Founding Cryptography on Tamper-Proof Hardware Tokens

Vipul Goyal[1,⋆], Yuval Ishai[2,⋆⋆], Amit Sahai[3,⋆⋆⋆], Ramarathnam Venkatesan[4], and Akshay Wadia[3]

[1] UCLA and MSR India
vipul.goyal@gmail.com
[2] Technion and UCLA
yuvali@cs.technion.ac.il
[3] UCLA
{sahai,awadia}@cs.ucla.edu
[4] Microsoft Research, India and Redmond
venkie@microsoft.com

Abstract. A number of works have investigated using tamper-proof hardware tokens as tools to achieve a variety of cryptographic tasks. In particular, Goldreich and Ostrovsky considered the problem of software protection via oblivious RAM. Goldwasser, Kalai, and Rothblum introduced the concept of *one-time programs*: in a one-time program, an honest sender sends a set of *simple* hardware tokens to a (potentially malicious) receiver. The hardware tokens allow the receiver to execute a secret program specified by the sender's tokens exactly once (or, more generally, up to a fixed t times). A recent line of work initiated by Katz examined the problem of achieving UC-secure computation using hardware tokens.

Motivated by the goal of unifying and strengthening these previous notions, we consider the general question of basing secure computation on hardware tokens. We show that the following tasks, which cannot be realized in the "plain" model, become feasible if the parties are allowed to generate and exchange tamper-proof hardware tokens.

– Unconditional and non-interactive secure computation. We show that by exchanging simple *stateful* hardware tokens, any functionality can be realized with *unconditional* security against malicious parties. In the case of two-party functionalities $f(x, y)$ which take their inputs from a sender and a receiver and deliver their output to the receiver, our protocol is non-interactive and only requires a unidirectional communication of simple stateful tokens from the

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-642-11799-2_36

⋆ Research supported in part by a Microsoft Research Graduate Fellowship and the grants of Amit Sahai mentioned below.

⋆⋆ Supported in part by ISF grant 1310/06, BSF grants 2008411, and NSF grants 0830803, 0716835, 0627781.

⋆⋆⋆ Research supported in part from NSF grants 0916574, 0830803, 0627781, and 0716389, BSF grant 2008411, an equipment grant from Intel, and an Okawa Foundation Research Grant.

D. Micciancio (Ed.): TCC 2010, LNCS 5978, pp. 308–326, 2010.
© Springer-Verlag Berlin Heidelberg 2010

sender to the receiver. This strengthens previous feasibility results for one-time programs both by providing *unconditional* security and by offering general protection against *malicious senders*. As is typically the case for unconditionally secure protocols, our protocol is in fact *UC-secure*. This improves over previous works on UC-secure computation based on hardware tokens, which provided computational security under cryptographic assumptions.

- Interactive secure computation from stateless tokens based on one-way functions. We show that *stateless* hardware tokens are sufficient to base general secure (in fact, UC-secure) computation on the existence of *one-way functions*.
- Obfuscation from stateless tokens. We consider the problem of realizing non-interactive secure computation from stateless tokens for functionalities which allow the receiver to provide an arbitrary number of inputs (these are the only functionalities one can hope to realize non-interactively with *stateless* tokens). By building on recent techniques for resettably secure computation, we obtain a general positive result under standard cryptographic assumptions. This gives the first general feasibility result for program obfuscation using *stateless* tokens, while strengthening the standard notion of obfuscation by providing security against a malicious sender.

1 Introduction

A number of works (e.g. [1,2,3,4,5,6,7,8,9,10,11,12,13]) have investigated using tamper-proof hardware tokens[1] as tools to achieve a variety of cryptographic goals. There has been a surge of research activity on this front of late. In particular, the recent work of Katz [9] examined the problem of achieving UC-secure [14] two party computation using tamper-proof hardware tokens. A number of follow-up papers [10,11,12] have further investigated this problem. In another separate (but related) work, Goldwasser et al. [13] introduced the concept of *one-time programs*: in a one-time program, a (semi-honest) sender sends a set of very simple hardware tokens to a (potentially malicious) receiver. The hardware tokens allow the receiver to execute a program specified by the sender's tokens exactly once (or, more generally, up to a fixed t times). This question is related to the more general goal of software protection using hardware tokens, which was first addressed by Goldreich and Ostrovsky [1] using the framework of oblivious RAM.

The present work is motivated by the observation that several of these previous goals and concepts can be presented in a unified way as instances of one general goal: realizing *secure computation* using tamper-proof hardware tokens. The lines of work mentioned above differ in the types of functionalities being

[1] Informally, a tamper-proof hardware token provides the holder of the token with black-box access to the functionality of the token. We will often omit the words "tamper-proof" when referring to hardware tokens, but all of the hardware tokens referred to in this paper are assumed to be tamper-proof.

considered (e.g., non-reactive vs. reactive), the type of interaction between the parties (interactive vs. non-interactive protocols), the type of hardware tokens (stateful vs. stateless, simple vs. complex), and the precise security model (stand-alone vs. UC, semi-honest vs. malicious parties). This unified point of view also gives rise to strictly stronger notions than those previously considered, which in turn give rise to new feasibility questions in this area.

The introduction of tamper-proof hardware tokens to the model of secure computation, as formalized in [9], invalidates many of the fundamental impossibility results in cryptography. Taking a step back to look at this general model from a foundational perspective, we find that a number of natural feasibility questions regarding secure computation with hardware tokens remain open. In this work we address several of these questions, focusing on goals that are impossible to realize in the plain model without tamper-proof hardware tokens:

- **Is it possible to achieve unconditional security for secure computation with hardware tokens?** We note that this problem is open even for stand-alone security, let alone UC security, and impossible in the plain model [15]. While in the semi-honest model this question is easy to settle by relying on unconditional protocols based on oblivious transfer (OT) [16,17,18,19], this question is more challenging when both parties as well as the tokens they generate can be malicious. (See Sections 1.2 and 3.1 for relevant discussion.) In the case of *stateless* tokens, which may be much easier to implement, security against unbounded adversaries cannot be generally achieved, since an unbounded adversary can "learn" the entire description of the token. A natural question in this case is **whether stateless tokens can be used to realize (UC) secure computation based on the assumption that one-way functions exist**.

