A conference version of this paper appeared at ProvSec 2015 [DKMN15a]. This is the full version.

From Stateful Hardware to Resettable Hardware Using Symmetric Assumptions

Nico Döttling [1,2], Daniel Kraschewski[2,3], Jörn Müller-Quade[3], and Tobias Nilges[3]

[1]Aarhus University, Denmark
[2]TNG Technology Consulting GmbH, Munich, Germany
[3]Karlsruhe Institute of Technology

Abstract

Universally composable multi-party computation is impossible without setup assumptions. Motivated by the ubiquitous use of secure hardware in many real world security applications, Katz (EUROCRYPT 2007) proposed a model of tamper-proof hardware as a UC-setup assumption. An important aspect of this model is whether the hardware token is allowed to hold a state or not. Real world examples of tamper-proof hardware that can hold a state are expensive hardware security modules commonly used in mainframes. Stateless, or resettable hardware tokens model cheaper devices such as smartcards, where an adversarial user can cut off the power supply, thus resetting the card's internal state.

A natural question is how the stateful and the resettable hardware model compare in their cryptographic power, given that either the receiver or the sender of the token (and thus the token itself) might be malicious. In this work we show that any UC-functionality that can be implemented by a protocol using a single untrusted stateful hardware token can likewise be implemented using a single untrusted resettable hardware token, assuming only the existence of one-way functions.

We present two compilers that transform UC-secure protocols in the stateful hardware model into UC-secure protocols in the resettable hardware model. The first compiler can be proven secure assuming merely the existence of one-way functions. However, it (necessarily) makes use of computationally rather expensive non-black-box techniques. We provide an alternative second compiler that replaces the expensive non-black-box component of the first compiler by few additional seed OTs. While this second compiler introduces the seed OTs as additional setup assumptions, it is computationally very efficient.

Keywords: tamper-proof hardware, universal composability, protocol compilers

[1]The authors acknowledge support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, within which part of this work was performed; and also from the CFEM research center (supported by the Danish Strategic Research Council) within which part of this work was performed.

[2]Supported by European Research Commission Starting Grant no. 279447.

[3]Part of work done while at Technion, Israel. Supported by the European Union's Tenth Framework Programme (FP10/2010-2016) under grant agreement no. 259426 – ERC Cryptography and Complexity.

1 Introduction

Tamper-proof hardware has gained a lot of interest in the design of UC-secure [Can01] protocols. Many cryptographic tasks that are impossible in the plain model can be realized using tamper-proof hardware, e.g. *Program Obfuscation* [BCG$^+$11]. It turns out that several flavors of tamper-proof hardware have different cryptographic strengths. On the one hand there are *stateful* tokens, which allow for very efficient and UC-secure oblivious transfer (OT) protocols even without computational assumptions [GIS$^+$10, DKMQ11, DKMQ12]. On the other hand, *resettable*, or equivalently *stateless*, tokens are strictly weaker: it was shown that unconditional OT cannot be achieved with stateless tokens [GIMS10]. Nevertheless the distinction between both models is very relevant, because in real-world applications it is considered to be much more difficult to manufacture stateful tamper-proof tokens than stateless or resettable tokens. Removing the dependency on stateful hardware by an improved protocol design can greatly simplify the manufacturing process and also reduce the costs. This leads to the following question:

Is it possible to implement any UC-functionality using a single resettable *tamper-proof hardware and assuming only one-way functions?*

We answer this question affirmatively. We shall first motivate the setting. There are protocol compilers by Kilian [Kil88] and Ishai et al. [IPS08] basing general (UC-)secure multi-party computation (MPC) on OT without additional computational assumptions. Recent results in the area of efficient information-theoretically secure OT protocols include [DKMQ11, DKMQ12]. Their results, however, are based on stateful tamper-proof hardware. Considering the above-mentioned impossibility result of Goyal et al. [GIMS10], who show that unconditionally secure OT with *any* number of resettable tokens is impossible, it becomes obvious that converting a protocol like [DKMQ12] such that only resettable tokens are necessary cannot be achieved without further assumptions.

One of the weakest common assumptions in cryptography is the existence of one-way functions and it turns out that these suffice for our goal. It was previously known that one-way functions suffice for UC-secure OT with stateless tokens [GIS$^+$10], but they need many tokens to obtain this result. For our solution, instead of adapting an OT protocol such that it uses a resettable token, we present two compilers that transform any protocol based on stateful tokens into a protocol based on resettable tokens. This allows for many improvements in previous protocols, because any statistically secure protocol using a single untrusted stateful token can be transformed into a computationally secure protocol using a single untrusted resettable token and one-way functions.

1.1 Our contribution

We present two compilers that basically use the same technique to achieve resettability of the token: The sender has to authenticate the query that the receiver wants to input into the token. For the first compiler, we generalize and improve upon a technique that is implicit in [DMMQN13], where resettability is obtained by having the sender sign every query the receiver will provide to the token. The second compiler is stated in the OT-hybrid model and makes no further assumptions.

In more detail, given a protocol where a stateful token is sent from the sender to the receiver, we extend the protocol as follows. For the first compiler, a key pair for a signature scheme is created by the sender. For each token invocation, the receiver commits to its input and sends the commitment to the sender. The sender signs the commitment and sends it back to the receiver.

At this point, care must be taken not to introduce a channel between the sender and the token. Otherwise, the token and/or the sender party could gather complete information about the messages sent and received by the receiver party. This would make aborts possible that are correlated with the receivers secret input and thus cannot be simulated in the UC framework. Therefore, the receiver does not show any signature to the token, but instead provides a resettably-sound zero-knowledge argument of knowledge (rsZKAoK) of the commitment and the signature. Recent results by Chung et al. [CPS13] and Bitansky and Paneth [BP13] show that such resettably-sound zero-knowledge arguments of knowledge can be based on the existence of one-way functions. This technique guarantees that any information generated by the sender during a protocol run remains oblivious to the token.

Additionally, we present a compiler that works in the OT-hybrid model (without any computational assumptions) and can, e.g., be used to implement many OT instances from few "seed OTs" and a resettable token. The raw version of this compiler uses one OT per bit sent from the receiver to the token, but by using Merkle trees (or sig-com trees [CPS13] respectively; see Section 2.5), we can compress the number of bits to be authenticated. The main idea is that the sender only has to authenticate a single message of small size, namely the root of such a tree. To stay true to the goal of using only one-way functions, however, we cannot directly use a Merkle tree. Instead, we show how the sig-com scheme proposed by [CPS13] can be applied to our scenario. The same compression technique applies to both our compilers, which also allows us to keep the amount of proofs for the rsZKAoK-based compiler at a minimum. For protocols that use more than one token, our compilers can be invoked successively, replacing all the stateful tokens by resettable tokens.

