Founding Cryptography on Tamper-Proof Hardware Tokens

Vipul Goyal
MSR India
vipul@microsoft.com

Yuval Ishai
Technion and UCLA
yuvali@cs.technion.ac.il

Amit Sahai
UCLA
sahai@cs.ucla.edu

Ramarathnam Venkatesan
MSR India and Redmond
venkie@microsoft.com

Akshay Wadia
UCLA
awadia@ucla.edu

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 goal 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 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 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*. 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.
- **Non-interactive secure computation from stateless tokens.** We consider the problem of designing non-interactive secure computation from stateless tokens for *stateless oblivious reactive functionalities*, i.e., reactive functionalities which allow unlimited queries from the receiver (these are the only functionalities one can hope to realize non-interactively with *stateless* tokens). By building on recent techniques from resettably secure computation, we give a general positive result for stateless oblivious reactive functionalities under standard cryptographic assumption. This result generalizes the notion of (unlimited-use) obfuscation by providing security against a malicious sender, and also provides the first general feasibility result for program obfuscation using *stateless* tokens.

1 Introduction

A number of works (e.g. [GO96, CP92, Bra93, CP93, ISW03, GLM+04, HMqU05b, MN05, Kat07, CGS08, MS08, DNW08, GKR08]) 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 [Kat07] examined the problem of achieving UC-secure [Can01a] two party computation using tamper-proof hardware tokens. A number of follow-up papers [CGS08, MS08, DNW08] have further investigated this problem. In another separate (but related) work, Goldwasser et al. [GKR08] 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 [GO96] using the framework of oblivious RAM.

This 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 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 (standalone 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 [Kat07], 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 [CK91]. While in the semi-honest model this question is easy to settle by relying on unconditional protocols based on oblivious transfer (OT) [Rab81, EGL85, Kil88, IPS08], this question appears to be much more challenging when both parties can be malicious. (See Section 4.1 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 work on secure two-party computation with hardware tokens [Kat07, CGS08, MS08] relied either on specific number theoretic assumptions (DDH) or the existence of oblivious transfer protocols in the plain model.

  A related question is: **is it possible to achieve unconditionally secure one-time programs for all polynomial-time computable functions?** The previous work of [GKR08] required the existence one-way functions in order to construct one-time programs.

- **Is it possible to achieve non-interactive secure two-party computation with hardware tokens?** Again, this problem is open even for stand-alone security, and impossible in the plain model. The work of Goldwasser et al. [GKR08] constructs (non-interactive) one-time programs using hardware tokens, however in their model,

[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.

the sender is semi-honest[2]. Thus an equivalent question is: **is it possible to achieve one-time programs tolerating a malicious sender?** We note that [GKR08] 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 [GMW87] paradigm 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).

- **What are the simplest kinds of tamper-proof hardware tokens needed to achieve UC-secure two-party computation?** For example, Goldwasser et al. [GKR08] introduce a very simple kind of tamper-proof hardware token that they call an OTM (one-time memory) token.[3] An OTM token stores two strings s_0 and s_1, and takes a single bit b as input, then outputs s_b and stops working (or self-destructs). In other words, an OTM token implements the one-out-of-two string OT functionality. (For this reason, we will call such tokens OT tokens.) An even simpler version of such a token would be one where the strings s_0 and s_1 are replaced with single bits, corresponding to the one-out-of-two bit OT functionality (we would call such a token a bit OT token). **Is it possible that we can get UC-secure two-party computation using only bit OT tokens?** We note that previous works on UC-secure two-party computation with hardware tokens [Kat07, CGS08, MS08] all make use of more complicated hardware tokens.

- **Which notions of software obfuscation be realized using 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 [GO96] (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?** To illustrate the motivation for the latter question, consider the goal of obfuscating a poker-playing program. The receiver of the obfuscator 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).

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 tamper-proof hardware tokens. We stress that in all results below, the code of our hardware tokens is **independent** of all parties' inputs[4].

- **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 simple stateful 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 strengthens previous feasibility results for one-time programs both by providing *unconditional* security and by offering general protection against *malicious senders* and by using only "bit-OTM" tokens.

[2] In the model of [GKR08], the sender is allowed to arbitrarily specify the functionality of the one-time program, and the receiver knows nothing about this functionality except an upper bound on its circuit size. (Thus, the issue of dishonest senders does not arise in their model.) In this work, by a one-time program tolerating a malicious sender, we mean that the receiver knows some partial specification of the functionality of the one-time program – modeled in the usual paradigm of secure computation.

[3] Goldwasser et al. [GKR08] additionally show that their constructions using OTM tokens are *leakage resilient* in a very strong sense; a feature our constructions using such tokens inherit as well.

[4] Thus, the tokens could theoretically be "mass-produced" before being used in any particular protocol with any particular inputs.

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 4.1 and 4.2 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 5 for details.

- **Non-interactive secure computation from stateless tokens.** We consider the problem of designing non-interactive secure computation from stateless tokens for *stateless oblivious reactive functionalities*, i.e., reactive functionalities which allow unlimited queries from the receiver (these are the only functionalities one can hope to realize non-interactively with *stateless* tokens). By building on recent techniques from resettably secure computation [GS09], we give a general positive result for stateless oblivious reactive functionalities under standard cryptographic assumption. We are not able to optimize the cryptographic assumptions because of inherent usage of non-black-box simulation and connections to other such problems. This result generalizes the notion of (unlimited-use) obfuscation by providing security against a malicious sender, and also provides the first general feasibility result for program obfuscation using *stateless* tokens. As a side result, we also propose constructions of non-interactive secure computation for general reactive functionalities with *stateful* tokens. See Section 6 for details.

We stress that in contrast to some previous results along this line (most notably, [GO96, GKR08]), 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. We leave a more detailed discussion of efficiency issues to the final version of this paper.

The first two results above are obtained by utilizing previous works [Kil88, IPS08] showing how to achieve secure computation based on OT, and thus a main ingredient in our constructions is showing how to securely implement OT using hardware tokens[5]. Note that in the case of non-interactive secure computation, additional tools are needed since the protocols of [Kil88, IPS08] are (necessarily) interactive.

Related Work. The use of tamper-proof hardware tokens for cryptographic purposes was first explored by Goldreich and Ostrovsky [GO96] in the context of software protection (one-time programs [GKR08] is a relaxation of this goal, generally called program obfuscation [BGI$^+$01]), and by Chaum, Pederson, Brands, and Cramer [CP92, Bra93, CP93] in the context of e-cash. Ishai, Sahai, and Wagner [ISW03] and Ishai, Sahai, Prabhakaran and Wagner [IPSW06] consider the question of how to construct tamper-proof hardware tokens when the hardware itself does not guarantee complete protection against tampering. Gennaro, Lysyanskaya, Malkin, Micali, and Rabin [GLM$^+$04] 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 [MN05] 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 [HMQU05a] consider a model similar to [Kat07] 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 [Kat07], where users can create and exchange tamper-proof hardware tokens, in the context of implementing UC-secure protocols. [Kat07] shows how to implement UC-secure two-party computation using stateful tokens, under the DDH assumption. Chandran, Goyal, Sahai [CGS08] implement UC-secure two-party computation using stateless tokens,

