Unconditional and Composable Security Using a Single Stateful Tamper-Proof Hardware Token

Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade

Institute of Cryptography and Security, Faculty of Informatics,
Karlsruhe Institute of Technology, Germany
doettling@kit.edu, kraschewski@kit.edu, mueller-quade@kit.edu

Abstract. Cryptographic assumptions regarding tamper proof hardware tokens have gained increasing attention. Even if the tamper-proof hardware is issued by one of the parties, and hence not necessarily trusted by the other, many tasks become possible: Tamper proof hardware is sufficient for universally composable protocols, for information-theoretically secure protocols, and even allow to create software which can only be used once (One-Time-Programs). However, all known protocols employing tamper-proof hardware are either indirect, i.e., additional computational assumptions must be used to obtain general two party computations or a large number of devices must be used. In this work we present the first protocol realizing universally composable two-party computations (and even trusted One-Time-Programs) with information-theoretic security using only one single tamper-proof device issued by one of the mutually distrusting parties.
Keywords: Secure Two-Party Computation, Universal Composability, Tamper-Proof Hardware, Information-Theoretical Security

1 Introduction

Recently, tamper-proof hardware tokens have received increasing attention. Tamper-proof hardware tokens allow information-theoretically secure protocols which are universally composable (UC) [4], they can be employed for protocols in the *globalized UC* framework [5,17], and they even allow for One-Time-Programs, i.e. circuits which can be evaluated only once [13]. However, all known protocols employing tamper-proof hardware are either indirect, i.e., the secure hardware is used to implement commitments or Zero Knowledge proofs and additional computational assumptions must be used to obtain general, composable two party computations [19,22,8,7], or a large number of devices must be used [13,15]. However, issuing multiple independent tamper-proof devices requires much stronger isolation assumptions. Not only the communication between the device and the issuer must be prevented, but also the many devices must be mutually isolated. This is especially difficult as the devices are not necessarily trusted (see [3] for the difficulty of isolating two devices in one location).

In this work we present the first protocol realizing universally composable two-party computations (and even trusted One-Time-Programs) with information-theoretic security using only one single (untrusted) tamper-proof device.

One of the main challenges is to prevent a corrupted token from encoding previous inputs in subsequent outputs.

2 Related Work

The idea of secure computation based on separation assumptions was introduced by Ben-Or et al. [2] to construct multi-prover interactive proof systems. Ben-Or et al. [2] construct an unconditionally secure protocol for Rabin-OT [23] between two provers and a verifier. Even though this result is not explicitly stated in the context of tamper-proof hardware[1] and is proven secure in a standalone, synchronous model, we suppose that an amplified variant of the protocol of [2] can be proven UC-secure if the sender is allowed to issue two tamper-proof hardware tokens.

The idea of explicitly using tamper-proof hardware for cryptographic purposes was introduced by Goldreich and Ostrovsky [12]. They showed that tamper-proof hardware can be used for the purpose of software-protection.

The interest in secure hardware and separation assumptions was renewed, when it was realized that universally secure multi-party computation can be based on the setup assumption of tamper-proof hardware tokens. The tamper-proof hardware must suffice strong separation conditions, even if a more recent result showed that the assumptions about the physical separation can be relaxed to some extent [8,7].

Generally, the work on secure multi-party computation with tamper-proof hardware assumption can be divided in works dealing with either stateful or stateless hardware-tokens. Katz [19] considers a scenario where all parties can create and issue stateful tamper-proof hardware tokens. Using additional number theoretic assumptions, [19] implements a multiple commitment functionality in this scenario. Subsequently Moran and Segev [22] improved upon Katz result, by constructing commitments in an asymmetric scenario, where only one out of two parties is able to issue stateful tamper-proof hardware tokens. Hofheinz, Mller-Quade and Unruh [17] use (stateless) signature cards, issued by a trusted authority to achieve universal composability with respect to global setup assumptions [5]. Fischlin et al. [10] show how set intersection can be computed securely using a single untrusted tamper-proof hardware token and additional computational assumptions.

Goldwasser et al. [13] show that using a minimalistic stateful tamper-proof hardware assumption called One-Time-Memory, a new cryptographic primitive called One-Time-Program can be implemented. Recently, Kolesnikov [21] implemented string oblivious transfer with stateless tamper-proof hardware tokens. Goyal et al. [15] consider a unified treatment of tamper-proof hardware assumptions. Important in the context of this work, they show that in a mutually mistrusting setting, trusted One-Time-Programs can be implemented statistically secure from a polynomial number of OTMs. Goyal et al. [14] show that a single stateless tamper-proof hardware token is sufficient to implement statistically

[1] [2] mention that the provers in their protocol might be implemented as bank-cards

secure commitments and statistical zero-knowledge. Furthermore, if stateless tokens can be encapsulated into other stateless tokens, general statistically secure composable multi-party computation is possible in this setting. [14] also show that unconditionally secure Oblivious Transfer cannot be realized from stateless tamper-proof hardware alone.

With the exception of [2], all of the above schemes based on stateful tamper-proof hardware either use additional complexity assumptions to achieve secure two party computations [17,19,22,13,8,7,21] or a large number of hardware tokens must be issued [13,15]. The question if one single tamper-proof device, issued by one of two mistrusting parties, suffices for information-theoretically secure two-party computations remains open in the literature.

3 Our Contribution

In this paper, we show that general, statistically secure, composable two-party computations are possible in a setting where a single untrusted stateful tamper-proof hardware token can be issued by one party. All previous solutions supposed that either the creator of the tamper-proof hardware is honest, that additional complexity assumptions are used, or that a larger number of independent tamper-proof hardware tokens is issued. As a reasonable abstraction for the primitives that can be implemented in our setting, we introduce a new primitive which we call *Sequential-One-Time-Memory*. Sequential-One-Time-Memories provide a large amount of single One-Time-Memories, with the additional guarantee that the memory cells can only be queried sequentially. Just like One-Time-Memories, Sequential-One-Time-Memories have the property of being non-signaling to their issuer once they have been sent. We show that the Oblivious Transfer (OT) functionality can be realized straightforwardly using Sequential-One-Time-Memories, thus our results for statistically secure, composable two party computations follow immediately by [18,20]. Our main contribution is a statistically secure, universally composable protocol that realizes the Sequential-One-Time-Memory functionality with a single, untrusted, tamper-proof hardware token. Our protocol construction is efficient, it realizes m sequential n-bit-string-OTMs while having a communication overhead of $O(n^2m)$ bits in its interactive send phase, where n is the security parameter. This is achieved by using tokens that compute blinded tensor-product functions. The algebraic structure of these functions allows the token-sender to encapsulate two random keys, one of which the token-receiver can learn. In turn, the token-receiver has the promise that the token will not learn his choice-bit from his function input and that the key he computes from the function-output solely depends on his choice-bit. Moreover, these functions can be computed by simple linear algebra operations. The technical challenge in the security-proof, compared to issuing a large number of independent devices, is that an untrusted, adversarial token might deviate arbitrarily from the specification of an honest token. It may thus try to correlate its output or abort-behavior with previous inputs. Our protocol is semi-interactive as it requires an interactive send-phase. This interaction is necessary, as we can

further show that any protocol that implements Sequential-OTMs using a single untrusted device without any further interaction is insecure. Furthermore, a simple modification of our protocol yields a completely non-interactive protocol that implements Sequential-One-Time-Memories from two tamper-proof hardware tokens. Finally, we can resolve a question left open by [22] positively. Moran and Segev asked if multiple commitments from the receiver of a token to its issuer can be realized with a single untrusted tamper-proof hardware token. As we can realize Oblivious Transfer, standard techniques (e.g. [9,18]) can be used to implement commitments.

