A conference version of this paper appears at ICITS 2015. This is the full version.

Weakening the Isolation Assumption of Tamper-proof Hardware Tokens

Rafael Dowsley Jörn Müller-Quade Tobias Nilges

Institute of Theoretical Informatics, Karlsruhe Institute of Technology
Am Fasanengarten 5, Geb. 50.34, 76131 Karlsruhe, Germany
`rafael.dowsley@kit.edu`; `mueller-quade@kit.edu`; `tobias.nilges@kit.edu`

Abstract

Recent results have shown the usefulness of tamper-proof hardware tokens as a setup assumption for building UC-secure two-party computation protocols, thus providing broad security guarantees and allowing the use of such protocols as buildings blocks in the modular design of complex cryptography protocols. All these works have in common that they assume the tokens to be completely isolated from their creator, but this is a strong assumption. In this work we investigate the feasibility of cryptographic protocols in the setting where the isolation of the hardware token is weakened.

We consider two cases: (1) the token can relay messages to its creator, or (2) the creator can send messages to the token after it is sent to the receiver. We provide a detailed characterization for both settings, presenting both impossibilities and information-theoretically secure solutions.

Keywords: Hardware Tokens, Isolation Assumption, UC security, One-Time Memory, Oblivious Transfer.

1 Introduction

Tamper-proof hardware tokens are a valuable resource for designing cryptographic protocols. It was shown in a series of recent papers that tamper-proof hardware tokens can be used as a cryptographic setup assumption to obtain Universally Composable (UC) [5] secure two-party computation protocols [20, 22, 18, 17, 13], thus achieving solutions that are secure according to one of the most stringent cryptographic models and can be used as buildings blocks in the modular design of complex cryptography protocols. Döttling et al. [13] showed that even a single tamper-proof hardware token generated by one of the mutually distrusting parties is enough to obtain information-theoretical security in the UC framework.

All these works have in common that the tokens are assumed to be completely isolated from their creator. In light of recent events this assumption becomes questionable at the least, apart from the fact that the tokens could contain internal clocks, which can be exploited in conjunction with the activation time to send information into the device (or to make the abort behavior dependent on the activation time, which is not modeled in the UC framework). We highlight that this problem lies skew to leakage and side-channel attacks, e.g. [2, 26], where a malicious token receiver tries to extract some of the contents of the token, i.e. the *tamper-resilience assumption* is weakened. In contrast, we consider a weakened *isolation assumption*. A similar scenario was studied by Damgård et al. [11], but only for a bandwidth-restricted channel and computational security. They showed that a partial physical separation of parties, e.g. in a token with a low-bandwidth covert channel, allows to perform UC-secure multiparty computation under standard cryptographic assumptions.

We consider an *unrestricted* channel and information-theoretical security. In this scenario, communication in both directions between the token and its creator without any restriction obviously renders the token useless as a setup assumption. Thus, there remain two different kinds of communication that can be considered to weaken the isolation assumption: either the tokens' creator can send messages to the tokens, or the tokens can send messages to their creator. While we deem the first case to be more realistic, we consider both cases. We emphasize that these one-way channels are available only for malicious parties and thus are not used by the honest parties during the protocol execution. This scenario is not directly comparable with the one by Damgård et al. [11], since here a broadband communication channel is available, but it is only one-way. This leads to the following question:

Is it possible to obtain UC-secure protocols even if there exists a broadband one-way communication channel between the tokens and their creator?

In this work, we provide a broad characterization from a feasibility standpoint for both malicious incoming and outgoing communication between the tokens and their creator. For our solutions, we only require that one party can create hardware tokens. We thus call this party Goliath, while the receiver of the token is called David and cannot create tokens of its own.

In more detail, we show that with one-way channels into the tokens, it is possible to basically use the One-Time Memory (OTM) protocol using two tokens of Döttling et al. [13] to obtain an information-theoretically UC-secure OTM with aborts (i.e., a malicious token creator can change the abort behavior of the token at runtime, which is unavoidable if one-way channels into the tokens are available) and we also provide a computationally UC-secure OTM protocol from a single token. Additionally, it is possible to obtain information-theoretically UC-secure Oblivious Transfer (OT) from a single hardware token. We prove an impossibility result for unconditionally secure OTM with a single token.

Concerning one-way channels from the tokens to their creator, we show that it is impossible to obtain even information-theoretically secure OT. We provide an information-theoretically UC-secure commitment scheme, which can then be used to obtain a computationally UC-secure OTM protocol with known techniques [25].

Further related work. Apart from the model of tamper-proof hardware as formalized by Katz [20], also weaker models such as resettable hardware tokens were proposed, e.g. [18]. With resettable hardware, it is not possible to obtain information-theoretically secure oblivious transfer [17], while commitments are still possible [17, 12]. Thus, the main focus of this research direction are efficient protocols based on computational assumptions while minimizing the amount of communication and tokens [6, 18, 21, 2, 14, 8]. Further results about hardware tokens can be founded in [7, 3, 9, 19, 15, 1].

Another UC hardware setup assumption are physically uncloneable functions (PUFs) [24, 4, 23], which have recently gained increasing interest. It was shown that PUFs can be used to achieve oblivious transfer [27] and UC-secure commitments [12]. However, if the PUFs can be created maliciously, oblivious transfer is impossible [10].

2 Preliminaries

2.1 Notation

We use standard information-theoretic measures: by $H(\cdot)$ we denote Shannon entropy, $H(\cdot|\cdot)$ denotes conditional entropy and $I(\cdot;\cdot)$ denotes the mutual information. Let in the following λ denote a security parameter. We use the cryptographic standard notions of negligible functions, as well as computational/statistical/perfect indistinguishability.

2.2 Model

We state and prove our results in the Universal Composability (UC) framework of Canetti [5] that allows for arbitrary composition of protocols. In this framework an ideal functionality $\mathcal{F}$ that captures the desired security requirements has to be modeled. A protocol Π that is supposed to instantiate $\mathcal{F}$ runs in the real world, where an adversary $\mathcal{A}$ can corrupt protocol parties. To prove the UC-security of Π, it has to be shown that there exists a simulator $\mathcal{S}$ that only interacts with the ideal functionality and simulates the behavior of any $\mathcal{A}$ in such a way that any environment $\mathcal{Z}$ that is plugged either into the real protocol or the simulated protocol cannot distinguish the real protocol run of Π from a simulated one.[1] For our results we assume static corruption, i.e. the adversary cannot adaptively corrupt protocol parties.

Target Functionalities. Ideally one would like to use tamper-proof hardware tokens to realize One-Time Memory (OTM) [16], as in the case where the token is modeled as being completely isolated from its creator [13]. See Figure 1 for the OTM functionality definition. This primitive resembles oblivious transfer, but the receiver can make his choice at any point in time and the sender is not notified about this event. OTM allows to build One-Time Programs [16, 18].

Impossibility of Realizing OTMs. Note that in the hybrid execution with a token and a channel into the token, a dishonest sender $\mathcal{G}$ has the ability to send an abortion message to $\mathcal{T}$ at any time, thus changing its abort behavior. In the ideal execution on the other hand, once the OTM functionality goes to the ready state, it is not possible to change its output/abort behavior

[1]In the case of computational security we allow the simulator to be expected polynomial time.

Functionality $\mathcal{F}^{\mathrm{OTM}}$

Parametrized by a security parameter λ. The variable F_{state} is initialized with wait.

