that $\mathsf{Real}_{\Pi_{\mathsf{PF}}}^{\tilde{\mathcal{A}}_{\mathsf{R}}}(\mathcal{Z}) \approx_c \mathsf{Ideal}_{\mathcal{F}_{\mathsf{PF}}}^{\mathcal{S}_{\mathsf{R}}}(\mathcal{Z})$. We will now show that for every sender-simulator $\mathcal{S}_{\mathsf{S}}$ there exists a PPT-environment $\mathcal{Z}^*$ such that the distributions $\mathsf{Real}_{\Pi_{\mathsf{PF}}}^{\tilde{\mathcal{A}}}(\mathcal{Z}^*)$ and $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{PF}}}^{\mathcal{S}_{\mathsf{S}}}(\mathcal{Z}^*)$ are efficiently distinguishable, contradicting the UC-security of Π_{PF}. This $\mathcal{Z}^*$ creates a malicious token $\mathcal{T}^*$ which behaves adaptively in the following sense. The token $\mathcal{T}^*$ internally simulates the simulator $\mathcal{S}_{\mathsf{R}}$ together with a malicious functionality $\mathcal{F}^*$, providing its interface with R to $\mathcal{S}_{\mathsf{R}}$. The malicious functionality $\mathcal{F}^*$ behaves as follows. Once it receives an input x for the first time, it checks whether x is equal to a secret random $\hat{x}_0$. If so, it will behave like the point function $\mathsf{PF}_{\hat{x}_0}$ in this call and all successive calls. If not, it will behave like a point function $\mathsf{PF}_{\hat{x}_1}$ (for a secret random $\hat{x}_1$) in this call and all successive calls. Observe now that from the view of R, the protocol Π_{PF} always implements a proper point function. However, a simulator $\mathcal{S}_{\mathsf{S}}$ must decide if it inputs $\hat{x}_0$ or $\hat{x}_1$ into the ideal functionality $\mathcal{F}_{\mathsf{PF}}$ without knowing the first input x of R. The environment $\mathcal{Z}$ can now distinguish between real and ideal as follows. It first flips an unbiased coin. If the outcome is 0, it provides input $x = \hat{x}_0$ to R, otherwise it provides input $x = \hat{x}_1$ to R. If $\mathcal{Z}^*$ is connected to the real experiment, then the output of R behaves according to the specification of $\mathcal{F}^*$.

In the ideal experiment however, the output of R behaves either like $\mathsf{PF}_{\hat{x}_0}$ or $\mathsf{PF}_{\hat{x}_1}$ (or completely different). Thus, with probability at least 1/2 $\mathcal{Z}^*$ notices a difference. This contradicts the UC-security of Π_{PF}.

For the second statement of the lemma, assume there exists a protocol Π_{PF} UC-realizing $\mathcal{F}_{\mathsf{PF}}$ and that there exists a black-box simulator $\mathcal{S}_{\mathsf{S}}$ against a corrupted sender $\tilde{\mathcal{A}}_{\mathsf{S}}$ such that it holds for all PPT-environments $\mathcal{Z}$ that $\mathsf{Real}^{\tilde{\mathcal{A}}_{\mathsf{S}}}_{\Pi_{\mathsf{PF}}}(\mathcal{Z}) \approx_c \mathsf{Ideal}^{\mathcal{S}_{\mathsf{S}}}_{\mathcal{F}_{\mathsf{PF}}}(\mathcal{Z})$. Such a simulator must be able to extract a point $\hat{x}$ from the malicious tokens $\mathcal{T}_1^*, \ldots, \mathcal{T}_n^*$ using only black-box techniques (i.e. rewinding). We will now construct an environment $\mathcal{Z}^*$ and a malicious receiver $\mathcal{A}_{\mathsf{R}}$ that extracts the secret $\hat{x}$ from the tokens $\mathcal{T}_1, \ldots, \mathcal{T}_n$. $\mathcal{A}_{\mathsf{R}}$ internally simulates $\mathcal{S}_{\mathsf{S}}$ and provides his interface with $\mathcal{T}_1, \ldots, \mathcal{T}_n$ to $\mathcal{S}_{\mathsf{S}}$. $\mathcal{A}_{\mathsf{R}}$ then outputs to $\mathcal{Z}$ whatever $\mathcal{S}_{\mathsf{S}}$ outputs. From the view of $\mathcal{S}_{\mathsf{S}}$, the simulation of $\mathcal{A}_{\mathsf{R}}$ is identically distributed to $\mathsf{Ideal}^{\mathcal{S}_{\mathsf{S}}}_{\mathcal{F}_{\mathsf{PF}}}(\mathcal{Z})$. Thus, $\mathcal{A}_{\mathsf{R}}$ outputs the secret point $\hat{x}$ with overwhelming probability. On the other hand, any simulator $\mathcal{S}_{\mathsf{R}}$ has only black-box access to the point function $\mathsf{PF}_{\hat{x}}$ via $\mathcal{F}_{\mathsf{PF}}$. Thus $\mathcal{S}_{\mathsf{R}}$ succeeds to learn $\hat{x}$ only with negligible probability. Therefore $\mathcal{Z}^*$ can efficiently distinguish $\mathsf{Real}^{\mathcal{A}_{\mathsf{R}}}_{\Pi_{\mathsf{PF}}}(\mathcal{Z})$ and $\mathsf{Ideal}^{\mathcal{S}_{\mathsf{R}}}_{\mathcal{F}_{\mathsf{PF}}}(\mathcal{Z})$ for any PPT-simulator $\mathcal{S}_{\mathsf{R}}$, contradicting the UC-security of Π_{PF}.