4 Framework

We state and prove our results in the Universal-Composability (UC) framework of [4]. In this framework security is defined by comparison of an *ideal model* and a *real model*. The protocol of interest is running in the latter, where an adversary $\mathcal{A}$ coordinates the behavior of all corrupted parties. In the ideal model, which is secure by definition, a simulator $\mathcal{S}$ tries to mimic the actions of $\mathcal{A}$. An environment $\mathcal{Z}$ is plugged either to the ideal or the real model and has to guess, which model it is actually plugged to.

By the random variable $\mathrm{View}_{\Pi}^{\mathcal{A}}(\mathcal{Z})$ we denote the complete view of the environment $\mathcal{Z}$ when plugged to the real model with protocol Π and adversary $\mathcal{A}$. Analogously, by the random variable $\mathrm{View}_{\mathcal{F}}^{\mathcal{S}}(\mathcal{Z})$ we denote the view of $\mathcal{Z}$ when plugged to the ideal model, where the functionality $\mathcal{F}$ realizes the protocol task and the simulator $\mathcal{S}$ coordinates the behavior of the corrupted parties. Therefore Π is a (statistically) UC-secure implementation of $\mathcal{F}$, if for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$, such that for all environments $\mathcal{Z}$ the random variables $\mathrm{View}_{\Pi}^{\mathcal{A}}(\mathcal{Z})$ and $\mathrm{View}_{\mathcal{F}}^{\mathcal{S}}(\mathcal{Z})$ are (statistically) close.

In our case the adversarial entities $\mathcal{A}, \mathcal{S}$ and the environment $\mathcal{Z}$ are computationally unbounded and a hybrid functionality $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$ models our tamper-proof hardware assumption (cf. Section 6). Moreover, as it was proven sufficient in [4], we always focus on the dummy adversary $\tilde{\mathcal{A}}$, which is completely controlled by the environment $\mathcal{Z}$.

5 Preliminaries

We use the notation $[m] := \{1, \ldots, m\}$. To formulate our protocol, we need some elementary concepts of linear algebra over finite fields. $\mathbb{F}_2$ is the finite field with two elements. We can canonically identify the vector-space $\mathbb{F}_2^n$ with the set $\{0,1\}^n$ of strings of length n. For two column-vectors $a \in \mathbb{F}_2^n$ and $b \in \mathbb{F}_2^k$ we write $ab^T = (a_i b_j)_{ij} \in \mathbb{F}_2^{n \times k}$ for the outer product (or tensor-product) of a and b. Similarly, for $a, b \in \mathbf{F}_2^n$ we denote the inner product by $a^T b = \sum_{i=1}^n a_i b_i \in \mathbb{F}_2$. Let $C \in \mathbb{F}_2^{n \times 2n}$. Then $\dim(\ker(C)) \geq n$. Let $B = \{b_1, \ldots, b_n\} \subseteq \ker(C)$ be a linearly independent set. We can choose a set $B^* = \{b_{n+1}, \ldots, b_{2n}\}$ such that $B \cup B^*$ is a basis of $\mathbb{F}_2^{2n}$. Let $e_i \in \mathbb{F}_2^n$ be the i-th unit-vector. Then there exists a matrix $G \in \mathbb{F}_2^{n \times 2n}$ such that $Gb_i = e_i$ for $i = 1, \ldots, n$ and $Gb_i = 0$

for $i = n+1, \ldots, 2n$. We call such a G a *complementary-matrix* to C. It holds that $\text{rank}(G) = n$ and $B^* \subseteq \ker(G)$. For such C and G, we can always solve linear equation systems $Cx = r$, $Gx = s$ by solving $Cx_{B^*} = r$ and $Gx_B = s$ independently for some $x_B \in \text{span}(B)$ and $x_{B^*} \in \text{span}(B^*)$. We can then set $x = x_B + x_{B^*}$. It holds $Cx = r$ and $Gx = s$, as $x_B \in \ker(C)$ and $x_{B^*} \in \ker(G)$.

6 Modeling Tamper-Proof Hardware

Our formulation of general stateful tamper-proof hardware resembles the meanwhile standard definitions of [19] and [22]. To model tamper-proof hardware, we employ the $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$ wrapper-functionality. A sender-party G (Goliath) provides as input a Turing-machine $\mathcal{M}$ to $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$. The receiver party D (David) can now query $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$ on inputs w, whereupon $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$ runs $\mathcal{M}$ on input w, sends the output y that $\mathcal{M}$ produced to D and stores the new state of $\mathcal{M}$. Every time D sends a new query w' to $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$, it resumes simulating $\mathcal{M}$ with its most recent state, sends the output to D and updates the stored state of $\mathcal{M}$. This captures the following properties one expects from tamper-proof hardware. First, G is unable to revoke $\mathcal{M}$ once it has sent a token to D. Second, D can run $\mathcal{M}$ on inputs of his choice, but the program-code and state of $\mathcal{M}$ are out of reach for D, due to the tokens tamper-proofness. Note that stateful tokens don't need a trusted source of randomness, as $\mathcal{M}$ can be provided with a sufficiently long hard-coded random-tape. Thus we can, w.l.o.g restrict $\mathcal{M}$ to be deterministic. See Figure 1.

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

Parametrized by an implicit security parameter n and a (polynomial) runtime bound $p(\cdot)$.

Creation Upon receiving a message (`create`,sid,P_i,P_j,$\mathcal{M}$) from party P_i, where $\mathcal{M}$ is a deterministic Turing-machine, go to state `ready` and store $(\mathcal{M}, \perp)$, where $\perp$ is the initial state of $\mathcal{M}$. Send (`ready`,sid,P_i,P_j) to party P_j.

Execution Upon receiving a message (`run`,sid,P_i,P_j,in) where in is an input word, first check if current state is `ready`. If so, run $\mathcal{M}(\text{state}, in)$ for at most $p(k)$ steps. Read *out* from the output tape of $\mathcal{M}$ and save its new state. Send (`out`,sid,P_i,P_j,*out*) to P_j.