Our rsZKAoK-based compiler can be used to obtain several improvements on existing protocols concerning resettable hardware. We highlight the following contribution: The combination of the OT protocol by Döttling et al. [DKMQ12] with our compiler yields a round-efficient OT protocol based on one-way functions and a single untrusted resettable token. This yields the first OT-protocol using a single resettable token and only symmetric computational assumptions. Prior to our result, the best known approach to obtain UC-secure OT with a single resettable token was to use the token to generate a common reference string [DMMQN13] and use an efficient OT-protocol in the CRS-model, e.g. the protocol of Peikert et al. [PVW08]. Implementing OT in the common reference string model, however, requires much stronger computational assumptions (e.g., doubly enhanced trapdoor permutations, number-theoretic or lattice assumptions). Alternatively, [GIS+10] presented a protocol for OT based on resettable hardware and one-way functions, but their construction needs polynomially many tokens. Thus the question of obtaining OT from a single resettable hardware token using only one-way functions was left open by prior works.

Döttling et al. [DMMQN13] and Choi et al. [CKS+14] showed that, if only a single resettable token is issued, then even simple functionalities cannot be implemented only with black-box techniques. We circumvent this impossibility result by using resettably-sound zero-knowledge argument of knowledge, which are inherently non-black-box. Moreover, Goyal et al. [GIMS10] showed there exists no unconditionally secure protocol implementing OT using any number of resettable tokens. Thus, computational assumptions are necessary to implement OT using a single resettable token. Considering the computational assumptions and number of resettable tokens used, our compiler yields an optimal UC-secure OT-protocol.

1.2 Efficiency

The compilers require one round of interaction between the token issuer and the receiver per token message. With a few exceptions in the area of non-interactive computation [GIS$^+$10, DKMQ12, DMMQN13], protocols based on tamper-proof hardware already require interaction between the sender and the receiver. Moreover, in the scenario of a single token, Döttling et al. [DMMQN13] show that interaction is necessary to UC-realize any functionality based on resettable hardware. Thus the induced overhead is minimal with respect to communication, even more so since typically token-based protocols are constant round.

The main efficiency drawback is incurred by the use of non-black-box zero-knowledge. However, in current protocols [CPS13, BP13] the honest prover is not required to execute a universal argument, so that the efficiency is comparable to a general zero-knowledge protocol. With a zero-knowledge protocol tailored to the statements in our protocol the efficiency can be further improved.

1.3 Further Related Work

The model of tamper-proof hardware considered in this paper was introduced by Katz [Kat07]. Further formalizations of different tamper-proof hardware tokens can be found in [GIS$^+$10] and [DMMQN13]. Physically uncloneable functions (PUFs) [Pap01, BFSK11, OSVW13] only allow for MPC if the PUFs are not malicious [Rüh10, DFK$^+$14], but this is out of the scope of our work.

Resettability was first considered in the context of zero-knowledge protocols. The case of a resettable prover was analyzed by Canetti et al. [CGGM00] while the case of resettable verifiers was treated by Barak et al. [BGGL01] with several follow up works, e.g. [DGS09, CPS13, BP13]. Later, simultaneously resettable zero-knowledge protocols were presented [DGS09, DFG$^+$11, BP13]. These works made it possible to transform stateful into stateless protocols. Goyal and Sahai [GS09] present a compiler that transforms any semi-honest secure protocol into a resettably secure protocol using similar techniques to ours. They also show that general resettable MPC with honest majority is possible where all parties are resettable. Another compiler due to Goyal and Maji [GM11] allows to compile almost any semi-honest secure protocol into a fully resettable protocol. However, neither [GS09] nor [GM11] achieve UC-security.

While all of the above-mentioned works do not make use of tamper-proof hardware, Chandran et al. [CGS08] present a protocol for general UC-secure MPC with resettable tokens, but they need to exchange several tokens and rely on strong cryptographic assumptions, namely enhanced trapdoor permutations. Goyal et al. [GIS$^+$10] construct a protocol for UC-secure MPC assuming only one-way functions, but they also need polynomially many resettable tokens.

In the context of statistically UC-secure OT, Goyal et al. [GIS$^+$10] present a protocol using several stateful tokens. The protocols of Döttling et al. [DKMQ11, DKMQ12] improve upon this result by achieving UC-secure OT using only a single stateful token. In [GIMS10] it was shown that statistically secure OT is impossible with stateless tokens (unless parties can encapsulate tokens into each other), but statistical commitments are possible. Their commitment construction was improved by [DS13] to achieve UC-security. Given an upper bound on the number of resets, Döttling et al. [DKMN15b] show that resettable tamper-proof hardware allows for unconditionally UC-secure OT. Another recent result by [CKS$^+$14] implements UC-secure OT from CRHFs and two bidirectionally exchanged stateless tokens. Leaky tokens, which reveal additional information to malicious receivers, were considered in [BCG$^+$11, PSW14], but this is again out of the scope of our work.

2 Preliminaries

In the following we denote by k a security parameter. We abbreviate probabilistic polynomial time by PPT. We use the standard notions of negligible functions, statistical indistinguishability and computational indistinguishability.

2.1 The UC-Framework

The *Universal Composability* (UC) framework was introduced by Canetti [Can01]. It differentiates between an *ideal model* and a *real model.* In the real model an adversary $\mathcal{A}$ coordinates the behavior of all corrupted parties while the uncorrupted parties follow the protocol. An environment $\mathcal{Z}$ representing an outside observer can read all messages and outputs of the protocol parties. The same setup also holds for the ideal model, but the adversary is replaced by a simulator $\mathcal{S}$ that simulates the behavior of $\mathcal{A}$, and all participating parties are replaced by dummy parties that only pass on any message that they receive. Security is proven by comparing a protocol Π in the real model with an ideal functionality $\mathcal{F}$ in the ideal model. An ideal functionality is secure per definition and represents a trusted third party that provides a functionality. All communication between a protocol party and the ideal functionality is assumed to be authenticated. Tamper-proof hardware is also modeled as an ideal functionality, further details can be found in Section 3.

By $\mathsf{Real}_{\Pi}^{\mathcal{A}}(\mathcal{Z})$ we denote the output of the environment $\mathcal{Z}$ when interacting with the real model, by $\mathsf{Ideal}_{\mathcal{F}}^{\mathcal{S}}(\mathcal{Z})$ we denote the output of $\mathcal{Z}$ when interacting with the ideal model. A protocol is said to compose securely if for any environment $\mathcal{Z}$, which is plugged to either the ideal model or the real model, the view is (computationally, statistically or perfectly) indistinguishable.