5 Resettable UC-Functionalities from Untrusted Hardware

In this section, we present the main result of this work. We provide two UC-secure protocols implementing the $\mathcal{F}^{hybrid1}_{wrap}$ and $\mathcal{F}^{hybrid2}_{wrap}$ UC-functionalities in the $\mathcal{F}_{wrap}$-hybrid model. The protocol for $\mathcal{F}^{hybrid1}_{wrap}$ requires only a single $\mathcal{F}_{wrap}$-instance, i.e. a single untrusted hardware token. However, the protocol requires an interactive initialization phase with the issuer of the token. The protocol $\mathcal{F}^{hybrid2}_{wrap}$ requires two $\mathcal{F}_{wrap}$-instances, but is on the other hand completely non-interactive. The blueprint for both protocols is the same. In a setup-phase, a common random reference string is negotiated via a variant of the Blum coin-tossing protocol tailored for the respective setting. In the second phase, a program M encapsulated in the wrapper-functionality (that has access to the common reference string) can be queried by the receiver.

We will start outlining the ideas behind the first protocol $\Pi^{hybrid1}$. On an intuitive level, the token is locked with a password. The receiver first needs to obtain the password a to use the token. The receiver obtains a after performing a Blum coin-toss with the sender which results in a common reference string. For this coin-toss, the issuer uses the token itself to commit to a random string y. More specifically, the sender programs the token to release y, after the token is convinced the the receiver is in possession of a. First the sender sends the token to the receiver to commit himself to the random string y. In the second step, the receiver sends a random x to the sender, who replies with y and the password a (which serves as unveil-information). The receiver now proves to the token that he is in possession of a, thereby obtaining (y', a') from the token. If it holds that $y = y'$, the receiver is convinced that the sender was a-priori committed to y

and a. If the receiver accepts, sender and receiver set $\mathsf{crs} = x \oplus y$ and the sender signs crs, so that the receiver can use it with the token.

One issue we did not address yet is how the receiver proves to the token that he is in possession of the password a. For that purpose, we first make the password a verifiable. We do this by choosing the string a uniformly at random and having the sender publish the image $b = \mathsf{F}(a)$ of a under a one-way function F. As the token is resettable, we will use a resettably-sound zero-knowledge argument of knowledge system for the receiver to prove to the token that he is in possession of the password a, i.e. he possesses an a such that $\mathsf{F}(a) = b$.

In our second protocol $\Pi^{hybrid2}$, where the sender issues two stateless tokens to the receiver and no communication between the sender and receiver is allowed, the above protocol fails. The reason for this is that once the receiver knows a, it can learn the string y, reset the second token (which acts in the role of the sender) and force the crs to a value of his liking by choosing his input x adaptively.

We now give an outline of our second protocol. In order to prohibit the receiver to choose his input x adaptively, he is now required to commit to x using a non-interactive perfectly binding commitment-scheme com. The protocol proceeds as follows. First, R computes $c = \mathsf{com}(x; r)$ (for random coins r). R then sends (x, c) to the first token $\mathcal{T}_1$ and proves that c is a proper commitment of x. Again, we use a resettably-sound zero-knowledge argument system, as $\mathcal{T}_1$ is resettable. If $\mathcal{T}_1$ accepts, it computes y pseudorandomly by $y = \mathsf{prf}(c)$, where prf is a pseudorandom function. Notice that a corrupted sender may choose prf maliciously, it is therefore essential that prf remains oblivious of x. Instead of releasing a password, $\mathcal{T}_1$ now computes a signature σ of c. This σ will now serve as a witness to unlock the second token $\mathcal{T}_2$ for a run with the commitment c. More specifically, R sends c to $\mathcal{T}_2$ and proves to $\mathcal{T}_2$, using a resettably-sound zero-knowledge argument system of knowledge, that it knows a valid signature of c under the verification key vk, i.e. it knows a σ such that verification succeeds. Once $\mathcal{T}_2$ is convinced that R knows such a σ, it computes $y' = \mathsf{prf}(c)$ and outputs y' to R. R now checks if $y = y'$ holds, if so it sets $\mathsf{crs} = x \oplus y$ and uses crs as common random reference string in a two-party computation with $\mathcal{T}_1$.

5.1 A Single Resettable Token

We now provide a formal statement of protocol $\Pi^{hybrid1}$ that UC-emulates $\mathcal{F}^{hybrid1}_{wrap}$. Let k be a security-parameter. Let $\mathcal{T} = \mathcal{F}_{\mathsf{wrap}}$ be a resettable hardware wrapper-functionality, let $\mathsf{F} : \{0,1\}^k \to \{0,1\}^m$ be a one-way function and (P, V) be a resettably-sound zero-knowledge argument system of knowledge for the language $L = \{b : \exists a \in \{0,1\}^k \text{ s.t. } b = F(a)\}$. Further let SIG be an EUF-CMA-secure signature scheme.

Let $\ell = \mathsf{poly}(k)$ be the desired length of the output common reference string.

Protocol $\Pi^{hybrid1}$

1. Sender S (setup step 1): The input of S is a program M
   – Choose $a \leftarrow \{0,1\}^k$ uniformly at random. Set $b = F(a)$.