Fig. 1. The wrapper functionality that models general stateful tamper-proof hardware

A very basic tamper-proof hardware primitive called One-Time-Memory (OTM) (Figure 2), that can be considered minimal in the sense that it only needs to be able to erase its contents after being queried, was defined in [13]. OTMs resemble the well known Oblivious Transfer (OT) functionality [23,9], with just

a slight difference. While OT notifies the sender when the receiver queries the primitive, this is not the case for OTM. Moreover, OTMs have the property of immediate delivery, which means that even a malicious sender cannot revoke or abort the functionality once it has been sent. Together with One-Time-Memories, [13] defined the notion of One-Time-Programs (OTP), and its generalization, k-Time-Programs (k-TP). OTP is a functionality that allows a receiver to evaluate a circuit on exactly one input, then it shuts down. Similarly, a k-TP can be evaluated exactly k-times, then shuts down. While [13] showed that OTPs can be realized with OTM and computational assumptions and assumed that the sender of the OTPs is trusted, [15] provided an unconditionally secure implementation of *trusted* OTPs using only OTMs, where the sender of the OTPs is untrusted.

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

Parametrized by a security parameter n.

Creation Upon receiving a message (`create`,sid,P_i,P_j,(s_0, s_1)) from party P_i, go to state `ready` and store (s_0, s_1). Send (`ready`,sid,P_i,P_j) to party P_j.

Choice Upon receiving a message (`choice`,sid,P_i,P_j,x) from P_j, check if current state is `ready`. If so, send (`out`,sid,P_i,P_j,s_x) to P_j and go to state `dead`.

Fig. 2. The One-Time-Memory Functionality

We introduce a new variant of the One-Time-Memory functionality that we call Sequential-One-Time-Memory (Seq-OTM) (Figure 3). Seq-OTM constitutes a set of m single OTM functionalities, with the restriction that they can only be queried in a fixed order. We refer to the single OTMs of Seq-OTM as its stages. We impose the sequential ordering to the stages, to model that the untrusted token might stop answering queries after an arbitrary stage. The limitation to sequential access makes applications impossible where random access is essential. All constructions where sequential access is sufficient can be implemented with Seq-OTM as well. As the construction of unconditionally secure trusted OTPs [15] using multiple OTM tokens is quite involved, we cannot present it here. However, we observe that the construction of [15] works using Seq-OTM instead of multiple independent OTM tokens. This holds because in this construction, an honest receiver queries all the OTM primitives he received in a fixed order anyway. Interestingly, the technical challenges [15] deals with in constructing unconditionally secure OTPs arises from the fact that a malicious receiver might query the OTMs *out of order*. Finally, a remark is in place. Even though Seq-OTM can be used to implement several OTPs, the sequential nature of Seq-OTM demands that those OTPs can only be executed in a fixed order. If one wishes to execute several OTPs in random order, multiple Seq-OTMs (and thus multiple hardware tokens) have to be issued.

The restriction to sequential access also bears advantages in certain applications. In [15], a construction was provided that realizes a single OT from polynomially many OTM tokens. The construction is rather expensive regarding the number of OTMs used, as it involves One-Time-Programs to issue a proof that the primitive has been queried. Such a proof is necessary to convince the OT-sender that the receiver has provided his input. Again, what makes this construction technically challenging is that the single OTM tokens can be queried in an arbitrary order.

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

Parametrized by a security parameter n and a parameter $m = \mathrm{poly}(n)$, which is the number of single OTMs that can be stored.

Creation Upon receiving a message (`create`,sid,P_i,P_j,$((s_{1,0}, s_{1,1}), \ldots, (s_{m,0}, s_{m,1}))$) from party P_i, go to state `ready` and store $((s_{1,0}, s_{1,1}), \ldots, (s_{m,0}, s_{m,1}))$. Set a counter $t = 1$. Send (`ready`,sid,P_i,P_j) to party P_j.

Choice Upon receiving a message (`choice`,sid,P_i,P_j,x) from P_j, check if current state is `ready` and $t \leq m$. If so, send (`out`,sid,P_i,P_j,$s_{t,x}$) to P_j and increment t by 1. If $t > m$, go to state `dead`.

Fig. 3. The Sequential-One-Time-Memory Functionality

We will now briefly outline how a polynomial number of OTs can be implemented using the Seq-OTM functionality. Given a Seq-OTM primitive with $2m$ single OTMs, $\mathrm{OTM}_1, \ldots, \mathrm{OTM}_{2m}$, the sender programs OTM_{2i-1} with his inputs for OT_i and programs a random-string r_i into OTM_{2i}. In the choice-phase, the receiver queries OTM_{2i-1} with his i-th choice-bit x_i and OTM_{2i} with choice-bit 0. To prove to the sender that OTM_{2i-1} has already been queried, the receiver sends r_i to the sender. As Seq-OTM forces the receiver to open $\mathrm{OTM}_1, \ldots, \mathrm{OTM}_{2m}$ sequentially, he has only negligible chance to guess r_i correctly unless he has opened OTM_{2i-1} already. Thus, this reduction is perfectly secure against the sender of the OT and statistically secure against the receiver of the OT. Noting that OT can be stored and reversed [1,24,25], we conclude that in the Seq-OTM hybrid-model, OT can be implemented in both ways (from the Seq-OTM sender to the Seq-OTM receiver and from the Seq-OTM receiver to the Seq-OTM sender).

7 The Necessity of Interaction

We will now show that any protocol, which implements a Seq-OTM with 2 or more stages from a single hardware token, requires some interaction between sender and receiver. The proof is a variant of the impossibility result of [6], which

states that Bit-Commitments cannot be realized in the plain UC-model. We show that any non-interactive protocol, realizing Seq-OTM from a single tamper-proof device, that is secure against a corrupted receiver is necessarily insecure against a corrupted sender. More precisely, once there exists a simulator against a corrupted receiver, a corrupted sender can use this simulator to construct a token that learns the receiver's inputs. Such a token can make the output of the second stage dependent on the input of the first stage. However, this behavior cannot be simulated, because in the ideal experiment, the sender's inputs to $\mathcal{F}^{\mathrm{Seq-OTM}}$ have to be provided before the choice-phase begins, and can thus not depend on any choice-bits.

Remark 1. This argument assumes that both the receiver and $\mathcal{T}$ are equivalent in their computational resources, as it uses the simulator against a corrupted receiver to construct a corrupted token. Thus, it only holds if the receiver is subject to the same computational restrictions as the token. If we allow the receiver (and thus the simulator against a corrupted receiver) to be computationally unbounded but require the token to run in polynomial time, there may exist non-interactive UC-secure protocols for $\mathcal{F}^{\mathrm{Seq-OTM}}$ from $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$ relative to some computational assumption.

