(Efficient) Universally Composable Oblivious Transfer Using a Minimal Number of Stateless Tokens

Seung Geol Choi[1,⋆], Jonathan Katz[2,⋆⋆], Dominique Schröder[3,⋆⋆⋆], Arkady Yerukhimovich[4,†], and Hong-Sheng Zhou[5,‡]

[1] United States Naval Academy, choi@usna.edu
[2] University of Maryland, jkatz@cs.umd.edu
[3] Saarland University, ds@ca.cs.uni-saarland.de
[4] MIT Lincoln Laboratory, arkady@cs.umd.edu
[5] Virginia Commonwealth University, hszhou@vcu.edu

Abstract. We continue the line of work initiated by Katz (Eurocrypt 2007) on using *tamper-proof hardware* for universally composable secure computation. As our main result, we show an efficient oblivious-transfer (OT) protocol in which two parties each create and exchange a single, stateless token and can then run an unbounded number of OTs. Our result yields what we believe is the most *practical* and *efficient* known approach for oblivious transfer based on tamper-proof tokens, and implies that the parties can perform (repeated) secure computation of arbitrary functions without exchanging additional tokens.

Motivated by this result, we investigate the minimal number of stateless tokens needed for universally composable OT/ secure computation. We prove that our protocol is *optimal* in this regard for constructions making *black-box* use of the tokens (in a sense we define). We also show that nonblack-box techniques can be used to obtain a construction using only a single stateless token.

1 Introduction

The universal composability (UC) framework [6] provides a way of analyzing protocols while ensuring strong security guarantees. In particular, protocols proven secure in this framework remain secure when run concurrently with

⋆ Work done in part at the University of Maryland and Columbia University.
⋆⋆ Work supported in part by NSF awards #1111599 and #1223623.
⋆⋆⋆ Work done in part at the University of Maryland, and supported by the German Ministry for Education and Research (BMBF) through funding for the Center for IT-Security, Privacy, and Accountability (CISPA www.cispa-security.org) and also by an Intel Early Career Award.
† Work done in part at the University of Maryland.
‡ Work done at the University of Maryland, and supported by an NSF CI postdoctoral fellowship.

arbitrary other protocols in a larger networked environment. Unfortunately, most interesting cryptographic tasks are impossible to realize in the "plain" UC framework when an honest majority cannot be assumed and, in particular, in the setting of two-party secure computation [8, 9, 36]. This stark negative result has motivated researchers to explore various extensions/variants of the UC framework in which secure computation can be achieved [7], with notable examples being the assumption of a common reference string (CRS) [6, 8, 10] or a public-key infrastructure [6, 3]. In the real world, implementing either of these approaches seems to require the existence of some *trusted entity* that parties agree to use (though see [11] for some ideas on using a naturally occurring high-entropy source in place of a CRS).

Katz [32] suggested using *tamper-proof hardware tokens* for UC computation. That is, Katz proposed a model where parties can construct hardware tokens to compute functions of their choice such that an adversary given a token $\mathcal{T}_F$ for a function F can do no more than observe the input/output characteristics of this token. The motivation for this being that the existence of tamper-proof hardware can be viewed, in principle, as a *physical* assumption rather than an assumption of trust in some external entity. (In fact, in Katz's model the parties may create the tamper-proof tokens themselves—rather than obtain them from a trusted provider [28]—and a malicious party can put any algorithm on a token it creates.) In addition, secure hardware may also potentially result in more efficient protocols; indeed, it has been suggested for improving efficiency in other settings (e.g., [17, 13, 14, 5, 27, 31, 34, 22]). In addition to introducing the model, Katz showed that tamper-proof hardware tokens can be used for universally composable computation of arbitrary functions. His work motivated an extensive amount of follow-up work [12, 37, 23, 15, 24, 25, 19] that we discuss in detail later.

As our main result, we show here a new protocol for universally composable 1-out-of-2 string oblivious transfer (OT) based on tamper-proof hardware tokens, secure against a static, malicious adversary. Our work yields what we believe to be the most *practical* and *efficient* known protocol since it simultaneously achieves all the following (which are not achieved all at once by any other solution; see Table 1 for a detailed comparison):

- Our protocol is based on *stateless* tokens, which seem easier/cheaper to create in practice and are (automatically) resistant to resetting attacks.
- Our protocol requires the parties to exchange a *single* pair of tokens. This can be done in advance, before the parties' inputs are known. Furthermore, the tokens can be used to implement an *unbounded* number of oblivious transfers, rather than requiring the parties to exchange a fresh pair of tokens for every oblivious transfer they want to compute. Thus, by relying on known completeness results [33, 30], the parties can use the same tokens to perform an unlimited number of secure computations (of possibly different functions, and on different inputs).
- Our protocol is efficient. It is *black-box*, and each OT needs mostly standard symmetric-key operations along with only a few (unique) digital signatures.

Moreover, any desired number of OTs can be obtained (in parallel) in *constant* rounds.
- If the total number of OTs is a priori bounded, then a variant of our protocol can realize any bounded number of OTs in constant rounds based only on the existence of collision-resistant hash functions.

Inspired by our result, we investigate the *minimal* number of stateless tokens needed for universally composable OT/secure computation. We show that two tokens—one created by each party—are needed even to obtain a single universally composable OT as long as only "black-box techniques" are used. (We explain what we mean by "black-box techniques" in the relevant section of our paper.) Our protocol, above, is thus optimal in this regard. Our impossibility result is somewhat surprising, since a single *stateful* token suffices for OT [19]. Our results thus demonstrate an inherent difference between stateful and stateless tokens.

Since protocols based on nonblack-box techniques tend to be impractical, our work pins down the minimal number of stateless tokens needed as far as practical protocols are concerned. From a theoretical point of view, however, it is still interesting to completely resolve the question. In this vein, we show a protocol for carrying out an unbounded number of secure computations using only a *single* (stateless) token. Our construction uses a variant of the nonblack-box simulation technique introduced by Barak [2].

In summary, our work shows that efficient, universally composable oblivious transfer can be realized from two stateless tokens without any additional setup assumptions, and is unlikely using a single stateless token. On the other hand, using (inefficient) nonblack-box techniques, a single stateless token serves as a sufficient setup for general universally-composable two-party computation.

1.1 Related Work

Katz's original protocol for secure computation using tamper-proof tokens [32] required stateful tokens and relied on number-theoretic assumptions (specifically, the DDH assumption). Subsequent work has mainly focused on improving one or both of these aspects of his work.

Several researchers have explored constructions using *stateless* tokens. Stateless tokens are presumably easier and/or cheaper to build, and are resistant to resetting attacks whereby an adversary cuts off the power supply and thus effectively "rewinds" the token. Chandran et al. [12] were the first to eliminate the requirement of stateful tokens. They construct UC commitments based on the existence of one-way functions, and oblivious transfer based on any enhanced trapdoor permutation (eTDP). They also introduce a variant security model in which an adversary need not know the code of the tokens he produces, thus capturing scenarios where an adversary may pass along tokens whose code he

doesn't know, e.g., via token replication. (We do not consider this model here.[6]) From a practical perspective, however, their work has several drawbacks. Their OT protocol makes nonblack-box use of the underlying primitives, runs in $\Theta(\lambda)$ rounds (where λ is the security parameter), and uses the heavy machinery of concurrent non-malleable zero-knowledge. Improving upon their work, Goyal et al. [25] show a black-box construction of oblivious transfer; their protocol runs in constant rounds assuming a collision-resistant hash function, or $\Theta(\lambda/\log\lambda)$ rounds based on one-way functions. However, their protocol requires the parties to exchange $\Theta(\lambda)$ tokens for *every* oblivious transfer the parties wish to execute. Compared with these results, our protocol is much more efficient at the expense of the stronger assumption of existence of unique signature schemes.

A second direction has explored the possibility of eliminating computational assumptions altogether. This line of work was initiated by Moran and Segev [37], who showed how to realize statistically secure UC commitments using a single stateful token. (Note that commitment does not imply OT, or general secure computation, in the unconditional setting.) Their construction can be used for any bounded number of commitments, still using only one token, and the authors note that they can achieve an unbounded number of commitments (with computational security) based on one-way functions. Goyal et al. [25] show an unconditional construction of oblivious transfer (and hence general secure computation) using $\Theta(\lambda)$ stateful tokens. Recently, Döttling et al. [19] show how to construct unconditionally secure OT using only a single stateful token. Goyal et al. [24] showed that unconditional security from stateless tokens is impossible (unless the token model is extended to allow tokens to encapsulate each other). If such encapsulation is allowed, then they show how to realize statistically secure OT in constant rounds using $\Theta(\lambda)$ stateless tokens.

Kolesnikov [34] showed an efficient construction of oblivious transfer from stateless tokens. However, his work is not in the UC setting, and he achieves only covert security [1] rather than security against malicious parties. Dubovitskaya et al. [21] constructed an OT protocol from two stateful tokens. Their work assumes tokens are not reused (it is not clear how this is enforced), and is also not in the UC setting.

Our work in relation to prior work. For our main result, we carefully combine the techniques of [25] and [19] to achieve the most *practical* and *efficient* known protocol for universally composable OT based on tamper-proof hardware tokens. Our protocol uses two *stateless* tokens (one per party), and can be used for either a bounded number of OTs (assuming the existence of collision-resistant hash functions) or for an unbounded number of OTs (additionally assuming the existence of unique signatures or, equivalently, verifiable random

[6] Although we do not formally consider this model, it appears that our efficient, two-token protocol would remain secure in that model since the simulator in our security proof does not refer to the code of the tokens.

	stateless tokens					stateful tokens		
	Here 1	Here 2	[12]	[25]	[24]	[32]	[25]	[19]
Tokens:	2	2	2	$\Theta(\lambda)$	$\Theta(\lambda)$	2	$\Theta(\lambda)$	1
Rounds:	$\Theta(1)$	$\Theta(1)$	$\Theta(\lambda)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Asmpt.:	CRHF, VRF	CRHF	eTDP	CRHF	TE	DDH	none	none
# OTs:	unbounded	bounded	unbounded	1	1	unbounded	1	bounded

Table 1. Universally composable OT based on tamper-proof hardware tokens. The security parameter is denoted by λ and TE means token encapsulation.

functions (VRFs)). Both instantiations run in constant rounds.[7] A detailed comparison of this protocol to relevant prior work is given in Table 1.

In addition to the above, we show two other results: there is no "black-box" construction of universally composable OT using fewer than two stateless tokens, but universally composable coin tossing (and hence OT) *can* be based on a single stateless token using nonblack-box techniques.

Concurrent work. Independently, Döttling et al. [20] show a different nonblack-box construction of UC coin-tossing from a single stateless token, and argue (without proof) that nonblack-box techniques are needed. Here, we provide a rigorous version of their argument. Our efficient, black-box OT protocol using two stateless tokens—which we consider our primary contribution—has no analog in their work.

2 Preliminaries

Let λ be the security parameter. For a set S, we let $x \leftarrow S$ denote choosing x uniformly at random from S. We assume readers are familiar with pseudorandom functions, collision-resistant hash functions, strong extractors, commitment schemes, digital signature schemes, message-authentication codes (MACs), and witness-indistinguishable arguments of knowledge (WI-AoKs). Due to space restrictions, we omit the formal definitions; here, we mainly introduce notation.