Creation. Upon receiving a message (create, sid, $\mathcal{G}, \mathcal{D}, s_0, s_1$) from $\mathcal{G}$ verify if $F_{state} = \mathsf{wait}$ and $s_0, s_1 \in \{0,1\}^\lambda$; else abort. Next, set $F_{state} \leftarrow \mathsf{sent}$, store ($\mathsf{sid}, \mathcal{G}, \mathcal{D}, s_0, s_1$) and send the message ($\mathsf{created}$, sid, $\mathcal{G}, \mathcal{D}$) to the adversary.

Deliver. Upon receiving a message ($\mathsf{deliver}$, sid, $\mathcal{G}, \mathcal{D}$) from the adversary, verify that $F_{state} = \mathsf{sent}$; else abort. Next, set $F_{state} \leftarrow \mathsf{ready}$, and send ($\mathsf{ready}$, sid, $\mathcal{G}, \mathcal{D}$) to $\mathcal{D}$.

Choice. Upon receiving a message (choice, sid, $\mathcal{G}, \mathcal{D}, c$) from $\mathcal{D}$ check if $F_{state} = \mathsf{ready}$; else abort. Next, set $F_{state} \leftarrow \mathsf{dead}$ and send (output, sid, $\mathcal{G}, \mathcal{D}, s_c$) to $\mathcal{D}$.

Figure 1: The One-Time Memory functionality.

anymore. Therefore it is not possible to realize the OTM functionality based on tokens that can receive communication from a malicious $\mathcal{G}$.

OTM with Abort. Given the above fact that online changes in the abort behavior are inherent in the setting with one-way communication into the token, we introduce an OTM functionality with abort, see Figure 2. For such a functionality, there is an initial delivering phase after which the adversary can only let the execution proceed correctly or switch off the functionality whenever he wants (independent of David inputs); but he cannot change the values stored in the functionality.

3 The Case of Incoming Communication

We first show that the existing solution of Döttling, Kraschewski and Müller-Quade [13] for OTM with 2 tokens can be modified to UC-realize OTM with abort. Then we show that using a single token, it is impossible to obtain an information-theoretically secure OTM protocol, if Goliath can send messages to the token. We sketch how a information-theoretically UC-secure OT protocol from a single token can be obtained and give a construction of a compuationally UC-secure OTM protocol from a single hardware token.

The formalization of the ideal functionality for stateful tamper-proof hardware tokens in this section uses a wrapper functionality as in the previous works [20, 22, 13], but as one-way communication from the token issuer to the token is now allowed, the wrapper functionality needs to be modified to capture this fact. A sender $\mathcal{G}$ (Goliath) provides as input to $\mathcal{F}^{\text{stateful}}_{\text{wrap-owc}}$ a deterministic Turing machine $\mathcal{T}$ (the token). Note that stateful tokens can be hard-coded with sufficiently long randomness tapes. The receiver $\mathcal{D}$ (David) can query $\mathcal{F}^{\text{stateful}}_{\text{wrap-owc}}$ to run $\mathcal{T}$ with inputs of his choice and receives the output produced by the token. The current state of $\mathcal{T}$ is stored between consecutive queries. In addition, and in order to capture the one-way communication property, we add the possibility of Goliath sending messages to the token, in which case $\mathcal{T}$ is run on the received string and changes to a new state. The complete description of the functionality is shown in Figure 3. This model captures the fact that on the one hand

Functionality $\mathcal{F}^{\text{OTM-with-Abort}}$

Parametrized by a security parameter λ. The variable F_{state} is initialized with wait and F_{abort} with $\top$. If any message other than ($\mathsf{switch\ on}$, sid, $\mathcal{G}, \mathcal{D}$) is received while $F_{abort} = \bot$, the functionality aborts.

Creation. Upon receiving a message (create, sid, $\mathcal{G}, \mathcal{D}, s_0, s_1$) from $\mathcal{G}$ verify if $F_{state} = \mathsf{wait}$ and $s_0, s_1 \in \{0,1\}^\lambda$; else abort. Next, set $F_{state} \leftarrow \mathsf{sent}$, store ($\mathsf{sid}, \mathcal{G}, \mathcal{D}, s_0, s_1$) and send the message ($\mathsf{created}$, sid, $\mathcal{G}, \mathcal{D}$) to the adversary.

Overwrite. Upon receiving a message ($\mathsf{overwrite}$, sid, $\mathcal{G}, \mathcal{D}, s_0', s_1'$) from $\mathcal{A}$ verify if $F_{state} = \mathsf{sent}$ and $s_0', s_1' \in \{0,1\}^\lambda$; else abort. Set $s_0 \leftarrow s_0'$; $s_1 \leftarrow s_1'$.

Deliver. Upon receiving a message ($\mathsf{deliver}$, sid, $\mathcal{G}, \mathcal{D}$) from the adversary, verify that $F_{state} = \mathsf{sent}$; else abort. Next, set $F_{state} \leftarrow \mathsf{ready}$, and send ($\mathsf{ready}$, sid, $\mathcal{G}, \mathcal{D}$) to $\mathcal{D}$.

Choice. Upon receiving a message (choice, sid, $\mathcal{G}, \mathcal{D}, c$) from $\mathcal{D}$ check if $F_{state} = \mathsf{ready}$; else abort. Next, set $F_{state} \leftarrow \mathsf{dead}$ and send (output, sid, $\mathcal{G}, \mathcal{D}, s_c$) to $\mathcal{D}$.

Switch Off. Upon receiving a message ($\mathsf{switch\ off}$, sid, $\mathcal{G}, \mathcal{D}$) from $\mathcal{A}$ set $F_{abort} \leftarrow \bot$.

Switch On. Upon receiving a message ($\mathsf{switch\ on}$, sid, $\mathcal{G}, \mathcal{D}$) from $\mathcal{A}$, set $F_{abort} \leftarrow \top$.

Figure 2: The One-Time Memory with Abort functionality.

the token cannot send messages to its creator, and on the other hand David cannot access the code or the internal state of $\mathcal{T}$.

3.1 Unconditionally Secure OTM with Two Tokens

Our solution is to use the non-interactive version of the protocol due to Döttling, Kraschewski and Müller-Quade [13]. The only function of Goliath in this protocol is creating the two tokens and sending them to David. David, on the other hand, interacts with both tokens in order to obtain his output and to check the correctness of the protocol execution. Intuitively, one of the tokens is used to generate a commitment to the input values and to send the input values encrypted using one-time pads. The second token only contains a random affine function which can be evaluated only a single time and allows David to recover the one-time pad key corresponding to one of the inputs. The specifications of the tokens can be found in Figure 4 and Figure 5. In the protocol David initially interacts with the token which has the inputs in order to obtain the commitments and the ciphertexts. After this point David considers the OTM as delivered. Then, whenever he wants to choose the input to be received, he simply queries the token that has the affine function on the appropriate input and obtains the one-time pad that he needs in order to recover his desired value. The description of the protocol is presented in Figure 6.

The fact that the protocol securely realizes $\mathcal{F}^{\text{OTM-with-Abort}}$ follows from a straightforward modification of the original security proof by Döttling et al. [13], which considered the same protocol but with isolated tokens and proved that it realizes $\mathcal{F}^{\text{OTM}}$ (i.e., without aborts) in such scenario.

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

Parametrized by a security parameter λ and a polynomial upper bound on the runtime $t(\cdot)$. The variable F_{state} is initialized with wait.