Theorem 1. *There is no non-interactive protocol which UC-realizes $\mathcal{F}^{\mathrm{Seq-OTM}}$ with two or more stages from a single tamper-proof hardware instance $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$, given that the receiver D and the token $\mathcal{T}$ ran by $\mathcal{F}_{\mathrm{wrap}}^{\mathrm{stateful}}$ are bounded by the same computational restrictions.*

Proof. Assume there exists a protocol Π that UC-realizes a Seq-OTM with two stages, which requires no further interaction between sender and receiver. For simplicity, we assume that the Seq-OTM stages are bit-OTMs. As Π is UC-secure, there exists a receiver-simulator $\mathcal{S}_D$ which extracts the choice-bits x_1 and x_2 from the dummy-adversary $\tilde{\mathcal{A}}_D$. As Π is non-interactive, $\tilde{\mathcal{A}}_D$ only interacts with a token $\mathcal{T}$. But that means that $\mathcal{S}_D$ must be able to extract $\tilde{\mathcal{A}}_D$'s input from the messages $\tilde{\mathcal{A}}_D$ has provided to $\mathcal{T}$. Consider the case of a corrupted sender. We will provide an environment $\mathcal{Z}'$ that distinguishes real and ideal with probability $\geq \frac{1}{2}$. $\mathcal{Z}'$ constructs a token $\mathcal{T}$, which does the following. The token $\mathcal{T}$ internally simulates the simulator $\mathcal{S}_D$ and some functionality $\mathcal{F}'$. $\mathcal{S}_D$ is wired to $\mathcal{F}'$ instead of $\mathcal{F}^{\mathrm{Seq-OTM}}$. $\mathcal{F}'$ is a "corrupted" version of $\mathcal{F}^{\mathrm{Seq-OTM}}$, which always gives 0 as its first-stage OTM-output and x_1 (its first-stage input) as its second-stage output. $\mathcal{Z}'$ instructs $\tilde{A}_G$ to input $\mathcal{T}$ into $\mathcal{F}_{\mathrm{stateful}}^{\mathrm{wrap}}$. In the choice-phase, $\mathcal{Z}'$ sets the receiver's first choice-bit x_1 to a uniformly random value and the second choice-bit x_2 to 0. This concludes the description of $\mathcal{Z}'$. We claim that for every simulator $\mathcal{S}_G$, the statistical distance between $\mathrm{View}_{\Pi}^{\mathcal{A}}(\mathcal{Z}')$ and $\mathrm{View}_{\mathcal{F}}^{\mathcal{S}}(\mathcal{Z}')$ is at least $\frac{1}{2}$. In the real experiment, the output s_2 that the receiver gets will always be the same as his first input x_1. In the ideal experiment however, the simulator $\mathcal{S}_G$ has to guess the choice-bit x_1 in advance. As x_1 depends solely on an internal coin-toss of $\mathcal{Z}'$, the chance of $\mathcal{S}_G$ guessing x_1 correctly is $\leq \frac{1}{2}$. Thus, the probability that D gives an incorrect output to $\mathcal{Z}'$ is $\geq \frac{1}{2}$.

8 The David-and-Goliath Sequential-OTM Protocol

In this Section, we will describe our protocol $\Pi_{\text{Seq-OTM}}$ for David-and-Goliath Sequential-OTM and give a sketch of the security proof. We will refer to the sender of the Seq-OTM as Goliath and to the receiver as David. Let n be the statistical security parameter. We first explain the token program that will be used in the protocol. The token is created with m affine functions stored in it. It has a counter t, which is initialized with 1. In the execution phase, if the token is queried with an input x, it will evaluate its t-th affine function on x and output the result. After each query the counter t is incremented by 1. When the counter reaches m, the token stops functioning.

Before we describe in more detail how these tokens can be used to implement Sequential-OTM, we will explain more precisely which kind of affine functions are stored on the token. Let $a \in \mathbb{F}_2^{2n}$, $B \in \mathbb{F}_2^{2n \times 2n}$ and $z \in \mathbb{F}_2^{2n}$ be an input vector. The functions have the form $V(z) = az^T + B$. Thus, $V(z)$ is an outer product az^T blinded by a matrix B. See Figure 4.

Program: Seq-OTM Token

Parametrized by a security parameter n.

Hardwired Inputs

- A parameter $m \in \mathbb{N}$, the maximum number of stages.
- $(a_1, B_1), \ldots, (a_m, B_m)$, where $a_i \in \mathbb{F}_2^{2n}$ and $B_i \in \mathbb{F}_2^{2n \times 2n}$ for $i \in [m]$

Stateful Variables

- A counter variable t that is, upon creation, initialized with $t = 1$

Execution Upon receiving an input $z \in \mathbb{F}_2^{2n}$

- Check if $t \leq m$, if not abort.
- Compute $V_t = a_t z^T + B_t$.
- Set $t \leftarrow t + 1$.
- Output V_t and wait for further inputs.

Fig. 4. The program that is run by an honest token for Sequential-One-Time-Memories.

We will first give a high-level picture of protocol $\Pi_{\text{Seq-OTM}}$. The protocol realizes a Seq-OTM with $m = \text{poly}(n)$ stages, where each stage is an n-bit-string OTM. It has three phases. In the first phase, Goliath creates a token with random affine functions stored on it, then sends the wrapped token to David. Hereafter Goliath is physically committed to the token. The second phase is the only interactive part of the protocol. It involves the sending of check-values and one-time-pad-encrypted OTM inputs. In the third phase, which is non-interactive,

David queries the token to get his desired one-time-pad keys. David's input to the token is a randomized value, in which his choice-bit is hidden. After receiving output from the token, David checks the output with the check-values provided by Goliath and aborts if the output does not pass the check. If the output passes the check, David computes a key and decrypts his desired OTM output.

We will now describe the protocol in detail. In the creation-phase, Goliath programs the token $\mathcal{T}$ with uniformly chosen $(a_i, B_i) \in \mathbb{F}_2^{2n} \times \mathbb{F}_2^{2n \times 2n}$, for $i = 1, \ldots, m$. Goliath then sends the wrapped token $\mathcal{T}$ to David.

The send-phase proceeds as follows. Let $(s_{i,0}, s_{i,1}) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$ be Goliath's i-th Seq-OTM input. First, David announces a randomly chosen check-matrix $C \in \mathbb{F}_2^{n \times 2n}$ to Goliath. Then for $i = 1, \ldots, m$, Goliath computes $(\tilde{a}_i, \tilde{B}_i) \in \mathbb{F}_2^n \times \mathbb{F}_2^{n \times 2n}$ by $\tilde{a}_i = Ca_i$ and $\tilde{B}_i = CB_i$. Goliath further chooses a matrix $G \in \mathbb{F}_2^{n \times 2n}$ such that G is a *complementary*-matrix of C (cf. Section 5). Goliath now sends the $(\tilde{a}_i, \tilde{B}_i)_{i \in [m]}$ and G to David. The $(\tilde{a}_i, \tilde{B}_i)$ can be seen as commitments that commit the token $\mathcal{T}$ to a unique output behavior, but reveal no meaningful information to David. After David has received $(\tilde{a}_i, \tilde{B}_i)_{i \in [m]}$ and G, he chooses a random $\mathbf{h} = (h_1, \ldots, h_m) \in \left(\mathbb{F}_2^{2n} \backslash \{0\}\right)^m$ and sends $\mathbf{h}$ to Goliath. Goliath computes *ciphertexts* $\tilde{s}_{i,0} = s_{i,0} + GB_ih_i$ and $\tilde{s}_{i,1} = s_{i,1} + GB_ih_i + Ga_i$ for $i = 1, \ldots, m$ and sends $((\tilde{s}_{i,0}, \tilde{s}_{i,1}))_{i \in [m]}$ to David. This concludes the send-phase.