Throughout the paper, a pseudorandom function is denoted by PRF, and it is assumed that the output length is sufficiently long so that it can be truncated to the appropriate length. We let $\mathsf{MAC} = (\mathsf{Sig}, \mathsf{Vrfy})$ be a deterministic message-authentication code, where Vrfy is a canonical verification procedure that checks the validity of a tag τ by recomputing it. We slightly abuse the notation to let $(\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vrfy})$ also denote a digital signature scheme (the context should make it obvious whether the notation indicates a MAC or a signature). A digital signature scheme is called *unique* if for every possible verification key vk and every message m, there is a unique signature σ such that $\mathsf{Vrfy}_{vk}(m, \sigma) = 1$. Dodis

[7] The CRHF assumption is only used to make our protocols constant round by instantiating constant round statistically-hiding commitments. Thus, we can instead instantiate our protocols in $\Theta(\lambda / \log \lambda)$ rounds using only one-way functions.

and Yampolskiy [18] give an efficient construction for unique signatures based on a certain number-theoretic assumption. Let $\mathsf{Ext} : \{0,1\}^{2\lambda} \times \{0,1\}^d \to \{0,1\}^\lambda$ denote a strong randomness extractor where the source has length 2λ and the seed has length d. If the min-entropy of the source is at least $2\lambda - O(\log \lambda)$, the output is statistically close to uniform.

We use SCom to denote a (possibly interactive) statistically-hiding and computationally-binding commitment scheme [16, 26] and Com to denote a (possibly interactive) computationally-hiding and strongly-binding commitment scheme. We let $\mathsf{com}_m \leftarrow \mathsf{Com}(m; r_m, \breve{r}_m)$ denote a commitment to a message m, where the sender (resp., receiver) uses uniform random coins r_m (resp., $\breve{r}_m$) and the final transcript is com_m; sometimes we omit $\breve{r}_m$ when it is clear from the context. (We also use similar notation for SCom.) In a strongly binding commitment scheme [29], with overwhelming probability over the receiver's coins $\breve{r}_m$, for any commitment com there is at most one (m, r_m) such that $\mathsf{com} = \mathsf{Com}(m; r_m, \breve{r}_m)$. Although this definition is stronger than usual, the Naor scheme [38] satisfies it. Without loss of generality, we assume a canonical decommit phase in which the sender sends m together with the randomness r_m (i.e., $\mathsf{decom}_m = (m, r_m)$). Then the receiver runs the algorithm $\mathsf{Open}(\mathsf{com}_m, m, r_m)$ which checks if (m, r_m) is consistent with the transcript com_m. If so, Open outputs m; otherwise it outputs $\perp$.

Linear algebra. By $\mathbb{F}_2$ we denote the finite field with two elements. If $a \in \mathbb{F}_2^\lambda$ and $b \in \mathbb{F}_2^k$ are two column-vectors, then $(ab^{\mathrm{T}}) = (a_i b_j)_{ij} \in \mathbb{F}_2^{\lambda \times k}$ is the outer product (or tensor product) of a and b, and $a^{\mathrm{T}} b = \sum_{i=1}^{\lambda} a_i b_i \in \mathbb{F}_2$ the inner product.

Let $C \in \mathbb{F}_2^{\lambda \times 2\lambda}$. Then $\dim(\ker(C)) \geq \lambda$. Let $B = \{b_1, \ldots, b_\lambda\} \subseteq \ker(C)$ be a linearly independent set. One can choose a set $B^* = \{b_{\lambda+1}, \ldots, b_{2\lambda}\}$ such that $B \cup B^*$ is a basis of $\mathbb{F}_2^{2\lambda}$. Let $e_i \in \mathbb{F}_2^\lambda$ be the ith unit-vector. Then, there exists a matrix $G \in \mathbb{F}_2^{\lambda \times 2\lambda}$ such that $Gb_i = e_i$ for $i = 1, \ldots, \lambda$ and $Gb_i = 0$ for $i = \lambda+1, \ldots, 2\lambda$. This matrix is called the *complementary matrix* of C and we denote by $G \leftarrow \mathsf{Comp}(C)$ its computation. It holds that $\mathrm{rank}(G) = \lambda$ and $B^* \subseteq \ker(G)$. For such C and G, we can always solve the linear system $Cx = r$, $Gx = s$ by solving $Cx_{B^*} = r$ and $Gx_B = s$ independently with $x_B \in \mathrm{span}(B) \subseteq \ker(C)$ and $x_{B^*} \in \mathrm{span}(B^*) \subseteq \ker(G)$, and then setting $x := x_B + x_{B^*}$.

Token functionality. We model tamper-proof hardware tokens as an ideal functionality in the UC framework, following Katz [32]; see Figure 1. Our ideal functionality models stateful tokens: although all our protocols use stateless tokens, an adversarially generated token may be stateful.

Oblivious-transfer functionality. The OT functionality is standard, but we wish here to model a multi-session variant where the sender and receiver repeatedly (in different sub-sessions) execute any agreed-upon number m of parallel OTs (in a given sub-session). We refer to this functionality as $\mathcal{F}_{\mathsf{multi\text{-}OT}}$, and describe it in Figure 2. We note that the sub-sessions are executed sequentially. Additionally, as highlighted in [25], the sender is notified each time the receiver obtains output. We define the bounded OT functionality similarly

Functionality $\mathcal{F}_{\mathsf{wrap}}$

The functionality is parameterized by a polynomial $p(\cdot)$ and an implicit security parameter λ.

Create: Upon receiving an input $(\text{CREATE}, \langle \mathsf{sid}, \mathsf{C}, \mathsf{U} \rangle, \mathcal{M})$ from a party C (i.e., the token creator), where U is another party (i.e., the token user) in the system and $\mathcal{M}$ is an interactive Turing machine, do:

> If there is no tuple of the form $\langle \mathsf{C}, \mathsf{U}, \star, \star, \star \rangle$ stored, store $\langle \mathsf{C}, \mathsf{U}, \mathcal{M}, 0, \emptyset \rangle$. Reveal $(\text{CREATE}, \langle \mathsf{sid}, \mathsf{C}, \mathsf{U} \rangle)$ to the adversary.

Ready: Upon receiving a message $(\text{READY}, \langle \mathsf{sid}, \mathsf{C}, \mathsf{U} \rangle)$ from the adversary, send $(\text{READY}, \langle \mathsf{sid}, \mathsf{C}, \mathsf{U} \rangle)$ to U.

Execute: Upon receiving an input $(\text{RUN}, \langle \mathsf{sid}, \mathsf{C}, \mathsf{U} \rangle, \mathsf{msg})$ from the user U, find the unique stored tuple $\langle \mathsf{C}, \mathsf{U}, \mathcal{M}, i, \mathsf{state} \rangle$. If no such tuple exists, do nothing. Otherwise, do:

> If the Turing machine $\mathcal{M}$ has never been used yet, i.e., $i = 0$, then choose ω uniformly at random from $\{0,1\}^{p(\lambda)}$ and set $\mathsf{state} := \omega$ before running the Turing machine. Run $(\mathsf{out}, \mathsf{state}') \leftarrow \mathcal{M}(\mathsf{msg}; \mathsf{state})$ for at most $p(\lambda)$ steps where out is the response and state' is the new state of $\mathcal{M}$ (set $\mathsf{out} := \perp$ and $\mathsf{state}' := \mathsf{state}$ if $\mathcal{M}$ does not respond in the allotted time). Send $(\text{RESPONSE}, \langle \mathsf{sid}, \mathsf{C}, \mathsf{U} \rangle, \mathsf{out})$ to U. Erase $\langle \mathsf{C}, \mathsf{U}, \mathcal{M}, i, \mathsf{state} \rangle$ and store $\langle \mathsf{C}, \mathsf{U}, \mathcal{M}, i+1, \mathsf{state}' \rangle$.

Fig. 1. The ideal $\mathcal{F}_{\mathsf{wrap}}$ functionality for stateful tokens.

except that the sender and receiver only execute a single sub-session of m parallel OTs. This allows the sender and receiver to execute any bounded number of OTs.

3 Efficient Oblivious Transfer Using Two Stateless Tokens

In this section, we first give the details of our unbounded OT protocol. Then, in Section 3.3 we briefly sketch how this protocol can be modified to achieve a bounded OT protocol using only CRHFs. We provide some intuition and background before giving the details of our protocol. Our starting point is the unconditionally secure OT protocol from [19], which uses a single *stateful* token. We sketch a simplified version of their protocol for the case of a single OT carried out between the sender S with input $(x_0, x_1) \in \{0,1\}^\lambda \times \{0,1\}^\lambda$ and the receiver R with input $b \in \{0,1\}$. We canonically identify the vector space $\mathbb{F}_2^\lambda$ with the set $\{0,1\}^\lambda$ of strings of length λ. The main steps of the protocol are as follows:

1. S creates a token $\mathcal{T}_{\mathsf{S}}$ holding random vector $a \in \mathbb{F}_2^{2\lambda}$ and matrix $B \in \mathbb{F}_2^{2\lambda \times 2\lambda}$ and gives it to R.

Functionality $\mathcal{F}_{\mathsf{multi\text{-}OT}}$

$\mathcal{F}_{\mathsf{multi\text{-}OT}}$ interacts with sender S, receiver R, and the adversary. The functionality is parameterized by a security parameter λ. It also maintains a variable $\mathsf{curr\text{-}id}$ initialized to $\perp$.

Upon receiving $(\text{SEND}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, \mathsf{ssid}, \langle m, \{(x_i^0, x_i^1)\}_{i=1}^m \rangle)$ from S with $x_i^0, x_i^1 \in \{0,1\}^\lambda$, if $\mathsf{curr\text{-}id} \notin \{\perp, \mathsf{ssid}\}$ then ignore it. Otherwise, set $\mathsf{curr\text{-}id} := \mathsf{ssid}$ and record $\langle \mathsf{ssid}, m, \{(x_i^0, x_i^1)\}_{i=1}^m \rangle$, and reveal $(\text{SEND}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, \mathsf{ssid})$ to the adversary. Ignore further $(\text{SEND}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, \mathsf{ssid}, \ldots)$ inputs with this ssid.

Upon receiving $(\text{RECEIVE}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, \mathsf{ssid}, \langle m, \{b_i\}_{i=1}^m \rangle)$ from R with $b_i \in \{0,1\}$, if $\mathsf{curr\text{-}id} \notin \{\perp, \mathsf{ssid}\}$ then ignore it. Otherwise, set $\mathsf{curr\text{-}id} := \mathsf{ssid}$ and record the tuple $\langle \mathsf{ssid}, m, \{b_i\}_{i=1}^m \rangle$, and reveal $(\text{RECEIVE}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, \mathsf{ssid})$ to the adversary. Ignore further $(\text{RECEIVE}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, \mathsf{ssid}, \ldots)$ inputs with this ssid.

Upon receiving $(\text{GO}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, \mathsf{ssid})$ from the adversary, ignore it if $\mathsf{curr\text{-}id} \neq \mathsf{ssid}$, or either $\langle \mathsf{ssid}, m, \{(x_i^0, x_i^1)\}_{i=1}^m \rangle$ or $\langle \mathsf{ssid}, m, \{b_i\}_{i=1}^m \rangle$ is not recorded. Otherwise, do the following: set $\mathsf{curr\text{-}id} := \perp$, return $(\text{RECEIVED}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, \mathsf{ssid})$ to S, and return $(\text{RECEIVED}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, \mathsf{ssid}, \{x_i^{b_i}\}_{i=1}^m)$ to R. Ignore further $(\text{GO}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, \mathsf{ssid})$ messages with this ssid from the adversary.

Fig. 2. The $\mathcal{F}_{\mathsf{multi\text{-}OT}}$ functionality.

2. R chooses a random matrix $C \in \mathbb{F}_2^{\lambda \times 2\lambda}$ and sends it to S. In turn, S computes $\tilde{a} := Ca$, $\tilde{B} := CB$, and a complementary matrix $G \in \mathbb{F}_2^{\lambda \times 2\lambda}$ to C and sends these to R.
3. R sends random $h \in \mathbb{F}_2^{2\lambda}$ to S. Then, S sends $\tilde{x}_0 := x_0 + GBh$ and $\tilde{x}_1 := x_1 + GBh + Ga$ to R.
4. R queries the token with a random $z \in \mathbb{F}_2^{2\lambda}$ such that $z^{\mathrm{T}} h = b$. The token will in turn output $V := az^{\mathrm{T}} + B$. Then R checks that $CV = \tilde{a}z^{\mathrm{T}} + \tilde{B}$ and, if this is the case, outputs $x_b := \tilde{x}_b - GVh$. (Otherwise it detects that S was cheating.)