[5]Note that for out first two results, the fact that we rely on OT immediately gives us the feature that the code of our hardware tokens can be made independent of all parties' inputs. This is simply because OT with random sender strings is (non-interactively) equivalent to OT with chosen sender strings [BG89].

under the assumption that oblivious transfer protocols exist in the plain model. Aside from just considering stateless tokens, [CGS08] also introduce a variant of the model of [Kat07] 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 [MS08] also implement UC-secure two-party computation under the same assumption as [CGS08], but using stateful tokens, and only requiring tokens to be passed in one direction. Damgård, Nielsen, and Wichs [DNW08] 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 [HL08] propose construction of truly efficient protocols for various problems of interest using trusted smartcards. Goldwasser, Kalai, and Rothblum [GKR08] introduced the notion of one-time programs, and showed how to achieve 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, motivated by quantum physics, Buhrman et al. [BCU$^+$] consider the application of *non-local boxes* to cryptography. Using non-local boxes, Buhrman et al. show an unconditional construction for secure two-party computation 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, we observe that 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, we can give a simple protocol (omitted here) that shows how to (interactively) build a trusted non-local-box using OTM tokens, giving an alternative to our "warm-up" construction (see Section 4.1) of unconditional secure computation from hardware tokens. However, this alternative construction does not seem useful as a building block to our first main result: *non-interactive* unconditional secure computation.

2 Preliminaries

In this section we briefly discuss some of the underlying definitions and concepts, and fix notation.

Definition 1. *(Computational Indistinguishability) Two distribution ensembles $X := \{ X_n \}_{n\in\mathbf{N}}$ and $Y := \{ Y_n \}_{n\in\mathbf{N}}$ are **computationally indistinguishable** (written as $X \sim Y$) if for every probabilistic polynomial-time algorithm D, every positive polynomial $p(\cdot)$, and all sufficiently large n's,*

$$|\Pr[\, D(X_n, 1^n) = 1\,] - \Pr[\, D(Y_n, 1^n) = 1\,]| \leq \frac{1}{p(n)}.$$

The *statistical difference* between two distribution ensembles $X := \{ X_n \}_{n\in\mathbf{N}}$ and $Y := \{ Y_n \}_{n\in\mathbf{N}}$ is defined by

$$\Delta(n) = \frac{1}{2}\sum_{\alpha} |\Pr[\, X_n = \alpha\,] - \Pr[\, Y_n = \alpha\,]|.$$

Ensembles X and Y are *statistically close* if the their statistical difference is negligible in n.

Now we define the next-message function of an interactive TM P:

Definition 2. *(Next Message Function) Let P be an interactive TM. The **next message function for round** i of P is the function $P_i^{x,r}()$, which on input $(m_1, \ldots, m_i, s_{i-1})$, outputs $(\hat{m}_i, s_i)$, where $\hat{m}_i$ is the message output by P on input x and random input r, after receiving $m_1, \ldots, m_i$ in previous rounds, and s_{i-1} and s_i contain auxiliary state information for rounds $i-1$ and i respectively.*

Commitments. We will use the two-round statistically binding commitment scheme com ([Nao91]). To recall, to commit to bit b, the first message in Naor's scheme is a random string r from receiver to sender. Then the sender responds with either $G(s)$ or $G(s) \oplus r$ depending of whether $b = 0$ or $b = 1$, where $G(\cdot)$ is a pseudo-random generator, and s is a randomly chosen seed. To decomit, the sender sends (b, s) to the receiver. We observe that Naor's scheme has the property that the same r can be used for committing to (polynomially) many bits. Thus, once

the receiver's string has been initially exchanged, we essentially have a non-interactive commitment function. For receiver's message r, denote sender's response by $\mathsf{com}_r(b)$, where b is the bit being committed. Abusing terminology, we will call $\mathsf{com}_r(b)$ the commitment to bit b. When the randomness used in the first message is clear from the context, we will drop the subscript r, and will denote the committed string by $\mathsf{com}(b)$. The opening of a commitment α is denoted by $\mathsf{open}(\alpha)$ and consists of the committed bit b and seed s (the randomness r is implicit).

Unconditional One-Time MAC. By (MAC, VF), we will denote an unconditional one-time message authentication scheme, where $MAC_k(m)$ represents tagging message m with key k, and $VF_k(m, \sigma)$ denotes the verification algorithm, which returns 1 if σ is the correct tag of m under key k. An example of such a MAC is as follows: the key k is a pair of κ length strings (a, b) (where κ is the security parameter). The tag of message m with key $k = (a, b)$ is $a \cdot m + b$, where all operations are in $GF(2^\kappa)$.

2.1 The Model

We use the UC-framework of Canetti [Can01b] 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. We begin by defining protocol syntax, and then informally review the UC-framework. For more details, see [Can01b].

Protocol syntax. Following [GMR89] and [Gol01], a protocol is represented as a system of probabilistic interactive Turing machines (ITMs), where each ITM represents the program to be run within a different party. Specifically, the input and output tapes model inputs and outputs that are received from and given to other programs running on the same machine, and the communication tapes model messages sent to and received from the network. Adversarial entities are also modeled as ITMs.

The construction of a protocol in the UC-framework proceeds as follows: first, an *ideal functionality* is defined, which is a "trusted party" that is guaranteed to accurately capture the desired functionality. Then, the process of executing a protocol in the presence of an adversary and in a given computational environment is formalized. This is called the *real-life* model. Finally, an *ideal process* is considered, where the parties only interact with the ideal functionality, and not amongst themselves. Informally, a protocol realizes an ideal functionality if running of the protocol amounts to "emulating" the ideal process for that functionality.

Let $\Pi = (P_1, P_2)$ be a protocol, and $\mathcal{F}$ be the ideal-functionality. We describe the ideal and real world executions.

The real-life model. The real-life model consists of the two parties P_1 and P_2, the environment $\mathcal{Z}$, and the adversary $\mathcal{A}$. Adversary $\mathcal{A}$ can communicate with environment $\mathcal{Z}$ and can corrupt any party. When $\mathcal{A}$ corrupts party P_i, it learns P_i's entire internal state, and takes complete control of P_i's input/output behaviour. The environment $\mathcal{Z}$ sets the parties' initial inputs. Let $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$ be the distribution ensemble that describes the environment's output when protocol Π is run with adversary $\mathcal{A}$.

The ideal process. The ideal process consists of two "dummy parties" $\hat{P}_1$ and $\hat{P}_2$, the ideal functionality $\mathcal{F}$, the environment $\mathcal{Z}$, and the ideal world adversary S, called the simulator. In the ideal world, the uncorrupted dummy parties obtain their inputs from environment $\mathcal{Z}$ and simply hand them over to $\mathcal{F}$. As in the real world, adversary S can corrupt any party. Once it corrupts party $\hat{P}_i$, it learns $\hat{P}_i$'s input, and takes complete control of its input/output behaviour. Let $\mathsf{IDEAL}^{\mathcal{F}}_{S,\mathcal{Z}}$ be the distribution ensemble that describes the environment's output in the ideal process.