The choice-phase is non-interactive and has m stages. In stage i David queries the i-th OTM of the Seq-OTM functionality being implemented. Let $x_i \in \mathbb{F}_2$ be David's i-th choice-bit. David uniformly samples an element $z_i \in \mathbb{F}_2^{2n}$ such that $z_i^T h_i = x_i$. That is, if $x_i = 0$ David samples z_i from the $2n-1$ dimensional hyperplane $A_{i,0} = \{z \in \mathbb{F}_2^{2n} : z^T h_i = 0\}$. Likewise, if $x_i = 1$, then z_i is sampled from the hyperplane $A_{i,1} = \{z \in \mathbb{F}_2^{2n} : z^T h_i = 1\}$. Let V_i be the output of the token when input z_i in stage i. If the token was created honestly, it holds that $V_i = a_i z_i^T + B_i$. David checks if $CV_i \stackrel{?}{=} \tilde{a}_i z^T + \tilde{B}_i$. If not, then the token computed a different function than the one Goliath committed to and David aborts the protocol. If the check is passed, David computes $s_{i,x_i} = \tilde{s}_{i,x_i} + GV_ih_i$ and outputs s_{i,x_i}. This concludes the description of the protocol. The full protocol is given in Figure 5.

Discussion. We will shortly sketch the ideas behind this construction. First notice, that from the view of the token, David's inputs look almost uniform. This comes from the fact that the token is oblivious of the randomly chosen vector h_i, that defines $A_{i,0}$ and $A_{i,1}$. In fact, it can be shown, that a complete input history $\mathbf{z} = (z_1, \ldots, z_m) \in \left(\mathbb{F}_2^{2n}\right)^m$, for adversarially chosen choice-bits of David, is statistically close to a uniformly chosen input history $\mathbf{u} = (u_1, \ldots, u_m) \in \left(\mathbb{F}_2^{2n}\right)^m$. This, in turn, means that a dishonest token is oblivious of David's input. The check-operation is performed to make sure that the token responds to queries in an unambiguous way and that it does not encode further information into Davids output. If the token answers dishonestly, David will notice that with overwhelming probability. The token's output V_i needs to be projected with the matrix G, so that David cannot learn anything about Goliath's inputs $s_{i,0}$ and $s_{i,1}$ from $\tilde{a}_i$ and $\tilde{B}_i$. Notice that Goliath commits to the check-values $(\tilde{a}_1, \tilde{B}_1), \ldots, (\tilde{a}_m, \tilde{B}_m)$

Protocol $\Pi_{\textbf{Seq-OTM}}$: David & Goliath Sequential OTM

Let n be the statistical security parameter and let $m = \text{poly}(n)$ be the number of sequential OTMs. Further let $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$ be the wrapper-functionality that models the tamper-proof hardware.

Creation-Phase: (**only Goliath**)

- For every index $i \in [m]$, choose $a_i \in \mathbb{F}_2^{2n}$ and $B_i \in \mathbb{F}_2^{2n \times 2n}$ uniformly at random.
- Program a Seq-OTM token $\mathcal{T}$ with hardwired inputs m and (a_i, B_i) for $i \in [m]$.
- Send $\mathcal{T}$ to $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$.

Send-Phase: Let $((s_{1,0}, s_{1,1}), \ldots, (s_{m,0}, s_{m,1})) \in (\mathbb{F}_2^n \times \mathbb{F}_2^n)^m$ be Goliath's Seq-OTM input.

1. (**David**) Wait until the `ready` message from $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$ was received. Then, choose a random matrix $C \in \mathbb{F}_2^{n \times 2n}$ and send C to Goliath.
2. (**Goliath**) Compute a matrix $G \in \mathbb{F}_2^{n \times 2n}$, such that G is complementary to C. For all $i \in [m]$, set $\tilde{a}_i = Ca_i$ and $\tilde{B}_i = CB_i$. Send $(G, (\tilde{a}_i, \tilde{B}_i)_{i \in [m]})$ to David.
3. (**David**) Choose $\mathbf{h} = (h_1, \ldots, h_m) \in \left(\mathbb{F}_2^{2n} \backslash \{0\}\right)^m$ uniformly a random and send $\mathbf{h}$ to Goliath.
4. (**Goliath**) For each $i \in [m]$, set $\tilde{s}_{i,0} = s_{i,0} + GB_ih_i$ and $\tilde{s}_{i,1} = s_{i,1} + GB_ih_i + Ga_i$. Send $(\tilde{s}_{i,0}, \tilde{s}_{i,1})_{i \in [m]}$ to David.

Choice-Phase (Stage $i \in [m]$): (**only David**) Let $x_i \in \mathbb{F}_2$ be David's i-th Seq-OTM input.

- Choose $z_i \in \mathbb{F}_2^{2n}$ uniformly at random such that $z_i^T h_i = x_i$ and input z_i into $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$.
- Let V_i be the output of $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$. Check if $CV_i = \tilde{a}_i z_i^T + \tilde{B}_i$. If not, go to abort state and output $\perp$ for this query and for all further queries.
- If the check is passed, output $s_{i,x_i} = \tilde{s}_{i,x_i} + GV_ih_i$.

Fig. 5. A protocol for sequential OTM from a single stateful token

before he sees the vectors $(h_1, \ldots, h_m)$. Thus, the condition on which David aborts is independent of the choice of $(h_1, \ldots, h_m)$. The use of the outer-product operation for the affine functions stems from a subtle issue. We need to ensure that the environment $\mathcal{Z}$ does not see any of the random coins David used to sample the z_i. If a corrupted Goliath could create a token that encodes several bits of z_i in its output, the environment might be able to tell apart real and ideal model in a later stage. Such a token might, for instance, abort in a later stage if some of the bits of z_i fulfill a certain condition (e.g. have odd parity or some special hash value). Given that the token actually computes the specified function (which is enforced by the check-operation), the outer-product form ensures that there are only two different outputs an honest David might produce in each stage. Particularly, let $V(z) = az^T + B$ be one of the functions computed by the token. Basically, the outer product form allows David to derandomize the

token's output. If x is such that $z^T h = x$, then the randomness of z is removed by $V(z)h = (az^T + B)h = az^T h + Bh = ax + Bh$. The term $ax + Bh$ is independent of the random coins used to sample z. Consequently, David's outputs s_{i,x_i} are (with overwhelming probability) independent of the random coins used to sample z.

Correctness of the Protocol. If both parties are honest, the correctness of the protocol can be seen straightforwardly. Let s'_{i,x_i} be one of David's outputs.

$$\begin{aligned} s'_{i,x_i} &= \tilde{s}_{i,x_i} + GV_i h_i = \tilde{s}_{i,x_i} + G(a_i z_i^T + B_i)h_i = \tilde{s}_{i,x_i} + Ga_i z_i^T h_i + GB_i h_i \\ &= \tilde{s}_{i,x_i} + Ga_i x_i + GB_i h_i = s_{i,x_i} \end{aligned}$$

8.1 Security of $\Pi_{\text{Seq-OTM}}$

The security of protocol $\Pi_{\text{Seq-OTM}}$ is summarized in Theorem 2.