The basic idea of their protocol is as follows. The receiver R performs a secret sharing of its input b into shares z and h; by using only h with the sender S and only z with the token, R maintains the privacy of b. In order to obtain the output x_b, the receiver R has to compute the mask (i.e., GBh or $GBh + Ga$). This is achieved by querying the token with z, since $GVh = G(az^{\mathrm{T}} + B)h = b(Ga) + GBh$. To ensure that the token outputs the correct value V, the receiver checks that $CV = \tilde{a}z^{\mathrm{T}} + \tilde{B}$. Since the token does not know C, incorrect behavior is detected with overwhelming probability. To achieve security against a malicious receiver, it is crucial that the receiver is only able to query the token once, and this is enforced by making the token *stateful*. In particular, the token "self destructs" after the first query by the receiver.

Using stateless tokens. We carefully combine the techniques of [19] with ideas from [25] so that we can use two *stateless* tokens instead of one stateful token. This entails several difficulties:

Multiple queries. A stateless token can be executed multiple times while the token remains oblivious about it (in contrast, stateful tokens "self destruct" after their use [23]), so we need to ensure that a malicious party queries the token only once. Motivated by similar techniques in [25], we handle this issue by modifying the token so that it only replies to *authenticated* inputs. That is, instead of querying z directly to the token, R queries $(\mathsf{com}_z, z, r_z, \sigma_z)$ where com_z is a commitment to z, the value σ_z is a digital signature on com_z (received from S) with respect to a verification key vk_{S}, and z, r_z is the decommitment of com_z (see the discussion below about why this technique is useful).

Extracting the inputs. Using a stateless token additionally introduces the difficulty of extracting the sender's inputs during the simulation. Extraction from a stateful token is possible by having the simulator rewind the token and query it multiple times. (The fact that the simulator can query the token multiple times is an advantage of the simulator over the real-world parties.) Once we move to a stateless token in the real world, and authenticate the queries as described above, even the simulator is no longer able to rewind and send multiple (authenticated) queries to the token.

To resolve this issue we introduce a second stateless token $\mathcal{T}_{\mathsf{R}}$ sent from the receiver to the sender which allows directly extracting the sender's inputs.

At a high level, the above results in the following changes to the protocol:

1. The receiver sends a token $\mathcal{T}_{\mathsf{R}}$ to the sender. (The behavior of this token will become clear below.)

2(a). S generates a statistically hiding[8] commitment $\mathsf{com} \leftarrow \mathsf{SCom}(a \| B)$ and sends com to R. Then, R chooses a matrix C and authenticates com by computing $\sigma := \mathsf{Sig}_{sk_{\mathsf{R}}}(\mathsf{com} \| C)$ (where sk_{R} is a secret key also embedded in $\mathcal{T}_{\mathsf{R}}$), and sends C and σ to S.

2(b). S queries $\mathcal{T}_{\mathsf{R}}$ on input $(C, \mathsf{com}, \mathsf{decom}, \sigma)$. The token checks that the signature σ is valid and that decom is a valid opening of com; otherwise it aborts. $\mathcal{T}_{\mathsf{R}}$ returns $\tilde{a} := Ca$ and $\tilde{B} := CB$ together with a signature $\sigma' := \mathsf{Sig}_{sk_{\mathsf{R}}}(\tilde{a} \| \tilde{B})$. Then S sends $(\tilde{a}, \tilde{B}, \sigma')$ to R.

Intuitively, the value (a, B) remains hidden from a corrupted R as before. For a corrupted sender, the simulator will emulate $\mathcal{F}_{\mathsf{wrap}}$ and observe the inputs to $\mathcal{T}_{\mathsf{R}}$. To guarantee that the simulator extracts the sender's inputs correctly, it is necessary that the sender queries the token exactly once with a value (a, B). The binding property of the commitment scheme, together with the unforgeability of the signature scheme, guarantees that S can make at most one valid query to the token. The unforgeability of the signature scheme assures that S must query the token at least once to generate a valid next message of the protocol. A similar argument holds for the inputs $(\mathsf{com}_z, z, r_z, \sigma_z)$ to the sender's token $\mathcal{T}_{\mathsf{S}}$.

[8] We were not able to prove security using a computationally hiding commitment scheme. Indeed, it is an interesting open problem to achieve a constant-round OT protocol with two stateless tokens based only on one-way functions.

On input (key):
ouput vk_{S}.

On input $(\mathsf{ssid}, i, \mathsf{com}_z, z, r_z, \sigma_z)$:
$v := \mathsf{Vrfy}_{vk_{\mathsf{S}}}(\mathsf{ssid}\|i\|\mathsf{com}_z, \sigma_z)$
if $z = \mathsf{Open}(\mathsf{com}_z, z, r_z)$ and $v = 1$
$a := \mathsf{PRF}_{k_a}(\mathsf{ssid}\|i)$
$B := \mathsf{PRF}_{k_B}(\mathsf{ssid}\|i)$
$V := az^{\mathrm{T}} + B$
$w := \mathsf{Sig}_{sk_{\mathsf{S}}}(\mathsf{ssid}\|i)$
output (V, w)
else output $(\perp, \perp)$

Fig. 3. The Turing machine $\mathcal{M}_{\mathsf{S}}$ to be embedded in the sender-created token $\mathcal{T}_{\mathsf{S}}$. It is initialized with $(sk_{\mathsf{S}}, vk_{\mathsf{S}}, k_a, k_B)$ given by the creator.

On input (key):
ouput vk_{R}.

On input $(\mathsf{ssid}, i, \mathsf{com}_{a\|B}, a, B, r_{a\|B}, \sigma_{a\|B})$:
$v := \mathsf{Vrfy}_{vk_{\mathsf{R}}}(\mathsf{ssid}\|i\|0\|\mathsf{com}_{a\|B}, \sigma_{a\|B})$
if $a\|B = \mathsf{Open}(\mathsf{com}_{a\|B}, a\|B, r_{a\|B})$
and $v = 1$
$\tilde{a} := Ca$; $\tilde{B} := CB$;
$\sigma_{\tilde{a}\|\tilde{B}} := \mathsf{Sig}_{sk_{\mathsf{R}}}(\mathsf{ssid}\|i\|1\|\tilde{a}\|\tilde{B})$
output $(\tilde{a}, \tilde{B}, \sigma_{\tilde{a}\|\tilde{B}})$
else output $(\perp, \perp, \perp)$

Fig. 4. The Turing machine $\mathcal{M}_{\mathsf{R}}$ to be embedded in the receiver-created token $\mathcal{T}_{\mathsf{R}}$. It is initialized with $(sk_{\mathsf{R}}, vk_{\mathsf{R}}, C)$ given by the creator.

Malicious tokens. There is one additional issue to be taken care of. The above protocol is secure assuming the tokens are generated as specified by the protocol. However, a malicious token may try to leak some information about the other party's queries to the token creator. For example, a malicious token $\mathcal{T}_{\mathsf{R}^*}$ may output $(\tilde{a}, \tilde{B}, \sigma')$ where the bits of σ' leak information about (a, B). To protect against this, we require the underlying signature scheme to be *unique*; then, σ' carries no more information about (a, B) than $(\tilde{a}, \tilde{B})$ does. This ensures that the only information leakage that can occur is from a *token abort*. For $\mathcal{T}_{\mathsf{S}}$, this type of leakage is already handled by the protocol of [19]. For $\mathcal{T}_{\mathsf{R}}$, we handle the leakage using strong extractors; see the formal description of the protocol below.

The above suffices for a single OT from each pair of tokens. To achieve an unbounded number of OTs we replace all the secret keys and secret inputs with pseudorandom values output by a pseudorandom function.

3.1 The Protocol

The protocol π between a sender S and a receiver R consists of an initial token-exchange phase after which the parties can carry out an unlimited number of oblivious transfers. We describe π now; see also Figure 8.

Token-exchange phase. Each party generates a single token and sends it to the other party. The sender's token $\mathcal{T}_{\mathsf{S}}$ encapsulates the code described in Figure 3, where $k_a, k_B, \leftarrow \{0,1\}^\lambda$ and $(sk_{\mathsf{S}}, vk_{\mathsf{S}}) \leftarrow \mathsf{Kg}()$. The receiver's token $\mathcal{T}_{\mathsf{R}}$ encapsulates the code described in Figure 4, where $(sk_{\mathsf{R}}, vk_{\mathsf{R}}) \leftarrow \mathsf{Kg}()$ and $C \leftarrow \mathbb{F}_2^{2\lambda \times 4\lambda}$. Formally, each party sends the relevant code to $\mathcal{F}_{\mathsf{wrap}}$ using an appropriate (CREATE, ...) message. Then, S runs $\mathcal{T}_{\mathsf{R}}(key)$ to obtain vk_{R}; likewise, R obtains vk_{S} by running $\mathcal{T}_{\mathsf{S}}(key)$. Finally, R sends C to S, and in turn S computes the complementary matrix $G \leftarrow \mathsf{Comp}(C)$ and sends it to R.

Oblivious transfer phase. Following the token-exchange phase, the parties can sequentially run an unbounded number of sub-sessions where in each sub-

session they carry out any desired number m of oblivious transfers. (Each sub-session uses only a constant number of rounds.) In each sub-session, S gets $(\text{SEND}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, \mathsf{ssid}, \langle m, \{(x_i^0, x_i^1)\}_{i=1}^m \rangle)$ and R gets $(\text{RECEIVE}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, \mathsf{ssid}, \langle m, \{b_i\}_{i=1}^m \rangle)$ from the environment, and they execute the following protocol:

- $\mathsf{S} \to \mathsf{R}$**:** For $i \in [m]$, the sender computes $a_i := \mathsf{PRF}_{k_A}(\mathsf{ssid} \| i)$ and $B_i := \mathsf{PRF}_{k_B}(\mathsf{ssid} \| i)$. Here, we have $a_i \in \mathbb{F}_2^{4\lambda}$, $B_i \in \mathbb{F}_2^{4\lambda \times 4\lambda}$. Then, the sender commits to (a_i, B_i) by executing $\mathsf{com}_{a_i \| B_i} \leftarrow \mathsf{SCom}(a_i \| B_i; r_{a_i \| B_i})$, and sends $\{\mathsf{com}_{a_i \| B_i}\}_{i=1}^m$ to R.
- $\mathsf{R} \to \mathsf{S}$**:** R chooses $h_i \leftarrow \mathbb{F}_2^{4\lambda}$ and $z_i \leftarrow \mathbb{F}_2^{4\lambda}$ subject to the constraint that $b_i = z_i^{\mathrm{T}} h_i$, and commits to z_i by executing $\mathsf{com}_{z_i} \leftarrow \mathsf{SCom}(z_i; r_{z_i})$. It next authenticates the commitment $\mathsf{com}_{a_i \| B_i}$ by computing $\sigma_{a_i \| B_i} := \mathsf{Sig}_{sk_\mathsf{R}}(\mathsf{ssid} \| i \| 0 \| \mathsf{com}_{a_i \| B_i})$ for $i \in [m]$. Finally, it sends $\{(\sigma_{a_i \| B_i}, \mathsf{com}_{z_i})\}_{i=1}^m$ to S.
- $\mathsf{S} \to \mathsf{R}$**:** The sender checks if $\mathsf{Vrfy}_{vk_\mathsf{R}}(\mathsf{ssid} \| i \| 0 \| \mathsf{com}_{a_i \| B_i}, \sigma_{a_i \| B_i}) = 1$ for $i \in [m]$; if the check fails, then S aborts the protocol. For $i \in [m]$, the sender runs the token $\mathcal{T}_\mathsf{R}$ with $(\mathsf{ssid},\ i,\ \mathsf{com}_{a_i \| B_i},\ a_i,\ B_i,\ r_{a_i \| B_i},\ \sigma_{a_i \| B_i})$ and obtains in return $(\tilde{a}_i, \tilde{B}_i, \sigma_{\tilde{a}_i \| \tilde{B}_i})$. Then S checks that $\tilde{a}_i = Ca_i$, $\tilde{B}_i = CB_i$, $\mathsf{Vrfy}_{vk_\mathsf{R}}(\mathsf{ssid} \| i \| 1 \| \tilde{a}_i \| \tilde{B}_i, \sigma_{\tilde{a}_i \| \tilde{B}_i}) = 1$ for $i \in [m]$; if the check fails, S aborts the protocol. Otherwise, for $i \in [m]$ it authenticates the commitment com_{z_i} by computing $\sigma_{z_i} := \mathsf{Sig}_{sk_\mathsf{S}}(\mathsf{ssid} \| i \| \mathsf{com}_{z_i})$. Finally, it sends $\{(\tilde{a}, \tilde{B}, \sigma_{\tilde{a}_i \| \tilde{B}_i}, \sigma_{z_i})\}_{i=1}^m$ to R.
- $\mathsf{R} \to \mathsf{S}$**:** The receiver checks that $\mathsf{Vrfy}_{vk_\mathsf{R}}(\mathsf{ssid} \| i \| 1 \| \tilde{a}_i \| \tilde{B}_i, \sigma_{\tilde{a}_i \| \tilde{B}_i}) = 1$, and $\mathsf{Vrfy}_{vk_\mathsf{S}}(\mathsf{ssid} \| i \| \mathsf{com}_{z_i}, \sigma_{z_i}) = 1$ for $i \in [m]$. If this check fails, it aborts the protocol. Otherwise, R runs the token $\mathcal{T}_\mathsf{S}$ with $(\mathsf{ssid}, i, \mathsf{com}_{z_i}, z_i, r_{z_i}, \sigma_{z_i})$ and obtains in return (V_i, w_i), for $i \in [m]$. The receiver checks that $\mathsf{Vrfy}_{vk_\mathsf{S}}(\mathsf{ssid} \| i, w_i) = 1$ and $CV_i = \tilde{a}_i z_i^{\mathrm{T}} + \tilde{B}_i$ for $i \in [m]$. If the check fails, then it aborts the protocol. Otherwise, it sends $\{(h_i, w_i)\}_{i=1}^m$ to S.
- $\mathsf{S} \to \mathsf{R}$**:** The sender checks that $\mathsf{Vrfy}_{vk_\mathsf{S}}(\mathsf{ssid} \| i, w_i) = 1$ for $i \in [m]$; if not, it aborts the protocol. Otherwise, for $i \in [m]$ it chooses $v_i^0, v_i^1 \leftarrow \{0,1\}^d$ and computes $\tilde{x}_i^0 := \mathsf{Ext}(GB_i h_i, v_i^0) \oplus x_i^0$ and $\tilde{x}_i^1 := \mathsf{Ext}(GB_i h_i + Ga_i, v_i^1) \oplus x_i^1$. Here d is an appropriate seed length for the extractor. Finally it sends $\{(v_i^0, v_i^1, \tilde{x}_i^0, \tilde{x}_i^1)\}_{i=1}^m$ to the receiver.
- $\mathsf{R} \to \mathsf{S}$**:** For $i \in [m]$, the receiver computes $x_i^{b_i} := \tilde{x}_i^{b_i} \oplus \mathsf{Ext}(GV_i h_i, v_i^{b_i})$ and outputs it.

Theorem 1 *Assume* PRF *is a pseudorandom function,* SCom *is statistically hiding and computationally binding, and* $(\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vrfy})$ *is a unique signature scheme. Then protocol* π *securely realizes* $\mathcal{F}_{\mathsf{multi\text{-}OT}}$ *in the* $\mathcal{F}_{\mathsf{wrap}}$*-hybrid model.*

3.2 Proof Idea

In this section, we briefly sketch the main ideas behind the proof of Theorem 1. A complete proof is deferred to the full version.

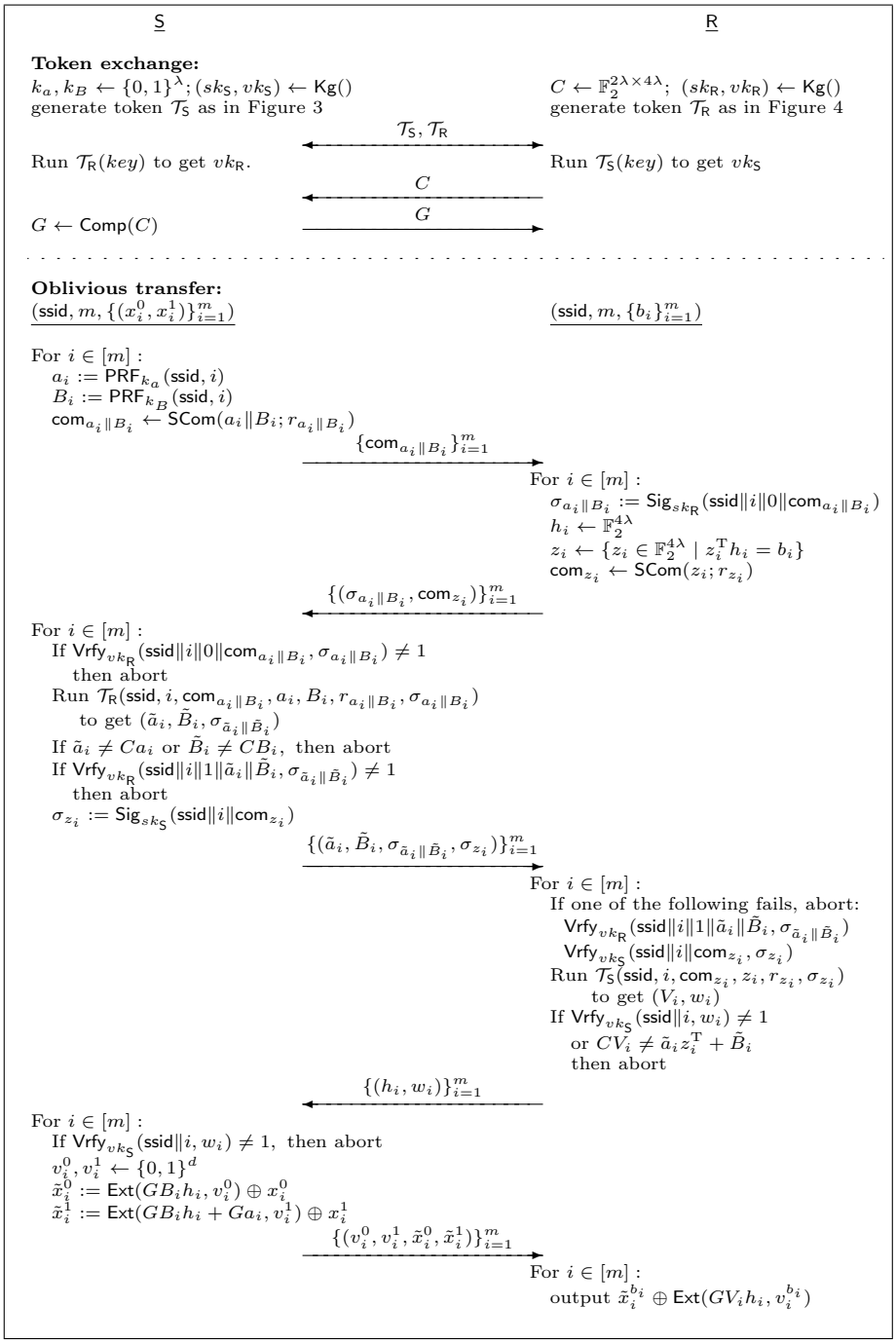

Fig. 5. An OT protocol π from two stateless tokens.

To show security of the protocol, we need to construct a simulator Sim for any non-uniform PPT environment $\mathcal{Z}$ such that $\text{EXEC}^{\mathcal{F}_{\mathsf{wrap}}}_{\pi,\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{\mathsf{multi\text{-}OT}},\mathsf{Sim},\mathcal{Z}}$, where $\mathcal{A}$ is the dummy adversary. Below, we briefly sketch the ideas used to construct such a simulator. Note that we assume w.l.o.g. that the adversary never asks the same query twice.

Sender corruption. First, recall that the token $\mathcal{T}_{\mathsf{R}}$ takes as input $(\mathsf{ssid}, i, \mathsf{com}_{a_i\|B_i}, a_i, B_i, r_{a_i\|B_i}, \sigma_{a_i\|B_i})$ and returns $(\tilde{a}_i, \tilde{B}_i, \sigma_{\tilde{a}_i\|\tilde{B}_i})$. Our protocol forces the malicious sender S^* to query $\mathcal{T}_{\mathsf{R}}$ on exactly one input $(a_i\|B_i)$ for a fixed index i. This allows Sim to extract the input from the protocol transcript. To see this, note that the receiver authenticates the inputs to its token $\mathcal{T}_{\mathsf{R}}$, which in turn authenticates its output. Thus, if the malicious sender does not query the token $\mathcal{T}_{\mathsf{R}}$ but sends a "valid" tuple $(\tilde{a}_i\|\tilde{B}_i, \sigma_{\tilde{a}_i\|\tilde{B}_i})$, then this immediately contradicts the unforgeability of the underlying signature scheme. Also, S^* cannot query $\mathcal{T}_{\mathsf{R}}$ with two different values $(a_i\|B_i), (a'_i\|B'_i)$ (for a fixed i) without contradicting the unforgeability of the signature $\sigma_{a_i\|B_i}$ or the binding of $\mathsf{com}_{a_i\|B_i}$.

Next, consider the maliciously generated token $\mathcal{T}_{\mathsf{S}^*}$. Its input consists of a tuple $(\mathsf{ssid}, i, \mathsf{com}_{z_i}, z_i, r_{z_i}, \sigma_{z_i})$, and its output is (V_i, w_i). The input is chosen carefully in combination with the protocol execution such that the output of the token does not reveal any information about the choice bit $b_i = z_i^{\mathrm{T}} h_i$ of the receiver. Observe that the signature σ_{z_i} depends only on vk_{S} and com_{z_i} in the information thoeretic sense, since we use a unique signature scheme. Therefore, the token knows nothing about h_i. Also, the signature w_i depends only on vk_{R} and (ssid, i) in the information theoretic sense, and it does not contain any information about z_i.

Still, the token $\mathcal{T}_{\mathsf{S}^*}$ may send some limited amount of information about the *previous* executions to S^* by aborting the protocol. Observe that according to our definition of the functionality, OT sub-sessions are executed in a sequential manner, and aborting is allowed *only once*. Thus, the number ℓ of successful sub-sessions so far can encode some information of the input history of $\mathcal{T}_{\mathsf{S}^*}$ up to $O(\log \lambda)$ bits. Therefore, even with this leakage, the adversary S^* has only a negligible advantage in predicting b_i. To see this, note that $b_i = z_i^{\mathrm{T}} h_i$ remains unfixed given $O(\log \lambda)$ bits of z_i.

Receiver corruption. First, recall that the input to the token $\mathcal{T}_{\mathsf{S}}$ is $(\mathsf{ssid}, i, \mathsf{com}_{z_i}, z_i, r_{z_i}, \sigma_{z_i})$ and that its output is (V_i, w_i). As in the malicious sender case, our protocol forces the malicious receiver R^* to query the token on exactly one input z_i for each i. By observing this query, the simulator Sim can extract the input $\hat{b}_i = z_i^{\mathrm{T}} h_i$ of R^*. This property is achieved using the unforgeability of the signature scheme and the binding of the commitment scheme SCom. That is, the unforgeability guarantees that the token runs only on the input (including com_{z_i}) authenticated by the honest sender, and the binding of com_{z_i} disables the malicious receiver R^* to query the token on two distinct z_is.