We assume static corruption (i.e. the adversary does not adaptively change corruption) and prove our results in this framework.

2.2 Signature Schemes

A signature scheme SIG consists of three PPT algorithms KeyGen, Sign and Verify.

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

We will use existentially unforgeable (EUF-CMA secure) signatures and will briefly recall the security definition. The experiment creates a key pair $(\mathsf{sgk}, \mathsf{vk})$. An adversary $\mathcal{A}$ has access to a verification key vk and a signing oracle $\mathcal{O}^{\mathsf{Sign}_{\mathsf{sgk}}(\cdot)}$. The adversary can now query the oracle with messages and obtains signatures to these messages. If $\mathcal{A}$ manages to create a signature σ^* for an arbitrary message m without querying $\mathcal{O}^{\mathsf{Sign}_{\mathsf{sgk}}(\cdot)}$ with m such that $\mathsf{Verify}_{\mathsf{vk}}(m, \sigma^*) = 1$ it wins the experiment.

A signature scheme SIG is called EUF-CMA-secure if the probability that a PPT adversary wins the above specified experiment is negligible. EUF-CMA secure signature schemes can be constructed from one-way functions [NY89, Rom90].

2.3 Commitment Schemes

We will use 2-move commitment schemes in our compiler. In such a commitment-scheme, the receiver first chooses a key k, sends k to the sender of the commitment, who computes a commitment $c = \mathsf{com}_k(m; r)$ for a message m using randomness r and sends c to the receiver. The sender can unveil the commitment by sending (m, r) to the receiver, who checks if $c = \mathsf{com}_k(m; r)$ holds.

We will require such a commitment scheme to be statistically binding, which means that for a given commitment $c = \mathsf{com}_k(m; r)$, the unveil (m, r) is unique, except with negligible probability over the choice of k. Naor [Nao91] constructs 2-move statistically binding commitment schemes using only pseudorandom generators, if one considers their first message from the receiver as the key. As the latter can be constructed from one-way functions [HILL99], this yields a 2-move statistically binding commitment scheme based on one-way functions.

2.4 Resettably-Sound Zero-Knowledge Arguments of Knowledge

Due to the fact that our protocol runs in the resettable token model, we use resettably-sound zero-knowledge arguments of knowledge for our proofs. We give a definition for resettably-sound zero-knowledge arguments of knowledge.

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 resettable, 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 resettable PPT V^* it holds that 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 $\Pr[\mathsf{Ext}(x, \mathsf{P}^*) \in w_L(x)] > \Pr[\langle \mathsf{P}^*, \mathsf{V}\rangle(x) = 1] - \nu$.

It will be convenient to rephrase the computational zero-knowledge property as follows. Given that a simulator Sim exists with $\{\langle \mathsf{P}(w), \mathsf{V}^*\rangle(x)\} \approx_c \{\mathsf{Sim}(x, \mathsf{V}^*)\}$, we can always construct a *prover-simulator* $\mathsf{P}_{\mathsf{Sim}}$ such that it holds that $\{\langle \mathsf{P}(w), \mathsf{V}^*\rangle(x)\} \approx_c \{\langle \mathsf{P}_{\mathsf{Sim}}(\mathsf{V}^*), \mathsf{V}^*\rangle(x)\}$. Such a prover-simulator can be constructed as follows. $\mathsf{P}_{\mathsf{Sim}}$ first runs $\mathsf{Sim}(x, \mathsf{V}^*)$ to obtain a simulated view of V^*. From this view it takes the prover-messages and uses these prover-messages in its own interaction with V^*. Thus it holds $\{\langle \mathsf{P}_{\mathsf{Sim}}(\mathsf{V}^*), \mathsf{V}^*\rangle(x)\} \approx_c \{\mathsf{Sim}(x, \mathsf{V}^*)\}$ and we are done.

Recent constructions of rsZK arguments of knowledge are based on one-way functions [CPS13, BP13].

2.5 Sig-Com Schemes

As an alternative to collision resistant hash functions, Chung et al. [CPS13] propose sig-com schemes. They show that such a scheme is compressing and has a collision resistance property similar to collision resistant hash functions, but requires only one-way functions. In comparison to hash functions, however, sig-com schemes require interaction between two parties: one party creates the signature and verification keys, and sends the verification key to the other party. The other party sends a commitment to its input and obtains a signature on the commitment, i.e. the party with the signature key acts as a signature oracle. This separation is due to the fact that if the party that holds the input for the sig-com tree also possesses the secret key to the signature scheme, the security of the signature scheme (and hence the collision resistance property) does no longer hold. The commitments to the input are necessary, because otherwise the sender could abort depending on the received message. The commit-then-sign step can be applied to create a tree analogous to Merkle trees.

Definition 2 ([CPS13])**.** Let $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a strong length-n signature scheme and let com be a non-interactive commitment scheme. Define $\mathsf{SIG}' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Verify}')$ to be a triple of PPT machines defined as follows:

- $\mathsf{Gen}' = \mathsf{Gen}$.
- $\mathsf{Sign}'_{\mathsf{sgk}}(m)$: compute a commitment $c = \mathsf{com}(m; r)$ using a uniformly selected r, and let $\sigma = \mathsf{Sign}_{\mathsf{sgk}}(c)$; output (σ, r).
- $\mathsf{Verify}'_{\mathsf{vk}}(m, \sigma, r)$: output 1 iff $\mathsf{Verify}_{\mathsf{vk}}(\mathsf{com}(m; r), \sigma) = 1$.

Definition 3 ([CPS13])**.** Let $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a strong length-n signature scheme, let com be a non-interactive commitment scheme, and let $\mathsf{SIG}' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Verify}')$ be the sig-com scheme corresponding to SIG and com. Let $(\mathsf{sgk}, \mathsf{vk})$ be a key pair of SIG', and s be a string of length 2^d. A sig-com tree for s w. r. t. $(\mathsf{sgk}, \mathsf{vk})$ is a complete binary tree of depth d, defined as follows.

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

Note that sig-com trees have a collision resistance property in the following sense: no adversary with oracle access to a signature oracle SIG can output a root and a sequence of signatures for both 0 and 1 for any leaf γ. This property stems from the binding property of the commitment and the unforgeability of the signature scheme.

3 Ideal Functionalities

In this section we define the ideal functionalities we will use later. Here we only consider the two-party case with a sender S and a receiver R. The following definition for a stateful wrapper functionality is based on [Kat07, GIS$^+$10].