  Previous positive results for secure two-party computation with hardware tokens relied either on specific number theoretic assumptions [9] or the existence of oblivious transfer protocols in the plain model [10,11], or alternatively offered weaker notions of security [20].

  A related question is: **is it possible to obtain unconditionally secure one-time programs for all polynomial-time computable functions?** The previous work of [13] required the existence one-way functions in order to construct one-time programs.
- **Is it possible to realize non-interactive secure two-party computation with simple hardware tokens?** Again, this problem is open[2] even for stand-alone security, and impossible in the plain model. Constructions of oblivious RAM [1] and one-time programs [13] provide partial solutions to

[2] All the previous questions were open even without any restriction on the size of the tokens. In the current and the following questions we restrict the tokens to be *simple* in the sense that the size of each token can only depend on the security parameter. This rules out a trivial solution of building a token which realizes a party in a secure two-party computation protocol.

this problem; however, in these models the sender is semi-honest.[3] Thus, in the context of one-time programs we ask: **is it possible to achieve one-time programs tolerating a malicious sender?** We note that [13] make partial progress towards this question by constructing one-time zero knowledge proofs, where the prover can be malicious. However, in the setting of hardware tokens, the GMW paradigm [21] of using zero knowledge proofs to compile semi-honest protocols into protocols tolerating malicious behavior does not apply, since one would potentially need to prove statements about hardware tokens (as opposed to ordinary NP statements).

- **Which notions of program obfuscation can be realized using simple hardware tokens?** Again, this problem can be captured in an elegant way within the framework of secure two-party computation, except that here we need to consider *reactive* functionalities which may take a single input from the "sender" and a sequence of (possibly adaptively chosen) inputs from the "receiver". Obfuscation can be viewed as a non-interactive secure realization of such functionalities. While this general goal is in some sense realized by the construction of oblivious RAM [1] (which employs stateful tokens), several natural questions remain: **Is it possible to achieve obfuscation using only stateless tokens? Is it possible to offer a general protection against a malicious sender using stateless or even stateful tokens?** To illustrate the motivation for the latter question, consider the goal of obfuscating a poker-playing program. The receiver of the obfuscated program would like to be assured that the sender did not violate the rules of the game (and in particular cannot bias the choice of the cards).
- **What are the simplest kinds of tamper-proof hardware tokens needed to realize the above goals?** For example, Goldwasser et al. [13] introduce a very simple kind of stateful token that they call an OTM (one-time memory) token.[4] An OTM token stores two strings s_0 and s_1, takes a single bit b as input, and then outputs s_b and stops working (or self-destructs). Note that an OTM token essentially implements the one-out-of-two string OT functionality; a subtle distinction between OTM and traditional OT is discussed in Section 3.1. An even simpler type of token is a *bit-OTM* token, where the strings s_0 and s_1 are restricted to be single bits. **Is it possible to realize unconditional, non-interactive, or UC-secure two-party computation using only bit-OTM tokens?** We note that previous works on secure two-party computation with hardware tokens [9,10,11,20] all make use of more complicated hardware tokens.

[3] In these models, the sender is allowed to arbitrarily specify the functionality of the oblivious RAM or the one-time program, and the receiver knows nothing about this functionality except an upper bound on its circuit size or running time. (Thus, the issue of dishonest senders does not arise in these models.) In the present work, by a one-time program tolerating a malicious sender, we mean that the receiver knows some partial specification of the functionality – modeled in the usual paradigm of secure two-party computation.

[4] The use of OTM tokens in [13] is motivated in part by the goal of achieving *leakage resilience,* a feature that our constructions using such tokens inherit as well.

1.1 Our Results

We show that the following tasks, which cannot be realized in the "plain" model, become feasible if the parties are allowed to generate and exchange simple tamper-proof hardware tokens.

- **Unconditional non-interactive secure computation.** We show that by exchanging *stateful* hardware tokens, any functionality can be realized with *unconditional* security against malicious parties. In the case of two-party functionalities $f(x, y)$ which take their inputs from a sender and a receiver and deliver their output to the receiver, our protocol is non-interactive and only requires a unidirectional communication of tokens from the sender to the receiver (in case an output has to be given to both parties, adding a reply from the receiver to the sender is sufficient). This result strengthens previous feasibility results for one-time programs by providing *unconditional* security, by offering general protection against *malicious senders*, and by using only bit-OTM tokens.

  As is typically the case for unconditionally secure protocols, our protocol is in fact *UC-secure*. This improves over previous works on UC-secure computation based on hardware tokens, which provided computational security under cryptographic assumptions.

  See Sections 3.1 and 3.2 for details of this result and a high level overview of techniques.
- **Interactive secure computation from stateless tokens based on one-way functions.** We show that *stateless* hardware tokens are sufficient to base general secure (in fact, UC-secure) computation on the existence of *one-way functions*. One cannot hope for security against unbounded adversaries with stateless tokens since an unbounded adversary could query the token multiple times to "learn" the functionality it contains. See Section 4 for details.
- **Obfuscation from stateless tokens.** We consider the problem of realizing non-interactive secure computation from stateless tokens for reactive functionalities which take a single input from the sender and an arbitrary sequence of inputs from the receiver (these are the only functionalities one can hope to realize non-interactively with *stateless* tokens). By building on recent techniques for resettably secure computation [22], we obtain a general positive result under standard cryptographic assumptions. This gives the first general feasibility result for program obfuscation using *stateless* tokens, while strengthening the standard notion of obfuscation by providing security against a malicious sender. We also propose constructions of non-interactive secure computation for general reactive functionalities with *stateful* tokens. See the full version for details.