Next, consider the maliciously generated token $\mathcal{T}_{\mathsf{R}^*}$. The input to the token and its output are carefully handled such that it does not reveal information about (x_i^0, x_i^1) of the sender. To see this, recall the token $\mathcal{T}_{\mathsf{R}^*}$ takes as input $(\mathsf{ssid}, i, \mathsf{com}_{a_i\|B_i}, a_i, B_i, r_{a_i\|B_i}, \sigma_{a_i\|B_i})$ and returns $(\tilde{a}_i, \tilde{B}_i, \sigma_{\tilde{a}_i\|\tilde{B}_i})$. In particular,

since we use a unique signature scheme, the signature $\sigma_{\tilde{a}_i \| \tilde{B}_i}$ information-theoretically depends only on $(vk_{\mathsf{R}}, \tilde{a}_i, \tilde{B}_i)$, so any malicious attempt by R^* and $\mathcal{T}_{\mathsf{R}^*}$ to gain information about (a_i, B_i) beyond $(\mathsf{com}_{a_i \| B_i}, \tilde{a}_i, \tilde{B}_i)$ is not possible.

Still, the token $\mathcal{T}_{\mathsf{R}^*}$ may reveal to the environment some limited amount of information about the previous executions to R^* by aborting the protocol (only allowed one-time), that is, the number ℓ of successful sub-sessions so far can encode some information of the input history of $\mathcal{T}_{\mathsf{R}^*}$. This value ℓ can encode at most $O(\log \lambda)$ bits. Using appropriate extractors in computing the masks for $\tilde{x}_i^0$ and $\tilde{x}_i^1$, we can handle this amount of leakage.

3.3 Bounded Oblivious Transfer from Collision Resistant Hash Functions

The bounded OT protocol between a sender S and a receiver R consists of an initial token-exchange phase after which the parties can carry out a bounded number of oblivious transfers in a single sub-session. We describe π now; see also Figure 8. Here, let $(\mathsf{Sig}, \mathsf{Vrfy})$ be a MAC scheme.

The main intuition for this protocol is that if we allow *only one sub-session* to be executed, then we only need to worry about covert channels that transmit information observed by each party/token so far up to the covert communication. We do not need to worry about a later sub-session sending information about a prior sub-session. This allows us to eliminate the need for unique signature and use MACs instead, thus giving a protocol based only on the existence of CRHFs.

We eliminate the unique signatures in two ways. First, some of the checks in the unbounded protocol can be removed. In particular, in the protocol, the receiver doesn't check the authentication on com_{z_i} before forwarding it to the sender-created token $\mathcal{T}_{\mathsf{S}}$. This is because it cannot contain any useful information about z_i beyond com_{z_i}. For other checks, unique signatures are replaced with (roughly) the following technique: a party A commits to a MAC key, authenticates the message using the MAC key, and later sends the decommitment. Due to the unforgeablity of MAC, the party A makes sure that B runs the token only once before receiving messages of B derived from the token execution result. Given the decommitment, the other party B can check that all the messages from A so far have been legitimate.

Token-exchange phase. Each party generates a single token and sends it to the other party. The sender's token $\mathcal{T}_{\mathsf{S}}$ encapsulates the code described in Figure 6, where $k_a, k_B, k_w, k_W, s_2 \leftarrow \{0,1\}^\lambda$. The receiver's token $\mathcal{T}_{\mathsf{R}}$ encapsulates the code described in Figure 7, where $s \leftarrow \{0,1\}^\lambda$ and $C \leftarrow \mathbb{F}_2^{2\lambda \times 4\lambda}$. Formally, each party sends the relevant code to $\mathcal{F}_{\mathsf{wrap}}$ using an appropriate (CREATE, ...) message. Finally, R sends C to S, and in turn S computes the complementary matrix $G \leftarrow \mathsf{Comp}(C)$ and sends it to R.

Oblivious-transfer phase. Following the token-exchange phase, the parties enter the oblivious transfer phase, where m OT instances are executed in parallel. In this phase, S gets $(\text{SEND}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, \langle m, \{(x_i^0, x_i^1)\}_{i=1}^m \rangle)$ and R gets

On input $(i, \mathsf{com}_z, z, r_z, \tau_z)$:
$v := \mathsf{Vrfy}_{s_2}(i \| \mathsf{com}_z, \tau_z)$
if $z = \mathsf{Open}(\mathsf{com}_z, z, r_z)$ and $v = 1$
$a := \mathsf{PRF}_{k_a}(i)$; $B := \mathsf{PRF}_{k_B}(i)$
$w := \mathsf{PRF}_{k_w}(i)$; $r_w := \mathsf{PRF}_{k_W}(i)$
$V := az^{\mathrm{T}} + B$
output (V, w, r_w)
else output $(\bot, \bot, \bot)$

Fig. 6. The Turing machine $\mathcal{M}_{\mathsf{S}}$ to be embedded in the sender-created token $\mathcal{T}_{\mathsf{S}}$. It is initialized with $(k_a, k_B, k_w, k_W, s_2)$ given by the creator.

On input $(i, \mathsf{com}_{a\|B}, a, B, r_{a\|B}, \tau_{a\|B})$:
$v := \mathsf{Vrfy}_s(i \| 0 \| \mathsf{com}_{a\|B}, \tau_{a\|B})$
if $a\|B = \mathsf{Open}(\mathsf{com}_{a\|B}, a\|B, r_{a\|B})$
and $v = 1$
$\tilde{a} := Ca$; $\tilde{B} := CB$;
$\tau_{\tilde{a}\|\tilde{B}} := \mathsf{Sig}_s(i\|1\|\tilde{a}\|\tilde{B})$
output $(\tilde{a}, \tilde{B}, \tau_{\tilde{a}\|\tilde{B}})$
else output $(\bot, \bot, \bot)$

Fig. 7. The Turing machine $\mathcal{M}_{\mathsf{R}}$ to be embedded in the receiver-created token $\mathcal{T}_{\mathsf{R}}$. It is initialized with (s, C) given by the creator

$(\text{RECEIVE}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, \langle m, \{b_i\}_{i=1}^m \rangle)$ from the environment, and they execute the following protocol:

$\mathsf{S} \to \mathsf{R}$: For $i \in [m]$, the sender computes $a_i := \mathsf{PRF}_{k_a}(i)$, $B_i := \mathsf{PRF}_{k_B}(i)$, $w_i := \mathsf{PRF}_{k_w}(i)$, and $r_{w_i} := \mathsf{PRF}_{k_W}(i)$, and chooses $u_i \leftarrow \{0,1\}^{8\lambda^2+3\lambda}$. Here, we have $a_i \in \mathbb{F}_2^{4\lambda}$, $B_i \in \mathbb{F}_2^{4\lambda \times 4\lambda}$, and $w_i \in \{0,1\}^{\lambda}$. Then, the sender commits to (a_i, B_i), w_i, and u_i by executing $\mathsf{com}_{a_i\|B_i} \leftarrow \mathsf{SCom}(a_i\|B_i; r_{a_i\|B_i})$, and $\mathsf{com}_{w_i} \leftarrow \mathsf{Com}(w_i; r_{w_i})$, and $\mathsf{com}_{u_i} \leftarrow \mathsf{Com}(u_i; r_{u_i})$, respectively.
The sender sends $\{(\mathsf{com}_{a_i\|B_i}, \mathsf{com}_{w_i}, \mathsf{com}_{u_i})\}_{i=1}^m$ to R.

$\mathsf{R} \to \mathsf{S}$: The receiver commits to s by executing $\mathsf{com}_s \leftarrow \mathsf{Com}(s; r_s)$. It also chooses $z_i \leftarrow \mathbb{F}_2^{4\lambda}$, and commits to z_i by executing $\mathsf{com}_{z_i} \leftarrow \mathsf{SCom}(z_i; r_{z_i})$. It next authenticates the commitment $\mathsf{com}_{a_i\|B_i}$ by computing

$$\tau_{a_i\|B_i} := \mathsf{Sig}_s(i\|0\|\mathsf{com}_{a_i\|B_i}) \text{ for } i = 1, \ldots, m.$$

Finally, it sends $(\mathsf{com}_s, \{(\tau_{a_i\|B_i}, \mathsf{com}_{z_i})\}_{i=1}^m)$ to S.

$\mathsf{S} \to \mathsf{R}$: For $i \in [m]$, the sender runs the token $\mathcal{T}_{\mathsf{R}}$ with $(i, \mathsf{com}_{a_i\|B_i}, a_i, B_i, r_{a_i\|B_i}, \tau_{a_i\|B_i})$ and obtains in return $(\tilde{a}_i, \tilde{B}_i, \tau_{\tilde{a}_i\|\tilde{B}_i})$. Then S checks that $\tilde{a}_i = Ca_i$ and $\tilde{B}_i = CB_i$ for $i \in [m]$. If the check fails, S aborts the protocol. Otherwise, S computes $U_i := u_i \oplus (\tilde{a}_i\|\tilde{B}_i\|\tau_{\tilde{a}_i\|\tilde{B}_i})$ for $i \in [m]$ and sends $\{U_i\}_{i=1}^m$ to R.

$\mathsf{R} \to \mathsf{S}$: R sends the decommitment (s, r_s) of com_s to S.

$\mathsf{S} \to \mathsf{R}$: The sender validates all values transmitted so far as follows: It checks that $s = \mathsf{Open}(\mathsf{com}_s, s, r_s)$ and that $\mathsf{Vrfy}_s(i\|0\|\mathsf{com}_{a_i\|B_i}, \tau_{a_i\|B_i}) = 1$, as well as $\mathsf{Vrfy}_s(i\|1\|\tilde{a}_i\|\tilde{B}_i, \tau_{\tilde{a}_i\|\tilde{B}_i}) = 1$ for $i \in [m]$. If the check fails, then S aborts the protocol. Otherwise, for $i \in [m]$ it authenticates the commitment com_{z_i} by computing $\tau_{z_i} := \mathsf{Sig}_{s_2}(i\|\mathsf{com}_{z_i})$. Finally, it sends $(\{(u_i, r_{u_i}, \tau_{z_i})\}_{i=1}^m)$ to R.