Creation. Upon receiving a message (create, sid, $\mathcal{G}, \mathcal{D}, \mathcal{T}$) from $\mathcal{G}$ where $\mathcal{T}$ is a deterministic Turing machine, verify if $F_{state} = \mathsf{wait}$; else ignore the input. Next, store (sid, $\mathcal{G}$, $\mathcal{D}$, $\mathcal{T}$,T_{state}) where T_{state} is the initial state of $\mathcal{T}$, set $F_{state} \leftarrow \mathsf{sent}$ and send the message ($\mathsf{created}$, sid, $\mathcal{G}, \mathcal{D}$) to the adversary.

Deliver. Upon receiving a message ($\mathsf{deliver}$, sid, $\mathcal{G}, \mathcal{D}$) from the adversary, verify that $F_{state} = \mathsf{sent}$; else ignore the input. Next, set $F_{state} \leftarrow \mathsf{ready}$, and send ($\mathsf{ready}$, sid, $\mathcal{G}, \mathcal{D}$) to $\mathcal{D}$.

Execution. Upon receiving a message ($\mathsf{execute}$, sid, $\mathcal{G}, \mathcal{D}, x$) from $\mathcal{D}$ where x is an input, check if $F_{state} = \mathsf{ready}$ and if it is, then run $\mathcal{T}(T_{state}, x)$ for at most $t(\lambda)$ steps. Save the new state of $\mathcal{T}$ in T_{state}, read the output y from its output tape and send (output, sid, $\mathcal{G}, \mathcal{D}, y$) to $\mathcal{D}$.

Incoming Communication. Upon receiving a message ($\mathsf{communication}$, sid, $\mathcal{G}, \mathcal{D}, m$) from $\mathcal{A}$, run $\mathcal{T}(T_{state}, m)$ for at most $t(\lambda)$ steps. Save the new state of $\mathcal{T}$.

Figure 3: The wrapper functionality allowing one-way communication.

Token - Random Values $\mathcal{T}_{\text{Random}}$

Parametrized by a security parameter λ. The token is hardwired with a random vector $a \xleftarrow{\$} \mathbb{F}_2^{2\lambda}$ and a random matrix $B \xleftarrow{\$} \mathbb{F}_2^{2\lambda \times 2\lambda}$. It is initialized with state $T_{state} = \mathsf{ready}$.

Output. Upon receiving a message (choice, z) from $\mathcal{D}$ check if $T_{state} = \mathsf{ready}$; else abort. Next, set $T_{state} \leftarrow \mathsf{dead}$, compute $V \leftarrow a \otimes z + B$ and send the message (output, V) to $\mathcal{D}$.

Figure 4: The first token, which only contains random values.

Theorem 3.1 *In the model where a malicious Goliath is allowed to send messages to the token, the protocol presented in Figure 6 UC-realizes the functionality $\mathcal{F}^{OTM\text{-}with\text{-}Abort}$ with statistical security against a corrupted Goliath and perfect security against a corrupted David.*

Proof: (Sketch) The correctness as well as the security against a corrupted David follow directly from Döttling's et al. proof of security. In the case of the security against a corrupted Goliath, note that the OTM is considered delivered at the point in which David has received $(G, \tilde{a}, \tilde{B}, \tilde{s_0}, \tilde{s_1})$ from $\mathcal{T}_{\text{Inputs}}$. From that point on, $\mathcal{T}_{\text{Inputs}}$ does not participate in the protocol anymore and it cannot send messages to the outside world. Hence neither Goliath nor $\mathcal{T}_{\text{Random}}$ know the matrix C which is used for the commitments, so they can cheat in the commitment's opening phase only with negligible probability. Both of them also do not know the value h, which is necessary together with z in order to determine David's input x. So the proof proceeds as in [13], the only difference here is that Goliath can still send messages to $\mathcal{T}_{\text{Random}}$ at any point, and thus he can modify the abort behavior. This can be dealt with by running Döttling's

Token - Inputs $\mathcal{T}_{\text{Inputs}}$

Parametrized by a security parameter λ. The token is initialized with Goliath's inputs s_0, s_1, and the vector a and matrix B that are used by $\mathcal{T}_{\text{Random}}$. It is initialized in state $T_{state} = \mathsf{ready}$.

Matrix Choice. Upon receiving a message ($\mathsf{matrix\ choice}$, C) from $\mathcal{D}$ check if $T_{state} = \mathsf{ready}$ and $C \in \mathbb{F}_2^{\lambda \times 2\lambda}$; else abort. Next, compute a matrix $G \in \mathbb{F}_2^{\lambda \times 2\lambda}$ that is complementary to C (i.e., G is determined by λ vectors of length 2λ which are linearly independent and G spans a subspace of the kernel of C), and also compute $\tilde{a} \leftarrow Ca$, $\tilde{B} \leftarrow CB$. Set $T_{state} \leftarrow \mathsf{committed}$ and send the message ($\mathsf{commitment}$, $G, \tilde{a}, \tilde{B}$) to $\mathcal{D}$.

Ciphertexts. Upon receiving a message ($\mathsf{vector\ choice}$, h) from $\mathcal{D}$ check if $T_{state} = \mathsf{committed}$ and $h \in \mathbb{F}_2^{2\lambda} \setminus \{0\}$; else abort. Next, compute $\tilde{s_0} \leftarrow s_0 + GBh$ and $\tilde{s_1} \leftarrow s_1 + GBh + Ga$, set $T_{state} \leftarrow \mathsf{dead}$ and send the message (output, $\tilde{s_0}, \tilde{s_1}$) to $\mathcal{D}$.

Figure 5: The second token, which stores Goliath's inputs.

et al. procedure to verify whether the token is going to abort or not (i.e., running a copy of the token in its current state with random inputs) after each incoming message from Goliath to the token. If the simulator notices that the abort behavior changed, he can make the appropriate change in $\mathcal{F}^{\text{OTM-with-Abort}}$ by using the Switch Off/Switch On commands. ▌

Sequential OTM with Abort. As done by Döttling et al. [13] for the OTM functionality, it is also possible to define a sequential version of the OTM-with-Abort functionality where there are many pairs of Goliath's inputs (i.e., there are multiple stages) which can only be queried sequentially by David. The functionality only needs to be modified to take pairs of inputs which can be queried sequentially by David and to allow an adversary to specify which stages are active/inactive at any time (if an inactive stage is queried by David, then the functionality aborts). In this case the two token solution of Döttling et al. [13] for sequential OTMs can be used. The security proof would be a straightforward modification of Döttling et al.'s proof in the same line as done above.

3.2 Impossibility of Unconditionally Secure OTM from a Single Token

Lemma 3.2 *Assume that there is only one token and that a malicious token is not computationally bounded. If a malicious Goliath is allowed to send messages to the token, then there is no protocol* Π *that realizes OTM with information-theoretic security from this single token.*

Proof: For the sake of contradiction assume that a correct and information-theoretically secure OTM protocol Π from a single stateful token exists. Assume that the parties' inputs are chosen as $s_0, s_1 \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and $c \overset{\$}{\leftarrow} \{0,1\}$. The sender's privacy of the OTM protocol should hold, i.e.

$$\begin{aligned} I(\mathsf{view}_{\mathcal{D}}; s_{1-c}) \leq \varepsilon \quad &\Leftrightarrow \quad H(s_{1-c}) - H(s_{1-c}|\mathsf{view}_{\mathcal{D}}) \leq \varepsilon \\ &\Leftrightarrow \quad H(s_{1-c}|\mathsf{view}_{\mathcal{D}}) \geq \lambda - \varepsilon, \end{aligned}$$

where $\mathsf{view}_{\mathcal{D}}$ is David's view of the protocol execution and ε is a function that is negligible in the security parameter.

Protocol

Parametrized by a security parameter λ.