- Choose $y \leftarrow \{0,1\}^\ell$ uniformly at random.
- Generate a key pair $(vk, sgk) = \mathsf{SIG.KeyGen}()$
- Program a stateless token T with the following functionality.
  - Upon receiving a message `unveil` from R, run the verifier V with input b. Forward the messages between R and V. If V rejects, output $\perp$.
  - If V accepts, send y to R.
  - Upon receiving a message (crs, σ), check if $\mathsf{SIG.Verify}_{vk}(\mathsf{crs}, \sigma) = 1$, if not abort.
  - Upon receiving input $(\mathtt{run}, w)$ from R, run M on input w starting from its most recent state, output whatever M outputs and save the new state of M and wait for the next message $(\mathtt{run}, w)$.
- Input T into $\mathcal{T}$ and send (vk, b) to R

2. Receiver R (setup step 2):
   - Wait for the `ready` message from $\mathcal{T}$ and a message (vk, b) from S.
   - Choose $x \leftarrow \{0,1\}^\ell$ uniformly at random.
   - Send x to S.
3. Sender S (setup step 3):
   - Upon receiving a message x from R, set $\mathsf{crs} = x \oplus y$, compute $\sigma = \mathsf{SIG.Sign}_{sgk}(\mathsf{crs})$. Send (y, a, σ) to R. Output crs.
4. Receiver R (setup step 4):
   - Wait for a message (y, a, σ) from S.
   - Check if $\mathsf{F}(a) = b$, if not abort.
   - Run the prover P with input b and witness-input a. Forward the messages between P and $\mathcal{T}$. Let y' be the output of $\mathcal{T}$.
   - If $y \neq y'$, abort.
   - Set $\mathsf{crs} = x \oplus y$.
   - Send (crs, σ) to $\mathcal{T}$
5. Receiver R (Execute Phase): Upon receiving input $(\mathtt{run}, w)$, send $(\mathtt{run}, w)$ to $\mathcal{T}$ and output whatever $\mathcal{T}$ outputs.

5.2 Proof of Security

We will prove computational UC-security against both corrupted sender and receiver.

Corrupted Receiver. We will start with a corrupted receiver. Let $\mathcal{A}_{\mathsf{R}}$ be the dummy-adversary for a corrupted receiver. Let Ext be the knowledge-extractor for the argument of knowledge-system (P, V). We will first state the simulator $\mathcal{S}_R$.

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

- Simulate the first round of a sender S, forward the message (vk, b) to $\mathcal{A}_{\mathsf{R}}$ and store the signature key sgk. Use the token-code T output by S to simulate the token $\mathcal{T}$ for $\mathcal{A}_{\mathsf{R}}$. Let a be the preimage of b under the one-way function F.

- When the simulated S receives a message x from $\mathcal{A}_\mathsf{R}$ do the following. Let crs be the common reference string output by $\mathcal{F}_{wrap}^{hybrid1}$. Set $y = \mathsf{crs} \oplus x$ and $\sigma = \mathsf{SIG.Sign}_{sgk}(\mathsf{crs})$. Output (y, a, σ) to $\mathcal{A}_\mathsf{R}$.
- If $\mathcal{A}_\mathsf{R}$ sends an `unveil`-message to $\mathcal{T}$ do the following. If $\mathcal{A}_\mathsf{R}$ has not yet sent x to S, output $\perp$, regardless if V would accept. Otherwise output the same y as S.
- If $\mathcal{A}_\mathsf{R}$ sends a tuple (crs', σ') to $\mathcal{T}$ do the following. If it holds that $\mathcal{A}_\mathsf{R}$ has not yet sent an x to S or $\mathsf{SIG.Verify}_{vk}(\mathsf{crs}', \sigma') = 0$ output $\perp$.
- If $\mathcal{A}_\mathsf{R}$ sends a tuple $(\mathtt{run}, w)$ to $\mathcal{T}$ do the following. If $\mathcal{A}_\mathsf{R}$ has not yet sent a tuple (crs, σ) to $\mathcal{T}_1$ with $\mathsf{SIG.Verify}_{vk}(\mathsf{crs}, \sigma) = 1$, output $\perp$. Otherwise forward $(\mathtt{run}, w)$ to $\mathcal{F}_{wrap}^{hybrid1}$ and output whatever $\mathcal{F}_{wrap}^{hybrid1}$ outputs.
- Whenever $\mathcal{A}_\mathsf{R}$ sends a message `reset` to $\mathcal{T}$ send `reset` to $\mathcal{F}_{wrap}^{hybrid1}$ and reset the state of $\mathcal{T}$.

Theorem 1. *For every PPT-environment $\mathcal{Z}$, it holds that the random variables $\mathsf{Real}_{\Pi^{hybrid1}}^{\mathcal{A}_\mathsf{R}}(\mathcal{Z})$ and $\mathsf{Ideal}_{\mathcal{F}_{wrap}^{hybrid1}}^{\mathcal{S}_\mathsf{R}}(\mathcal{Z})$ are computationally indistinguishable.*

To prove the theorem, we will show indistinguishability of the following experiments.

Experiment 1. Simulator $\mathcal{S}_1$ simulates the real protocol $\Pi^{hybrid1}$.

Experiment 2. Identical to experiment 1, except that $\mathcal{T}$ outputs $\perp$ if $\mathcal{A}_\mathsf{R}$ sends an `unveil`-query for which the verifier V accepts before $\mathcal{A}_\mathsf{R}$ has sent his coins x to S.

Experiment 3. Identical to experiment 2, except that S's coins y are computed by $y = \mathsf{crs} \oplus y$, where crs is a common random reference string chosen uniformly at random.

Experiment 4. Identical to experiment 3, except that $\mathcal{T}$ aborts if $\mathcal{A}_\mathsf{R}$ sends a a tuple (crs, σ), where it holds that $\mathsf{SIG.Verify}_{vk}(\mathsf{crs}, \sigma) = 1$ and crs has not been signed by S. This is the ideal experiment.