Theorem 2. *Protocol $\Pi_{Seq\text{-}OTM}$ UC-realizes the $\mathcal{F}^{\text{Seq}-\text{OTM}}$ functionality, with perfect security against a corrupted receiver and statistical security against a corrupted sender.*

We will provide the security-proof against a corrupted receiver and outline the security-proof against a corrupted sender

Corrupted Receiver. We will first consider the case of a corrupted receiver, as this is the easy case. Let $\tilde{\mathcal{A}}$ be the dummy adversary for David. We will provide a simulator $\mathcal{S}_D$ and show, that for any environment $\mathcal{Z}$ the distributions $\text{View}^{\tilde{\mathcal{A}}}_{\Pi_{Seq-OTM}}(\mathcal{Z})$ and $\text{View}^{\mathcal{S}_D}_{\mathcal{F}_{Seq-OTM}}(\mathcal{Z})$ are identical. Simulator $\mathcal{S}_D$ runs in strict polynomial time. The simulator $\mathcal{S}_D$ is given in Figure 6.

First notice that there is always a solution V_i for $CV_i = \tilde{a}_i z_i^T + \tilde{B}_i$ and $GV_i h_i = \tilde{s}_{i,x_i} + s_{i,x_i}$, as G is a complementary matrix of C and has rank n. Furthermore, from $\tilde{\mathcal{A}}$'s view, each V_i with $CV_i = \tilde{a}_i z_i^T + \tilde{B}_i$ and $GV_i h_i = \tilde{s}_{i,x_i} + s_{i,x_i}$ is equally likely. Thus we obtain perfect indistinguishability for $\text{View}^{\tilde{\mathcal{A}}}_{\Pi_{Seq-OTM}}(\mathcal{Z})$ and $\text{View}^{\mathcal{S}_D}_{\mathcal{F}_{Seq-OTM}}(\mathcal{Z})$.

Corrupted Sender. The case of a corrupted sender is the technically challenging part of this proof. Let $\tilde{\mathcal{A}}$ be the dummy adversary for Goliath. We will provide a simulator $\mathcal{S}_G$ and show, that for any environment $\mathcal{Z}$ the distributions $\text{View}^{\tilde{\mathcal{A}}}_{\Pi_{Seq-OTM}}(\mathcal{Z})$ and $\text{View}^{\mathcal{S}_G}_{\mathcal{F}_{Seq-OTM}}(\mathcal{Z})$ are statistically close. Simulator $\mathcal{S}_G$ runs in expected polynomial time. The simulator $\mathcal{S}_G$ is given in Figure 7.

Remark 2. Let $\mathcal{T}$ be a (possibly malicious) token program. Write $(V_1, \ldots, V_m) = \mathcal{T}(z_1, \ldots, z_m)$ for a token run with inputs $(z_1, \ldots, z_m)$ and outputs $(V_1, \ldots, V_m)$.

Simulator $\mathcal{S}_D$

- Setup a Goliath-machine $\mathcal{G}$ and provide $\mathcal{G}$ with a random tape.
- Simulate the creation phase with machine $\mathcal{G}$ and extract a Seq-OTM token program $\mathcal{T}$
- Send the `ready` message to $\tilde{\mathcal{A}}$.
- If $\tilde{\mathcal{A}}$ queries $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$ with a z_i, before h_i has been received, wire the input to $\mathcal{T}$ and forward $\mathcal{T}$'s output V_i to $\tilde{\mathcal{A}}$.
- Simulate the send phase between $\mathcal{G}$ and $\tilde{\mathcal{A}}$. For all indices $i \in [m]$ such that $\tilde{\mathcal{A}}$ has queried $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$ before sending h_i, upon receiving h_i, set $x_i = z_i^T h_i$. Query $\mathcal{F}^{\text{Seq-OTM}}$ with x_i and receive s_{i,x_i}. Set $\tilde{s}_{i,x_i} = s_{i,x_i} + GV_i h_i$. Except for that, continue the interaction between $\mathcal{G}$ and $\tilde{\mathcal{A}}$ normally.
- For every index i such that $\tilde{\mathcal{A}}$ queries $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$ with input z_i after h_i has been received, compute $x_i = z_i^T h_i$. Query $\mathcal{F}^{\text{Seq-OTM}}$ with x_i and receive s_{i,x_i}. Uniformly sample $V_i \in \mathbb{F}_2^{2n \times 2n}$ such that $CV_i = \tilde{a}_i z_i^T + \tilde{B}_i$ and $GV_i h_i = \tilde{s}_{i,x_i} + s_{i,x_i}$. Forward V_i to $\tilde{\mathcal{A}}$.

Fig. 6. The simulator for a corrupted receiver.

The proof proceeds roughly as follows. In a hybrid argument, we show for each stage i, that if David does not abort in stage i, then for both inputs $x_i = 0$ and $x_i = 1$, the simulator can extract outputs $\hat{V}_{i,0}$ and $\hat{V}_{i,1}$ from $\mathcal{T}$ that David both would accept. The extraction is done using a rewind-to-the-beginning technique. This means, instead of rewinding an existing run to the previous stage, we sample a completely new run. We can show that the event that the simulator fails to extract happens only with negligibly small probability. Next, we show that the token's output in the hybrid real part matches the corresponding extracted output with overwhelming probability (over the simulators coins). We can thus modify the simulator to use the extracted token outputs instead of the token output from the hybrid real part. After this transformation, David's outputs are ideal. Thus, only the stage at which David aborts, supposed that he aborts, depends on the hybrid real part (and thus on input made by $\mathcal{Z}$). We will call the random variable that describes at which stage David aborts the stoptime S. To determine the stage at which to abort in the ideal part, the simulator runs the token with purely random inputs $u_1^*, \ldots, u_m^*$. Let $(V_1^*, \ldots, V_m^*) = \mathcal{T}(u_1^*, \ldots, u_m^*)$. Let i_0 be the first index such that $CV_i \neq \tilde{a}_i u_i^{*T} + \tilde{B}_i$. The simulator sets David's outputs to be $\perp$ for indices $i_0, \ldots, m$. We can show that from $\mathcal{Z}$'s view, the stoptime S of the hybrid real part is statistically close to the stoptime S' of the hybrid ideal part.

Consider the following sequence of games. In game i, the environment $\mathcal{Z}$ interacts with simulator $\mathcal{S}_i$. As the send-phase is identical for the real protocol and the simulation, we only care about the choice-phase. Recall that it has stages $1, \ldots, m$.

Game 0 Simulator $\mathcal{S}_0$ simulates the real protocol $\Pi_{\text{Seq-OTM}}$.