Definition 3. *(Realizing an Ideal Functionality) Let $n \in \mathbf{N}$. Let $\mathcal{F}$ be an ideal functionality, and Π be a protocol. We say Π **realizes** $\mathcal{F}$ if for any real-world adversary $\mathcal{A}$, there exists an ideal process adversary S such that for every*

environment $\mathcal{Z}$,

$$\mathsf{IDEAL}^{\mathcal{F}}_{S,\mathcal{Z}} \sim \mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}.$$

3 Modeling Tamper-Proof Hardware

Our model for tamper-proof hardware is similar to that of Katz ([Kat07]). However, as we consider both stateful and stateless tokens, we define different ideal functionalities for the two. Here, we formally define the ideal functionalities $\mathcal{F}_{wrap}^{single}$, for single use stateful tokens, and $\mathcal{F}_{wrap}^{stateless}$ for stateless tokens.

The functionality $\mathcal{F}_{wrap}^{single}$ is used to model hardware tokens that can be executed only once. Thus, the only state these tokens keep is a flag which indicates whether the token has been run or not. To model this behaviour, $\mathcal{F}_{wrap}^{single}$ deletes the token from its memory after it has been run once. The formal description of $\mathcal{F}_{wrap}^{single}$ is presented in Figure 1.

Functionality $\mathcal{F}_{wrap}^{single}$

$\mathcal{F}_{wrap}^{single}$ is parameterized by a polynomial $p(\cdot)$ and an implicit security parameter κ.

Create Upon receiving $(\mathsf{create}, \mathsf{sid}, P_i, P_j, \mathsf{mid}, M)$ from P_i, where M is a Turing machine and mid is machine id, do:

1. Send $(\mathsf{create}, \mathsf{sid}, P_i, P_j, \mathsf{mid})$ to P_j.
2. Store $(P_i, P_j, \mathsf{mid}, M)$.

Execute Upon receiving $(\mathsf{run}, \mathsf{sid}, P_i, \mathsf{mid}, \mathsf{msg})$ from P_j, find the unique stored tuple $(P_i, P_j, \mathsf{mid}, M)$ (if no such tuple exists, do nothing). Choose random $r \leftarrow \{0,1\}^{p(k)}$. Run $M(\mathsf{msg}; r)$ for at most $p(k)$ steps, and let out be the response ($\mathsf{out} = \perp$ if M does not halt in $p(k)$ steps). Send $(\mathsf{sid}, P_i, \mathsf{mid}, \mathsf{out})$ to P_j, and delete $(P_i, P_j, \mathsf{mid}, M)$.

Figure 1: Ideal functionality for single-use stateful tokens

Next, we define the functionality $\mathcal{F}_{wrap}^{stateless}$ that models stateless tokens. The idea of $\mathcal{F}_{wrap}^{stateless}$, described in Figure 2, is to model the following real-world functionality: party P_i sends a stateless token M to party P_j. Since the token is stateless, P_j can run M multiple times on inputs of its choice. Thus, $\mathcal{F}_{wrap}^{stateless}$ saves the description of the Turing machines it gets from a party in create messages, and lets the other party run them multiple times. Each machine is uniquely identified by a machine identifier mid.

One can also consider a variant of $\mathcal{F}_{wrap}^{stateless}$ which allows a malicious sender to generate *stateful* tokens. Our protocols which use stateless tokens are secure in this more adversarial setting as well. (This is automatically the case in all protocols for which an honest receiver makes only a single use of each token.)

We are interested in non-interactive protocols in which the communication involves a single batch of tokens sent from a "sender" to a "receiver". (One could also allow the sender to send a message to a receiver; however, from a feasibility point of view this could also be done in the simpler model in which only tokens are sent.)

Definition 4. *(Non-Interactive Protocols in the Tamper-Proof Hardware Model) A two-party protocol $\Pi = (P_1, P_2)$ is **non-interactive** if the only messages sent by P_1 are create messages to $\mathcal{F}_{wrap}^{single}$ (or $\mathcal{F}_{wrap}^{stateless}$) and the only messages sent by P_2 are run messages to $\mathcal{F}_{wrap}^{single}$ (or $\mathcal{F}_{wrap}^{stateless}$).*

Functionality $\mathcal{F}_{wrap}^{stateless}$

$\mathcal{F}_{wrap}^{stateless}$ is parameterized by a polynomial $p(\cdot)$ and an implicit security parameter κ.

Create Upon receiving $(\mathsf{create}, \mathsf{sid}, P_i, P_j, \mathsf{mid}, M)$ from P_i, where M is a Turing machine, do:

1. Send $(\mathsf{create}, \mathsf{sid}, P_i, P_j, \mathsf{mid})$ to P_j.
2. Store $(P_i, P_j, \mathsf{mid}, M)$.

Execute Upon receiving $(\mathsf{run}, \mathsf{sid}, P_i, \mathsf{mid}, \mathsf{msg})$ from P_j, find the unique stored tuple $(P_i, P_j, \mathsf{mid}, M)$. If no such tuple exists, do nothing. Run $M(\mathsf{msg})$ for at most $p(k)$ steps, and let out be the response ($\mathsf{out} = \perp$ if M does not halt in $p(k)$ steps). Send $(\mathsf{sid}, P_i, \mathsf{mid}, \mathsf{out})$ to P_j.

Figure 2: Ideal functionality for stateless tokens

3.1 One-Time Programs

Informally, a *one-time program* (OTP) [GKR08] for a function f lets a party evaluate f on only one input chosen by that party at run time. The intuitive security goal is that no efficient adversary, after evaluating the one-time program on x, can learn anything about $f(y)$ for some $y \neq x$, other than what can be inferred from $f(x)$. In our constructions, OTPs will be used within other protocols, thus it would be convenient for us to view them as two-party non-interactive protocols in the hardware token model, which are secure against malicious receivers. We thus view OTP as implementing a two-party functionality $f(\cdot, \cdot)$ (where the description of f is known to both parties), where the first (secret) input is fixed by the sender during construction.

Figure 3 defines the ideal functionality for a one-time program for function $f(\cdot, \cdot)$.

Functionality $\mathcal{F}_f^{OTP}$

Create Upon receiving $(\mathsf{create}, \mathsf{sid}, P_i, P_j, x)$ from P_i, where x is a string, do:

1. Send $(\mathsf{create}, \mathsf{sid}, P_i, P_j)$ to P_j.
2. Store (P_i, P_j, x).

Execute Upon receiving $(\mathsf{run}, \mathsf{sid}, P_i, y)$ from P_j, find the stored tuple (P_i, P_j, x) (if no such tuple exists, do nothing). Send $f(x, y)$ to P_j and delete tuple (P_i, P_j, x).

Figure 3: Ideal functionality for One-time Program for function $f(\cdot, \cdot)$.

Now we define OTPs in the $\mathcal{F}_{wrap}^{single}$-hybrid model.

Definition 5. *(One-Time Program for $f(\cdot, \cdot)$) A one-time program for function $f(\cdot, \cdot)$ is a two-party non-interactive protocol $\Pi = (P_1, P_2)$ in the $\mathcal{F}_{wrap}^{single}$-hybrid model, such that for every probabilistic polynomial time adversary $\mathcal{A}$ corrupting P_2, there exists a probabilistic polynomial time ideal-world adversary S called the simulator, such that for every environment $\mathcal{Z}$,*

$$\mathsf{IDEAL}_{S,\mathcal{Z}}^{\mathcal{F}_f^{OTP}} \sim \mathsf{REAL}_{\Pi,\mathcal{A},Z}.$$

One way of implementing OTPs using hardware tokens is to construct stateful hardware tokens that contain the entire code of the function $f(x, \cdot)$. However, like in [GKR08] we would like to use only the simplest kind of hardware tokens in our protocols. To this end, we focus on using one-time-memory (OTM) tokens only. OTM tokens realize the ideal functionality defined in Figure 5.