Deliver. $\mathcal{D}$ waits until $\mathcal{G}$ send the tokens $\mathcal{T}_{\text{Random}}$ and $\mathcal{T}_{\text{Inputs}}$. Then he chooses a random matrix $C \in \mathbb{F}_2^{\lambda \times 2\lambda}$ and sends the message $(\mathsf{matrix\ choice}, C)$ to $\mathcal{T}_{\text{Inputs}}$ in order to get the answer $(\mathsf{commitment}, G, \tilde{a}, \tilde{B})$. After that, $\mathcal{D}$ picks a random vector $h \in \mathbb{F}_2^{2\lambda} \setminus \{0\}$ and sends the message $(\mathsf{vector\ choice}, h)$ to $\mathcal{T}_{\text{Inputs}}$ in order to get the output $(\mathsf{output}, \tilde{s_0}, \tilde{s_1})$.

Choice Phase. When $\mathcal{D}$ gets his input $c \in \mathbb{F}_2$, he chooses $z \xleftarrow{\$} \mathbb{F}_2^{2\lambda}$ such that $z^T h = c$ and sends the message (choice, z) to $\mathcal{T}_{\text{Random}}$ to get the output (output, V). Then $\mathcal{D}$ checks if $CV = \tilde{a}z^T + \tilde{B}$. If it is not, $\mathcal{D}$ aborts; otherwise, he outputs $s_c = \tilde{s_c} + GVh$.

Figure 6: The unconditionally secure protocol that realizes $\mathcal{F}^{\text{OTM-with-Abort}}$.

By definition of the OTM functionality David can choose his input c at any time after he receives the token and Goliath should not learn when David queried the OTM functionality. So David can choose his input c at a point in the future far after receiving the token, when all initial communication between the parties is already finished, and then he interacts with the token to receive s_c. But then, at the moment right before David's choice c is made, its entropy is still 1 from the point of view of all parties. Therefore, due to the sender's privacy, at this point it should hold that

$$H(s_0 | \mathsf{view}'_{\mathcal{D}}) \geq \lambda - \varepsilon$$

and

$$H(s_1 | \mathsf{view}'_{\mathcal{D}}) \geq \lambda - \varepsilon,$$

where $\mathsf{view}'_{\mathcal{D}}$ is David's view of the protocol execution until this point. But if a malicious Goliath is allowed to send messages to the token, he can forward his complete view to the token. The token then gets to know all protocol interactions so far and due to the correctness of the OTM protocol (i.e., it should work for any pair of inputs in $\{0,1\}^\lambda$) he is able, for almost any $s'_c \in \{0,1\}^\lambda$, to find a strategy to follow for the rest of the protocol that makes David accept s'_c. Hence the values s_0 and s_1 are not fixed up to the point when David inputs c. But in the OTM functionality the values s_0 and s_1 are fixed once it is sent, and thus we get a contradiction. ▌

3.3 Unconditionally Secure OT with a Single Token

Döttling et al. [13] also presented an unconditionally secure solution with one token only, in which the interactions which are performed between David and $\mathcal{T}_{\text{Inputs}}$ in the previously described protocol are instead performed between David and Goliath in an initial interactive phase that is used to send the commitments and the ciphertexts. Note that such a version of the protocol would not be secure in the setting where one-way communication is allowed into the token since Goliath could simply forward the matrix C to $\mathcal{T}_{\text{Random}}$, which would then be able to open the commitments to any value and thus be able to change the outputs at any time. But we should mention that it is possible to obtain an oblivious transfer protocol with only one token by letting the single token act like $\mathcal{T}_{\text{Inputs}}$ in the above protocol and letting the interactions between David and $\mathcal{T}_{\text{Random}}$ be replaced by identical interactions between David and Goliath. The proof of

Token $\mathcal{T}$

Parametrized by a security parameter λ. The token is hardwired with the shares $(v_{i,0}, v_{i,1})$ for $i = 1, \dots, \lambda$, the inputs s_0, s_1 and Goliath's public key pk. It is initialized with state $T_{state} = \mathsf{ready}$ and $j = 0$.

Message Commitment. Upon receiving a message ($\mathsf{challenge}$, k_j) from $\mathcal{D}$ check if $T_{state} = \mathsf{ready}$ and k_j is a bit; else abort Set $j \leftarrow j + 1$. If $j = \lambda$, then set $T_{state} \leftarrow \mathsf{message\ committed}$. Send the message ($\mathsf{message\ commitment}$, v_{j,k_j}) to $\mathcal{D}$.

Inputs Commitment. Upon receiving a message (commit, crs, σ) from $\mathcal{D}$ check if $T_{state} = \mathsf{message\ committed}$ and if σ is a valid signature of $\mathcal{G}$ on crs; else abort. Set $T_{state} \leftarrow \mathsf{inputs\ committed}$. Commit to the values s_0, s_1 using a computationally UC-secure commitment protocol that uses the common reference string crs and send the commitments to $\mathcal{D}$. Let d_0, d_1 denote the information to open the commitments.

Output. Upon receiving the message output from $\mathcal{D}$ check if $T_{state} = \mathsf{inputs\ committed}$; else abort. Next, set $T_{state} \leftarrow \mathsf{dead}$ and execute with $\mathcal{D}$ a computationally UC-secure oblivious transfer protocol using the common reference string crs and with inputs $(s_0 \| d_0, s_1 \| d_1)$.

Figure 7: The token for a computationally secure OTM protocol with a single token.

security would follow in the same line as before since Goliath would never get to know C and h. Note that the drawback of having to know the OT inputs before sending the token can be easily overcome by performing the OTs with random inputs and derandomizing them afterwards.

3.4 Computationally Secure OTM from a Single Token

If one considers the scenario where only one token is available, it is possible to obtain a protocol that realizes $\mathcal{F}^{\text{OTM-with-Abort}}$ with computational security. The idea is to compute as an initial step (i.e., during the delivery phase) the commitment functionality by using the token and interactions between Goliath and David. With access to this commitment functionality it is possible to obtain a common reference string between David and the token[2], which in turn allows to run a computationally secure UC-commitment protocol between them in order to commit to the input values. After receiving from the token the commitments to the input values, David considers the deliver complete, and whenever he wants to get his output he just executes an oblivious transfer protocol with the token with his desired choice bit as input. He checks the correctness of the output using the commitment. The crucial point for the simulation to go through is that the simulator should be able to extract the first commitment before its opening, so that he can choose the common reference string as he wishes. In order to accomplish that in face of a potentially malicious token which possibly only answer correctly queries to certain values, we will commit to a message m by using λ pairs of random shares $(v_{i,0}, v_{i,1})$ where for each pair $v_{i,0} + v_{i,1} = m$. During the committing phase, $\mathcal{D}$ interacts with the token and can choose to receive either $v_{i,0}$ or $v_{i,1}$ for each pair. To open the commitment, $\mathcal{G}$ reveals all the shares. The specification of the token can be found in Figure 7 and of the protocol in Figure 8.

[2]The common reference string is actually obtained by Goliath and David, but can be forwarded from Goliath to the token via David by using a digital signature to ensure that the value that the token obtains is exactly the same one that Goliath sent.

Protocol

Parametrized by a security parameter λ.