In all of the above results, the size of each hardware token is either constant or polynomial in the security parameter, and its code is independent of the inputs of the parties. Thus, the tokens could theoretically be "mass-produced" before being used in any particular protocol with any particular inputs.

We stress that in contrast to some previous results along this line (most notably, [1,13,20]), our focus is almost entirely on *feasibility* questions, while only briefly discussing more refined efficiency considerations. However, in most cases our stronger feasibility results can be realized while also meeting the main efficiency goals pursued in previous works.

The first two results above are obtained by utilizing previous protocols for secure computation based on OT [18,19], and thus a main ingredient in our constructions is showing how to securely implement OT using hardware tokens. Note that in the case of non-interactive secure computation, additional tools are needed since the protocols of [18,19] are (necessarily) interactive.

1.2 Related Work

The use of tamper-proof hardware tokens for cryptographic purposes was first explored by Goldreich and Ostrovsky [1] in the context of software protection (one-time programs [13] is a relaxation of this goal, generally called program obfuscation [23]), and by Chaum, Pederson, Brands, and Cramer [2,3,4] in the context of e-cash. Ishai, Sahai, and Wagner [5] and Ishai, Prabhakaran, Sahai and Wagner [24] consider the question of how to construct tamper-proof hardware tokens when the hardware itself does not guarantee complete protection against tampering. Gennaro, Lysyanskaya, Malkin, Micali, and Rabin [6] consider a similar question, when the underlying hardware guarantees that part of the hardware is tamper-proof but readable, while the other part of the hardware is unreadable but susceptible to tampering. Moran and Naor [8] considered a relaxation of tamper-proof hardware called "tamper-evident seals," and given number of constructions of graphic tasks based on this relaxed notion. Hofheinz, Müller-Quade, and Unruh [25] consider a model similar to [9] in the context of UC-secure protocols where tamper-proof hardware tokens (signature cards) are issued by a trusted central authority.

The model that we primarily build on here is due to Katz [9], who considers a setting in which users can create and exchange tamper-proof hardware tokens where malicious users have full control over the functionality realized by each token they create. The main result of [9] is a general protocol for UC-secure two-party computation using stateful tokens, under the DDH assumption. Chandran, Goyal, Sahai [10] implement UC-secure two-party computation using stateless tokens, under the assumption that oblivious transfer protocols exist in the plain model. Aside from just considering stateless tokens, [10] also introduce a variant of the model of [9] that allows for the adversary to pass along tokens, and in general allows the adversary not to know the code of the tokens he produces. We do not consider this model here. Moran and Segev [11] also implement UC-secure two-party computation under the same assumption as [10], but using stateful tokens, and only requiring tokens to be passed in one direction. Damgård, Nielsen, and Wichs [12] show how to relax the "isolation" requirement of tamper-proof hardware tokens, and consider a model in which tokens can communicate a fixed number of bits back to its creator. Hazay and Lindell [20] propose constructions of practical protocols for various problems of interest using trusted stateful

tokens. Very recently and independently of our work, practical oblivious transfer protocols using stateless tokens and relying only on one-way functions were suggested by Kolesnikov [26]. In contrast to the corresponding feasibility result from our work, these protocols either provide a weaker security guarantee or assume that tokens are well-formed, but on the other hand they offer better practical efficiency.

Goldwasser, Kalai, and Rothblum [13] introduced the notion of one-time programs, and showed how to realize it under the assumption that one-way functions exist, as we have already discussed. They also construct one-time zero-knowledge proofs under the same assumption. Their results focus mainly on achieving efficiency in terms of the number of tokens needed, and a non-adaptive use of the tokens by the receiver.

Finally, in a seemingly unrelated work which is motivated by quantum physics, Buhrman, Christandl, Unger, Wehner and Winter [27] consider the application of *non-local boxes* to cryptography. Using non-local boxes, Buhrman et al. show an unconditional construction for oblivious transfer in the interactive setting. A non-local box implements a *trusted* functionality taking input and giving output to both the parties (as opposed to OTM tokens which could be prepared maliciously). However, the key problem faced by Buhrman et al. is similar to a problem we face as well: delayed invocation of the non-local box by a malicious party. Indeed, one can give a simple interactive protocol (omitted here) for building a trusted non-local-box using OTM tokens. This provides an alternative to the interactive variant of our construction of unconditional secure computation from hardware tokens described in Section 3.1.

2 Preliminaries

In this section we briefly discuss some of the underlying definitions and concepts. The reader is referred to the full version for the details.

We use the UC-framework of Canetti [28] to capture the general notion of secure computation of (possibly reactive) functionalities. Our main focus is on the two-party case. We will usually refer to one party as a "sender" and to another as a "receiver". A *non-reactive* functionality may receive an input from each party and deliver output to each party (or only to the receiver). A *reactive* functionality may have several rounds of inputs and outputs, possibly maintaining state information between rounds.

Our model for tamper-proof hardware is similar to that of Katz [9]. As we consider both stateful and stateless tokens, we define different ideal functionalities for the two. By $\mathcal{F}_{wrap}^{single}$ we denote an ideal functionality that allows a sender to generate a "one-time token" which can be invoked by its designated receiver. A one-time token is a stateful token which takes an input from the receiver and returns a function which is specified in advance by the sender. (Note that if the sender is malicious, this function can be arbitrary.) After being invoked by the receiver, such a token "self-destructs". Thus, the only state these tokens keep is a flag which indicates whether the token has been run or not. Simple tokens of this type were used in [13].

We also define an ideal functionality $\mathcal{F}_{wrap}^{stateless}$ for *stateless* tokens. Here the token computes some (deterministic) function specified by the sender, and the receiver can query the token an unbounded number of times. Note that this makes stateless tokens useless if the receiver has enough resources to "learn" the token's description (either because the token is too small or the receiver is too powerful). [5]

By a *non-interactive* protocol we refer to a protocol in which the communication only involves a single batch of tokens, possibly along with an additional message, communicated from a sender to a receiver.