3.2 Flavors of OT

The ideal OT functionality required by unconditionally secure protocols in the OT-hybrid model [Kil88, IPS08] is denoted by $\mathcal{F}^{\mathrm{OT}}$ and is formally defined in Figure 4.

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

On input $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$ from party P_i, send $(P_i, P_j, \mathsf{sid}, \mathsf{id})$ to P_j and store the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$.
On receiving $(P_i, \mathsf{sid}, \mathsf{id}, c)$ from party P_j, if a tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$ exists, return $(P_i, \mathsf{sid}, \mathsf{id}, s_c)$ to P_j, **send an acknowledgment** $(P_j, \mathsf{sid}, \mathsf{id})$ **to** P_i, and delete the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$. Else, do nothing.

Figure 4: Ideal functionality for OT

Following [GKR08], we refer to simple hardware tokens that implement a single OT call as *OTM (one-time-memory)* tokens. We give the formal definition of OTM tokens here, and discuss these tokens in detail in Section 4.1. OTM tokens are defined in Figure 5.

Functionality OTM

On input $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$ from party P_i, send $(P_i, P_j, \mathsf{sid}, \mathsf{id})$ to P_j and store the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$.
On receiving $(P_i, \mathsf{sid}, \mathsf{id}, c)$ from party P_j, if a tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$ exists, return $(P_i, \mathsf{sid}, \mathsf{id}, s_c)$ to P_j and delete the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$. Else, do nothing.

Figure 5: Ideal functionality for OTM tokens

We also define a parallel version of the OTM functionality, denoted pOTM, which allows the receiver to make several parallel OTM choice. Note that unlike a direct implementation using separate OTM tokens, this functionality requires the receiver to make all its choices at once, and does not allow a malicious receiver to determine its choice bits in an adaptive fashion. The ideal functionality for pOTM is defined in Figure 6.

Functionality pOTM

pOTM is parameterized by an integer k.
On input $(P_i, P_j, \mathsf{sid}, \mathsf{id}, ((s_0^1, s_1^1), \ldots, (s_0^k, s_1^k)))$ from party P_i, send $(P_i, P_j, \mathsf{sid}, \mathsf{id})$ to P_j and store the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, ((s_0^1, s_1^1), \ldots, (s_0^k, s_1^k)))$.
On receiving $(P_i, \mathsf{sid}, \mathsf{id}, (c^1, \ldots, c^k))$ from party P_j, if a tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, ((s_0^1, s_1^1), \ldots, (s_0^k, s_1^k)))$ exists, return $(P_i, \mathsf{sid}, \mathsf{id}, (s_{c^1}^1, \ldots, s_{c^k}^k))$ to P_j and delete the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, ((s_0^1, s_1^1), \ldots, (s_0^k, s_1^k)))$. Else, do nothing.

Figure 6: Ideal functionality for parallel-OTM.

Finally, the ExtOTM functionality differs from OTM in that it takes an additional input r from the sender, and delivers this input to the receiver together with its chosen string s_c. This functionality is described in Figure 7. It is easy to see that a protocol for the ExtOTM functionalities allows us to realize the $\mathcal{F}^{\mathrm{OT}}$functionality as required by [Kil88, IPS08].

Functionality ExtOTM

On input $(P_i, P_j, \mathsf{sid}, \mathsf{id}, ((s_0, s_1), r))$ from party P_i, send $(P_i, P_j, \mathsf{sid}, \mathsf{id})$ to P_j and store the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, ((s_0, s_1), r))$.
On receiving $(P_i, \mathsf{sid}, \mathsf{id}, c)$ from party P_j, if a tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, ((s_0, s_1), r))$ exists, return $(P_i, \mathsf{sid}, \mathsf{id}, s_c, r)$ to P_j and delete the tuple $(P_i, P_j, id, ((s_0, s_1), r))$. Else, do nothing.

Figure 7: Ideal functionality for ExtOTM

4 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 4.1 we present as a "warmup" an interactive protocol for arbitrary functionalities, which requires the parties to engage in multiple rounds of interaction. This section will introduce some useful building blocks that are used in the next subsection. This gives an unconditional version of previous protocols for UC-secure computation based on hardware tokens [Kat07, CGS08, MS08], which all relied on computational assumptions.[6]

In Subsection 4.2 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 [GKR08] by providing *unconditional* security (in fact, UC-security) and by offering general protection against *malicious senders*.

4.1 Warmup: The Interactive Setting

Unconditionally secure two-party computation is impossible to realize for most nontrivial functionalities, even with semi-honest parties [BOGW88, Kus92]. However, if the parties are given oracle access to a simple ideal functionality such as Oblivious Transfer (OT) [Rab81, EGL85], then it becomes possible not only to obtain unconditionally secure computation with semi-honest parties [GV87, GHY87, Gol04], but also unconditional *UC-security* against *malicious* parties [Kil88, IPS08]. 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 [GKR08]. Following the terminology of [GKR08], we refer to such tokens as OTM (one-time-memory) tokens. OTM tokens were formally defined in Section 3.2.

An appealing feature of OTM tokens is their simplicity, which can also lead to better resistance against side-channel attacks (see [GKR08] for discussion). This simplicity feature served as the main motivation for using OTM

[6]The work of [MS08] realizes an unconditionally UC-secure *commitment* from stateful tokens. This does not directly yield protocols for secure computation without additional computational assumptions.

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 [GKR08], 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.[7]

Attacks on simple solution ideas. A natural way to handle the above type of attacks is by having the sender pick a secret key r for each OTM and append this key to both strings (s_0, s_1) it stores in the OTM. Then, to emulate a single call to an OT oracle, the sender sends such an OTM token to the receiver, and expects the receiver to send back r before proceeding with the protocol. This approach can indeed be used to protect a semi-honest sender from out-of-order token invocations by a malicious receiver. However, it completely exposes the receiver to an attack by a malicious sender who puts a pair distinct strings (r_1, r_2) in the same OTM. Note that in the context of one-time programs where the sender is *semi-honest*, a variant of this approach [GKR08] does suffice to solve the problem. However, in the context of *malicious* senders that we consider here, the approach of [GKR08] does not suffice.

A more subtle attack, known as *selective abort*, arises if one tries to fix the problem above in naïve ways such as asking the sender to send a randomized hash of r, and then instructing the receiver to abort if the sender's message is different from the hash of the r that it received from the OTM. This would allow a malicious sender to cause the receiver to abort based on its private choice bit, which is not allowed by the ideal OT functionality (and can lead to real attacks on secrecy).

Extending the OTM functionality. To fix the above idea, 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 (see Figure 7). 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 acknowledgment to the sender. This flavour of the OT functionality, which we denote by $\mathcal{F}^{\mathrm{OT}}$(see Figure 4), 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 [Kil88, IPS08]. We refer the reader to Appendix **??** for a UC-style definition of the OTM, ExtOTM, and $\mathcal{F}^{\mathrm{OT}}$functionalities.

Realizing ExtOTM using general[8] 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.