Remarks. Experiment 1 and experiment 2 are computationally indistinguishable, given that the one-way function F is strongly one way. Experiment 2 and experiment 3 are identically distributed, as both x and crs are uniformly and independently distributed. The indistinguishability of experiment 3 and experiment 4 follows easily from the EUF-CMA-security of SIG.

Lemma 2. *From $\mathcal{Z}$'s view, experiment 1 and experiment 2 are computationally indistinguishable, given that F is a one-way function and the argument system (P, V) suffices the proof of knowledge property.*

Proof. From $\mathcal{Z}$'s view, experiment 1 and experiment 2 are identically distributed conditioned to the event that $\mathcal{A}_\mathsf{R}$ does not convince V that it possesses an a such that $\mathsf{F}(a) = b$ before sending his own coins x to S. Thus, a $\mathcal{Z}$ distinguishing between experiment 1 and experiment 2 must succeed in making $\mathcal{A}_{\mathsf{R}}$ convince $\mathcal{T}$ of this before receiving such a value a from S.

Assume that $\mathcal{Z}$ causes this event with non-negligible probability ϵ. We will construct an adversary B that inverts the one-way function F with non-negligible probability. Let $m = \mathsf{poly}(k)$ be an upper bound on the number of `unveil` calls that $\mathcal{A}_{\mathsf{R}}$ sends to $\mathcal{T}$. Let b' be the image on which $\mathcal{B}$ is supposed to invert F. $\mathcal{B}$ first chooses an index $i \in \{1, \dots, m\}$ uniformly at random.

Let $\mathcal{S}_1'$ be a simulator the behaves exactly like $\mathcal{S}_1$, except for one modification. $\mathcal{S}_1'$ sets $b = b'$ instead of generating b by $b = \mathsf{F}(a)$. $\mathcal{B}$ then simulates the interaction between $\mathcal{Z}$ and $\mathcal{S}_1'$ until $\mathcal{A}_{\mathsf{R}}$ makes the i'th `unveil`-call. $\mathcal{B}$ now halts the computation of $\mathcal{Z}$. If the computation of $\mathcal{Z}$ continued after this point, the subsequent messages passed by $\mathcal{A}_{\mathsf{R}}$ correspond to the messages of a malicious prover P^* for the argument system (P, V). Thus, $\mathcal{B}$ can construct P^* from the state of the halted $\mathcal{Z}$ which basically continues the simulation of $\mathcal{Z}$ at its current state and forwards messages between $\mathcal{A}_{\mathsf{R}}$ and an external verifier V. $\mathcal{B}$ can now take the code of P^* and run the extractor $\mathsf{Ext}(b, \mathsf{P}^*)$. Let a be the output of Ext. $\mathcal{B}$ outputs a and terminates.

First notice, that from $\mathcal{Z}$'s view, this simulation is identically distributed to experiment 1. Thus, the event that $\mathcal{A}_{\mathsf{R}}$ succeeds in convincing $\mathcal{T}$ that it possesses a preimage a of b happens with probability at least ϵ. With probability at least $1/m$, the index i chosen by $\mathcal{B}$ matches the index of the proof where this event happens. Therefore, it holds that $\Pr[\langle \mathsf{P}^*, \mathsf{V}(c) \rangle = 1] \geq \epsilon/m$. Due to the proof of knowledge property of the argument system (P, V) it holds that $\Pr[\mathsf{Ext}(b, \mathsf{P}^*) \in w_{L_1}(b)] > \Pr[\langle \mathsf{P}^*, \mathsf{V}(c) \rangle = 1] - \nu \geq \epsilon - \nu$ for some negligible ν. Thus, with probability $\epsilon - \nu$, which is non-negligible, $\mathcal{B}$ outputs a preimage a of b' under F, thus breaking the one-wayness property of F.

Corrupted Sender. Next, we will prove computational UC-security for the case of a corrupted sender. Let $\mathcal{A}_{\mathsf{S}}$ be the dummy-adversary for a corrupted sender and let Sim be the non-black-box simulator for the argument system (P, V), that takes as input a statement (k, b) and the code V^* of a malicious verifier. The simulator $\mathcal{S}_{\mathsf{S}}$ is given as follows.

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

- Let T^* be the token sent by $\mathcal{A}_{\mathsf{S}}$ and (vk, b) be the message sent by $\mathcal{A}_{\mathsf{S}}$.
- Simulate $\mathcal{T}$ using T^*.
- Construct a malicious verifier V^* for the argument system (P, V) that basically simulates the zero-knowledge step of T^* and outputs the state of T^* after the zero-knowledge step is over.
- Run the non-black-box simulator Sim on input b and auxiliary input V^*. The output of Sim is the state of T^* after the zero-knowledge protocol. Continue the simulation of $\mathcal{T}$ from this state until it outputs y.
- Let crs be the common reference string sent by $\mathcal{F}_{wrap}^{hybrid1}$. Set $x = \mathsf{crs} \oplus y$.
- Send x to S. Let (y', a, σ) be the response of S.
- If $y \neq y'$ or $\mathsf{F}(a) \neq b$ abort. Otherwise run $\mathcal{T}$ on input (crs, σ).

– Input T^* with the most recent state taken from $\mathcal{T}$ hardwired into $\mathcal{F}_{wrap}^{hybrid1}$ and output crs to S.