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

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

We use the wrapper functionality for resettable functionalities as defined in [DMMQN13].

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

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

In the sequel, we will use the notation $\mathcal{P}$ for programs (given as code, Turing-machine etc.) and $\mathcal{T}$ for the instance of the wrapper-functionality $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$, resp. $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$, in which $\mathcal{P}$ runs.

4 Compiler

In the following we present two compilers that allow to convert a protocol that makes a single call to a *stateful* token into a protocol that uses a *resettable* token.

We need to make some assumptions on the structure of the underlying stateful protocol Π_s. W.l.o.g the protocol can be divided into the following phases.

- A setup phase in which the sender sends a token program T to $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$.
- Communication between the sender and the receiver.
- An invocation of the token by the receiver.

The token program from the setup phase will be used in the resettable protocol as well, albeit the setup phase will be extended by additional steps. Any interaction between the two parties of the protocol (not the communication with the token) will be left untouched.

The basic idea underlying our compilers is to let the sender *authenticate* the message for the token, while being oblivious of what the actual message to the token is. Instead of invoking the stateful token in the underlying protocol directly, we will additionally insert a communication step with the sender. Though the receiver can still perform reset-attacks, it will not be able to change its input after a reset.

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

4.1 Protocol Using Resettably-Sound Zero-Knowledge

4.1.1 Outline

The compiler $\mathcal{C}_{\mathrm{ZK}}$ (cf. Figure 1) alters the underlying protocol as follows. Before the execution of Π_s a signature key-pair $(\mathsf{sgk}, \mathsf{vk})$ and a key k for a commitment scheme (cf. Section 2.3) are created by the sender and sent to the receiver. Then Π_s is carried out. When the token code of the underlying protocol is sent to the wrapper functionality, the sender chooses a seed s for a pseudorandom function and constructs a new token.

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

We stress that it is essential that the token cannot learn the signature σ on the commitment c, otherwise both token and sender have a covert channel by which they can communicate, which cannot be simulated. To eliminate this channel, we use a zero-knowledge proof which hides the signature from the token.

4.1.2 Proof of Security

Corrupted Receiver. Let $\mathcal{A}_{\mathsf{R}}$ be the dummy-adversary for a corrupted receiver for the protocol Π_r. We will construct an adversary $\mathcal{A}'_{\mathsf{R}}$ (cf. Figure 2) against the protocol Π_s. $\mathcal{A}'_{\mathsf{R}}$ needs to simulate a resettable token to $\mathcal{A}_{\mathsf{R}}$, while it has itself access to a non-resettable stateful token.

Lemma 1. *For every PPT-environment $\mathcal{Z}$, it holds that the random variables $\mathsf{Real}_{\Pi_r}^{\mathcal{A}_{\mathsf{R}}}(\mathcal{Z})$ and $\mathsf{Real}_{\Pi_s}^{\mathcal{A}'_{\mathsf{R}}}(\mathcal{Z})$ are computationally indistinguishable.*

As Π_s is UC-secure, there exists a simulator $\mathcal{S}_{\mathsf{R}}$ such that $\mathsf{Real}_{\Pi_s}^{\mathcal{A}'_{\mathsf{R}}}(\mathcal{Z}) \approx \mathsf{Ideal}_{\mathcal{F}}^{\mathcal{S}_{\mathsf{R}}}(\mathcal{Z})$. This yields the desired $\mathsf{Real}_{\Pi_r}^{\mathcal{A}_{\mathsf{R}}}(\mathcal{Z}) \approx \mathsf{Ideal}_{\mathcal{F}}^{\mathcal{S}_{\mathsf{R}}}(\mathcal{Z})$.

Proof. Let $\mathcal{Z}$ be a PPT environment. We will prove the indistinguishability of $\mathsf{Real}_{\Pi_r}^{\mathcal{A}_{\mathsf{R}}}(\mathcal{Z})$ and $\mathsf{Real}_{\Pi_s}^{\mathcal{A}'_{\mathsf{R}}}(\mathcal{Z})$ by a series of indistinguishable hybrid experiments.

Hybrid $\mathcal{H}_0$. Simulator $\mathcal{S}_0$ simulates $\mathsf{Real}_{\Pi_r}^{\mathcal{A}_{\mathsf{R}}}$.

Hybrid $\mathcal{H}_1$. Identical to $\mathcal{H}_0$, except that simulator $\mathcal{S}_1$ replaces the pseudo"-random-function $\mathsf{F}_s(\cdot)$ by a random-oracle H.

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

Let $\mathcal{F}$ be a two-party UC-functionality. Let com_k denote a 2-move commitment scheme and $\mathsf{SIG} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ an EUF-CMA secure signature-scheme. Let (P, V) be a resettably-sound zero-knowledge argument-of-knowledge system for the NP-language $L = \{(\mathsf{vk}, \mathsf{inp}) | \exists \sigma, c, r : \mathsf{Verify}_{\mathsf{vk}}(c, \sigma) = 1 \wedge c = \mathsf{com}(\mathsf{inp}; r)\}$. Further let F be a pseudorandom function.

Input: Protocol Π_s UC-implementing $\mathcal{F}$ in the $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$-hybrid model.

Output: Protocol Π_r UC-implementing $\mathcal{F}$ in the $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$-hybrid model.

Setup (Before execution of Π_s):

- **(Sender)** Generate a key pair $(\mathsf{sgk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and choose a key $k \in \{0,1\}^\lambda$ for the commitment scheme uniformly at random. Send $(\mathtt{setup}, \mathsf{vk}, k)$ to R.
- **(Receiver)** Upon receiving a message $(\mathtt{setup}, \mathsf{vk}, k)$ from S, store vk and k.

Rewriting the token-code:

(Sender) Once S inputs a token code T into $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$ do the following.

- Choose a seed $s \in \{0,1\}^\lambda$ for the pseudorandom function F uniformly at random.
- Construct a token-code T' which upon receiving a message $(\mathtt{input}, \mathsf{inp})$ from R sets up a verifier V with input $(\mathsf{vk}, \mathsf{inp})$, random-tape $\mathsf{F}_s(\mathsf{inp})$ and runs V. It forwards the messages sent by V to R and vice versa. If V rejects, it aborts. If V accepts, it continues the execution of T with input inp and forwards T's output to R.
- Input T' into $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$.