Deliver. $\mathcal{G}$ generates a pair of signing sk and public pk keys for a signature scheme. Then he picks a random message $m' \xleftarrow{\$} \mathbb{F}_2^\lambda$ and random vectors $v_{i,0} \xleftarrow{\$} \mathbb{F}_2^\lambda$ for $i = 1, \ldots, \lambda$ and sets $v_{i,1} = m' - v_{i,0}$. He creates the token $\mathcal{T}$ (described in Figure 7) with the hardwired vectors $(v_{i,0}, v_{i,1})$, s_0, s_1 and pk, and sends it to $\mathcal{D}$. Upon receiving the token $\mathcal{T}$, $\mathcal{D}$ queries it with random bits k_i for $i = 1, \ldots, \lambda$ in order to get v_{i,k_i}. $\mathcal{D}$ picks a random message $m'' \xleftarrow{\$} \mathbb{F}_2^\lambda$ and sends it to $\mathcal{G}$. Then $\mathcal{G}$ opens the commitment to m' by sending all the shares $(v_{i,0}, v_{i,1})$ to $\mathcal{D}$. $\mathcal{D}$ checks if $m' = v_{i,0} + v_{i,1}$ for all $i = 1, \ldots, \lambda$, aborting the protocol if this is not the case. Both $\mathcal{G}$ and $\mathcal{D}$ use $m = m' + m''$ to generate a common reference string crs. $\mathcal{G}$ signs crs with his signing key sk and sends the signature σ to $\mathcal{D}$. $\mathcal{D}$ sends crs and σ to $\mathcal{T}$ in order to receive the commitments to s_0 and s_1.

Choice Phase. When $\mathcal{D}$ gets his input $c \in \mathbb{F}_2$, he sends the message output to $\mathcal{T}$ and executes a computationally UC-secure oblivious transfer protocol with the token using the common reference string crs and with input c in order to get the output $s_c \| d_c$, where $\|$ denotes concatenation. $\mathcal{D}$ checks the correctness of s_c using the commitment that he received previously and the opening information d_c.

Figure 8: The computationally secure OTM protocol using one token.

Theorem 3.3 *In the model where a malicious Goliath is allowed to send messages to the token, the protocol presented in Figure 8 UC-realizes the functionality $\mathcal{F}^{\textit{OTM-with-Abort}}$ with computational security.*

Proof: The correctness of the protocol can be trivially verified. The simulation for the cases that both parties are corrupted or no parties are corrupted are trivial. We describe below how the simulation proceeds in the other cases.

Corrupted Sender: If Goliath is corrupted (and thus also the token), the simulator will simulate an interaction of the protocol with the adversary and has to extract both s_0 and s_1 from this interaction in order to given them as input for the OTM functionality. The key point to do this is that the simulator should be able to extract the value m' before sending m'', so that he can choose the common reference string crs as he wishes, thus being able to create a trapdoor to extract s_0 and s_1 from the committed values.

We have that only Goliath can program the token, so the environment machine will provide the code to Goliath (and hence to the simulator). To extract the value m' the simulator does the following. When the commitment step happens, whenever David sends a valid message $(\mathsf{challenge}, k_j)$ to receive a share $\tilde{v}_{j,k_j}$, the simulator first executes the token with the input $1 - k_j$ obtaining an answer $\tilde{v}_{j,1-k_j}$, and then resets the token to the point before this query and executes the token with input k_j to obtain $\tilde{v}_{j,k_j}$ and forward it to David. Let $\tilde{m}'_j = \tilde{v}_{j,0} + \tilde{v}_{j,1}$. After all the λ challenges are done, the simulator fixes $\tilde{m}'$ as the value that appeared more often in the tuple $(\tilde{m}'_1, \ldots, \tilde{m}'_\lambda)$. He then chooses $m'' = m - \tilde{m}'$ for any m he wants. Lets now analyze this extraction procedure. Let $(\hat{v}_{j,0}, \hat{v}_{j,1})$ denote the values that Goliath reveals in the opening phase. Note that the protocol will be aborted unless $\hat{v}_{j,0} + \hat{v}_{j,1} = \hat{m}'$ for all j and some fixed message $\hat{m}'$. For any j, if $\hat{v}_{j,0} \neq \tilde{v}_{j,0}$ and $\hat{v}_{j,1} \neq \tilde{v}_{j,1}$ then the protocol will be aborted anyway and we do not need to worry about the extracted value. If for the majority of the j's it holds that $\hat{v}_{j,0} = \tilde{v}_{j,0}$ and $\hat{v}_{j,1} = \tilde{v}_{j,1}$, then $\tilde{m}' = \hat{m}'$ and thus the extraction procedure works properly. The remaining case is the one in which at least half of the j's are such that either $\hat{v}_{j,0} = \tilde{v}_{j,0}$ or $\hat{v}_{j,1} = \tilde{v}_{j,1}$, but not both equalities hold. For each such j the probability that the opening check succeeds for this pair of vectors is 1/2 since Goliath cannot get any information from the token. Therefore if half or more of the j's are in this condition, the protocol will abort with overwhelming probability in the security parameter λ.

Given that the extraction worked properly, the simulator can create the common reference string as he wishes and so he is able to have a trapdoor to extract the values s_0 and s_1 from the commitments and give them as input to the OTM functionality. To learn the abort behavior, the simulator simulates, at onset and also after each incoming message from Goliath to the token, a choice phase execution between David and the token. The simulator can then use the Switch Off/Switch On commands to adapt $\mathcal{F}^{\text{OTM-with-Abort}}$'s abort behavior properly.

Corrupted Receiver: If David is corrupted, the simulator gets to know all David's challenges k_j in the first commitment. Hence, after seeing m'', he can choose any m' he wants (and thus any resulting m and crs) and appropriate shares $(\hat{v}_{j,0}, \hat{v}_{j,1})$ that are correct from David's point of view. By picking a common reference string together with an appropriate trapdoor, the simulator can learn the choice bit c and query it to the functionality $\mathcal{F}^{\text{OTM-with-Abort}}$ to learn s_c. Using the equivocability of the UC-commitment the simulator can find an appropriate opening information d_c and feed $s_c \| d_c$ to David in the OT protocol. ▮

Note that the above protocol can be trivially extended to the case of sequential OTMs.

4 The Case of Outgoing Communication

In the complementary problem, we consider tokens which have a one-way channel that allow them to send messages to Goliath, but which cannot receive any information from Goliath. In this scenario we would like to implement $\mathcal{F}^{\text{OTM}}$. Note that in this case Goliath cannot control online the abort behavior of the token. We first show an impossibility result for unconditionally secure protocols and then present a computationally secure protocol using a single token.

4.1 Impossibility of Information-Theoretically Secure OT(M)

Lemma 4.1 *If the tokens can send messages to Goliath, then there is no protocol* Π *that realizes OTM, or even oblivious transfer, with information-theoretic security.*

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

Token $\mathcal{T}$

Parametrized by a security parameter λ. The token is hardwired with the shares $(v_{n,i,0}, v_{n,i,1})$ for $i = 1, \ldots, 2\lambda, n = 1, \ldots, \lambda$ and an opening key $\mathsf{cok} \in \{0,1\}^\lambda$. It is initialized with state $T_{state} = \mathsf{ready}$.

Shares Opening. Upon receiving a message $(\mathsf{challenge}, n_1, \ldots, n_{\lambda/2})$ from $\mathcal{D}$ check if $T_{state} = \mathsf{ready}$ and $\{n_1, \ldots, n_{\lambda/2}\} \subset \{1, \ldots, \lambda\}$ are the specifications of the shares $\mathcal{D}$ wants to be revealed; else abort. Set $T_{state} \leftarrow \mathsf{message\ committed}$. Send the message $(\mathsf{shares\ opening}, (v_{n_j,i,0}, v_{n_j,i,1})_{j=1,\ldots,\lambda/2, i=1,\ldots,2\lambda})$ to $\mathcal{D}$.