Let $\mathcal{Z}$ be a PPT environment. We will now prove computational indistinguishability between $\mathsf{Real}_{\Pi_{\mathsf{CRS}}}^{\mathcal{A}_{\mathsf{R}}}(\mathcal{Z})$ and $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{CRS}}}^{\mathcal{S}_{\mathsf{R}}}(\mathcal{Z})$ directly, given that the argument system (P, V) is zero-knowledge.

Theorem 2. $\mathsf{Real}_{\Pi^{hybrid1}}^{\mathcal{A}_{\mathsf{S}}}(\mathcal{Z})$ *and* $\mathsf{Ideal}_{\mathcal{F}_{wrap}^{hybrid1}}^{\mathcal{S}_{\mathsf{S}}}(\mathcal{Z})$ *are computationally indistinguishable.*

To prove the theorem, we will show indistinguishability of the following experiments.

Experiment 1. Simulator S_1 simulates the protocol $\Pi^{hybrid1}$

Experiment 2. Identical to experiment 1, except that when R sends an `unveil`-message to T^*, S_2 runs the non-black-box simulator Sim on the verifier V^* (as constructed in the description of the simulator $\mathcal{S}_{\mathsf{S}}$) instead of letting the prover P interact with T^*. S_2 then uses the output of Sim as most recent state to continue the computation of T^*.

Experiment 3. Identical to experiment 2, except that S_3 runs the `unveil`-phase of $\mathcal{T}$ before interacting with $\mathcal{A}_{\mathsf{S}}$, thereby obtaining y. Set $x = \mathsf{crs} \oplus y$, where crs is a uniformly random common reference string. This is the ideal experiment.

Remarks. The indistinguishability of experiment 1 and experiment 2 follows from the computational zero-knowledge property of the argument system (P, V). Experiment 2 and experiment 3 are identically distributed, as the interactions of R with $\mathcal{T}$ and $\mathcal{A}_{\mathsf{S}}$ are independent of one another in both experiments and thus exchangeable.

Lemma 3. *From $\mathcal{Z}$'s view, experiment 1 and experiment 2 are computationally indistinguishable, provided that the argument system* (P, V) *is computational zero-knowledge.*

Proof. Fix a PPT-environment $\mathcal{Z}$. Assume for contradiction that $\mathcal{Z}$ distinguishes experiment 1 and experiment 2 with non-negligible advantage ϵ. We will construct a malicious verifier V^* and a distinguisher $\mathcal{D}$, such that $\mathcal{D}$ distinguishes the random variables $\langle \mathsf{P}(a), \mathsf{V}^* \rangle(b)$ and $\mathsf{Sim}(b, \mathsf{V}^*)$ with advantage ϵ, for some a and b. Fix the random tape of $\mathcal{Z}$ such that $\mathcal{Z}$ with these fixed coins distinguishes between experiment 1 and experiment 2 with advantage ϵ. By a simple averaging argument, such coins must exist. Let (a, b) with $\mathsf{F}(a) = b$ be the fixed tuple that corresponds with this $\mathcal{Z}$. The malicious verifier V^* is constructed as in the description of the simulator. The distinguisher $\mathcal{D}$ is obtained by plugging the machine $\mathcal{Z}$ and $\mathcal{S}_2$ together, with modification to the simulator $\mathcal{S}_2$ that it does not obtain the state of $\mathcal{T}$ by running Sim on b and V^*, but using its own input as state of $\mathcal{T}$. Clearly, if $\mathcal{D}$'s input is distributed by $\langle \mathsf{P}(a), \mathsf{V}^* \rangle(b)$, then $\mathcal{Z}$'s view is distributed identical as in experiment 1. If, on the other hand, $\mathcal{D}$'s input is distributed according to $\mathsf{Sim}(b, \mathsf{V}^*)$, then $\mathcal{Z}$'s view is distributed identical to experiment 2. Thus $\mathcal{D}$ and V^* contradict the zero-knowledge property of (P, V).

5.3 Two Resettable Tokens

In this section we will describe our protocol $\Pi^{hybrid2}$ that implements $\mathcal{F}_{wrap}^{hybrid2}$. Let $\{\mathsf{PRF}\}$ be a family of pseudorandom functions, $\mathsf{com}(\cdot,\cdot)$ be a non-interactive perfectly binding commitment scheme and $\mathsf{SIG} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be an EUF-CMA-secure signature scheme with deterministic signing algorithm. We further need two resettably-sound zero-knowledge argument of knowledge systems. Let $(\mathsf{P}_1, \mathsf{V}_1)$ be such a system for the language $L_1 = \{(x, c) | \exists r : c = \mathsf{com}(x; r)\}$ and $(\mathsf{P}_2, \mathsf{V}_2)$ be such a system for the language $L_2 = \{(vk, c) | \exists \sigma : \mathsf{Verify}(c, \sigma) = 1\}$. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two $\mathcal{F}_{wrap}$ functionalities.