3 Unconditional Non-interactive Secure Computation Using Stateful Tokens

In this section we establish the feasibility of unconditionally non-interactive secure computation based on *stateful* hardware tokens. As is typically the case for unconditionally secure protocols, our protocols are in fact *UC secure*.

This section is organized as follows. In Subsection 3.1 we present an interactive protocol for arbitrary functionalities, which requires the parties to engage in multiple rounds of interaction. This gives an unconditional version of previous protocols for UC-secure computation based on hardware tokens [9,10,11], which all relied on computational assumptions. [6] This subsection also introduces some useful building blocks that are used for the non-interactive solution in the next subsection.

In Subsection 3.2 we consider the case of secure evaluation of two-party functionalities which deliver output to only one of the parties (the "receiver"). We strengthen the previous result in two ways. First, we show that in this case interaction can be completely eliminated: it suffices for the sender to non-interactively send tokens to the receiver, without any additional communication. Second, we show that even very simple, constant-size stateful tokens are sufficient for this purpose. This strengthens previous feasibility results for one-time programs [13] by providing *unconditional* security (in fact, UC-security), by offering general protection against *malicious senders*, and by using constant-size tokens.

3.1 The Interactive Setting

Unconditionally secure two-party computation is impossible to realize for most nontrivial functionalities, even with semi-honest parties [29,30]. However, if the parties are given oracle access to a simple ideal functionality such as Oblivious

[5] While the formal definition of this functionality forces a malicious sender to also use only *stateless* tokens, this requirement can be relaxed without affecting the security of our protocols. See Section 4 for details.

[6] The work of [11] realizes an unconditionally UC-secure *commitment* from stateful tokens. This does not directly yield protocols for secure computation without additional computational assumptions.

Transfer (OT) [16,17], then it becomes possible not only to obtain unconditionally secure computation with semi-honest parties [31,32,33], but also unconditional *UC-security* against *malicious* parties [18,19]. This serves as a natural starting point for our construction.

In the OT-hybrid model, the two parties are given access to the following ideal OT functionality: the input of P_1 (the "sender") consists of a pair of k-bit strings (s_0, s_1), the input of P_2 (the "receiver") is a choice bit c, and the receiver's output is the chosen string s_c. The natural way to implement a single OT call using stateful hardware tokens is by having the sender send to the receiver a token which, on input c, outputs s_c and erases s_{1-c} from its internal state. The use of such hardware tokens was first suggested in the context of one-time programs [13]. Following the terminology of [13], we refer to such tokens as OTM (one-time-memory) tokens.

An appealing feature of OTM tokens is their simplicity, which can also lead to better resistance against side-channel attacks (see [13] for discussion). This simplicity feature served as the main motivation for using OTM tokens as a basis for one-time programs. Another appealing feature, which is particularly important in our context, is that the OTM functionality does not leave room for bad sender strategies: whatever badly formed token a malicious sender may send is equivalent from the point of view of an honest receiver to having the sender send a well-formed OTM token picked from some probability distribution. (This is not the case for tokens implementing more complex functionalities, such as 2-out-of-3 OT or the extended OTM functionality discussed below, for which badly formed tokens may not correspond to any distribution over well-formed tokens.)

Given the above, it is tempting to hope that our goal can be achieved by simply taking any unconditionally secure protocol in the OT-hybrid model, and using OTM tokens to implement OT calls. However, as observed in [13], there is a subtle but important distinction between the OT-hybrid model and the OTM-hybrid model: while in the former model the sender knows the point in the protocol in which the receiver has already made its choice and received its output, in the latter model invoking the token is entirely at the discretion of the receiver. This may give rise to attacks in which the receiver adaptively invokes the OTM tokens "out of order," and such attacks may have a devastating effect on the security of protocols even in the case of unconditional security. A more detailed discussion of such attacks and simple solution ideas (that do not work) is included in the full version.

Extending the OTM functionality. To solve the above problem, we will realize an extended OTM functionality which takes from the sender a pair of strings (s_0, s_1) along with an auxiliary string r, takes from the receiver a choice bit c, and delivers to the receiver both s_c and r. We denote this functionality by ExtOTM. What makes the ExtOTM functionality nontrivial to realize using hardware tokens is the need to protect the receiver from a malicious sender who may try to make the received r depend on the choice bit c while *at the same*

time protecting the sender from a malicious receiver who may try to postpone its choice c until after it learns r.

Using the ExtOTM functionality, it is easy to realize a UC-style version of the OT functionality which not only delivers the chosen string to the receiver (as in the OTM functionality) but also delivers an acknowledgement to the sender. This flavor of the OT functionality, which we denote by $\mathcal{F}^{\mathrm{OT}}$, can be realized by having the sender invoke ExtOTM with (s_0, s_1) and a randomly chosen r, and having the receiver send r to the sender. In contrast to OTM, the $\mathcal{F}^{\mathrm{OT}}$ functionality allows the sender to force any subset of the OT calls to be completed before proceeding with the protocol. This suffices for instantiating the OT calls in the unconditionally secure protocols from [18,19]. We refer the reader to the full version of this paper for a UC-style definition of the OTM, ExtOTM, and $\mathcal{F}^{\mathrm{OT}}$ functionalities.

Realizing ExtOTM using general[7] stateful tokens. As discussed above, we cannot directly use a stateful token for realizing the ExtOTM functionality, because this allows the sender to correlate the delivered r with the choice bit c. On the other hand, we cannot allow the sender to directly reveal r to the receiver, because this will allow the receiver to postpone its choice until after it learns r. In the following we sketch our protocol for realizing ExtOTM using stateful tokens. This protocol is non-interactive (i.e., it only involves tokens sent from the sender to the receiver) and will also be used as a building block towards the stronger results in the next subsection. We refer the reader to the full version of this paper for a formal description of the protocol and its proof of security. Below we include a detailed overview.