$\mathsf{R} \to \mathsf{S}$: For $i \in [m]$, the receiver checks that $u_i = \mathsf{Open}(\mathsf{com}_{u_i}, u_i, r_{u_i})$; if not, it aborts. Then R sets $\tilde{a}_i\|\tilde{B}_i\|\tau_{\tilde{a}_i\|\tilde{B}_i} := U_i \oplus u_i$. It next checks that

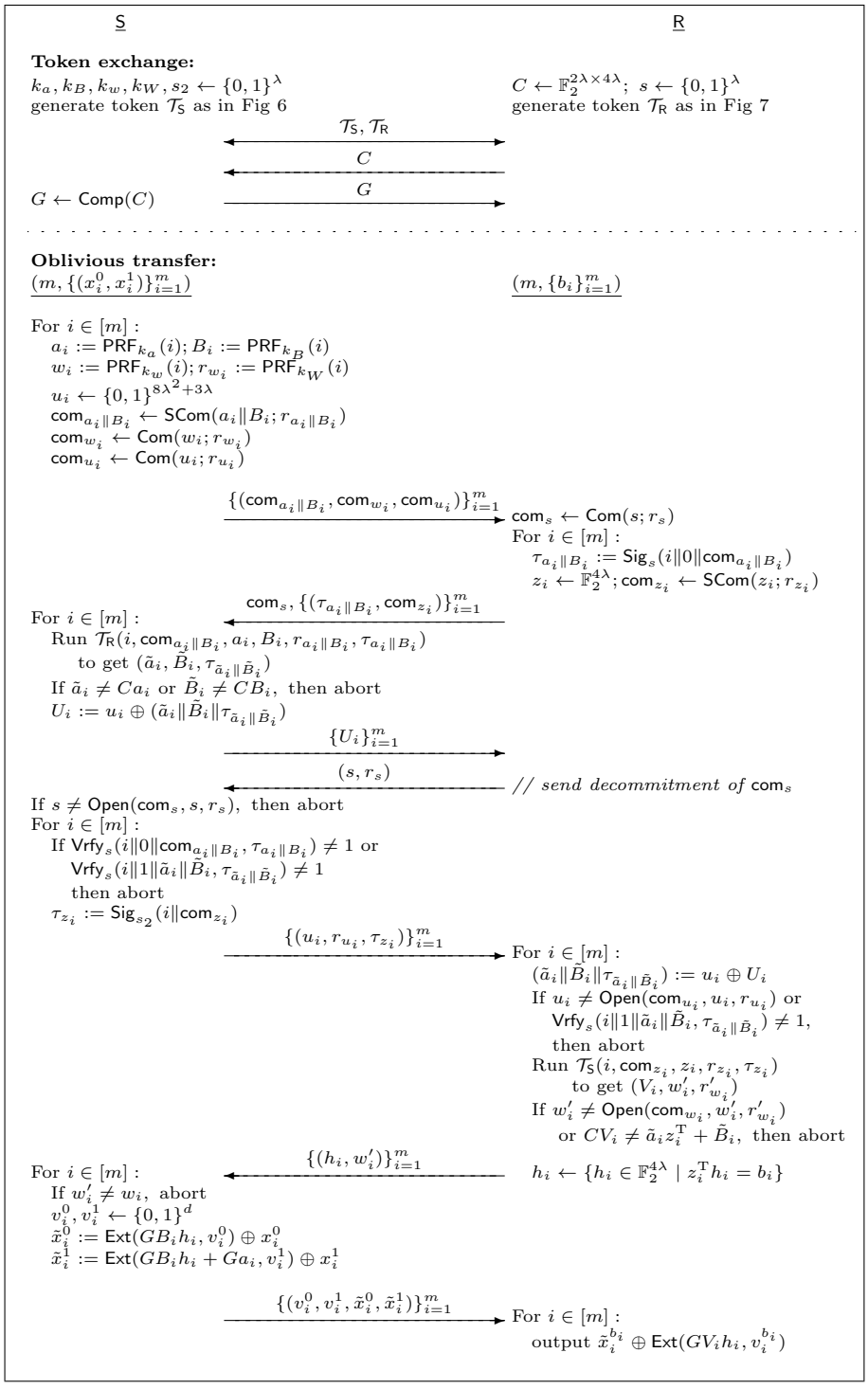

Fig. 8. A bounded OT protocol π from two stateless tokens.

$\mathsf{Vrfy}_s(i\|1\|\tilde{a}_i\|\tilde{B}_i, \tau_{\tilde{a}_i\|\tilde{B}_i}) = 1$ for $i \in [m]$. If this check does not hold, it aborts the protocol. Otherwise, for $i \in [m]$, it runs the token $\mathcal{T}_{\mathsf{S}}$ with $(i, \mathsf{com}_{z_i}, z_i, r_{z_i}, \tau_{z_i})$ and obtains in return (V_i, w'_i, r'_{w_i}). The receiver checks that $w'_i = \mathsf{Open}(\mathsf{com}_{w_i}, w'_i, r'_{w_i})$, that $CV_i = \tilde{a}_i z_i^{\mathrm{T}} + \tilde{B}_i$, and that $w'_i \neq \perp$ for $i \in [m]$. If the check fails, then it aborts the protocol. Otherwise, it chooses $h_i \leftarrow \mathbb{F}_2^{4\lambda}$ subject to the constraint that $b_i = z_i^{\mathrm{T}} h_i$ for $i \in [m]$, and sends $\{(h_i, w'_i)\}_{i=1}^m$ to S.

$\mathsf{S} \to \mathsf{R}$**:** The sender checks that $w_i = w'_i$ for $i \in [m]$; if not, it aborts the protocol. Otherwise, for $i \in [m]$ it chooses $v_i^0, v_i^1 \leftarrow \{0,1\}^d$ and computes $\tilde{x}_i^0 := \mathsf{Ext}(GB_i h_i, v_i^0) \oplus x_i^0$ and $\tilde{x}_i^1 := \mathsf{Ext}(GB_i h_i + Ga_i, v_i^1) \oplus x_i^1$. Finally it sends $\{(v_i^0, v_i^1, \tilde{x}_i^0, \tilde{x}_i^1)\}_{i=1}^m$ to the receiver.

$\mathsf{R} \to \mathsf{S}$**:** For $i \in [m]$, the receiver computes $x_i^{b_i} := \tilde{x}_i^{b_i} \oplus \mathsf{Ext}(GV_i h_i, v_i^{b_i})$ and outputs it.

4 Black-Box Impossibility of OT Using One Stateless Token

In the previous section we showed that an unbounded number of universally composable OTs (and hence universally composable secure computation) is possible by having the parties exchange two stateless tokens. We show here that a construction of (even a single) OT using *one* stateless token is impossible using black-box techniques alone. Here, by "black-box" we refer to the simulator's access to the code $\mathcal{M}$ encapsulated in the token. Note that the token model as defined by Katz [32] is inherently *non*black-box and, in particular, the simulator is given the code $\mathcal{M}$ that the malicious party submits to $\mathcal{F}_{\mathsf{wrap}}$. Nevertheless, an examination of the proof of security of our two-token protocol in Section 3— as well as the proofs of security in almost all prior work [12, 37, 23, 15, 24, 25, 19]— shows that the simulator only uses this code in a black-box fashion, namely, by running the code to observe its input/output behavior but without using any internal structure of the code itself. We formalize this in what follows.

Specifically, we consider simulators of the form $\mathsf{Sim} = (\mathsf{Sim}_{code}, \mathsf{Sim}_{bb})$, where Sim_{code} gets the code $\mathcal{M}$ that the adversary submits to $\mathcal{F}_{\mathsf{wrap}}$, and then runs Sim_{bb} as a subroutine while giving it oracle access to $\mathcal{M}$. Inspired by [8, 9] we show that, restricting to such simulators, constructions of OT from one stateless token are impossible. Intuitively, for any such protocol proven secure using black-box techniques, Sim_{bb} must be able to extract the input of a corrupted token-creating party by interacting with $\mathcal{M}$ in a black-box fashion. The real-world adversary can then use Sim_{bb} to extract the input of the honest token-creating party by running Sim_{bb} and answering its oracle queries by *querying the token* itself; for *stateless* tokens, querying the token (in the real world) is equivalent to black-box access to $\mathcal{M}$ (in the ideal world).

Theorem 2 *There is no protocol π that uses one stateless token to securely realize $\mathcal{F}_{\mathsf{OT}}$ in the $\mathcal{F}_{\mathsf{wrap}}$-hybrid model whose security is proven using a simulator $\mathsf{Sim} = (\mathsf{Sim}_{code}, \mathsf{Sim}_{bb})$ as defined above.*

Proof (Sketch) For completeness, the (single-session) OT functionality $\mathcal{F}_{\mathsf{OT}}$ is given in Figure 9. Let π be a protocol between a sender S and a receiver R, in which a single token is sent from the sender to the receiver. (The other case is handled analogously.) Consider the following environment $\mathcal{Z}'$ and dummy adversary $\mathcal{A}'$ corrupting the sender: $\mathcal{Z}'$ chooses random bits s_0, s_1, and b, and instructs the sender to run π honestly on input (s_0, s_1), including submitting some code to $\mathcal{F}_{\mathsf{wrap}}$ to create a token. Once the honest receiver outputs s, then $\mathcal{Z}'$ outputs 1 if $s = s_b$ and 0 otherwise.

Suppose that π securely realizes $\mathcal{F}_{\mathsf{OT}}$, where security is proved via a simulator $\mathsf{Sim} = (\mathsf{Sim}_{code}, \mathsf{Sim}_{bb})$ as previously described. In the course of the proof, Sim_{bb} *plays the role of a receiver* while interacting with $\mathcal{A}'$, and Sim_{code} provides Sim_{bb} with black-box access to whatever code $\mathcal{A}'$ submits to $\mathcal{F}_{\mathsf{wrap}}$. At some point during its execution, Sim_{bb} must send some inputs $(\tilde{s}_0, \tilde{s}_1)$ to the ideal functionality $\mathcal{F}_{\mathsf{OT}}$. It is not hard to see that we must have $(\tilde{s}_0, \tilde{s}_1) = (s_0, s_1)$ with all but negligible probability.

We now consider a different environment $\mathcal{Z}$ and an adversary $\mathcal{A}$ *corrupting the receiver*. $\mathcal{Z}$ chooses random bits s_0, s_1, b and provides (s_0, s_1) as input to the honest sender; it outputs 1 iff $\mathcal{A}$ outputs (s_0, s_1). Note that $\mathcal{A}$ receives a token from the honest sender as specified by π. Adversary $\mathcal{A}$ works as follows:

> Run Sim_{bb}, relaying messages from the honest sender to this internal copy of Sim_{bb}. Whenever Sim_{bb} makes a query to Sim_{code} to run $\mathcal{M}$ (the code created by the honest sender) on some input q, adversary $\mathcal{A}$ runs the token with input q (formally, $\mathcal{A}$ sends $(\text{RUN}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, q)$ to $\mathcal{F}_{\mathsf{wrap}}$ and gives the response to Sim_{bb}). At some point, Sim_{bb} sends $(\tilde{s}_0, \tilde{s}_1)$ to $\mathcal{F}_{\mathsf{OT}}$, at which point $\mathcal{A}$ outputs $(\tilde{s}_0, \tilde{s}_1)$ and halts.

The key point is that *because the token is stateless*, there is no difference between Sim_{code} running the code $\mathcal{M}$, and $\mathcal{A}$ querying the token via $\mathcal{F}_{\mathsf{wrap}}$. Thus, $\mathcal{A}$ provides a perfect simulation for Sim_{bb}, and we conclude that $(\tilde{s}_0, \tilde{s}_1) = (s_0, s_1)$ with all but negligible probability in an execution of π in the $\mathcal{F}_{\mathsf{wrap}}$-hybrid world. But this occurs with probability at most $1/2$ in an ideal-world evaluation of $\mathcal{F}_{\mathsf{OT}}$. Thus, $\mathcal{Z}$ can distinguish between the real- and ideal-world executions, contradicting the claimed security of π. □

5 Coin Tossing Using One Stateless Token

In the previous section we showed that universally composable OT cannot be realized from one stateless token if only "black-box techniques" are used. We complement that result by showing that UC secure computation from one stateless token *is* feasible via *non*black-box techniques. Here, we find it somewhat simpler to construct a protocol for universally composable coin tossing rather than OT. (The coin-tossing functionality $\mathcal{F}_{\mathsf{coin}}$ is defined in the natural way; see Figure 10 .) Note that coin tossing suffices for general secure computation under a variety of cryptographic assumptions [10, 35].

Functionality $\mathcal{F}_{\mathsf{OT}}$

$\mathcal{F}_{\mathsf{OT}}$ interacts with sender S, receiver R, and the adversary.

Upon receiving an input $(\text{SEND}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, \langle x_0, x_1 \rangle)$ from S with $x_0, x_1 \in \{0,1\}$, record the tuple $\langle x_0, x_1 \rangle$, and reveal $(\text{SEND}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle)$ to the adversary. Ignore further $(\text{SEND}, \ldots)$ inputs.

Upon receiving an input $(\text{RECEIVE}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, b)$ from R with $b \in \{0,1\}$, record the bit b, and reveal $(\text{RECEIVE}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle)$ to the adversary. Ignore further $(\text{RECEIVE}, \ldots)$ inputs.