Token-invocation:

- **(Receiver)** Let inp be R's input to the token. Compute $c = \mathsf{com}_k(\mathsf{inp}; r)$ and send $(\mathtt{query}, c)$ to S.
- **(Sender)** Upon receiving a message $(\mathtt{query}, c)$ from R, compute $\sigma = \mathsf{Sign}_{\mathsf{sgk}}(c)$. Send $(\mathtt{ack}, \sigma)$ to R.
- **(Receiver)** Upon receiving a message $(\mathtt{ack}, \sigma)$ from S, check if $\mathsf{Verify}_{\mathsf{vk}}(c, \sigma) = 1$ holds, if not abort. Otherwise send $(\mathtt{input}, \mathsf{inp})$ to the token. Setup a prover P with input $(\mathsf{vk}, \mathsf{inp})$, witness-input (σ, c, r) and run P. Forward the messages sent by P to the token and vice versa. Continue R's computation once the token outputs out.

Figure 1: Stateless compiler using resettably-sound zero-knowledge.

Hybrid $\mathcal{H}_2$. Identical to $\mathcal{H}_1$, except for the following. $\mathcal{S}_2$ checks – after V accepts – if a tuple $(\mathsf{inp}', \mathsf{out}')$ has already been stored. If so and $\mathsf{inp}' \neq \mathsf{inp}$, it aborts. Moreover, if no such tuple exists it will store $(\mathsf{inp}, \mathsf{out})$, where out is the output of the token. From the view of $\mathcal{Z}$, this is identical to $\mathsf{Real}_{\Pi_s}^{\mathcal{A}'_{\mathsf{R}}}$.

Computational indistinguishability of $\mathcal{H}_0$ and $\mathcal{H}_1$ follows straightforwardly by the pseudorandomness of the pseudorandom-function F_s. The interesting part is the computational indistinguishability of $\mathcal{H}_1$ and $\mathcal{H}_2$.

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

- **Setup:** Generate a key pair $(\mathsf{sgk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^k)$ and choose $k \in \{0,1\}^n$ uniformly at random. Send $(\mathtt{setup}, \mathsf{vk}, k)$ to $\mathcal{A}_{\mathsf{R}}$. Setup a simulated-random oracle H.
- **Token-Invocation:**
  - Once $\mathcal{A}_{\mathsf{R}}$ wants to send a message $(\mathtt{query}, c)$ to S, compute $\sigma = \mathsf{Sign}_{\mathsf{sgk}}(c)$. Send $\mathtt{query}$ to S. Once S responds with $\mathtt{ack}$, send $(\mathtt{ack}, \sigma)$ to $\mathcal{A}_{\mathsf{R}}$.
  - Once $\mathcal{A}_{\mathsf{R}}$ wants to input a message $(\mathtt{input}, \mathsf{inp})$ to $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$, setup a verifier V with input $(\mathsf{vk}, \mathsf{inp})$, random-tape $\mathsf{H}(\mathsf{inp})$ and run V. Forward the messages sent by V to $\mathcal{A}_{\mathsf{R}}$ and vice versa. If V rejects, abort. If V accepts, check if a tuple $(\mathsf{inp}', \mathsf{out}')$ has been stored. If yes and it holds $\mathsf{inp}' \neq \mathsf{inp}$, abort. If yes and it holds $\mathsf{inp}' = \mathsf{inp}$, send out' to $\mathcal{A}_{\mathsf{R}}$. If no, send inp to $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$, let out be the corresponding output, send out to $\mathcal{A}_{\mathsf{R}}$ and store the tuple $(\mathsf{inp}, \mathsf{out})$.

Figure 2: Adversary-simulator $\mathcal{A}'_{\mathsf{R}}$ for $\mathcal{C}_{\text{ZK}}$.

We claim that $\mathcal{H}_1$ and $\mathcal{H}_2$ are computationally indistinguishable, provided that the argument system (P, V) fulfills the computational resettable soundness property, the commitment scheme com is statistically binding and the signature scheme SIG is EUF-CMA secure.

Clearly, if $\mathcal{S}_2$ does not abort after V accepts, the views of $\mathcal{Z}$ are identical in $\mathcal{H}_1$ and $\mathcal{H}_2$. We will thus show that this event happens at most with negligible probability, establishing indistinguishability of $\mathcal{H}_1$ and $\mathcal{H}_2$.

Since the commitment-scheme com is statistically binding, the event that there exist distinct (inp_1, r_1) and (inp_2, r_2) with $\mathsf{com}_k(\mathsf{inp}_1; r_1) = \mathsf{com}_k(\mathsf{inp}_2; r_2)$ has only negligible probability (over the choice of k). We can thus assume that each commitment c has a unique unveil (inp, r).

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

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

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

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

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

$\mathcal{B}$ now runs the extractor Ext on P^* and obtains a witness (σ^*, c^*, r^*) for a statement $(\mathsf{vk}, \mathsf{inp}^*)$. If it holds $\mathsf{Verify}_{\mathsf{vk}}(c^*, \sigma^*) = 1$, then $\mathcal{B}$ outputs the forge (c^*, σ^*). Otherwise it outputs $\perp$.

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

Corrupted Sender. We move on to prove the security against a corrupted sender by stating a simulator (cf. Figure 3).

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

- **Setup:** Once $\mathcal{A}_{\mathsf{S}}$ sends a message $(\mathtt{setup}, \mathsf{vk}, k)$ store vk and k.
- **Rewriting the Token-code:** Once S inputs a token code T^* into $\mathcal{F}_{\text{wrap}}^{\text{resettable}}$ construct a token-code $\mathsf{T}^\dagger$ with the following functionality.

  Upon receiving a message $(\mathtt{input}, \mathsf{inp})$ from R, run T^* with input $(\mathtt{input}, \mathsf{inp})$. Halt the computation of T^* and construct a corrupted verifier V^* from T^*. Setup a prover-simulator $\mathsf{P}_{\mathsf{Sim}}$ with input $(\mathsf{vk}, \mathsf{inp})$ and witness-input V^* and run $\mathsf{P}_{\mathsf{Sim}}$. Forward the messages between $\mathsf{P}_{\mathsf{Sim}}$ and T^* and vice versa. Once T^* outputs $\mathtt{out}$, send $\mathtt{out}$ to R.

  Input $\mathsf{T}^\dagger$ into $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$.
- **Token-Invocation:** Upon receiving a message $\mathtt{query}$ from R, compute $c = \mathsf{com}_k(0; r)$. Send $(\mathtt{query}, c)$ to $\mathcal{A}_{\mathsf{S}}$. Let $(\mathtt{ack}, \sigma)$ be the output of $\mathcal{A}_{\mathsf{S}}$. Check if it holds $\mathsf{Verify}_{\mathsf{vk}}(c, \sigma) = 1$, if not abort. Otherwise send $\mathtt{ack}$ to R.