Message Opening. Upon receiving a message $(\mathsf{reveal\ message}, \overline{\mathsf{cok}})$ from $\mathcal{D}$ check if $T_{state} = \mathsf{message\ committed}$ and $\overline{\mathsf{cok}} = \mathsf{cok}$; else abort. Set $T_{state} \leftarrow \mathsf{message\ opened}$. Send the message $(\mathsf{opening}, (v_{n,i,0}, v_{n,i,1})_{n=1,\ldots,\lambda, i=1,\ldots,2\lambda})$ to $\mathcal{D}$.

Figure 9: The token for the commitment protocol with outgoing communication.

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

4.2 Unconditionally Secure Commitment with a Single Token

The idea here is to commit to a message m by using pairs of random shares $(v_{i,0}, v_{i,1})$ such that for each pair $v_{i,0} + v_{i,1} = m$, the shares are known to both the token and Goliath. The commitment phase is done by interactions between David and Goliath, where for each pair David can choose to receive either $v_{i,0}$ or $v_{i,1}$. In order to guarantee the binding property, the opening phase is executed between David and the token: David receives an opening key from Goliath and forwards it to the token, who checks it and reveals all the shares to David. To guarantee that on one hand David cannot guess the opening key correctly (and thus open the commitment whenever he wants), but on the other hand the opening key does not contain enough information to allow the token to learn David's choices during the commitment phase (and thus successfully open the commitment to any value), we have opening keys that are random λ-bit strings and we use 2λ pairs of random shares. This commitment scheme is secure, but not yet extractable. In order to get extractability, instead of committing to the message itself, we first use the $(\lambda, \lambda/2+1)$-Shamir's secret share scheme to create λ shares $(m_1, \ldots, m_\lambda)$ of the message, then commit to each share using the above scheme (in the opening phase a single opening key of λ-bits is given to the token in order to open all the commitments), but we additionally make David ask the token to open $\lambda/2$ shares $m_{n_1}, \ldots m_{n_{\lambda/2}}$ (without sending the opening key) already in the commitment phase, which do not reveal any information about m. The specification of the token can be found in Figure 9 and of the protocol in Figure 10.

Theorem 4.2 *In the model where malicious tokens are allowed to send messages to Goliath, the protocol presented in Figure 10 UC-realizes the commitment functionality $\mathcal{F}^{COM}$ with unconditional security.*

The proof is in Appendix A.

Commitment Protocol

Parametrized by a security parameter λ.

Commitment Phase. $\mathcal{G}$ generates an opening key $\mathsf{cok} \xleftarrow{\$} \mathbb{F}_2^\lambda$. Then he generates λ shares $(m_1, \ldots, m_\lambda)$ of the message m using Shamir's secret sharing scheme. For each share m_n, $\mathcal{G}$ picks random vectors $v_{n,i,0} \xleftarrow{\$} \mathbb{F}_2^\lambda$ for $i = 1, \ldots, 2\lambda$ and sets $v_{n,i,1} = m_n - v_{n,i,0}$. He creates the token $\mathcal{T}$ (described in Figure 9) with the hardwired cok and vectors $(v_{n,i,0}, v_{n,i,1})$ for $n = 1, \ldots, \lambda, i = 1, \ldots, 2\lambda$, and sends it to $\mathcal{D}$. Upon receiving the token $\mathcal{T}$, $\mathcal{D}$ queries $\mathcal{G}$ with random bits $k_{n,i}$ for $n = 1, \ldots, \lambda, i = 1, \ldots, 2\lambda$ in order to get $v_{n,i,k_{n,i}}$. Then $\mathcal{D}$ picks a random subset $\{n_1, \ldots, n_{\lambda/2}\} \subset \{1, \ldots, \lambda\}$ and asks the token to reveal $(v_{n_j,i,0}, v_{n_j,i,1})$ for $j = 1, \ldots, \lambda/2, i = 1, \ldots, 2\lambda$, which he checks against the information he received from $\mathcal{G}$; aborting if they do not match.

Opening Phase. $\mathcal{G}$ sends to $\mathcal{D}$ the message shares $(m_1, \ldots, m_\lambda)$ and also the commitment opening key cok, which $\mathcal{D}$ forwards to $\mathcal{T}$ in order to get all the shares $(v_{n,i,0}, v_{n,i,1})$. $\mathcal{D}$ checks if $m_n = v_{n,i,0} + v_{n,i,1}$ for all i and n, aborting the protocol if this is not the case. Then he reconstructs m from the shares; aborting if m is not uniquely determined by the shares.

Figure 10: The unconditionally secure commitment protocol using one token for the case of outgoing communication.

4.3 Computationally Secure OTM with a Single Token

For the case of computational security, it is possible to obtain an OTM protocol which uses only one token. The approach is briefly described below. Using the ideas from the previous section the parties can compute the commitment functionality, which can then be used to establish a common reference string between David and the token. The common reference string in turn can be used to run computationally UC-secure commitment and OT protocols between the token and David. The token commits to the input values using the computationally UC-secure commitment protocol, at which point David considers the deliver complete. Afterwards, whenever David wants to obtain his output, he engages in a computationally UC-secure OT protocol with the token in order to get the desired output and the commitment verification information.

Theorem 4.3 *In the model where malicious tokens are allowed to send messages to Goliath, there is a protocol using a single token which UC-realizes the functionality $\mathcal{F}^{OTM}$ with computational security.*

The description of the token and the protocol, as well as the security proof can be found in Appendix B.

5 Conclusion

In this work we investigated a weaker isolation model for tamper-proof hardware, namely one-way (broadband) communication channels are allowed either for the token creator to the tokens or in the opposite direction. In the case that the tokens can receive incoming communication from their creators we showed the following: (1) there is an unconditionally secure One-Time Memory (OTM) protocol using two tokens, (2) it is impossible to realize OTM with unconditional security from a single token, (3) there is an unconditionally secure oblivious transfer protocol using a single token, (4) there is a computationally secure OTM protocol using a single token. In the case that the tokens can send outgoing communication to their creator we showed the following: (1) it is impossible to realize OTM or oblivious transfer with unconditional security, (2) there is an unconditionally secure commitment protocol using a single token, (3) there is a computationally secure OTM protocol using a single token.

References

[1] Shashank Agrawal, Prabhanjan Ananth, Vipul Goyal, Manoj Prabhakaran, and Alon Rosen. Lower bounds in the hardware token model. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 663–687, San Diego, CA, USA, February 24–26, 2014. Springer, Berlin, Germany. (Cited on page 2.)

[2] Nir Bitansky, Ran Canetti, Shafi Goldwasser, Shai Halevi, Yael Tauman Kalai, and Guy N. Rothblum. Program obfuscation with leaky hardware. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 722–739, Seoul, South Korea, December 4–8, 2011. Springer, Berlin, Germany. (Cited on page 1, 2.)

[3] Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 302–318, Santa Barbara, CA, USA, August 22–26, 1993. Springer, Berlin, Germany. (Cited on page 2.)

[4] Christina Brzuska, Marc Fischlin, Heike Schröder, and Stefan Katzenbeisser. Physically uncloneable functions in the universal composition framework. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 51–70, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Berlin, Germany. (Cited on page 2.)

[5] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, Nevada, USA, October 14–17, 2001. IEEE Computer Society Press. (Cited on page 1, 2.)

[6] Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for UC secure computation using tamper-proof hardware. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 545–562, Istanbul, Turkey, April 13–17, 2008. Springer, Berlin, Germany. (Cited on page 2.)