Upon receiving a message $(\text{GO}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle)$ from the adversary, ignore the message if $\langle x_0, x_1 \rangle$ or b is not recorded. Otherwise, do the following: return $(\text{RECEIVED}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle)$ to S, and return $(\text{RECEIVED}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle, x_b)$ to R. Ignore further $(\text{GO}, \langle \mathsf{sid}, \mathsf{S}, \mathsf{R} \rangle)$ messages from the adversary.

Fig. 9. The ideal $\mathcal{F}_{\mathsf{OT}}$ functionality for bit-OT.

Functionality $\mathcal{F}_{\mathsf{coin}}$

$\mathcal{F}_{\mathsf{coin}}$ interacts with two parties, Alice A and Bob B, and the adversary. The functionality is parameterized by a security parameter λ. It also maintains variables $(b_{\mathsf{A}}, b_{\mathsf{B}}, \mathsf{coins})$ initialized to $(\mathsf{false}, \mathsf{false}, \perp)$.

Upon receiving an input $(\text{TOSS}, \langle \mathsf{sid}, \mathsf{A}, \mathsf{B} \rangle)$ from party $P \in \{\mathsf{A}, \mathsf{B}\}$, then set $b_P := \mathsf{true}$, and reveal $(\text{TOSS}, \langle \mathsf{sid}, \mathsf{A}, \mathsf{B} \rangle, P)$ to the adversary. Ignore further $(\text{TOSS}, \langle \mathsf{sid}, \mathsf{A}, \mathsf{B} \rangle)$ inputs from the party P.

Upon receiving a message $(\text{GO}, \langle \mathsf{sid}, \mathsf{A}, \mathsf{B} \rangle, P)$ from the adversary for $P \in \{\mathsf{A}, \mathsf{B}\}$, ignore the message if $b_{\mathsf{A}} = \mathsf{false}$ or $b_{\mathsf{B}} = \mathsf{false}$. Otherwise, do the following: if $\mathsf{coins} = \perp$, i.e, coins has not been set yet, then randomly choose $u \leftarrow \{0,1\}^{\lambda}$ and set $\mathsf{coins} := u$; return $(\text{COIN}, \langle \mathsf{sid}, \mathsf{A}, \mathsf{B} \rangle, \mathsf{coins})$ to party P. Ignore further $(\text{GO}, \langle \mathsf{sid}, \mathsf{A}, \mathsf{B} \rangle, P)$ messages for the party P from the adversary.

Fig. 10. The ideal $\mathcal{F}_{\mathsf{coin}}$ functionality for coin tossing.

At a high level, our protocol follows the general structure of Blum's coin-tossing protocol [4]. This protocol consists of three moves. In the first move (B1), Alice commits to a random x and sends com_x to Bob, who in return chooses a random value y and sends it to Alice (B2). In the last move (B3), Alice sends the decommitment to x and both parties output $x \oplus y$.

To obtain a UC coin-tossing protocol that follows this basic approach, we need to be able to *simulate* each party. In particular, if Alice is malicious then the simulator needs to extract the message contained in com_x (extractability), whereas when Bob is corrupted the simulator needs to open the commitment in an arbitrary way (equivocation). We achieve both goals by having Bob send a single stateless token $\mathcal{T}_{\mathsf{B}}$ to Alice. This token behaves in two different ways, depending on its input. The first task of the token is to generate a random value

e upon seeing com_x (we discuss the details later); the second task is to generate a notification t for Bob that Alice knows the decommitment x. The value e is used for equivocation, and the notification t gives the simulator the ability to extract x. We give further details next. In the high-level description here, we assume the token is created honestly; we deal with a potentially malicious token when we formally define the protocol, below.

Achieving extractability. Similar to Section 3, the token only works on authenticated inputs. That is, whenever Alice wants to query the token on some inputs, she needs to first ask Bob to compute a MAC on those inputs. In order to achieve extractability, we let Alice query the token on input $(\mathsf{com}_x, x, r_x, \tau_x)$, where $\mathsf{com}_x \leftarrow \mathsf{SCom}(x; r_x)$ comes from Alice and τ_x is a MAC tag on com_x. The output of the token is a random value t that can be seen as a notification to Bob that Alice knows the decommitment x. As in the previous OT protocol, the authentication of the inputs guarantees that Alice makes exactly one valid query to the token: more than one valid query would imply that Alice has violated security of the MAC or binding of com_x; if Alice makes no query, then she cannot guess t. By forcing Alice to query the token exactly once, we can now easily construct a simulator that extracts the value x while emulating $\mathcal{F}_{\mathsf{wrap}}$. Modifying Blum's coin-tossing protocol, we have the following step:

(B1) Alice commits to x by executing $\mathsf{com}_x \leftarrow \mathsf{SCom}(x; r_x)$. She sends the commitment com_x to Bob, who in turn computes a tag τ_x on com_x and sends τ_x to Alice. Alice runs the token with $(\mathsf{com}_x, x, r_x, \tau_x)$ to obtain output t, and sends t to Bob. Bob checks if t is correct.

Achieving equivocation. Toward this goal, we further modify the protocol and the token. Alice sends com_x and a dummy commitment com_M before getting the tag τ_x on $\mathsf{com}_x \| \mathsf{com}_M$ from Bob. The token also gets as an additional input this commitment com_M; it outputs a random value e on input $(\mathsf{com}_x, \mathsf{com}_M, \tau_x)$ and a random value t on input $(\mathsf{com}_x, \mathsf{com}_M, x, r_x, \tau_x)$. In step (B3), instead of sending the decommitment (x, r_x) of com_x to Bob, Alice sends x together with a witness-indistinguishable (WI) proof that either x is a valid decommitment of com_x, or com_M contains code that outputs the actual output e of the token $\mathcal{T}_{\mathsf{B}}$. (This is where we use the nonblack-box techniques of Barak [2].)
Due to the binding property of com_M, and because the notification e is unpredictable, Alice cannot commit to such code in com_M. The simulator, however, takes advantage of the fact that it obtains the code of the token generated by Bob while emulating $\mathcal{F}_{\mathsf{wrap}}$. Then, as in [2], the simulator's ability to predict the output of the token beforehand can be used to achieve equivocation. We remark that in contrast to Barak's work, we do not need to use universal arguments; this is because $\mathcal{F}_{\mathsf{wrap}}$ is parameterized with a fixed polynomial bounding the running time of the token (whereas Barak had to handle any polynomial running time).

More formally, we change the protocol as follows:

Token The token input is either $(\mathsf{com}_x, \mathsf{com}_M, \tau_x)$ or $(\mathsf{com}_x, x, r_x, \mathsf{com}_M, \tau_x)$ and in both cases it checks the validity of τ_x before responding. In the first

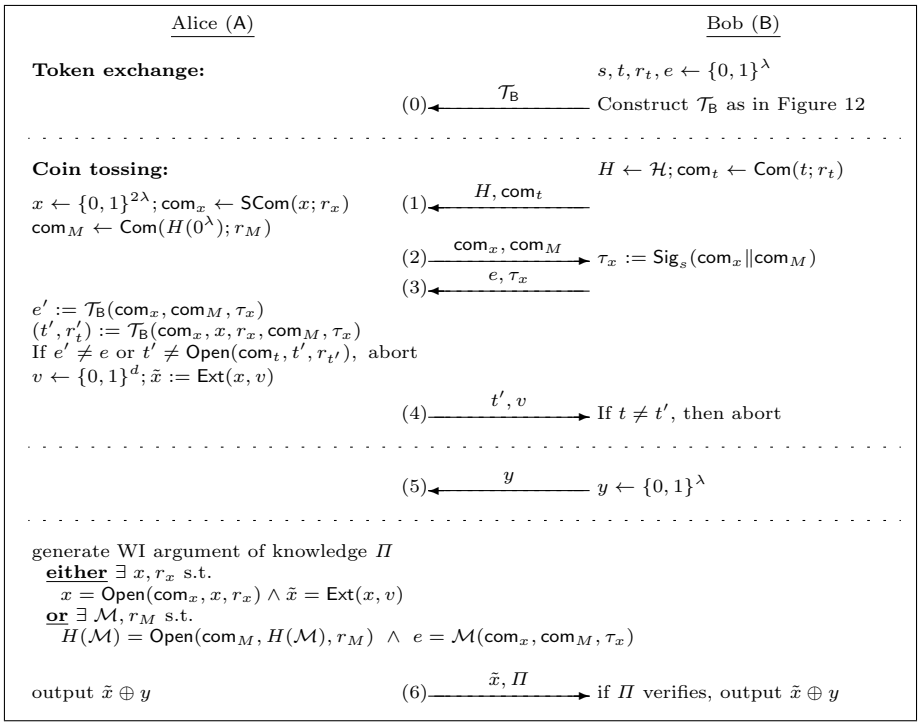

Fig. 11. A coin-tossing protocol ψ from a single stateless token.

case, the token outputs a random value e. In the second case, it returns a random value t.

Protocol Once Alice obtains the token, she runs the following protocol with Bob:

(B1) Alice commits to x and sends com_x together with a dummy commitment com_M to Bob. In turn, Bob authenticates $\mathsf{com}_x \| \mathsf{com}_M$ using a MAC and sends (τ_x, e) to Alice. Alice invokes the token twice, first with $(\mathsf{com}_x, \mathsf{com}_M, \tau_x)$ and then with $(\mathsf{com}_x, x, r_x, \mathsf{com}_M, \tau_x)$, obtaining the values e' and t', respectively. She checks if $e = e'$ and, if so, sends t' to Bob. Finally, Bob checks if $t' = t$.

(B2) Bob sends Alice a random value y.

(B3) Alice sends x together with a WI proof that either (i) (x, r_x) is the decommitment of com_x, or (ii) com_M is a commitment to a Turing machine $\mathcal{M}$ such that $\mathcal{M}(\mathsf{com}_x, \mathsf{com}_M, \tau_x) = e$.

5.1 Formal Description of the Protocol

The protocol ψ between Alice A and Bob B consists of an initial token-exchange phase, followed by a coin-tossing phase for generating a random λ-bit string. We now describe ψ formally; see also Figure 11.

On input $(\mathsf{com}_x, \mathsf{com}_M, \tau_x)$ do:
 if $\mathsf{Vrfy}_s(\mathsf{com}_x \| \mathsf{com}_M, \tau_x) = 1$
  output e
 else output $\perp$

On input $(\mathsf{com}_x, x, r_x, \mathsf{com}_M, \tau_x)$ do:
 if $\mathsf{Vrfy}_s(\mathsf{com}_x \| \mathsf{com}_M, \tau_x) = 1$
  and $x = \mathsf{Open}(\mathsf{com}_x, x, r_x)$
  output (t, r_t)
 else output $(\perp, \perp)$

Fig. 12. The Turing machine $\mathcal{M}$ embedded in the sender-created token $\mathcal{T}_{\mathsf{B}}$. Here, $e, s, t, r_t \in \{0,1\}^\lambda$ are chosen uniformly at random and embedded in the token.

Token-exchange phase. Bob generates a single token $\mathcal{T}_{\mathsf{B}}$ and sends it to Alice. Bob's token $\mathcal{T}_{\mathsf{B}}$ encapsulates the code $\mathcal{M}$ described in Figure 12, where s, e, t, r_t are each chosen uniformly from $\{0,1\}^\lambda$.

Coin-tossing phase. In this phase, Alice and Bob proceed as follows.

$\mathsf{B} \to \mathsf{A}$: Bob chooses a collision-resistant hash function $H \leftarrow \mathcal{H}$. He also commits to the value t (that he used when creating the token) by executing $\mathsf{com}_t \leftarrow \mathsf{Com}(t; r_t)$. He sends H, com_t to Alice.

$\mathsf{A} \to \mathsf{B}$: Alice chooses a value $x \leftarrow \{0,1\}^{2\lambda}$ and commits to x and $H(0^\lambda)$ by executing $\mathsf{com}_x \leftarrow \mathsf{SCom}(x; r_x)$ and $\mathsf{com}_M \leftarrow \mathsf{Com}(H(0^\lambda); r_M)$. She sends com_x and com_M to Bob.