Figure 3: Adversary-simulator $\mathcal{A}'_{\mathsf{S}}$ for $\mathcal{C}_{\text{ZK}}$.

Lemma 2. *For every PPT-environment $\mathcal{Z}$, it holds that the random variables $\mathsf{Real}_{\Pi_r}^{\mathcal{A}_{\mathsf{S}}}(\mathcal{Z})$ and $\mathsf{Real}_{\Pi_s}^{\mathcal{A}'_{\mathsf{S}}}(\mathcal{Z})$ are computationally indistinguishable.*

Again, as Π_s is UC-secure, there exists a simulator $\mathcal{S}_{\mathsf{S}}$ such that $\mathsf{Real}_{\Pi_s}^{\mathcal{A}'_{\mathsf{S}}}(\mathcal{Z}) \approx \mathsf{Ideal}_{\mathcal{F}}^{\mathcal{S}_{\mathsf{S}}}(\mathcal{Z})$, which yields the desired $\mathsf{Real}_{\Pi_r}^{\mathcal{A}_{\mathsf{S}}}(\mathcal{Z}) \approx \mathsf{Ideal}_{\mathcal{F}}^{\mathcal{S}_{\mathsf{S}}}(\mathcal{Z})$

Proof. Let $\mathcal{Z}$ be a PPT environment. We will prove the indistinguishability of $\mathsf{Real}_{\Pi_r}^{\mathcal{A}_{\mathsf{S}}}(\mathcal{Z})$ and $\mathsf{Real}_{\Pi_s}^{\mathcal{A}'_{\mathsf{S}}}(\mathcal{Z})$ by a series of indistinguishable hybrid experiments.

Hybrid $\mathcal{H}_0$. Simulator $\mathcal{S}_0$ simulates $\mathsf{Real}_{\Pi_r}^{\mathcal{A}_{\mathsf{S}}}$.

Hybrid $\mathcal{H}_1$. Identical to $\mathcal{H}_0$, except that during invocation of the token, R does not setup and run a prover P with input $(\mathsf{vk}, \mathsf{inp})$ and witness-input (σ, c, r), but instead runs the prover-simulator $\mathsf{P}_{\mathsf{Sim}}$ with input $(\mathsf{vk}, \mathsf{inp})$ and witness-input V^*, where V^* is a corrupted verifier that is constructed from T^*.

Hybrid $\mathcal{H}_2$. Identical to $\mathcal{H}_1$, except that the commitment c sent to $\mathcal{A}_{\mathsf{S}}$ is computed by $c = \mathsf{com}_k(0; r)$ instead of $c = \mathsf{com}_k(\mathsf{inp}; r)$. From the view of $\mathcal{Z}$, this is identical to $\mathsf{Real}_{\Pi_s}^{\mathcal{A}'_{\mathsf{S}}}$.

Indistinguishability of the hybrids $\mathcal{H}_0$ and $\mathcal{H}_1$ follows directly from the computational zero-knowledge property of the system (P, V). Since the commitment scheme com is computationally binding, the hybrids $\mathcal{H}_1$ and $\mathcal{H}_2$ are computationally indistinguishable from the view of $\mathcal{Z}$ as well. This can be established by a simple hybrid-argument, where a $\mathcal{Z}$ distinguishing the two experiments can be used to to break the hiding-property of com. □

Remark 3. The above compiler can easily be extended to allow for multiple messages. Then, in each step of the token invocation the token receiver has to query the sender on a commitment and provide a proof to the token, that this commitment was signed by the sender. For each message, a counter is added such that the receiver cannot query the token "‘out-of-sync"’. If the token is reset, its counter will not match the counter of the sender and thus the token will abort.

4.2 Protocol Using UC-Secure Seed-OTs

4.2.1 Outline

The compiler $\mathcal{C}_{\mathrm{OT}}$ depicted in Figure 4 adds a step to the underlying protocol Π_s, which authenticates the token input. This time, the authentication is done by using UC-secure OTs. Before the execution of Π_s, the token sender creates two random strings (s_0^i, s_1^i) for every bit of the message inp that the receiver will input into the token. Let $\mathsf{inp}(i)$ denote the i-th bit of inp. During the setup, the receiver obtains one of these random strings, namely $s^i_{\mathsf{inp}(i)}$, for each of his input bits. Since the receiver does not learn any $s^i_{1-\mathsf{inp}(i)}$, he is bitwise committed to his input, while the sender does not learn anything about it.

All random values $((s_0^1, s_1^1), \ldots, (s_0^k, s_1^k))$ that the sender created are stored in the token functionality. When the token is invoked on input $(\mathsf{inp}, (s^1_{j_1}, \ldots, s^k_{j_k}))$, the tokens checks that these values are consistent with the random values of the OTs. If that is the case, the token will evaluate the underlying token functionality on inp and forward the output out.

4.2.2 Proof of Security

Please note that the security reduction is information-theoretic, but depending on the realization of $\mathcal{F}$, the protocol might still only be computationally secure.

Corrupted Receiver Let $\mathcal{A}_{\mathsf{R}}$ be the dummy-adversary for a corrupted sender for the protocol Π_r. We will construct an adversary $\mathcal{A}'_{\mathsf{R}}$ against the protocol Π_s (cf. Figure 5).

Lemma 4. *For every (PPT-)environment $\mathcal{Z}$, it holds that the random variables $\mathsf{Real}^{\mathcal{A}_{\mathsf{R}}}_{\Pi_r}(\mathcal{Z})$ and $\mathsf{Real}^{\mathcal{A}'_{\mathsf{R}}}_{\Pi_s}(\mathcal{Z})$ are indistinguishable.*

Proof. The only difference between $\mathsf{Real}^{\mathcal{A}_{\mathsf{R}}}_{\Pi_r}(\mathcal{Z})$ and $\mathsf{Real}^{\mathcal{A}'_{\mathsf{R}}}_{\Pi_s}(\mathcal{Z})$ is the abort of $\mathcal{A}'_{\mathsf{R}}$ in case $\mathsf{inp}' \neq \mathsf{inp}$. For this event to happen, $\mathcal{A}_{\mathsf{R}}$ has to guess a string $s^i_j \in S$ of length λ for any $i \in \{1, \ldots, k\}, j \in \{0, 1\}$. The probability for this event is obviously negligible in the security parameter λ. □

Since the protocol Π_s is UC-secure, there exists a simulator S_{R} such that $\mathsf{Real}^{\mathcal{A}'_{\mathsf{R}}}_{\Pi_s}(\mathcal{Z}) \approx \mathsf{Ideal}^{\mathcal{S}_{\mathsf{R}}}_{\mathcal{F}}(\mathcal{Z})$. Thus, $\mathsf{Real}^{\mathcal{A}_{\mathsf{R}}}_{\Pi_r}(\mathcal{Z}) \approx \mathsf{Ideal}^{\mathcal{S}_{\mathsf{R}}}_{\mathcal{F}}(\mathcal{Z})$ follows from the above lemma.