Protocol $\Pi^{hybrid2}$

1. Sender S (setup step 1): The input of S is a program M
   - Sample a pseudorandom function $\mathsf{prf} \leftarrow \{\mathsf{PRF}\}$, generate signature and verification keys $(sgk, vk) = \mathsf{SIG.KeyGen}()$.
   - Choose a random tape for the token T_1 and program T_1 as follows.
     - Set flag $\mathtt{ready} = 0$.
     - Upon receiving input (x, c) from R, run the verifier V_1 with input (x, c). Forward the messages between R and V_1. If V_1 rejects, output $\perp$.
     - If V_1 accepts, compute $y = \mathsf{prf}(c)$ and $\sigma = \mathsf{SIG.Sign}_{sgk}(c)$. Set flag $\mathtt{ready} = 1$ and output (y, σ)
     - Upon receiving input $(\mathtt{run}, w)$ from R, run M on input w starting from its most recent state, output whatever M outputs and save the new state of M and wait for the next message $(\mathtt{run}, w)$.
   - Choose a random tape for the token T_2 and program T_2 as follows.
     - Upon receiving input c from R, run the verifier V_2 with input (vk, c). Forward the messages between R and V_2. If V_2 rejects, output $\perp$.
     - If V_1 accepts, compute $y = \mathsf{prf}(c)$ and output y.
   - Input T_1 into $\mathcal{T}_1$ and T_2 into $\mathcal{T}_2$. Send vk to R.
2. Receiver R (setup step 2):
   - Choose $x \leftarrow \{0, 1\}^k$ uniformly at random.
   - Compute $c = \mathsf{com}(x; r)$ with a uniformly random chosen r.
   - Send (x, c) to $\mathcal{T}_1$ and run the prover P_1 with input (x, c) and witness-input r. Forward the messages between P_1 and $\mathcal{T}_1$. Let (y, σ) be the output of $\mathcal{T}_1$.
   - Check if $\mathsf{SIG.Verify}_{vk}(c, \sigma) = 1$, if not abort.
   - Send c to $\mathcal{T}_2$ and run the prover P_2 with input (vk, c) and witness-input σ. Let y' be the output of $\mathcal{T}_2$.
   - Check whether $y = y'$, if not abort. Otherwise set $\mathsf{crs} = x \oplus y$.
3. Receiver R (Execute Phase): Upon receiving input $(\mathtt{run}, w)$, send $(\mathtt{run}, w)$ to $\mathcal{T}_1$ and output whatever $\mathcal{T}_1$ outputs.

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

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

- Simulate the first round of a sender S and forward the message vk to $\mathcal{A}_\mathsf{R}$ and store sgk. Use the token-codes T_1 and T_2 output by S to simulate $\mathcal{T}_1$ and $\mathcal{T}_2$ for $\mathcal{A}_\mathsf{R}$. Setup a counter $j = 1$.
- If $\mathcal{A}_\mathsf{R}$ sends a message (x, c) to $\mathcal{T}_1$ do the following. Continue the simulation of $\mathcal{T}_1$ until the verifier V_1 either accepts or rejects. If it accepts, even though a tuple (x', y', c', j') has been stored, for some $x' \neq x$ and $c' = c$, output $\perp$. Otherwise, if V_1 accepts, compute $\sigma = \mathsf{SIG.Sign}_{sgk}(c)$ and output (y, σ), for which a tuple (x, y, c, j') has been stored. Send $(\mathtt{reset}, j')$ to $\mathcal{F}_{wrap}^{hybrid2}$. If no such tuple has been stored before, increment j by 1, send $(\mathtt{reset}, j')$ to the $\mathcal{F}_{wrap}^{hybrid2}$ functionality to get a string crs, set $y = x \oplus \mathsf{crs}$, compute $\sigma = \mathsf{SIG.Sign}_{sgk}(c)$, store the tuple (x, y, c, j) and output (y, σ).
- If $\mathcal{A}_\mathsf{R}$ sends a message c to $\mathcal{T}_2$ do the following. Continue the simulation of $\mathcal{T}_2$ until the verifier V_2 either accepts or rejects. If it accepts, check if a tuple (x', y', c', j') with $c' = c$ has been stored. If not, output $\perp$. Otherwise output y'.
- Whenever $\mathcal{A}_\mathsf{R}$ sends a message $\mathtt{reset}$ to $\mathcal{T}_2$, reset $\mathcal{T}_2$.

Theorem 3. *For every PPT-environment $\mathcal{Z}$, it holds that the random variables* $\mathsf{Real}^{\mathcal{A}_\mathsf{R}}_{\Pi^{hybrid2}}(\mathcal{Z})$ *and* $\mathsf{Ideal}^{\mathcal{S}_\mathsf{R}}_{\mathcal{F}^{hybrid2}_{wrap}}(\mathcal{Z})$ *are computationally indistinguishable.*

Again, to prove the theorem, we will show indistinguishability of the following experiments.

Experiment 1. Simulator $\mathcal{S}_1$ simulates the protocol $\Pi^{hybrid2}$. This is the real experiment.

Experiment 2. Identical to experiment 1, except that y is not computed as $y = \mathsf{prf}(c)$ but as follows. If a tuple (x', y', c', j') has been stored with $c' = c$, set $y = y'$. Otherwise, choose crs uniformly at random and set $y = x \oplus \mathsf{crs}$. Also store the tuple (x, y, c).

Experiment 3. Identical to experiment 2, except that $\mathcal{T}_1$ outputs $\perp$ if V_1 accepts, even though a tuple (x', y', c', j') has been stored, with $x' \neq x$ and $c' = c$.

Experiment 4. Identical to experiment 3, except that $\mathcal{T}_2$ outputs $\perp$ if V_2 accepts, even though no tuple (x', y', c', j') with $c' = c$ has been stored. This is the ideal experiment.