As mentioned above, at a high level, the challenge we face is to prevent unwanted correlations in an information-theoretic way for both malicious senders and malicious receivers. This is a more complex situation than a typical similar situation where only one side needs to be protected against (c.f. [34,35]). To accomplish this goal, we make use of secret-sharing techniques combined with additional token-based "verification" techniques to enforce honest behavior.

Our ExtOTM protocol Π_{ExtOTM} starts by having the sender break its auxiliary string r into $2k$ additive shares r^i, and pick $2k$ pairs of random strings (q_0^i, q_1^i). (Each of the strings q_b^i and r^i is k-bit long, where k is a statistical security parameter.) It then generates $2k$ OTM tokens, where the i-th token contains the pair $(q_0^i \circ r^i, q_1^i \circ r^i)$ (where '$\circ$' is the concatenation operator). Note that a malicious sender may generate badly formed OTM tokens which correlate r^i with the i-th choice of the receiver; we will later implement a token-based verification strategy that convinces an honest receiver that the sender did not cheat (too much) in this step.

Now the receiver breaks its choice bit c into $2k$ additive shares c^i, and invokes the $2k$ OTM tokens with these choice bits. Let $(\hat{q}^i, \hat{r}^i)$ be the pair of k-bit strings obtained by the receiver from the i-th token. Note that if the sender is honest, the

[7] Here, we make use of general tokens. Later in this section, we will show how to achieve the ExtOTM functionality (and in fact every poly-time functionality) using only very simple tokens – just bit OTM tokens.

receiver can already learn r. We would like to allow the receiver to learn its chosen string s_c while convincing it that the sender did not correlate all of the auxiliary strings $\hat{r}^i$ with the corresponding choice bits c_i. (The latter guarantee is required to assure an honest receiver that $\hat{r} = \sum \hat{r}^i$ is independent of c as required.)

This is done as follows. The sender prepares an additional single-use hardware token which takes from the receiver its $2k$ received strings $\hat{q}^i$, checks that for each $\hat{q}^i$ there is a valid selection $\hat{c}_i$ such that $\hat{q}^i = q^i_{\hat{c}_i}$ (otherwise the token returns $\perp$), and finally outputs the chosen string $s_{\hat{c}^1 \oplus \ldots \oplus \hat{c}^{2k}}$. (All tokens in the protocol can be sent to the receiver at one shot.) Note that the additive sharing of r in the first $2k$ tokens protects an honest sender from a malicious receiver who tries to learn $s_{\hat{c}}$ where $\hat{c}$ is significantly correlated with r, as it guarantees that the receiver effectively commits to c before obtaining any information about r. The receiver is protected against a malicious sender because even a badly formed token corresponds to some (possibly randomized) ideal-model strategy of choosing (s_0, s_1).

Finally, we need to provide to the receiver the above-mentioned guarantee that a malicious sender cannot correlate the receiver's auxiliary output $\hat{r} = \sum \hat{r}^i$ with the choice bit c. To explain this part, it is convenient to assume that both the sender and the badly formed tokens are deterministic. (The general case is handled by a standard averaging argument.) In such a case, we call each of the first $2k$ tokens well-formed if the honest receiver obtains the same r^i regardless of its choice c^i, and we call it badly formed otherwise. By the additive sharing of c, the only way for a malicious sender to correlate the receiver's auxiliary output with c is to make *all* of the first $2k$ tokens badly formed. To prevent this from happening, we require the sender to send a final token which proves that it knows all of the $2k$ auxiliary strings $\hat{r}^i$ obtained by the receiver. This suffices to convince the receiver that not all of the first $2k$ tokens are badly formed. Note, however, that we cannot ask the sender to send these $2k$ strings r^i in the clear, since this would (again) allow a malicious receiver to postpone its choice c until after it learns r.

Instead, the sender generates and sends a token which first verifies that the receiver knows r (by comparing the receiver's input to the k-bit string r) and only then outputs all $2k$ shares r^i. The verification step prevents correlation attacks by a malicious receiver. The final issue to worry about is that the string r received by the token (which may be correlated with the receiver's choices c_i) does not reveal to the sender enough information to pass the test even if all of its first $2k$ tokens are badly formed. This follows by a simple information-theoretic argument: in order to pass the test, the token must correctly guess *all* $2k$ bits c_i, but this cannot be done (except with $2^{-\Omega(k)}$ probability) even when given arbitrary k bits of information about the c_i.

The above protocol shows the following (see full version for proof):

Claim. Protocol Π_{ExtOTM} realizes ExtOTM with statistical UC-security in the $\mathcal{F}^{single}_{wrap}$-hybrid model.

We are now ready to prove the main feasibility result of this subsection.

Theorem 1 (Interactive unconditionally secure computation using stateful tokens). *Let f be a (possibly reactive) polynomial-time computable functionality. Then there exists an efficient, statistically UC-secure interactive protocol which realizes f in the $\mathcal{F}_{wrap}^{single}$-hybrid model.*

Proof. We compose three reductions. The protocols of [18,19] realize unconditionally secure two-party (and multi-party) computation of general functionalities using $\mathcal{F}^{\mathrm{OT}}$. A trivial reduction described above reduces $\mathcal{F}^{\mathrm{OT}}$ to ExtOTM. Finally, the above Claim reduces ExtOTM to $\mathcal{F}_{wrap}^{single}$.

3.2 The Non-interactive Setting

In this subsection we restrict the attention to the case of securely evaluating two-party functionalities $f(x, y)$ which take an input x from the sender and an input y from the receiver, and deliver $f(x, y)$ to the receiver. We refer to such functionalities as being *sender-oblivious*. Note that here we consider only *non-reactive* sender-oblivious functionalities, which interact with the sender and the receiver in a single round. The reactive case will be discussed in the full version.

Unlike the case of general functionalities, here one can hope to obtain *non-interactive* protocols in which the sender unidirectionally send tokens (possibly along with additional messages[8]) to the receiver.