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

Let $\mathcal{F}$ be a two-party UC-functionality. Let $k = |\mathsf{inp}|$ be the input length of the token receiver's message inp to the token in Π_s. S and R have access to k $\mathcal{F}_{\mathrm{OT}}$-functionalities.

Input: Protocol Π_s UC-implementing $\mathcal{F}$ in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$-hybrid model.

Output: Protocol Π_r UC-implementing $\mathcal{F}$ in the $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$-hybrid model.

Setup (Before execution of Π_s):

- **(Sender)** S creates $2k$ random strings $S = ((s_0^1, s_1^1), \dots, (s_0^k, s_1^k)), s_j^i \in \{0,1\}^\lambda$ and inputs them into the k $\mathcal{F}_{\mathrm{OT}}$-functionalities.
- **(Receiver)** R inputs $\mathsf{inp}(1), \dots, \mathsf{inp}(k)$ into the corresponding $\mathcal{F}_{\mathrm{OT}}$ and obtains $(s^1_{\mathsf{inp}(1)}, \dots, s^k_{\mathsf{inp}(k)})$.

Rewriting the token-code:

(Sender) Once S inputs a token code T into $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$ do the following. Construct a token-code T' which upon receiving a message $(\mathtt{input}, \mathsf{inp}, (s^1_{j_1}, \dots, s^k_{j_k})), j \in \{0,1\}$ from R checks that $s^i_j \in S$ for all $i \in \{1, \dots, k\}$. If this is the case, it continues the execution of T with input inp and forwards whatever T outputs. Then input T' into $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$.

Token-invocation:

1. R sends a message $\mathtt{query}$ to S, who replies with a message $\mathtt{ack}$.
2. R sends the previously obtained random strings with the message $(\mathtt{input}, \mathsf{inp}, (s^1_{j_1}, \dots, s^k_{j_k}))$ to the token and continues the normal computation once the token outputs out.

Figure 4: Stateless compiler using k seed-OTs.

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

- **Setup:** Create $2k$ random strings $S = ((s_0^1, s_1^1), \dots, (s_0^k, s_1^k))$ and simulate k $\mathcal{F}_{\mathrm{OT}}$ instances. Obtain the choice-bits $c_i, i \in \{1, \dots, k\}$ and thereby learn inp.
- **Token-Invocation:**
  - Once $\mathcal{A}_{\mathsf{R}}$ sends $\mathtt{query}$, forward $\mathtt{query}$ to S. Once S responds with $\mathtt{ack}$, send $\mathtt{ack}$ to $\mathcal{A}_{\mathsf{R}}$.
  - Once $\mathcal{A}_{\mathsf{R}}$ wants to input a message $(\mathtt{input}, \mathsf{inp}', (s^1_{j_1}, \dots, s^k_{j_k}))$ to $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$, check if $\mathsf{inp} = \mathsf{inp}'$ and $s^i_j \in S \forall i$, if not abort. Query $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$ on inp and let out be the result. Send out to $\mathcal{A}_{\mathsf{R}}$.

Figure 5: Adversary-simulator $\mathcal{A}'_{\mathsf{R}}$ for $\mathcal{C}_{\mathrm{OT}}$.

Corrupted Sender Let $\mathcal{A}_{\mathsf{S}}$ be the dummy-adversary for a corrupted sender for the protocol Π_r. We will construct an adversary $\mathcal{A}'_{\mathsf{S}}$ against the protocol Π_s (cf. Figure 6).

Lemma 5. *For every (PPT-)environment $\mathcal{Z}$, it holds that the random variables $\mathsf{Real}^{\mathcal{A}_{\mathsf{S}}}_{\Pi_r}(\mathcal{Z})$ and $\mathsf{Real}^{\mathcal{A}'_{\mathsf{S}}}_{\Pi_s}(\mathcal{Z})$ are indistinguishable.*

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

- **Setup:** Simulate the $\mathcal{F}_{\mathrm{OT}}$-functionalities and obtain all values $((s_0^1, s_1^1), \ldots, (s_0^k, s_1^k))$.
- **Rewriting the Token-code:** Once S inputs a token code T^* into $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{resettable}}$, create a token code $\mathsf{T}^\dagger$ as follows:
  - On input a message $(\mathtt{input}, \mathsf{inp})$, select $(s^1_{\mathsf{Inp}(1)}, \ldots, s^k_{\mathsf{Inp}(k)})$.
  - Run T^* with input $(\mathtt{input}, \mathsf{inp}, (s^1_{\mathsf{Inp}(1)}, \ldots, s^k_{\mathsf{Inp}(k)}))$ and let out be the result. Send out to R.

Input $\mathsf{T}^\dagger$ into $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$.

Figure 6: Adversary-simulator $\mathcal{A}'_{\mathsf{S}}$ for $\mathcal{C}_{\mathrm{OT}}$.

Proof. $\mathsf{Real}_{\Pi_r}^{\mathcal{A}_{\mathsf{S}}}(\mathcal{Z})$ and $\mathsf{Real}_{\Pi_s}^{\mathcal{A}'_{\mathsf{S}}}(\mathcal{Z})$ are identically distributed, because after obtaining all labels $((s_0^1, s_1^1), \ldots, (s_0^k, s_1^k))$, a normal protocol run is simulated. □

5 Optimizations

Recall that the compiler $\mathcal{C}_{\mathrm{ZK}}$ can straightforwardly be extended to allow for multiple messages between token and receiver. However, this would lead to an inefficient zero-knowledge proof for each message. Also, it seems difficult to change the compiler $\mathcal{C}_{\mathrm{OT}}$ such that it allows for more than a single message due to the fixed amount of seed-OTs.

In case that the receiver has non-adaptive queries for the token, these problems can be overcome. By non-adaptive queries, we mean that the i-th token query does not depend on the $(i-1)$-th query. A very simple solution is to just concatenate all messages into a single message and have the sender authenticate this message. However, this needs quite a lot of seed-OTs and also the amount of data that has to be sent to the sender is very large.