[7] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105, Santa Barbara, CA, USA, August 16–20, 1992. Springer, Berlin, Germany. (Cited on page 2.)

[8] Seung Geol Choi, Jonathan Katz, Dominique Schröder, Arkady Yerukhimovich, and Hong-Sheng Zhou. (efficient) universally composable oblivious transfer using a minimal number of stateless tokens. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 638–662, San Diego, CA, USA, February 24–26, 2014. Springer, Berlin, Germany. (Cited on page 2.)

[9] Ronald Cramer and Torben P. Pedersen. Improved privacy in wallets with observers (extended abstract). In Tor Helleseth, editor, *Advances in Cryptology – EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 329–343, Lofthus, Norway, May 23–27, 1993. Springer, Berlin, Germany. (Cited on page 2.)

[10] Dana Dachman-Soled, Nils Fleischhacker, Jonathan Katz, Anna Lysyanskaya, and Dominique Schröder. Feasibility and infeasibility of secure computation with malicious PUFs. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of

Lecture Notes in Computer Science, pages 405–420, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Berlin, Germany. (Cited on page 2.)

[11] Ivan Damgård, Jesper Buus Nielsen, and Daniel Wichs. Universally composable multiparty computation with partially isolated parties. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 315–331. Springer, Berlin, Germany, March 15–17, 2009. (Cited on page 1.)

[12] Ivan Damgård and Alessandra Scafuro. Unconditionally secure and universally composable commitments from physical assumptions. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 100–119, Bengalore, India, December 1–5, 2013. Springer, Berlin, Germany. (Cited on page 2.)

[13] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 164–181, Providence, RI, USA, March 28–30, 2011. Springer, Berlin, Germany. (Cited on page 1, 2, 3, 4, 5, 6, 7.)

[14] Nico Döttling, Thilo Mie, Jörn Müller-Quade, and Tobias Nilges. Implementing resettable UC-functionalities with untrusted tamper-proof hardware-tokens. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 642–661, Tokyo, Japan, March 3–6, 2013. Springer, Berlin, Germany. (Cited on page 2.)

[15] Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 258–277, Cambridge, MA, USA, February 19–21, 2004. Springer, Berlin, Germany. (Cited on page 2.)

[16] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 39–56, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Berlin, Germany. (Cited on page 2.)

[17] Vipul Goyal, Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. Interactive locking, zero-knowledge pcps, and unconditional cryptography. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 173–190, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Berlin, Germany. (Cited on page 1, 2.)

[18] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 308–326, Zurich, Switzerland, February 9–11, 2010. Springer, Berlin, Germany. (Cited on page 1, 2.)

[19] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Berlin, Germany. (Cited on page 2.)

[20] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128, Barcelona, Spain, May 20–24, 2007. Springer, Berlin, Germany. (Cited on page 1, 2, 3.)

[21] Vladimir Kolesnikov. Truly efficient string oblivious transfer using resettable tamper-proof tokens. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 327–342, Zurich, Switzerland, February 9–11, 2010. Springer, Berlin, Germany. (Cited on page 2.)

[22] Tal Moran and Gil Segev. David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 527–544, Istanbul, Turkey, April 13–17, 2008. Springer, Berlin, Germany. (Cited on page 1, 3.)

[23] Rafail Ostrovsky, Alessandra Scafuro, Ivan Visconti, and Akshay Wadia. Universally composable secure computation with (malicious) physically uncloneable functions. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 702–718, Athens, Greece, May 26–30, 2013. Springer, Berlin, Germany. (Cited on page 2.)

[24] Ravikanth Srinivasa Pappu. *Physical One-Way Functions*. PhD thesis, MIT, 2001. (Cited on page 2.)

[25] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Berlin, Germany. (Cited on page 2.)

[26] Manoj Prabhakaran, Amit Sahai, and Akshay Wadia. Secure computation using leaky tokens. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP 2014: 41st International Colloquium on Automata, Languages and Programming, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 907–918, Copenhagen, Denmark, July 8–11, 2014. Springer, Berlin, Germany. (Cited on page 1.)

[27] Ulrich Rührmair. Oblivious transfer based on physical unclonable functions. In *TRUST*, pages 430–440, 2010. (Cited on page 2.)

A Proof of Theorem 4.2

Proof: The correctness of the protocol is trivial, as well as the simulation for the cases that both parties are corrupted or no parties are corrupted. We describe below how the simulation proceeds in the other cases.

Corrupted Sender: We have that only Goliath can program the token, so the environment machine will provide the code to Goliath (and hence to the simulator). To extract the value m the simulator does the following when David asks the token to open $\lambda/2$ shares $m_{n_1}, \dots m_{n_{\lambda/2}}$ of m during the commitment step. Let Q_0 denote the subset that David chose to be opened. First note that given the token's answer to any opening query and the commitment information received during the interaction with Goliath, it is possible to distinguish with overwhelming probability if the token opened the shares correctly or not. So the simulator first tests if the query Q_0 was answered correctly by the token. If Q_0 is not answered correctly by the token, we do not need to worry about the extraction since the protocol will be aborted anyway. If it was answered correctly, the simulator also runs internally other executions of this procedure for opening half of the shares, in each of them asking a random subset $Q_1, Q_2, \dots$ (with $|Q_i| = \lambda/2$) of the shares to be opened (the token is started in each execution from the same state that it was in just before Q_0 and there is a fixed exponential upper bound on the number of executions that can be performed). The procedure is repeated until some Q_j is answered correctly. Note that the probability that the token answers any of the queries $Q_0, Q_1, \dots$ correctly is the same since they are chosen from the same distribution and let p denote this probability. If p is not negligible, then the expected numbers of iterations needed to find a second query Q_j that is answered correctly is polynomial. From Q_0 and Q_j the simulator can recover $\hat{m}$ (the unique value that can possibly be accepted with non-negligible probability in the tests performed in the commitment's opening phase). If $m \neq \hat{m}$, we do not need to worry about the extraction since the protocol will be aborted anyway. Therefore the simulator can perform the extraction correctly with overwhelming probability.

Corrupted Receiver: If David is corrupted, the simulator gets to know all David's challenges $k_{n,i}$ as well as which shares of m were opened. Hence after the commitment phase he can still choose any $\hat{m}$ he wants and appropriate shares $(\hat{m}_1, \ldots, \hat{m}_\lambda)$ and $(\hat{v}_{n,i,0}, \hat{v}_{n,i,1})$ that will be accepted by David in the opening phase. ▌

B Computationally Secure OTM with a Single Token

Here we describe the computationally secure OTM protocol with a single token in the case of outgoing communication. It uses the commitment protocol as a building block. We describe the full protocol for the sake of completeness.

As in the protocol presented in Section 3.4, the idea is to use initial interactions (i.e., during the delivering phase) between Goliath and David, and also between David and the token in order to compute the commitment functionality and use it to establish a common reference string between David and the token, which can then be use to run computationally UC-secure commitment and OT protocols between the token and David. The token then commits to the input values, at which point David considers the deliver complete. Afterwards, whenever David wants to obtain his output, he engages in a computationally UC-secure OT protocol with the token in order to get the desired output and the commitment verification information. The specification of the token can be found in Figure 11 and of the protocol in Figure 12.

Theorem B.1 *The protocol presented in Figure 12 UC-realizes the functionality $\mathcal{F}^{OTM}$ with computational security.*

Proof: The correctness of the protocol is trivial, as well as the simulation for the cases that both parties are corrupted or no parties are corrupted. We describe below how the simulation proceeds in the other cases.