[7] To illustrate the effect of such attacks, consider the following functionality f. The functionality takes from the sender a pair of k-bit strings (x_1, x_2) and from the receiver a k-bit string y. If $y = x_1$, the functionality delivers (x_1, x_2) to the receiver, otherwise it delivers only x_1. Now, let Π be any OT-based protocol for f which consists of only one round of OTs from the sender to the receiver, where the receiver's OT choices are its input bits. (A simple protocol of this type with perfect security against a malicious receiver and a semi-honest sender is given in [Kil88] for any f in NC^1.) Modify Π into a new protocol Π' in which the sender, following the round of OT calls, reveals x_1 to the receiver. The protocol Π' is still perfectly secure against a malicious receiver in the OT-hybrid model, but if OT calls are realized by sending OTM tokens, the new protocol allows the receiver to always learn x_2 by first observing x_1 and then invoking the OTM tokens with $y = x_1$.

[8] 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.

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 this subsection, we present 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 start with a detailed discussion of the intuition of the protocol and its security proof.

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. [Kil90, LP07]). 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 $(q_0^i \circ r^i, q_1^i \circ r^i)$. 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 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 $\hat{q}^i = q_{\hat{c}_i}^i$ (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. We describe the protocol formally now.

Protocol Π_{ExtOTM}.

- ***Input:*** *P_1 gets as input k-bit strings (s_0, s_1, r), and P_2 gets as input a bit c.*

- ***Specified Output:*** *P_2 should receive (s_c, r).*

  1. *P_1 chooses $4k$ distinct strings of length k, $((s_0^1, s_1^1), \ldots, (s_0^{2k}, s_1^{2k}))$. Now P_1 chooses another $2k-1$ strings of length k, $(\rho^1, \ldots, \rho^{2k-1})$, and sets ρ^{2k} such that $\bigoplus_{i=1}^{2k} \rho^i = r$. Then,*
     (a) *For $1 \leq i \leq 2k$, send $(\mathsf{create}, \mathsf{sid}, P_1, P_2, \mathsf{mid}, M^i)$ to $\mathcal{F}_{wrap}^{single}$, where M^i implements the following functionality: on input bit c^i, output $(s_{c^i}^i, \rho^i)$.*
     (b) *P_1 constructs and sends an unconditional OTP for the following functionality F_1:*
        . Receive input $(\hat{s}^1, \ldots, \hat{s}^{2k})$. For $1 \leq i \leq 2k$, let $\hat{c}^i \in \{0,1\}$ be such that $\hat{s}^i = s_{\hat{c}^i}^i$. If no such $\hat{c}^i$s exist, output $\perp$. Else output $s_{\hat{c}^1 \oplus \ldots \oplus \hat{c}^{2k}}$.
     (c) *P_1 constructs and sends an unconditional OTP for the following functionality F_2:*
        . On input $\rho \in \{0,1\}^k$, check if $\rho = \bigoplus_{i=1}^{2k} \rho^i$. If not, output $\perp$. Else, output $\rho^1 \circ \ldots \circ \rho^{2k}$.
  2. *P_2 picks random bits $c^1, \ldots, c^{2k-1}$, and sets c^{2k} such that $c = \bigoplus_{i=1}^{2k} c^i$. For $1 \leq i \leq 2k$, P_2 runs M^i with input c^i and obtains $(\hat{s}^i, \rho^i)$. It runs the OTP for F_1 on input $(\hat{s}^1, \ldots, \hat{s}^{2k})$ and obtains string s. If the OTP aborts, then P_2 sets $s = 0^k$. Next, P_2 runs OTP for F_2 on input $\bigoplus_{i=1}^{2k} \rho^i$, and obtains string ρ. If $\rho \neq \rho^1 \circ \ldots \circ \rho^{2k}$, P_2 aborts. Else, it outputs $(s, \bigoplus_{i=1}^{2k} \rho^i)$.*

Claim 6. *Protocol Π_{ExtOTM} realizes* ExtOTM *with statistical UC-security in the* OTM*-hybrid model.*

Proof First consider the case of malicious sender. Let $\mathcal{A}$ be an adversary controlling P_1, and let $\mathcal{Z}$ be any environment. We define the simulator S_1^{ExtOTM} as follows:

Simulator S_1^{ExtOTM}

1. Receive input (s_0, s_1, r) from $\mathcal{Z}$ for $\mathcal{A}$. Start internal simulation of $\mathcal{A}$ with the given input.
2. Receive TMs $M^1, \ldots, M^{2k}$, and OTPs for functionalities F_1 and F_2 from $\mathcal{A}$.
3. For $1 \leq i \leq 2k$, run M^i with input 0 to obtain (s_0^i, ρ_0^i). Now rewind M^i and run it with input 1 to obtain (s_1^i, ρ_1^i). If for every $1 \leq i \leq 2k$, $\rho_0^i \neq \rho_1^i$, abort.
4. Choose $2k-1$ random bits $c^1, \ldots, c^{2k-1}$, and set c^{2k} such that $\bigoplus_{i=1}^{2k} c^i = 1$. Then,
   (a) If for any index i, $s_{c^i}^i = \perp$, abort. Run the OTP corresponding to F_1 with input $(s_{c^1}^1, \ldots, s_{c^{2k}}^{2k})$ to obtain s_1. Let j be the index such that $\rho_0^j = \rho_1^j$. Run F_1 again with input $(s_{c^1}^1, \ldots, s_{c^{j-1}}^{j-1}, s_{c^j \oplus 1}^j, s_{c^{j+1}}^{j+1}, \ldots, s_{c^{2k}}^{2k})$ to obtain s_0. If the OTP aborts in either case, set that string to the default value, 0^k.
   (b) Set $r = \bigoplus_{i=1}^{2k} \rho_{c^i}^i$. Run OTP for F_2 with input r and obtain string ρ. If $\rho \neq \rho_{c^1}^1 \circ \ldots \circ \rho_{c^2 k}^{2k}$, abort. Else, send (s_0, s_1, r) to ExtOTM.

We proceed to show that $\mathsf{REAL}_{ExtOTM,\mathcal{A},\mathcal{Z}}$ and $\mathsf{IDEAL}^{\mathrm{ExtOTM}}_{S_1^{ExtOTM},\mathcal{Z}}$ are statistically close. Consider the following hybrids:

Hybrid $\mathcal{H}_0$: In this experiment, $\mathcal{Z}$ interacts with simulator S_1^{extOT} only. The simulator receives inputs (s_0, s_1, r) for $\mathcal{A}$ on one hand, and bit c for P_2 on the other. It then internally simulates a real execution of the protocol between $\mathcal{A}$ and P_2, and outputs whatever the simulated P_2 outputs. Clearly, $\mathcal{H}_0$ is identical to $\mathsf{REAL}_{ExtOTM,\mathcal{A},\mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: In this experiment, S_1^{extOT} runs M^1 with input 0, and then rewinds M^1 and runs it with input 1 to obtain both the outputs $s_0^1 \circ \rho_0^1$ and $s_1^1 \circ \rho_1^1$. Let c^1 be P_2's query to M^1. Then, instead of running M^1, the simulator responds with $(s_{c^1}^1, \rho_{c^1}^1)$ (if M^1 outputs $\perp$ on input bit c^1, then S_1^{extOT} returns $\perp$ to P_2).

Note that in both $\mathcal{H}_0$ and $\mathcal{H}_1$, P_2 receives the same value when it queries M^1. Thus, $\mathcal{H}_0$ and $\mathcal{H}_1$ are identical.