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

This improvement leads to a single message of small size during the authentication step of $\mathcal{C}_{\mathrm{ZK}}$ and $\mathcal{C}_{\mathrm{OT}}$ respectively. This construction has one drawback: the Merkle tree relies on collision resistant hash functions. Considering our initial goal to achieve a compiler using only one-way functions, we replace the Merkle tree with the recent construction of *sig-com trees* [CPS13].

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

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

6 Implications

In this section we will briefly discuss the implications of applying our compiler to existing protocols. We want to focus on UC-secure oblivious transfer protocols. Previous constructions based on resettable tokens were either dependent on the fact that several hardware tokens had to be exchanged [CKS+14] or made use of stronger computational assumptions [CGS08]. In fact, it was shown that, using only black-box techniques, OT can only be achieved by exchanging two tokens [CKS+14] or by sending a large amount of tokens in one direction [CGS08, GIS+10].

The only known solution using a single resettable hardware token can be constructed by using the recent work of [DMMQN13] (which makes inherent use of non-black-box techniques). They create a CRS with a single resettable token and by plugging in an efficient OT protocol in the CRS model, e.g. [PVW08], an OT protocol using a single resettable token can be obtained. OT protocols in the CRS model, however, cannot be based on one-way functions and thus stronger cryptographic assumptions are needed. In the context of stateful tokens, very efficient constructions are known, e.g. [DKMQ12]. By plugging the protocol of Döttling et al. [DKMQ11, DKMQ12] into one of our compilers, we obtain the most efficient OT-protocol based on resettable hardware to date (the protocol of [DKMQ11, DKMQ12] only gives an a priori fixed amount of OTs). The compiler $\mathcal{C}_{\mathrm{ZK}}$ uses non-black-box techniques, so the above-mentioned impossibility result does not hold. We can further improve the efficiency of this protocol by performing random OTs with non-adaptive token inputs. This allows us to use the optimization from Section 5, thereby making only a single call to the sender. Additionally, the compiler $\mathcal{C}_{\mathrm{OT}}$ allows to extend a fixed amount of UC-OTs (the seed-OTs of the compiler) to a (fixed but independent) number of UC-OTs by using the protocol of [DKMQ11, DKMQ12].

References

[BCG+11] Nir Bitansky, Ran Canetti, Shafi Goldwasser, Shai Halevi, Yael Tauman Kalai, and Guy N. Rothblum. Program obfuscation with leaky hardware. In *ASIACRYPT*, pages 722–739, 2011.

[BFSK11] Christina Brzuska, Marc Fischlin, Heike Schröder, and Stefan Katzenbeisser. Physically uncloneable functions in the universal composition framework. In *CRYPTO*, pages 51–70, 2011.

[BGGL01] Boaz Barak, Oded Goldreich, Shafi Goldwasser, and Yehuda Lindell. Resettably-sound zero-knowledge and its applications. In *FOCS*, pages 116–125, 2001.

[BP13] Nir Bitansky and Omer Paneth. On the impossibility of approximate obfuscation and applications to resettable cryptography. In *STOC*, 2013.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *STOC*, pages 235–244, 2000.

[CGS08] Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for uc secure computation using tamper-proof hardware. In *EUROCRYPT*, pages 545–562, 2008.

[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 *TCC*, pages 638–662, 2014.

[CPS13] Kai-Min Chung, Rafael Pass, and Karn Seth. Non-black-box simulation from one-way functions and applications to resettable security. In *STOC*, 2013.

[DFG$^+$11] Yi Deng, Dengguo Feng, Vipul Goyal, Dongdai Lin, Amit Sahai, and Moti Yung. Resettable cryptography in constant rounds - the case of zero knowledge. In *ASIACRYPT*, pages 390–406, 2011.

[DFK$^+$14] Dana Dachman-Soled, Nils Fleischhacker, Jonathan Katz, Anna Lysyanskaya, and Dominique Schröder. Feasibility and infeasibility of secure computation with malicious pufs. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 405–420, 2014.

[DGS09] Yi Deng, Vipul Goyal, and Amit Sahai. Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. In *FOCS*, pages 251–260, 2009.

[DKMN15a] Nico Döttling, Daniel Kraschewski, Jörn Müller-Quade, and Tobias Nilges. From stateful hardware to resettable hardware using symmetric assumptions. In *Provable Security - 9th International Conference, ProvSec 2015, Kanazawa, Japan, November 24-26, 2015, Proceedings*, pages 23–42, 2015.

[DKMN15b] Nico Döttling, Daniel Kraschewski, Jörn Müller-Quade, and Tobias Nilges. General statistically secure computation with bounded-resettable hardware tokens. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*, pages 319–344, 2015.

[DKMQ11] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In *TCC*, pages 164–181, 2011.

[DKMQ12] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. David & goliath oblivious affine function evaluation - asymptotically optimal building blocks for universally composable two-party computation from a single untrusted stateful tamper-proof hardware token. *IACR Cryptology ePrint Archive*, 2012:135, 2012.

[DMMQN13] Nico Döttling, Thilo Mie, Jörn Müller-Quade, and Tobias Nilges. Implementing resettable uc-functionalities with untrusted tamper-proof hardware-tokens. In *TCC*, pages 642–661, 2013.

[DS13] Ivan Damgård and Alessandra Scafuro. Unconditionally secure and universally composable commitments from physical assumptions. In *ASIACRYPT*, pages 100–119, 2013.

[GIMS10] Vipul Goyal, Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. Interactive locking, zero-knowledge pcps, and unconditional cryptography. In *CRYPTO*, pages 173–190, 2010.

[GIS$^+$10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.

[GM11] Vipul Goyal and Hemanta K. Maji. Stateless cryptographic protocols. In *FOCS*, pages 678–687, 2011.

[GS09] Vipul Goyal and Amit Sahai. Resettably secure computation. In *EUROCRYPT*, pages 54–71, 2009.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.

[Kat07] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In *EUROCRYPT*, pages 115–128, 2007.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2), 1991.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *STOC*, pages 33–43, 1989.

[OSVW13] Rafail Ostrovsky, Alessandra Scafuro, Ivan Visconti, and Akshay Wadia. Universally composable secure computation with (malicious) physically uncloneable functions. In *EUROCRYPT*, 2013.

[Pap01] Ravikanth Srinivasa Pappu. *Physical One-Way Functions.* PhD thesis, MIT, 2001.

[PSW14] Manoj Prabhakaran, Amit Sahai, and Akshay Wadia. Secure computation using leaky tokens. In *ICALP*, 2014.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, pages 554–571, 2008.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *STOC*, pages 387–394, 1990.

[Rüh10] Ulrich Rührmair. Oblivious transfer based on physical unclonable functions. In *TRUST*, pages 430–440, 2010.