For sender-oblivious functionalities, the main result of this subsection strengthens the results of Section 3.1 in two ways. First, it shows that a non-interactive protocol can indeed realize such functionalities using stateful tokens. Second, it pushes the simplicity of the tokens to an extreme, relying only on OTM tokens which contain pairs of *bits*.

Below we provide only a high-level description of the construction and the underlying ideas. We refer the reader to the full version for the full description of the protocols and their analysis.

One-time programs. Our starting point is the concept of a *one-time program* (OTP) [13]. A one-time program can be viewed in our framework as a non-interactive protocol for $f(x, y)$ which uses only OTM tokens, and whose security only needs to hold for the case of a *semi-honest sender* (and a malicious receiver).[9] The main result of [13] establishes the feasibility of computationally-secure OTPs for any polynomial-time computable f, based on the existence of one-way functions. The construction is based on Yao's garbled circuit technique [37]. Our initial observation is that if f is restricted to the complexity class NC^1, one can replace Yao's construction by an efficient perfectly secure variant (cf. [38]). This yields perfectly secure OTPs for NC^1. Alternatively, we

[8] Since our main focus is on establishing feasibility results, the distinction between the "hardware" part and the "software" part is not important for our purposes.

[9] The original notion of OTP from [13] is syntactically different in that it views f as a function of the receiver's input, where a description of f is given to the sender. This can be captured in our framework by letting $f(x, y)$ be a universal functionality.

also present a general construction of a OTP from any "decomposable randomized encoding" of f. This can be used to derive perfectly secure OTPs for larger classes of functions (including NL) based on randomized encoding techniques from [39,38]. See the full version for further details.

A next natural step is to construct unconditionally secure OTPs for any polynomial-time computable function f. In the full version of this paper, we describe a direct and self-contained construction which uses the perfect OTPs for NC^1 described above to build a statistically secure construction for any f. However, this result will be subsumed by our main result, which can be proved (in a less self-contained way) without relying on the latter construction.

Handling malicious senders. As in Section 3.1, the main ingredient in our solution is an interactive secure protocol Π for f. The high level idea of our construction is obtain a non-interactive protocol for f which emulates Π by having the sender generate and send a one-time token which computes the sender's next message function for each round of Π (a similar idea was used in [13] to construct one time proofs). Using the above procedure, we transform Π into a non-interactive protocol Π' which uses very complex one-time tokens (for implementing the next message functions of Π). The next idea is that we can break each such complex token into simple OTM tokens by using a one-time program realization of each complex token. More details are provided in the full version.

From the plain model to the OT-hybrid model. So far we assumed the protocol Π to be secure in the plain model. This rules out unconditional security as well as UC-security, which are our main goals in this section. A natural approach for obtaining unconditional UC-security is to extend the above compiler to protocols in the OT-hybrid model. This introduces a subtle difficulty which was already encountered in Section 3.1: the sender cannot directly implement the OT calls by using OTM tokens. To solve this problem, we build on the (non-interactive) ExtOTM protocol from Section 3.1. See full version for details.

From string-OTM to bit-OTMs. As a final optimization, in the full version we show how to use an unconditionally UC-secure non-interactive implementation of a string-OTM token using bit-OTM tokens.

This yields the following main result of this section:

Theorem 2 (Non-interactive unconditionally secure computation using bit-OTM tokens). *Let $f(x, y)$ be a non-reactive, sender-oblivious, polynomial-time computable two-party functionality. Then there exists an efficient, statistically UC-secure non-interactive protocol which realizes f in the $\mathcal{F}_{wrap}^{single}$-hybrid model in which the sender only sends bit-OTM tokens to the receiver.*

4 Two-Party Computation with Stateless Tokens

In this section, we again address the question of achieving interactive two-party computation protocols, but asking the following questions: (1) Can we rely on

stateless tokens while only assuming that one-way functions exist? (2) Can the above be achieved without requiring that the complexity or number of the tokens grows with the complexity of the function being computed, as was the case in the previous section? We show how to positively answer both questions: We use stateless tokens, whose complexity is polynomial in the security parameter, to implement the OT functionality. Since (as discussed earlier) secure protocols for any two-party task exist given OT, this suffices to achieve the claimed result.

Before turning to our protocols, we make a few observations about stateless tokens to set the stage. First, we observe that with stateless tokens, it is always possible to have protocols where tokens are exchanged *only at the start of the protocol*. This is simply because each party can create a "universal" token that takes as input a pair (c, x), where c is a (symmetric authenticated/CCA-secure) encryption[10] of a machine M, and outputs $M(x)$. Then, later in the protocol, instead of sending a new token T, a party only has to send the encryption of the code of the token, and the other party can make use of that encrypted code and the universal token to emulate having the token T. The proof of security and correctness of this construction is straightforward.

Dealing with dishonestly created *stateful* tokens. The above discussion, however, assumes that dishonest players also only create stateless tokens. If that is not the case, then re-using a dishonestly created token may cause problems with security. If we allow dishonest players to create stateful tokens, then a simple solution is to repeat the above construction and send separate universal tokens for each future *use* of any token by the other player, where honest players are instructed to only use each token once. Since this forces all tokens to be used in a stateless manner, this simple fix is easily shown to be correct and secure; however, it may lead to a large number of tokens being exchanged. To deal with this, as was discussed in the previous section, we observe that by Beaver's OT extension result [36] (which requires only one-way functions), it suffices to implement $O(k)$ OTs, where k is the security parameter, in order to implement any polynomial number of OTs. Thus, it suffices to exchange only a polynomial number of tokens even in the setting where dishonest players may create stateful tokens.

Convention for intuitive protocol descriptions. In light of the previous discussions, in our protocol descriptions, in order to be as intuitive as possible, we describe tokens as being created at various points during the protocol. However, as noted above, our protocols can be immediately transformed into ones where a bounded number of tokens (or in the model where statelessness is guaranteed, only one token each) are exchanged in an initial setup phase.