Hybrids $\mathcal{H}_2 \ldots \mathcal{H}_{2k}$: In hybrid $\mathcal{H}_i$ for $2 \leq i \leq 2k$, the simulator extracts both the values of M^i. When P_2 queries M^i, the simulator responds without running M^i again, as it did in the case of M^1 in $\mathcal{H}_1$. As above, all these hybrids are identical.

Hybrids $\mathcal{H}_{2k+1}$: If for all i, $1 \leq i \leq 2k$, $\rho_0^i \neq \rho_1^i$, simulator aborts.

If for each i the ρ^is are different, then F_2 must guess a $2k$ length binary string. However, its input is only of length k, and thus with probability negligibly close 1, F_2 will output the wrong sequence of ρ^is, causing P_2 to abort in $\mathcal{H}_{2k}$. Thus, $\mathcal{H}_{2k}$ and $\mathcal{H}_{2k+1}$ are statistically close.

Hybrid $\mathcal{H}_{2k+2}$: S_1^{extOT} chooses $2k-1$ random bits $c^1, \ldots, c^{2k-1}$, and sets c^{2k} such that $c^{2k} = \bigoplus_{i=1}^{2k-1} c^i$. If for any index i, $s^i_{c^i} = \perp$, the simulator aborts. Else, it runs F_1 with input $(s^1_{c^1}, \ldots, s^{2k}_{c^{2k}})$, and obtains s_0. Let j be the index such that $\rho_0^j = \rho_1^j$. Now S_1^{extOT} runs F_1 with input $(s^1_{c^1}, \ldots, s^{j-1}_{c^{j-1}}, s^j_{c^j \oplus 1}, s^{j+1}_{c^{j+1}}, \ldots, s^{2k}_{c^{2k}})$ and obtains s_1. If F_1 aborts in either execution, it sets the corresponding string to the default value 0^k. Then it runs F_2 with input $r := \bigoplus_{i=1}^{2k} \rho^i_{c^i}$, and obtains output ρ. If $\rho \neq \rho^1_{c^1} \circ \ldots \circ \rho^{2k}_{c^{2k}}$, simulator aborts. Finally, S_1^{extOT} ignores P_2 and outputs (s_c, r) as P_2's output.

Note that P_2's input bits to the M^is are $2k-1$-wise independent. Thus, as there is at least one M^i which does not abort on either input, the probability that P_2 aborts in $\mathcal{H}_{2k+1}$ (before querying F_1) is the same as the probability that S_1^{extOT} aborts in $\mathcal{H}_{2k+1}$ (before running F_1).

Next, consider the joint distribution of inputs to F_1 and F_2. Since $\rho_0^j = \rho_1^j$, the joint distribution of inputs to F_1 and F_2 is identical in $\mathcal{H}_{2k+1}$ and $\mathcal{H}_{2k+2}$. Thus, the joint distribution of outputs from F_1 and F_2 is identical in the two experiments. Thus, the output of P_2 is distributed identically in $\mathcal{H}_{2k+1}$ and $\mathcal{H}_{2k+2}$.

Hybrid $\mathcal{H}_{2k+3}$: This is the ideal world experiment. The simulator S_1^{extOT} extracts (s_0, s_1, r) as above, and sends it to the ideal functionality. The output of P_2 in this case is exactly the output in $\mathcal{H}_{2k+2}$. Thus, this experiment is identical to $\mathcal{H}_{2k+2}$.

Now we handle the case of malicious P_2. Let $\mathcal{A}$ be an adversary controlling P_2, and let $\mathcal{Z}$ be an environment. Let S_{F_1} and S_{F_2} be the simulators for the OTPs for F_1 and F_2 respectively. The ideal-world adversary S_2^{ExtOTM} is defined as follows:

Simulator S_2^{ExtOTM}

1. Receive input bit c from $\mathcal{Z}$ for $\mathcal{A}$. Start internal simulation of $\mathcal{A}$ with given input.
2. For $1 \leq i \leq 2k$, send $(\mathsf{create}, \mathsf{sid}, P_1, P_2, \mathsf{mid}^i)$ to $\mathcal{A}$. Also, run simulators S_{F_1} and S_{F_2} and convey their messages to $\mathcal{A}$.
3. Answer token-queries from $\mathcal{A}$ as follows:
   (a) Let $(\mathsf{run}, \mathsf{sid}, P_1, P_2, \mathsf{mid}^{i_l}, c^{i_l})$ be a query to one of the M^is. For all but the last such query, answer with a random s^{i_l}, and a random ρ^{i_l}. When $\mathcal{A}$ asks the last query, set $c' = \bigoplus_{j=1}^{2k} c^j$. Send c' to ExtOTM and obtain (s, r). Set $\rho^{i_{2k}} = r \oplus \bigoplus_{l=1}^{2k-1} \rho^{i_l}$. Choose a random $s^{i_{2k}}$, and reply with $(s^{i_{2k}}, \rho^{i_{2k}})$.
   (b) For queries to OTPs for F_1 and F_2, forward them to S_{F_1} and S_{F_2}, and pass their responses back to $\mathcal{A}$.
4. Let $(\hat{s}^1, \ldots, \hat{s}^{2k})$ be S_{F_1}'s query to the ideal functionality for F_1's OTP. If this query occurs before all M^is have been queried, return $\perp$. For $1 \leq j \leq 2k$, check if $\hat{s}^j = s^j$. If not, return $\perp$. Else, return s.
5. Let ρ be S_{F_2}'s query to the ideal functionality for F_2's OTP. If this query occurs before all M^i's have been queried, or if $\rho \neq r$, return $\perp$. Else, return $\rho^1 \circ \ldots \circ \rho^{2k}$ to S_{F_2}.

We proceed to show that $\mathsf{REAL}_{ExtOTM,\mathcal{A},\mathcal{Z}}$ and $\mathsf{IDEAL}^{\mathrm{ExtOTM}}_{S_2^{ExtOTM},\mathcal{Z}}$ are statistically close. Consider the following hybrids:

Hybrid $\mathcal{H}_0$: In this experiment, $\mathcal{Z}$ interacts with simulator S_2^{extOT} only. The simulator receives inputs (s_0, s_1, r) for P_1 on one hand, and bit c for $\mathcal{A}$ on the other. It then internally simulates a real execution of the protocol between P_1 and $\mathcal{A}$, and outputs whatever the simulated $\mathcal{A}$ outputs. Clearly, $\mathcal{H}_0$ is identical to $\mathsf{REAL}_{ExtOTM,\mathcal{A},\mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: S_2^{extOT} runs the simulator S_{F_1} for F_1, and passes its messages to $\mathcal{A}$. When S_{F_1} queries its ideal functionality, S_2^{extOT} runs the correct OTP for F_1 sent by P_1, and returns the output to S_{F_1}. It follows from the security of OTPs that $\mathcal{H}_1$ and $\mathcal{H}_0$ are identical.

Hybrid $\mathcal{H}_2$: If $\mathcal{A}$ queries the OTP for F_1 without querying all the M^is, S_2^{extOT} causes S_{F_1} to abort.

Let M^j be a token not queried by $\mathcal{A}$ before querying F_1. Unless $\mathcal{A}$ guesses one of the outputs of M^j, F_1 will output $\perp$. Thus, $\mathcal{H}_1$ and $\mathcal{H}_2$ are statistically close.