Simulator $\mathcal{S}_G$

Simulation

- Setup a David-machine $\mathcal{D}$ and provide $\mathcal{D}$ with a random tape.
- Wait until $\tilde{\mathcal{A}}$ sends a token $\mathcal{T}$ to $\mathcal{F}_{\text{wrap}}^{\text{stateful}}$.
- Simulate the send phase between $\tilde{\mathcal{A}}$ and $\mathcal{D}$

Extraction Once the send-phase is complete, proceed as follows. The variables $C, G, h_i, \tilde{a}_i, \tilde{B}_i, \tilde{s}_{i,0}, \tilde{s}_{i,1}$ are taken from the view of $\mathcal{D}$.

- Choose $\mathbf{u}^* = (u_1^*, \ldots, u_m^*) \in \left(\mathbb{F}_2^{2n}\right)^m$ uniformly. Compute $(V_1^*, \ldots, V_m^*) = \mathcal{T}(\mathbf{u}^*)$.
- For $i = \{1, \ldots, m\}$
  - If $CV_i^* \neq \tilde{a}_i u_i^{*T} + \tilde{B}_i$, then set $s_{j,0} = s_{j,1} = \perp$ for $j \geq i$ and abort loop.
  - Otherwise, for $x = 0$ and $x = 1$, try for $2^{\frac{n}{4}}$ times
    - Choose $\mathbf{u} \in \left(\mathbb{F}_2^{2n}\right)^{i-1}$ uniformly at random.
    - Choose $z \in \mathbb{F}_2^{2n}$ uniformly at random such that $z^T h_i = x$.
    - Let $(V_1', \ldots, V_i') = \mathcal{T}(\mathbf{u}, z)$. If for all $j \in [i-1]$ it holds that $CV_j' = \tilde{a}_j u_j^T + \tilde{B}_j$ and $CV_i' = \tilde{a}_i z^T + \tilde{B}_i$, then set $s_{i,x} = \tilde{s}_{i,x} + GV_i' h_i$ and exit loop, otherwise repeat trying.
- Input $((s_{1,0}, s_{1,1}), \ldots, (s_{m,0}, s_{m,1}))$ into $\mathcal{F}^{\text{Seq-OTM}}$.

Fig. 7. The simulator for a corrupted sender.

Game i (for $i = 1, \ldots, m$) $\mathcal{S}_i$ does the same as $\mathcal{S}_{i-1}$, except for the following. In stage i, if V_i passes the test $CV_i = \tilde{a}_i z_i^T + \tilde{B}_i$ sample $\hat{V}_{i,0}$ and $\hat{V}_{i,1}$ in the following manner. For both $x = 0$ and $x = 1$ repeat for at most $2^{\frac{n}{4}}$ times:

- Uniformly choose input-history $(u_1, \ldots, u_{i-1}) \in \left(\mathbb{F}_2^{2n}\right)^{i-1}$.
- Uniformly choose $z \in \mathbb{F}_2^{2n}$ such that $z^T h_i = x$.
- Let $(V_1', \ldots, V_i') = \mathcal{T}(u_1, \ldots, u_i, z)$. If for all $j \in [i]$ it holds that $CV_j' = \tilde{a}_j u_j^T + \tilde{B}_j$ and $CV_i' = \tilde{a}_i z^T + \tilde{B}_i$, then set $\hat{V}_{i,x} = V_i'$ and exit loop.

If $\mathcal{S}_i$ fails to sample either $\hat{V}_{i,0}$ or $\hat{V}_{i,1}$ after the $2^{\frac{n}{4}}$ iterations, $\mathcal{S}_i$ halts.

Game $m + i$ (for $i = 1, \ldots, m$) The same as game $m + i - 1$, except that now David's i-th output is $G\hat{V}_{i,x_i} h_i + \tilde{s}_{i,x_i}$ instead of $GV_i h_i + \tilde{s}_{i,x_i}$.

Game $2m + 1$ The same as game $2m$, except that now the stage at which David aborts is determined differently. $\mathcal{S}_{2m+1}$ samples $(u_1^*, \ldots, u_m^*) \in \left(\mathbb{F}_2^{2n}\right)^m$ uniformly at random. Let $(V_1^*, \ldots, V_m^*) = \mathcal{T}(u_1^*, \ldots, u_m^*)$. For every stage $i \in [m]$, $\mathcal{S}_{2m+1}$ replaces the abort condition $CV_i = \tilde{a}_i z_i^T + \tilde{B}_i$ by $CV_i^* = \tilde{a}_i u_i^{*T} + \tilde{B}_i$. This is the ideal game.

Proof Techniques. We will briefly sketch the proofs that establish indistinguishability between successive games. The indistinguishability of games $i - 1$

and i, for $i = 1, \ldots, m$ can be established as follows. Once a token $\mathcal{T}$ and a check-matrix C (together with $(\tilde{a}_1, \tilde{B}_1), \ldots, (\tilde{a}_m, \tilde{B}_m)$) are fixed, we can define a set $D_{\mathcal{T},C}$ of accepting input-histories. That is, $D_{\mathcal{T},C}$ consists of all input histories $(z_1, \ldots, z_i)$ for which David accepts $\mathcal{T}$'s corresponding outputs. As h_i is chosen independently of $D_{\mathcal{T},C}$, almost every choice of h_i is *good*, in the sense that it partitions $D_{\mathcal{T},C}$ in two sets of almost (up to negligible) the same size, one for which $z_i^T h_i = 0$ and $z_i^T h_i = 1$ for the other. As an analogy, h_i can be thought of as a universal hash function. If we fix h_i to be good in the above-mentioned sense, the chance to find an accepting run that belongs to choice-bit 0 roughly equals the chance to find an accepting run for choice bit 1. It can be shown that the probability of the hybrid real part producing a run that lies in $D_{\mathcal{T},C}$ is statistically close to a uniformly chosen run lying in $D_{\mathcal{T},C}$. Hence, the simulator's probability of extraction-failure is negligible (over its random coins).

The indistinguishability for games $m + i - 1$ and $m + i$, for $i = 1, \ldots, m$ is proven as follows. We need to show that for a fixed h_i and a fixed choice-bit x_i, Davids output $\tilde{s}_{i,x_i} + GV_i h_i$ in the hybrid real part of game $m + i$ and the extracted output $\tilde{s}_{i,x_i} + G\hat{V}_{i,x_i} h_i$ are identical. Assume there was a token $\mathcal{T}$ so that $\mathcal{T}$ succeeds to output different V_i and $\hat{V}_{i,x_i}$ such that $V_i h_i \neq \hat{V}_{i,x_i} h_i$, that are both accepted by David. Let z be the token-input corresponding to $\hat{V}_{i,x_i}$. If David accepts both runs, it must hold that

$$CV_i = \tilde{a}_i z_i^T + \tilde{B}_i \tag{1}$$

$$C\hat{V}_{i,x_i} = \tilde{a}_i z^T + \tilde{B}_i. \tag{2}$$

This implies that

$$CV_i h_i = \tilde{a}_i z_i^T h_i + \tilde{B}_i h_i = \tilde{a}_i x_i + \tilde{B}_i h_i \tag{3}$$

$$C\hat{V}_{i,x_i} h_i = \tilde{a}_i z^T h_i + \tilde{B}_i h_i = \tilde{a}_i x_i + \tilde{B}_i h_i, \tag{4}$$

and thus $CV_i h_i = C\hat{V}_{i,x_i} h_i$ holds. But this means that $\mathcal{T}$ could as well form hash-collisions for the universal hash-function C, of which it is oblivious. However, this event has only negligible probability.