Remarks. Given that prf is a pseudorandom function, we can replace the outputs of prf with truly random values, thus experiment 1 and experiment 2 are indistinguishable. The indistinguishability of experiment 2 and experiment 3 follows from the binding property of the commitment scheme com. The event that $\mathcal{T}_1$ outputs $\perp$ even though V_1 accepts happens only when $\mathcal{A}_\mathsf{R}$ manages to convince $\mathcal{T}_1$ that c is a commitment to two different values x and x'. The proof of knowledge property of the argument system $(\mathsf{P}_1, \mathsf{V}_1)$ guarantees that the corresponding unveils r and r' can be extracted with high probability. The indistinguishability of experiment 3 and experiment 4 follows from the EUF-CMA-property of the signature scheme SIG. If $\mathcal{T}_2$ outputs $\perp$, even though the verifier V_2 accepts, then $\mathcal{A}_\mathsf{R}$ has convinced V_2 that it possesses a signature on a commitment c for which it never received a signature σ from $\mathcal{T}_1$. The proof of knowledge property enables us to extract such a forged signature, contradicting the EUF-CMA-security of SIG. For the complete proof refer to the full version of this paper.

Corrupted Sender. We will prove computational UC-security against a corrupted sender S. Let therefore $\mathcal{A}_\mathsf{S}$ be the dummy-adversary and $\mathcal{Z}$ be a PPT-environment. We first state the sender-simulator $\mathcal{S}_\mathsf{S}$. This simulator programs a simulator-token $\mathsf{T}_\mathcal{S}$ and sends this token to $\mathcal{F}_{wrap}^{hybrid2}$.

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

- Let T_1 and T_2 be the inputs of $\mathcal{A}_\mathsf{S}$.
- Simulate $\mathcal{T}_2$ with the code T_2.
- Set $c = \mathsf{com}(0; r)$ for a randomly chosen r.
- Use the code T_2 to construct the code V_2^* of a verifier that runs the verifier-stage of $\mathcal{T}_2$ when its input is c. The output of V_2^* is the the same output that T_2 would provide to R.
- Run Sim_2 with input (vk, c) and witness-input V^*. Let y' be the output of Sim_2
- Program a token $\mathsf{T}_\mathcal{S}$ as follows.
  - Read the common reference string crs provided by the $\mathcal{F}_{wrap}^{hybrid2}$ functionality. Set $x = \mathsf{crs} \oplus y'$.
  - Use the code T_1 to construct the code V_1^* of a verifier that runs the verifier stage of T_1 when its input is (x, c). The output of V_1^* is the the same output that T_1 would provide to R.
  - Run Sim_1 with input (x, c) and witness-input V_1^*. Let (y, σ) be the output of Sim_1.
  - Check whether $y = y'$. If not abort.
  - Upon receiving input $(\mathtt{run}, w)$ from R, run M on input w starting from its most recent state, output whatever M outputs and save the new state of M and wait for the next message $(\mathtt{run}, w)$.
- Input $\mathsf{T}_\mathcal{S}$ into $\mathcal{F}_{wrap}^{hybrid2}$

Theorem 4. *For every PPT-environment $\mathcal{Z}$, it holds that the random variables $\mathsf{Real}_{\Pi^{hybrid2}}^{\mathcal{A}_\mathsf{S}}(\mathcal{Z})$ and $\mathsf{Ideal}_{\mathcal{F}_{wrap}^{hybrid2}}^{\mathcal{S}_\mathsf{S}}(\mathcal{Z})$ are computationally indistinguishable.*

To prove the theorem, we will show indistinguishability of the following experiments.

Experiment 1. Simulator $\mathcal{S}_1$ simulates the protocol $\Pi^{hybrid2}$. This is the real experiment.

Experiment 2. Identical to experiment 1, except that $\mathcal{S}_2$ does the following. Instead of running P_2 with input (vk, c) and witness-input σ, it runs Sim_2 with input (vk, c) and witness-input V_2^* (where V_2^* is as defined in the description of simulator $\mathcal{S}_\mathcal{S}$).

Experiment 3. Identical to experiment 2, except that S_3 does the following. Instead of running P_1 with input (x, c) and witness-input r, it runs Sim_1 with input (x, c) and witness-input V_1^* (where V_1^* is as defined in the description of simulator $\mathcal{S}_\mathcal{S}$).

Experiment 4. Identical to experiment 3, except that the commitment c is computed as $c = \mathsf{com}(0; r)$ instead of $c = \mathsf{com}(x; r)$.

Experiment 5. Identical to experiment 4, except that $\mathcal{S}_5$ first interacts with $\mathcal{T}_2$ and then with $\mathcal{T}_1$, instead of vice versa. Moreover, it sets $x = \mathsf{crs} \oplus y'$ instead of choosing x uniformly at random. This is the ideal experiment.

Remarks. Experiment 1 and experiment 2 are indistinguishable given that the argument system $(\mathsf{P}_1, \mathsf{V}_1)$ is zero-knowledge. Similarly, experiment 2 and experiment 3 are indistinguishable given that $(\mathsf{P}_2, \mathsf{V}_2)$ is zero-knowledge. Both indistinguishability proofs are almost identical to the proof of Lemma 3 and thus omitted. The indistinguishability of experiment 3 and experiment 4 follows straightforwardly from the hiding property of the commitment scheme com. Experiment 4 and experiment 5 are identically distributed, as in both experiments y is independently uniformly distributed.

6 Applications

One application for our protocols is implementing a UC-secure functionality we call conditional decryption, $\mathcal{F}_{\mathsf{CONDEC}}$. In essence, what can be achieved through the conditional decryption functionality is that the computational workload is transfered from the hardware token to the user of the protocol, similar in concept to delegation of computation [23, 24].