Hybrid $\mathcal{H}_3$: S_2^{extOT} runs the simulator S_{F_2} for F_2, and passes its messages to $\mathcal{A}$. When S_{F_2} queries its ideal functionality, S_2^{extOT} runs the correct OTP for F_2 sent by P_1, and returns the output to S_{F_2}. It follows from the security of OTPs that $\mathcal{H}_2$ and $\mathcal{H}_3$ are identical.

Hybrid $\mathcal{H}_4$: If $\mathcal{A}$ queries F_2 without querying all the M^is, S_2^{extOT} causes S_{F_2} to abort.

Let M^j be a token not queried by $\mathcal{A}$ before querying F_1. Unless $\mathcal{A}$ guesses one the output of M^j, F_1 will output $\perp$. Thus, $\mathcal{H}_1$ and $\mathcal{H}_2$ are statistically close.

Hybrid $\mathcal{H}_5$: For each i, $1 \leq i \leq 2k$, let $\hat{c}^i$ be $\mathcal{A}$'s query to M^i, and let $s^i_{\hat{c}^i} \circ \rho^i$ be its response. Let $(\hat{s}^1, \ldots, \hat{s}^{2k})$ be S_{F_1}'s query to its ideal functionality. S_2^{extOT} checks, for all $1 \leq i \leq 2k$, if $\hat{s}^i = s^i_{\hat{c}^i}$. If not, it returns $\perp$ to S_{F_1}. Else, it returns $s_{\hat{c}^1 \oplus \ldots \oplus \hat{c}^{2k}}$.

If $s^i_{\hat{c}^i} = \hat{s}^i$ for all i, then $\mathcal{H}_4$ and $\mathcal{H}_5$ are identical. If not, then there exists index j, such that $\hat{s}^j$ is not the value $\mathcal{A}$ received from M^j. Unless $\mathcal{A}$ is able to guess the other output of M^j, F_1 aborts. Thus, $\mathcal{H}_4$ and $\mathcal{H}_5$ are statistically close.

Hybrid $\mathcal{H}_6$: For each i, $1 \leq i \leq 2k$, let $\hat{c}^i$ be $\mathcal{A}$'s query to M^i, and let $s^i_{\hat{c}^i} \circ \rho^i$ be its response. Let ρ be S_{F_2}'s query to its ideal functionality. S_2^{extOT} checks, if $\rho = r$. If not, it returns $\perp$ to S_{F_2}. Else, it returns $\rho^1 \circ \ldots \circ \rho^{2k}$ to S_{F_2}. This is exactly what F_2 does, therefore, $\mathcal{H}_5$ and $\mathcal{H}_6$ are identical.

Hybrid $\mathcal{H}_7$: Let $(\hat{c}^{i_1}, \ldots, \hat{c}^{i_{2k}})$ be $\mathcal{A}$'s queries to the M^is, and let $s^{i_j}_{\hat{c}^{i_j}} \circ \rho^{i_j}$, $1 \leq j \leq 2k-1$ be the first $(2k-1)$ responses. For the final query, instead of running $M^{i_{2k}}$, simulator chooses random k-bit string $t^{i_{2k}}$, and sets $\gamma^{i_{2k}} = r \oplus \bigoplus_{j=1}^{2k-1} \rho^{i_j}$. When S_{F_1} queries its ideal functionality, S_2^{extOT} checks if $\hat{s}^{i_{2k}} = t^{i_{2k}}$ and $\hat{s}^{i_j} = s^{i_j}_{\hat{c}^{i_j}}$ for $i_j \neq i_{2k}$. If all coordinates match, S_2^{extOT} returns $s_{\hat{c}^1 \oplus \ldots \oplus \hat{c}^{2k}}$ to S_{F_1}.

As S_2^{extOT} picks $t^{i_{2k}}$ uniformly at random, $t^{i_{2k}}$ and $s^{i_{2k}}_{c^{i_{2k}}}$ are identically distributed. Also, as $\gamma^{i_{2k}} \oplus \bigoplus_{j=1}^{2k-1} \rho^{i_j} = r$, $\gamma^{i_{2k}}$ and $\rho^{i_{2k}}$ are identically distributed. Thus, the output of $\mathcal{A}$ is identically distributed in experiments $\mathcal{H}_6$ and $\mathcal{H}_7$.

Hybrids $\mathcal{H}_8 \ldots \mathcal{H}_{2k+6}$: In each of these hybrids $\mathcal{H}_{6+l}$, $2 \leq l \leq 2k$, the simulator S_2^{extOT} replaces the outputs of $M^{i_{2k-l+1}}$ with random outputs, as in $\mathcal{H}_7$. By the same argument, these hybrids are identical.

Hybrid $\mathcal{H}_{2k+7}$: This is the ideal world experiment. For the first $2k-1$ queries to M^is, c^{i_j}, S_2^{extOT} responds with random values $s^{i_j} \circ \rho^{i_j}$. For the last query, S_2^{extOT} queries the ideal functionality with bit $\bigoplus_{i=1}^{2k} c^i$, and obtains (s_c, r). Then it sets $\rho^{i_{2k}} = r \oplus \bigoplus_{j=1}^{2k-1} \rho^{i_j}$. This is identical to $\mathcal{H}_{2k+6}$.

□

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

Theorem 7. (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 [Kil88, IPS08] 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, Claim 6 reduces ExtOTM to $\mathcal{F}_{wrap}^{single}$.

□

While our main focus here is on *feasibility* questions, a couple of remarks about efficiency are in place. First, the protocol Π_{ExtOTM}uses stateful tokens of size $\mathrm{poly}(k)$, where k is a statistical security parameter. In the next subsection we will show that the tokens can be further simplified to OTM tokens, each containing a pair of *bits*. Second, the number of stateful tokens employed by the above protocol is proportional to the *computational* complexity of f. This seems unavoidable given the current state of the art in the area of unconditionally secure MPC. However, if one is willing to settle for computational UC-security based on *one-way functions*, Beaver's OT extension technique [Bea96] can be used to reduce the number of tokens to $\mathrm{poly}(k)$, independently of the complexity of f. Moreover, all of these $\mathrm{poly}(k)$ tokens can be sent at one shot in the beginning of the protocol. We defer a more detailed discussion of these optimizations to the final version.

4.2 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 Section 6.

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[9]) to the receiver.

For sender-oblivious functionalities, the main result of this subsection strengthens the results of Section 4.1 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*.

4.2.1 One-time programs.

Our starting point is the concept of a *one-time program* (OTP) [GKR08]. 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).[10] The main result of [GKR08] 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 [Yao86]. 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. [IK02]). This yields perfectly secure OTPs for NC^1. We now 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 [FKN94, IK02].