$\mathsf{B} \to \mathsf{A}$: Bob generates a MAC tag $\tau_x := \mathsf{Sig}_s(\mathsf{com}_x \| \mathsf{com}_M)$, and sends (e, τ_x) to Alice. Recall that Bob has already chosen e and embedded the value in the token $\mathcal{T}_{\mathsf{B}}$ in the token exchange phase.

$\mathsf{A} \to \mathsf{B}$: Alice runs the token $\mathcal{T}_{\mathsf{B}}$ with $(\mathsf{com}_x, \mathsf{com}_M, \tau_x)$ and obtains e' in response. Then she runs the token with $(\mathsf{com}_x, x, r_x, \mathsf{com}_M, \tau_x)$ and obtains (t', r'_t). Alice checks if $e' = e$ and $t' = \mathsf{Open}(\mathsf{com}_t, t', r'_t)$. If not, she aborts the protocol. Otherwise, she chooses $v \leftarrow \{0,1\}^d$ and sends (t', v) to Bob, where d is an appropriate seed length for the extractor.

$\mathsf{B} \to \mathsf{A}$: Bob checks that $t = t'$, and aborts if not. Otherwise he chooses $y \leftarrow \{0,1\}^\lambda$ and sends it to Alice.

$\mathsf{A} \to \mathsf{B}$: Alice sends $\tilde{x} := \mathsf{Ext}(x, v)$ and gives a WI argument of knowledge that $(H, \mathsf{com}_x, \mathsf{com}_M, e, \tau_x, v, \tilde{x})$ belongs to the $\mathcal{NP}$ language L defined by the following relation R_L:

$R_L((H, \mathsf{com}_x, \mathsf{com}_M, e, \tau_x, v, \tilde{x}), (\alpha, \beta)) = 1$ if either one of the following holds:

(i) $\alpha = \mathsf{Open}(\mathsf{com}_x, \alpha, \beta)$ and $\tilde{x} = \mathsf{Ext}(\alpha, v)$.

(ii) $H(\alpha) = \mathsf{Open}(\mathsf{com}_M, H(\alpha), \beta)$. In addition, treating α as the description of a Turing machine, the execution of $\alpha(\mathsf{com}_x, \mathsf{com}_M, \tau_x)$ outputs e in time at most $p(\lambda)$, where p is the polynomially bounded running time defined by $\mathcal{F}_{\mathsf{wrap}}$.

If the proof succeeds, both parties output $\tilde{x} \oplus y$.

Theorem 3 *Assume* Com *is computationally hiding and strongly binding,* SCom *is statistically hiding and computationally binding,* MAC *is a deterministic, unforgeable message-authentication code, $\mathcal{H}$ is a family of collision-resistant hash functions, and the proof system is a witness-indistinguishable argument of knowledge. Then ψ securely realizes $\mathcal{F}_{\mathsf{coin}}$ in the $\mathcal{F}_{\mathsf{wrap}}$-hybrid model.*

5.2 Proof Idea

In this section, we briefly sketch the main ideas behind the proof of Theorem 3. A complete proof is deferred to the full version.

To show the security of the protocol, we need to construct a simulator Sim for any PPT environment $\mathcal{Z}$ such that $\text{EXEC}^{\mathcal{F}_{\mathsf{wrap}}}_{\mathcal{A},\pi,\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{\mathsf{coin}},\mathsf{Sim},\mathcal{Z}}$, where $\mathcal{A}$ is the dummy adversary. Below, we briefly provide ideas for simulation.

Corrupting Alice. Note that Alice cannot commit to a TM $\mathcal{M}$ in com_M that will output e, since e is chosen at random independently. Therefore, from the collision-resistance of H, Alice has to show that $\tilde{x} = \mathsf{Ext}(x, v)$ in the WI proof. Since v appears in the communication transcript, by forcing malicious Alice A^* to query $\mathcal{T}_{\mathsf{B}}$ with x exactly once, the simulator Sim can extract the value $\tilde{x}$ from binding of com_x. Then, upon receiving the random string coins from $\mathcal{F}_{\mathsf{coin}}$, the simulator can send $y = \tilde{x} \oplus \mathsf{coins}$ to A^*. To see how the protocol forces exactly one query to the token, note that malicious Alice is not able to generate a valid t without querying $\mathcal{T}_{\mathsf{B}}$, since t is random and com_t is hiding. Also, note that A^* cannot query $\mathcal{T}_{\mathsf{B}}$ with different xs without contradicting the unforgeability of the tag τ_x and/or the binding of com_x. This allows the simulator to extract the value of x.

Corrupting Bob. The simulator Sim needs to equivocate the value $\tilde{x}$ so that it may hold that $\tilde{x} = \mathsf{coins} \oplus y$, where coins is the random string from $\mathcal{F}_{\mathsf{coin}}$. This is achieved as follows. While emulating $\mathcal{F}_{\mathsf{wrap}}$, the simulator obtains the token code $\mathcal{M}$ generated by Bob, then it generates $\mathsf{com}_M \leftarrow \mathsf{Com}(H(\mathcal{M}); R_M)$. Given the hiding property of com_M, the simulated transcript is indistinguishable from that in the $\mathcal{F}_{\mathsf{wrap}}$-hybrid world. Now, with the witness $(\mathcal{M}, R_M)$ in the WI proof, the simulator can send any value $\tilde{x}$.

References

1. Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *4th Theory of Cryptography Conference—TCC 2007*, volume 4392 of *LNCS*, pages 137–156. Springer, 2007.
2. B. Barak. How to go beyond the black-box simulation barrier. In *42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 106–115. IEEE, 2001.
3. B. Barak, R. Canetti, J. B. Nielsen, and R. Pass. Universally composable protocols with relaxed set-up assumptions. In *45th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 186–195. IEEE, 2004.
4. M. Blum. Coin flipping by telephone. In *Proc. IEEE COMPCOM*, pages 133–137, 1982.
5. S. Brands. Untraceable off-line cash in wallets with observers (extended abstract). In *Advances in Cryptology—Crypto '93*, volume 773 of *LNCS*, pages 302–318. Springer, 1994.
6. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145. IEEE, 2001. Latest version available at `http://eprint.iacr.org/2000/067`, December 2005.

7. R. Canetti. Obtaining universally compoable security: Towards the bare bones of trust (invited talk). In *Advances in Cryptology — Asiacrypt 2007*, volume 4833 of *LNCS*, pages 88–112. Springer, 2007.
8. R. Canetti and M. Fischlin. Universally composable commitments. In *Advances in Cryptology — Crypto 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, 2001.
9. R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *Journal of Cryptology*, 19(2):135–167, 2006.
10. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 494–503. ACM Press, 2002.
11. R. Canetti, R. Pass, and A. Shelat. Cryptography from sunspots: How to use an imperfect reference string. In *48th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 249–259. IEEE, 2007.
12. N. Chandran, V. Goyal, and A. Sahai. New constructions for UC secure computation using tamper-proof hardware. In *Advances in Cryptology — Eurocrypt 2008*, volume 4965 of *LNCS*, pages 545–562. Springer, 2008.
13. D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Advances in Cryptology — Crypto '92*, volume 740 of *LNCS*, pages 89–105. Springer, 1993.
14. R. Cramer and T. P. Pedersen. Improved privacy in wallets with observers (extended abstract). In *Advances in Cryptology — Eurocrypt '93*, volume 765 of *LNCS*, pages 329–343. Springer, 1993.
15. I. Damgård, J. B. Nielsen, and D. Wichs. Universally composable multiparty computation with partially isolated parties. In *6th Theory of Cryptography Conference — TCC 2009*, volume 5444 of *LNCS*, pages 315–331. Springer, 2009.
16. I. Damgård, T. P. Pedersen, and B. Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. *Journal of Cryptology*, 10(3):163–194, 1997.
17. Y. Desmedt and J.-J. Quisquater. Public-key systems based on the difficulty of tampering (is there a difference between DES and RSA?). In *Advances in Cryptology — Crypto '86*, volume 263 of *LNCS*, pages 111–117. Springer, 1987.
18. Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In S. Vaudenay, editor, *8th Intl. Workshop on Theory and Practice in Public Key Cryptography(PKC 2005)*, volume 3386 of *LNCS*, pages 416–431. Springer, Jan. 2005.
19. N. Döttling, D. Kraschewski, and J. Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In *8th Theory of Cryptography Conference — TCC 2011*, volume 6597 of *LNCS*, pages 164–181. Springer, 2011.
20. N. Döttling, T. Mie, J. Müller-Quade, and T. Nilges. Basing obfuscation on simple tamper-proof hardware assumptions. Cryptology ePrint Archive, Report 2011/675, 2011. `http://eprint.iacr.org/`.
21. M. Dubovitskaya, A. Scafuro, and I. Visconti. On efficient non-interactive oblivious transfer with tamper-proof hardware. Cryptology ePrint Archive, Report 2010/509, 2010.
22. M. Fischlin, B. Pinkas, A.-R. Sadeghi, T. Schneider, and I. Visconti. Secure set intersection with untrusted hardware tokens. In *Cryptographers' Track — RSA 2011*, volume 6558 of *LNCS*, pages 1–16. Springer, 2011.
23. S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. One-time programs. In *Advances in Cryptology — Crypto 2008*, volume 5157 of *LNCS*, pages 39–56. Springer, 2008.

24. V. Goyal, Y. Ishai, M. Mahmoody, and A. Sahai. Interactive locking, zero-knowledge PCPs, and unconditional cryptography. In *Advances in Cryptology — Crypto 2010*, volume 6223 of *LNCS*, pages 173–190. Springer, 2010.
25. V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia. Founding cryptography on tamper-proof hardware tokens. In *7th Theory of Cryptography Conference — TCC 2010*, volume 5978 of *LNCS*, pages 308–326. Springer, 2010.
26. I. Haitner and O. Reingold. Statistically-hiding commitment from any one-way function. In *39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10. ACM Press, 2007.
27. C. Hazay and Y. Lindell. Constructions of truly practical secure protocols using standard smartcards. In *ACM CCS '08: 15th ACM Conf. on Computer and Communications Security*, pages 491–500. ACM Press, 2008.
28. D. Hofheinz, J. Müller-Quade, and D. Unruh. Universally composable zero-knowledge arguments and commitments from signature cards. In *Proc. 5th Central European Conference on Cryptology (MoraviaCrypt)*, 2005.
29. O. Horvitz and J. Katz. Bounds on the efficiency of "black-box" commitment schemes. In *ICALP 2005: 32nd Intl. Colloquium on Automata, Languages, and Programming (ICALP)*, volume 3580 of *LNCS*, pages 128–139. Springer, 2005.
30. Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In D. Wagner, editor, *Advances in Cryptology — Crypto 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, 2008.
31. K. Järvinen, V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Embedded SFE: Offloading server and network using hardware tokens. In *Financial Cryptography and Data Security 2010*, volume 6052 of *LNCS*, pages 207–221. Springer, 2010.
32. J. Katz. Universally composable multi-party computation using tamper-proof hardware. In *Advances in Cryptology — Eurocrypt 2007*, volume 4515 of *LNCS*, pages 115–128. Springer, 2007.
33. J. Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.
34. V. Kolesnikov. Truly efficient string oblivious transfer using resettable tamper-proof tokens. In *7th Theory of Cryptography Conference — TCC 2010*, volume 5978 of *LNCS*, pages 327–342. Springer, 2010.
35. H. Lin, R. Pass, and M. Venkitasubramaniam. A unified framework for concurrent security: Universal composability from stand-alone non-malleability. In *41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 179–188. ACM Press, 2009.
36. Y. Lindell. General composition and universal composability in secure multiparty computation. *Journal of Cryptology*, 22(3):395–428, 2009.
37. T. Moran and G. Segev. David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In *Advances in Cryptology — Eurocrypt 2008*, volume 4965 of *LNCS*, pages 527–544. Springer, 2008.
38. M. Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.