Corrupted Sender: If Goliath is corrupted (and thus also the token), the simulator should be able to extract both s_0 and s_1 to give them as input to the OTM functionality. In short, the simulator should be able to extract the value m' before sending m'', so that he can choose the common reference string crs as he wishes, thus being able to create a trapdoor to extract s_0 and s_1 from the committed values. The extraction happens in the same way as in the proof in Appendix A and the expected polynomial time simulator can perform the extraction correctly with overwhelming probability.

If the extraction works properly, the simulator is able to choose the common reference string and therefore is able to have a trapdoor that allows him to extract the values s_0 and s_1 from the commitments. To learn about the abort behavior during the choice phase, the simulator simulates a choice phase between David and the token with random inputs. Hence he is able to forward the correct inputs to the OTM functionality.

Token $\mathcal{T}$

Parametrized by a security parameter λ. The token is hardwired with the shares $(v_{n,i,0}, v_{n,i,1})$ for $i = 1, \ldots, 2\lambda, n = 1, \ldots, \lambda$, the inputs s_0, s_1, Goliath's public key pk and an opening key $\mathsf{cok} \in \{0,1\}^\lambda$. It is initialized with state $T_{state} = \mathsf{ready}$.

Shares Opening. Upon receiving a message $(\mathsf{challenge}, n_1, \ldots, n_{\lambda/2})$ from $\mathcal{D}$ check if $T_{state} = \mathsf{ready}$ and $\{n_1, \ldots, n_{\lambda/2}\} \subset \{1, \ldots, \lambda\}$ are the specifications of the shares $\mathcal{D}$ wants to be revealed; else abort. Set $T_{state} \leftarrow \mathsf{message\ committed}$. Send the message $(\mathsf{shares\ opening}, (v_{n_j,i,0}, v_{n_j,i,1})_{j=1,\ldots,\lambda/2, i=1,\ldots,2\lambda})$ to $\mathcal{D}$.

Message Opening. Upon receiving a message $(\mathsf{reveal\ message}, \overline{\mathsf{cok}})$ from $\mathcal{D}$ check if $T_{state} = \mathsf{message\ committed}$ and $\overline{\mathsf{cok}} = \mathsf{cok}$; else abort. Set $T_{state} \leftarrow \mathsf{message\ opened}$. Send the message $(\mathsf{opening}, (v_{n,i,0}, v_{n,i,1})_{n=1,\ldots,\lambda, i=1,\ldots,2\lambda})$ to $\mathcal{D}$.

Inputs Commitment. Upon receiving a message $(\mathsf{commit}, \mathsf{crs}, \sigma)$ from $\mathcal{D}$ check if $T_{state} = \mathsf{message\ opened}$ and if σ is a valid signature of $\mathcal{G}$ on crs; else abort. Set $T_{state} \leftarrow \mathsf{inputs\ committed}$. Commit to the values s_0, s_1 using a computationally UC-secure commitment protocol that uses the common reference string crs and send the commitments to $\mathcal{D}$. Let d_0, d_1 denote the information to open the commitments.

Output. Upon receiving the message output from $\mathcal{D}$ check if $T_{state} = \mathsf{inputs\ committed}$; else abort. Next, set $T_{state} \leftarrow \mathsf{dead}$ and execute with $\mathcal{D}$ a computationally UC-secure oblivious transfer protocol using the common reference string crs and with inputs $(s_0 \| d_0, s_1 \| d_1)$.

Figure 11: The token for the OTM protocol with outgoing communication.

Corrupted Receiver: If David is corrupted, the simulator gets to know all David's challenges $k_{n,i}$ in the first commitment as well as which shares of m' were opened. Hence, after seeing m'', he can choose any $\hat{m}'$ he wants (and thus any resulting m and crs) and appropriate shares $(\hat{m}'_1, \ldots, \hat{m}'_\lambda)$ and $(\hat{v}_{n,i,0}, \hat{v}_{n,i,1})$ that are correct from David's point of view. By picking a common reference string together with an appropriate trapdoor, the simulator can learn the choice bit c and query it to the functionality $\mathcal{F}^{\mathrm{OTM}}$ to learn s_c. Using the equivocability of the UC-commitment the simulator can find an appropriate opening information d_c and feed $s_c \| d_c$ to David in the OT protocol. ▌

Protocol

Parametrized by a security parameter λ.

Deliver. $\mathcal{G}$ generates a pair of signing sk and public pk keys for a signature scheme and also an opening key $\mathsf{cok} \overset{\$}{\leftarrow} \mathbb{F}_2^\lambda$. Then he picks a random message $m' \overset{\$}{\leftarrow} \mathbb{F}_2^\lambda$ and generates the λ shares of it $(m'_1, \ldots, m'_\lambda)$ using Shamir's secret sharing scheme. For each share m'_n, $\mathcal{G}$ picks random vectors $v_{n,i,0} \overset{\$}{\leftarrow} \mathbb{F}_2^\lambda$ for $i = 1, \ldots, 2\lambda$ and sets $v_{n,i,1} = m'_n - v_{n,i,0}$. He creates the token $\mathcal{T}$ (described in Figure 9) with the hardwired s_0, s_1, pk, cok and vectors $(v_{n,i,0}, v_{n,i,1})$ for $n = 1, \ldots, \lambda, i = 1, \ldots, 2\lambda$, and sends it to $\mathcal{D}$. Upon receiving the token $\mathcal{T}$, $\mathcal{D}$ queries $\mathcal{G}$ with random bits $k_{n,i}$ for $n = 1, \ldots, \lambda, i = 1, \ldots, 2\lambda$ in order to get $v_{n,i,k_{n,i}}$. Then $\mathcal{D}$ picks a random subset $\{n_1, \ldots, n_{\lambda/2}\} \subset \{1, \ldots, \lambda\}$ and ask the tokens to reveal $(v_{n_j,i,0}, v_{n_j,i,1})$ for $j = 1, \ldots, \lambda/2, i = 1, \ldots, 2\lambda$, which he checks against the information he received from $\mathcal{G}$; aborting if they do not match. $\mathcal{D}$ picks a random message $m'' \overset{\$}{\leftarrow} \mathbb{F}_2^\lambda$ and sends it to $\mathcal{G}$. $\mathcal{G}$ sends to $\mathcal{D}$ the message shares $(m'_1, \ldots, m'_\lambda)$ and also the commitment opening key cok, which $\mathcal{D}$ forwards to $\mathcal{T}$ in order to get all the shares $(v_{n,i,0}, v_{n,i,1})$. $\mathcal{D}$ checks if $m'_n = v_{n,i,0} + v_{n,i,1}$ for all i and n, aborting the protocol if this is not the case. Then he reconstruct m' from the shares; aborting if m' is not uniquely determined by the shares. Both $\mathcal{G}$ and $\mathcal{D}$ use $m = m' + m''$ to generated a common reference string crs. $\mathcal{G}$ signs crs with his signing key sk and sends the signature σ to $\mathcal{D}$. $\mathcal{D}$ sends crs and σ to $\mathcal{T}$ in order to receive the commitments to s_0 and s_1.

Choice Phase. When $\mathcal{D}$ gets his input $c \in \mathbb{F}_2$, he sends the message output to $\mathcal{T}$ and execute with the token a computationally UC-secure oblivious transfer protocol using the common reference string crs and with input c in order to get the output $s_c \| d_c$. $\mathcal{D}$ checks the correctness of s_c using the commitment that he received previously and the opening information d_c.

Figure 12: The computationally secure OTM protocol using one token for the case of outgoing communication.