The $\mathcal{F}_{\mathsf{CONDEC}}$-functionality takes as sender-input a private key sk for a fully homomorphic encryption scheme FHE. The receiver can send decryption-queries c to $\mathcal{F}_{\mathsf{CONDEC}}$. Before $\mathcal{F}_{\mathsf{CONDEC}}$ decrypts such queries, the receiver must prove that the query is well-formed. This proof is implemented using a universal argument system [25]. Such a conditional decryption has a straightforward application in the context of obfuscation. Given $\mathcal{F}_{\mathsf{CONDEC}}$, a sender initializes the conditional decryption functionality. He can then encrypt an arbitrary program and send it to the receiver. All the receiver has to do is to homomorphically evaluate the encrypted program and send the result together with a proof to the token. If the homomorphic evaluation was carried out correctly, the token will decrypt the result and send it to the receiver. The advantage of this approach is that the sender can obfuscate programs arbitrarily without having to send a new token each time. A similar construction can be found in [26]. Due to space limitations, the above high level description omits several important details which are necessary for a UC-proof. A detailed description of the protocols and a full proof can be found in a preliminary version of this paper [27].

7 Conclusion

In this work, we investigated the cryptographic strength of untrusted resettable hardware tokens in the UC-framework. We devised two protocols that use resettable hardware tokens to realize intermediate functionalities that are sufficient to UC-emulate arbitrary resettable functionalities. The first protocol uses one resettable token and two rounds of interaction in an initialization phase, after which no further interaction takes place. In the second protocol, messages and hardware tokens are only passed from the sender to the receiver. However, this protocol requires two resettable token. Given these protocols, it is possible to UC-realize any resettable two-party computation. We showed that a completely non-interactive coin-toss with only one resettable token is impossible. Thus, both our protocols are optimal if one of the conditions (no interaction or just a single token) is dropped.

References

1. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (Im)possibility of Obfuscating Programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (2001)
2. Canetti, R., Fischlin, M.: Universally Composable Commitments. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 19–40. Springer, Heidelberg (2001)
3. Canetti, R., Goldreich, O., Goldwasser, S., Micali, S.: Resettable zero-knowledge (extended abstract). In: STOC, pp. 235–244 (2000)
4. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: STOC, pp. 494–503 (2002)
5. Barak, B., Lindell, Y., Vadhan, S.P.: Lower bounds for non-black-box zero knowledge. In: FOCS (2003)
6. Deng, Y., Lin, D.: Instance-Dependent Verifiable Random Functions and Their Application to Simultaneous Resettability. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 148–168. Springer, Heidelberg (2007)
7. Barak, B., Goldreich, O., Goldwasser, S., Lindell, Y.: Resettably-sound zero-knowledge and its applications. In: FOCS, pp. 116–125 (2001)
8. Deng, Y., Goyal, V., Sahai, A.: Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. In: FOCS, pp. 251–260 (2009)
9. Deng, Y., Feng, D., Goyal, V., Lin, D., Sahai, A., Yung, M.: Resettable Cryptography in Constant Rounds – The Case of Zero Knowledge. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 390–406. Springer, Heidelberg (2011)
10. Cho, C., Ostrovsky, R., Scafuro, A., Visconti, I.: Simultaneously Resettable Arguments of Knowledge. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 530–547. Springer, Heidelberg (2012)

11. Goyal, V., Sahai, A.: Resettably Secure Computation. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 54–71. Springer, Heidelberg (2009)
12. Goyal, V., Maji, H.K.: Stateless cryptographic protocols. In: FOCS, pp. 678–687 (2011)
13. Goyal, V., Ishai, Y., Mahmoody, M., Sahai, A.: Interactive Locking, Zero-Knowledge PCPs, and Unconditional Cryptography. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 173–190. Springer, Heidelberg (2010)
14. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS, pp. 136–145 (2001)
15. Canetti, R., Kushilevitz, E., Lindell, Y.: On the limitations of universally composable two-party computation without set-up assumptions. J. Cryptology 19(2), 135–167 (2006)
16. Fischlin, M.: Universally Composable Oblivious Transfer in the Multi-party Setting. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 332–349. Springer, Heidelberg (2006)
17. Goyal, V., Ishai, Y., Sahai, A., Venkatesan, R., Wadia, A.: Founding Cryptography on Tamper-Proof Hardware Tokens. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 308–326. Springer, Heidelberg (2010)
18. Katz, J.: Universally Composable Multi-party Computation Using Tamper-Proof Hardware. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 115–128. Springer, Heidelberg (2007)
19. Choi, S.G., Katz, J., Schröder, D., Yerukhimovich, A., Zhou, H.S.: (Efficient) universally composable two-party computation using a minimal number of stateless tokens. IACR Cryptology ePrint Archive 2011, 689 (2011)
20. Naor, M., Yung, M.: Universal one-way hash functions and their cryptographic applications. In: STOC, pp. 33–43 (1989)
21. Rompel, J.: One-way functions are necessary and sufficient for secure signatures. In: STOC, pp. 387–394 (1990)
22. Goldreich, O.: Foundations of Cryptography. Cambridge University Press (2001)
23. Gennaro, R., Gentry, C., Parno, B.: Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg (2010)
24. Chung, K.-M., Kalai, Y., Vadhan, S.: Improved Delegation of Computation Using Fully Homomorphic Encryption. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 483–501. Springer, Heidelberg (2010)
25. Barak, B., Goldreich, O.: Universal arguments and their applications. In: IEEE Conference on Computational Complexity, pp. 194–203 (2002)
26. Bitansky, N., Canetti, R., Goldwasser, S., Halevi, S., Kalai, Y.T., Rothblum, G.N.: Program Obfuscation with Leaky Hardware. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 722–739. Springer, Heidelberg (2011)
27. Döttling, N., Mie, T., Müller-Quade, J., Nilges, T.: Basing obfuscation on simple tamper-proof hardware assumptions. IACR Cryptology ePrint Archive 2011, 675 (2011)