4.1 Protocol Intuition

We now discuss the intuition behind our protocol for realizing OT using stateless tokens; due to the complexity of the protocol, we do not present the intuition

[10] An "encrypt-then-MAC" scheme would suffice here.

for the entire protocol all at once, but rather build up intuition for the different components of the protocol and why they are needed, one component at a time. For this intuition, we will assume that the sender holds two *random* strings s_0 and s_1, and the receiver holds a choice bit b. Note that OT of random strings is equivalent to OT for chosen strings [41].

The Basic Idea. Note that, since stateless tokens can be re-used by malicious players, if we naively tried to create a token that output s_b on input the receiver's choice bit b, the receiver could re-use it to discover both s_0 and s_1. A simple idea to prevent this reuse would be the following protocol, which is our starting point:

1. Receiver sends a commitment $c = \mathsf{com}(b; r)$ to its choice bit b.
2. Sender sends a token, that on input (b, r), checks if this is a valid decommitment of c, and if so, outputs s_b.
3. Receiver feeds (b, r) to the token it received, and obtains $w = s_b$

Handling a Malicious Receiver. Similar to the problem discussed in the previous section, there is a problem that the receiver may choose not to use the token sent by the sender until the end of the protocol (or even later!). In our context, this can be dealt with easily. We can have the sender commit to a random string π at the start of the protocol, and require that the sender's token must, in addition to outputting s_b, also output a valid decommitment to π. We then add a last step where the receiver must report π to the sender. Only upon receipt of the correct π value does the sender consider the protocol complete.

Proving Knowledge. While this protocol seems intuitive, we note that it is actually *insecure* for a fairly subtle reason. A dishonest sender could send a token that on input (b, r), simply outputs (b, r) (as a string). This means that at the end of the protocol, the dishonest sender can output a specific commitment c, such that the receiver's output is a decommitment of c showing that it was a commitment to the receiver's choice bit b. It is easy to see that this is impossible in the ideal world, where the sender can only call an ideal OT functionality.

To address the issue above, we need a way to prevent the sender from creating a token that can adaptively decide what string it will output. Thinking about it in a different way, we want the sender to "prove knowledge" of two strings before he sends his token. We can accomplish this by adding the following preamble to the protocol above:

1. Receiver chooses a pseudo-random function (PRF) $f_\gamma : \{0,1\}^{5k} \to \{0,1\}^k$, and then sends a token that on input $x \in \{0,1\}^{5k}$, outputs $f_\gamma(x)$.
2. Sender picks two strings $x_0, x_1 \in \{0,1\}^{5k}$ at random, and feeds them (one-at-a-time) to the token it received, and obtains y_0 and y_1. The sender sends (y_0, y_1) to the receiver.
3. Sender and receiver execute the original protocol above with x_0 and x_1 in place of s_0 and s_1. The receiver checks to see if the string w that it obtains from the sender's token satisfies $f_\gamma(w) = y_b$, and aborts if not.

The crucial feature of the protocol above is that a dishonest sender is effectively committed to two values x_0 and x_1 after the second step (and in fact the simulator can use the PRF token to extract these values), such that later on it must output x_b on input b, or abort.

Note that a dishonest receiver may learn k bits of useful information about x_0 and x_1 each from its token, but this can be easily eliminated later using the Leftover Hash Lemma (or any strong extractor).

Preventing correlated aborts. A final significant subtle obstacle remains, however. A dishonest sender can still send a token that causes an abort to be correlated with the receiver's input, *e.g.* it could choose whether or not to abort based on the inputs chosen by the receiver (see full version for a discussion of why this is a problem).

To prevent a dishonest sender from correlating the probability of abort with the receiver's choice, the input b of the receiver is additively shared into bits $b_1, \ldots, b_k$ such that $b_1 + b_2 + \cdots + b_k = b$. The sender, on the other hand, chooses strings $z_1, \ldots, z_k$ and r uniformly at random from $\{0,1\}^{5k}$. Then the sender and receiver invoke k parallel copies of the above protocol (which we call the *Quasi-OT* protocol), where for the ith execution, the sender's inputs are $(z_i, z_i + r)$, and the receiver's input is b_i. Note that at the end of the protocol, the receiver either holds $\sum z_i$ if $b = 0$, or $r + \sum z_i$ if $b = 1$.

Intuitively speaking, this reduction (variants of which were previously used by, e.g. [34,35]) forces the dishonest sender to make one of two bad choices: If each token that it sends aborts too often, then with overwhelming probability at least one token will abort and therefore the entire protocol will abort. On the other hand, if few of the sender's tokens abort, then the simulator will be able to perfectly simulate the probability of abort, since the bits b_i are $(k-1)$-wise independent (and therefore all but one of the Quasi-OT protocols can be perfectly simulated from the receiver's perspective). We make the receiver commit to its bits b_i using a statistically hiding commitment scheme (which can be constructed from one-way functions [42]) to make this probabilistic argument go through.

This completes the intuition behind our protocol. The result of this section is summarized by the following theorem, whose proof appears in full version.

Theorem 3 (Interactive UC-secure computation using stateless tokens). *Let f be a (possibly reactive) polynomial-time computable functionality. Then, assuming one-way functions exist, there exists a computationally UC-secure interactive protocol which realizes f in the $\mathcal{F}_{wrap}^{stateless}$-hybrid model. Furthermore, the protocol only makes a black-box use of the one-way function.*

Oblivious Reactive Functionalities in the Non-Interactive Setting. In the full version, we generalize our study of non-interactive secure computation to the case of reactive functionalities. Roughly speaking, reactive functionalities are the ones for which in the ideal world, the parties might invoke the ideal trusted party multiple times and this trusted party might possibly keep state between

different invocations. For the interactive setting (i.e. when the parties are allowed multiple rounds of interaction in the $\mathcal{F}_{wrap}$-hybrid models) there are standard techniques using which, given protocol for non-reactive functionality, protocol for securely realizing reactive functionality can be constructed. However, these techniques fail in the non-interactive setting. In the full version, we study what class of reactive functionalities can be securely realized in the non-interactive setting for the case of stateless as well as stateful hardware token.