Finally, in game $2m$, all of David's output values are determined in the hybrid ideal part. From $\mathcal{Z}$'s view, the probability that either of the simulators $\mathcal{S}_{2m}$ or $\mathcal{S}_{2m+1}$ halts due to an extraction error is negligible. Thus we need to show that the stoptimes S and S' for game $2m$ and game $2m+1$ are statistically close from the view of $\mathcal{Z}$. The proof needs to take into account that all the C and h_i are contained in $\mathcal{Z}$'s view. Again, it can be shown that almost all choices for the h_i are *good*, which shows that even for fixed C and h_i the stoptimes S and S' in game $2m$ and S' in game $2m + 1$ are statistically close.

8.2 A Non-Interactive Protocol using two Tokens

The send-phase in protocol $\Pi_{\mathrm{Seq-OTM}}$ (Figure 5) is interactive. We have proven this interaction to be necessary, given that only one token is issued (Theorem

1). However, if we allow the sender Goliath to issue a second stateful token $\mathcal{T}'$ to David, then we obtain a non-interactive protocol $\Pi'_{\text{Seq-OTM}}$ for Seq-OTM. The second token $\mathcal{T}'$ runs Goliath's program for the send-phase of $\Pi_{\text{Seq-OTM}}$. Instead of running the send-phase interactively with David, Goliath sends $\mathcal{T}'$ to a second instance of $\mathcal{F}_{\text{stateful}}^{\text{wrap}}$ in the creation-phase. David then runs the send-phase with $\mathcal{T}'$. The security-proof for $\Pi_{\text{Seq-OTM}}$ still holds for $\Pi'_{\text{Seq-OTM}}$, due to the following observations. In protocol $\Pi_{\text{Seq-OTM}}$, a corrupted sender Goliath is controlled by the environment $\mathcal{Z}$. In $\Pi'_{\text{Seq-OTM}}$, $\mathcal{Z}$ can only control Goliath during the creation-phase. Afterwards, the tokens $\mathcal{T}$ and $\mathcal{T}'$ cannot communicate with $\mathcal{Z}$ anymore. Thus a corrupted sender in $\Pi'_{\text{Seq-OTM}}$ is strictly less powerful than in $\Pi_{\text{Seq-OTM}}$, and we can conclude that $\Pi'_{\text{Seq-OTM}}$ is UC-secure against a corrupted sender. It is also UC-secure against a corrupted receiver David, as from the view of a corrupted receiver, $\Pi_{\text{Seq-OTM}}$ and $\Pi'_{\text{Seq-OTM}}$ are the same (it makes no difference if David interacts with Goliath or $\mathcal{T}'$).

9 Memory Limited Tokens

The protocol presented in the last Section guarantees perfect security against David. However, to achieve this, the token needs to be able to store $O(n^2m)$ bits of information (for $m = \text{poly}(n)$). For large m, this contradicts the idea of a tamper-proof hardware token being a small and simple device. Moran and Segev [22] noted, that if David is computationally bounded, then the functions stored on the token could be chosen to be pseudorandom [11,16]. The same is true for our construction. It suffices that the token stores a succinct seed of length $O(n)$ for a pseudorandom function F. The token can answer queries z by temporarily computing $(a_t, B_t) = F(t)$ and outputting $V_t = a_t z^T + B_t$.

10 Conclusion

In this paper, we showed that a single (untrusted) tamper-proof hardware token is sufficient for non-interactive, composable computation. We require no additional complexity assumptions. As we only need a single device and no zero-knowledge proofs, our approach is more efficient than previous solutions. Our new primitive, Sequential-One-Time-Memory, is sufficient to realize unconditionally secure One- and k-Time-Programs. However, Sequential-One-Time-Memory is restricted to sequential-access, thus it cannot be used directly to implement several independent One-Time-Programs that can be executed in random order. We consider it an interesting open problem whether it is possible to implement multiple random-access One-Time-Memories with a single untrusted tamper-proof hardware token. However, this seems improbable. Any protocol realizing multiple random-access One-Time-Memories with a single token needs to effectively hide the order in which the One-Time-Memories are queried from the token.

Acknowledgments. We would like to thank the anonymous reviewers of TCC 2011 for their helpful comments.

References

1. Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *STOC*, pages 479–488, 1996.
2. Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *STOC*, pages 113–131, 1988.
3. Julien Brouchier, Tom Kean, Carol Marsh, and David Naccache. Temperature attacks. *IEEE Security & Privacy*, 7(2):79–82, 2009.
4. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.
5. Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *TCC*, pages 61–85, 2007.
6. Ran Canetti and Marc Fischlin. Universally composable commitments. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 19–40, London, UK, 2001. Springer-Verlag.
7. Ivan Damgård, Jesper Buus Nielsen, and Daniel Wichs. Isolated proofs of knowledge and isolated zero knowledge. In *EUROCRYPT*, pages 509–526, 2008.
8. Ivan Damgrd, Jesper Buus Nielsen, and Daniel Wichs. Universally composable multiparty computation with partially isolated parties, 2007.
9. Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.
10. Marc Fischlin, Benny Pinkas, Ahmad-Reza Sadeghi, Thomas Schneider, and Ivan Visconti. Secure set intersection with untrusted hardware tokens. In *11th Cryptographers' Track at the RSA Conference (CT-RSA'11)*, 2011.
11. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.
12. Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.
13. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In *CRYPTO*, pages 39–56, 2008.
14. Vipul Goyal, Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. Interactive locking, zero-knowledge pcps, and unconditional cryptography. In *CRYPTO*, pages 173–190, 2010.
15. Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.
16. 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.
17. Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. Universally composable zero-knowledge arguments and commitments from signature cards. In *In Proc. of the 5th Central European Conference on Cryptology MoraviaCrypt 2005*, 2005.
18. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.

19. Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In *EUROCRYPT*, pages 115–128, 2007.
20. Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.
21. Vladimir Kolesnikov. Truly efficient string oblivious transfer using resettable tamper-proof tokens. In *TCC*, pages 327–342, 2010.
22. Tal Moran and Gil Segev. David and goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In *EUROCRYPT*, pages 527–544, 2008.
23. Michael O. Rabin. How to exchange secrets by oblivious transfer. technical report tr-81. Technical report, Aiken Computation Laboratory, Harvard University, 1981.
24. Stefan Wolf and Jürg Wullschleger. Oblivious transfer is symmetric. In *EUROCRYPT*, pages 222–232, 2006.
25. Jürg Wullschleger. Oblivious-transfer amplification. *CoRR*, abs/cs/0608076, 2006.