Acknowledgements. We thank Jürg Wullschleger for pointing out the relevance of [27] and for other helpful comments. We thank Guy Rothblum for useful discussions.

References

1. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious rams. J. ACM 43(3), 431–473 (1996)
2. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993)
3. Brands, S.: Untraceable off-line cash in wallets with observers (extended abstract). In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 302–318. Springer, Heidelberg (1994)
4. Cramer, R., Pedersen, T.P.: Improved privacy in wallets with observers (extended abstract). In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 329–343. Springer, Heidelberg (1994)
5. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)
6. Gennaro, R., Lysyanskaya, A., Malkin, T., Micali, S., Rabin, T.: Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 258–277. Springer, Heidelberg (2004)
7. Hofheinz, D., Müller-quade, J., Unruh, D.: Universally composable zero-knowledge arguments and commitments from signature cards. In: Proc. of the 5th Central European Conference on Cryptology MoraviaCrypt 2005, Mathematical Publications (2005)
8. Moran, T., Naor, M.: Basing cryptographic protocols on tamper-evident seals. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 285–297. Springer, Heidelberg (2005)
9. Katz, J.: Universally composable multi-party computation using tamper-proof hardware. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 115–128. Springer, Heidelberg (2007)
10. Chandran, N., Goyal, V., Sahai, A.: New constructions for UC secure computation using tamper-proof hardware. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 545–562. Springer, Heidelberg (2008)
11. Moran, T., Segev, G.: David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 527–544. Springer, Heidelberg (2008)

12. Damgård, I., Nielsen, J.B., Wichs, D.: Isolated proofs of knowledge and isolated zero knowledge. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 509–526. Springer, Heidelberg (2008)
13. Goldwasser, S., Kalai, Y.T., Rothblum, G.: One-time programs. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 39–56. Springer, Heidelberg (2008)
14. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS, pp. 136–145 (2001)
15. Chor, B., Kushilevitz, E.: A zero-one law for boolean privacy. SIAM J. Discrete Math. 4(1), 36–47 (1991)
16. Rabin, M.O.: How to exchange secrets with oblivious transfer (1981)
17. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Commun. ACM 28(6), 637–647 (1985)
18. Kilian, J.: Founding cryptography on oblivious transfer. In: STOC, pp. 20–31 (1988)
19. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer - efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (2008)
20. Hazay, C., Lindell, Y.: Constructions of truly practical secure protocols using standardsmartcards. In: Ning, P., Syverson, P.F., Jha, S. (eds.) ACM Conference on Computer and Communications Security, pp. 491–500. ACM, New York (2008)
21. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC, pp. 218–229 (1987)
22. Goyal, V., Sahai, A.: Resettably secure computation. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 54–71. Springer, Heidelberg (2009)
23. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (2001)
24. Ishai, Y., Prabhakaran, M., Sahai, A., Wagner, D.: Private circuits ii: Keeping secrets in tamperable circuits. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 308–327. Springer, Heidelberg (2006)
25. Hofheinz, D., Müller-Quade, J., Unruh, D.: Universally composable zero-knowledge arguments and commitments from signature cards. In: 5th Central European Conference on Cryptology (2005), http://homepages.cwi.nl/~hofheinz/card.pdf
26. Kolesnikov, V.: Truly efficient string oblivious transfer using resettable tamper-proof tokens. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978. Springer, Heidelberg (2010)
27. Buhrman, H., Christandl, M., Unger, F., Wehner, S., Winter, A.: Implications of superstrong nonlocality for cryptography. Proceedings of the Royal Society A 462(2071), 1919–1932
28. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS, pp. 136–145 (2001)
29. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: STOC, pp. 1–10 (1988)
30. Kushilevitz, E.: Privacy and communication complexity. SIAM J. Discrete Math. 5(2), 273–284 (1992)
31. Goldreich, O., Vainish, R.: How to solve any protocol problem - an efficiency improvement. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 73–86. Springer, Heidelberg (1988)

32. Galil, Z., Haber, S., Yung, M.: Cryptographic computation: Secure faut-tolerant protocols and the public-key model. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 135–155. Springer, Heidelberg (1988)
33. Goldreich, O.: Foundations of Cryptography: Basic Applications. Cambridge University Press, Cambridge (2004)
34. Kilian, J.: Uses of Randomness in Algorithms and Protocols. MIT Press, Cambridge (1990)
35. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007)
36. Beaver, D.: Correlated pseudorandomness and the complexity of private computations. In: STOC, pp. 479–488 (1996)
37. Yao, A.: How to generate and share secrets. In: FOCS, pp. 162–167 (1986)
38. Ishai, Y., Kushilevitz, E.: Perfect constant-round secure computation via perfect randomizing polynomials. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 244–256. Springer, Heidelberg (2002)
39. Feige, U., Kilian, J., Naor, M.: A minimal model for secure computation (extended abstract). In: STOC, pp. 554–563 (1994)
40. Brassard, G., Crépeau, C., Santha, M.: Oblivious transfers and intersecting codes. IEEE Transactions on Information Theory 42(6), 1769–1780 (1996)
41. Beaver, D., Goldwasser, S.: Multiparty computation with faulty majority (extended announcement). In: FOCS, pp. 468–473. IEEE, Los Alamitos (1989)
42. Haitner, I., Reingold, O.: Statistically-hiding commitment from any one-way function. In: STOC, pp. 1–10 (2007)
43. Anderson, W.E.: On the secure obfuscation of deterministic nite automata. Cryptology ePrint Archive, Report 2008/184 (2008)
44. Canetti, R., Goldreich, O., Goldwasser, S., Micali, S.: Resettable zero-knowledge (extended abstract). In: STOC, pp. 235–244 (2000)