The construction uses *randomized encodings* for functions:

Definition 8. ***(Perfect Randomized Encodings** [AIK06]) Let $f : \{0,1\}^n \to \{0,1\}^l$ be a function. We say that a function $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$ is a perfect randomized encoding of f, if it satisfies the following:*

- **Correctness** *There exists a deterministic algorithm $D_{\hat{f}}$, called a 'decoder', such that for every input $x \in \{0,1\}^n$, $\Pr\left[D_{\hat{f}}(\hat{f}(x, \mathcal{U}_m)) \neq f(x) \right] = 0$.*
- **Privacy** *There exists a randomized algorithm $S_{\hat{f}}$, called the 'simulator', such that for every $x \in \{0,1\}^n$, $\delta(S_{\hat{f}}(f(x)), \hat{f}(x, \mathcal{U}_m)) = 0$.*

A perfect randomized encoding is called 'efficient' if $\hat{f}$ can be computed in time polynomial in the length of x. A perfect randomized encoding is called 'decomposable' if every output bit of $\hat{f}$ depends upon a single bit of x.

We note that the Correctness and Privacy conditions can be relaxed to obtain computational and statistical randomized encodings.

We will need the following theorem on randomized encodings for $\mathbf{NC}^1$ functions:

Theorem 9. *([Kil88], [IK02]) Let f be an $\mathbf{NC}^1$ function. Then there exists an efficient, perfect decomposable randomized encoding for f.*

Let $f(\cdot, \cdot)$ be any function admitting a decomposable randomized encoding. We now construct an OTP for $f(\cdot, \cdot)$ in the pOTM-hybrid model.

Protocol Π_1. *(One-Time Program for f)*

[9] 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.

[10] The original notion of OTP from [GKR08] 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.

- ***Input:*** *P_1 has input $x \in \{0,1\}^n$.*
- ***Output:*** *P_2 should receive $f(x,y)$, for $y \in \{0,1\}^n$.*
- ***The Protocol:***

  1. *P_1 constructs a decomposable randomized encoding $\hat{f}(\cdot,\cdot)$ of the function f. Let $\hat{f}_x$ be the restriction of the encoding to x. Choose random string r, and let $\hat{f}_x(y,r) = (f_1(y_1,r),\ldots,f_n(y_n,r))$. Send $(\mathsf{create}, \mathsf{sid}, P_1, P_2, \mathsf{mid}, ((f_1(0,r),f_1(1,r)),\ldots,(f_n(0,r),f_n(1,r))))$ to* pOTM.
  2. *P_2 sends $(\mathsf{run}, \mathsf{sid}, P_1, P_2, \mathsf{mid}, (y_1,\ldots,y_n))$ to* pOTM *and obtains $s := \hat{f}_x(y,r)$. Let $D_{\hat{f}}(\cdot)$ be the decoder for $\hat{f}(\cdot,\cdot)$. Party P_2 outputs $D_{\hat{f}}(s)$.*

□

We now show that the above construction is indeed an OTP for f .

Claim 10. *For any PPT adversary $\mathcal{A}$ corrupting P_2, there exists a PPT Sim_f, such that for every environment $\mathcal{Z}$,*

$$\mathsf{IDEAL}^{\mathcal{F}_f^{OTP}}_{Sim_f,\mathcal{Z}} = \mathsf{REAL}_{\Pi_1,\mathcal{A},Z}.$$

Proof We construct the ideal world simulator Sim_f as follows:

Simulator Sim_f

1. Send $(\mathsf{create}, \mathsf{sid}, P_1, P_2, \mathsf{mid})$ to $\mathcal{A}$.
2. Receive $(\mathsf{run}, \mathsf{sid}, P_1, P_2, \mathsf{mid}, (y_1,\ldots,y_n))$ from P_2. Send $y = y_1 \ldots y_n$ to $\mathcal{F}_f^{OTP}$ and obtain $f(x,y)$. Let $S_{\hat{f}}(\cdot)$ be the simulator for the randomized encoding $\hat{f}$. Run $S_{\hat{f}}(f(x,y))$ to obtain $(\rho_1,\ldots,\rho_n)$. Send $(\rho_1,\ldots,\rho_n)$ to $\mathcal{A}$.

It directly follows from Theorem 9 that $\mathsf{IDEAL}^{\mathcal{F}_f^{OTP}}_{Sim_f,\mathcal{Z}} = \mathsf{REAL}_{\Pi_1,\mathcal{A},Z}$.

□

Implementing Parallel-OT tokens by simple OT tokens. The above protocol uses parallel-OT tokens. Now we construct a non-interactive protocol in the OTM-hybrid model that realizes pOTM.

Protocol Π_2 .

- ***Input:*** *P_1's input is a tuple of n-bit strings $((s_0^1,s_1^1),\ldots,(s_0^k,s_1^k))$. Party P_2's input is a tuple of bits $(c^1,\ldots,c^k)$.*
- ***Output:*** *P_2 should receive $(s_{c^1}^1,\ldots,s_{c^k}^k)$.*
- ***The Protocol:***

  1. *P_1 chooses k random strings r_i, for $1 \le i \le k$. Let $r = r_1 \circ \ldots \circ r_k$. Now P_1 additively shares r into k random shares $\rho_1,\ldots,\rho_k$. For $1 \le i \le k$, party P_1 sends $(\mathsf{create}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_i, ((s_0^i \oplus r_i) \circ \rho_i, (s_1^i \oplus r_i) \circ \rho_i))$ to* OTM.

2. *For* $1 \leq i \leq k$*, party* P_2 *sends* $(\mathsf{run}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_i, c^i)$ *to* OTM *and obtains* $a_i \circ b_i$*. Now party* P_2 *computes* $r' = \sum_{i=1}^{k} b_i$*. Let* $r'_1, \ldots, r'_k$ *be successive* n*-bit substrings of* r'*. Party* P_2 *outputs* $(a_1 \oplus r'_1, \ldots, a_k \oplus r'_k)$.

□

Claim 11. *For any PPT adversary* $\mathcal{A}$ *corrupting* P_2 *there exists a PPT* S_2^{pOTM}*, such that for every environment* $\mathcal{Z}$*,*

$$\mathsf{IDEAL}^{\mathrm{pOTM}}_{S_2^{pOTM},\mathcal{Z}} = \mathsf{REAL}_{\Pi_2,\mathcal{A},\mathcal{Z}}.$$

Proof We define the ideal world adversary S_2^{pOTM} as follows.

Simulator S_2^{pOTM}

1. For $1 \leq i \leq k$, send $(\mathsf{create}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_i)$ to $\mathcal{A}$.
2. On input $(\mathsf{run}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_i, c^j)$ from $\mathcal{A}$, do
   (a) if this is not the k^{th} query from $\mathcal{A}$, choose random strings $a_j \in \{0,1\}^n$ and $b_j \in \{0,1\}^{kn}$, and return $a_j \circ b_j$ to $\mathcal{A}$.
   (b) if this is the k^{th} query from $\mathcal{A}$, send $(c^1, \ldots, c^k)$ to pOTM and obtain $(s^1_{c^1}, \ldots, s^k_{c^k})$. Choose a random $a_j \in \{0,1\}^n$. For $1 \leq i \leq k$, set $r_i = a_i \oplus s^i_{c^i}$. Set $b_j = \sum_{i=1, i\neq j}^{k} b_i + r_1 \circ \ldots \circ r_k$. Send $a_j \circ b_j$.

We proceed to show that for every environment $\mathcal{Z}$, $\mathsf{IDEAL}^{\mathrm{pOTM}}_{S_2^{pOTM},\mathcal{Z}} = \mathsf{REAL}_{\Pi_2,\mathcal{A},\mathcal{Z}}$ by considering the following intermediate hybrids. In the following, the symbols $\mathcal{H}_0, \mathcal{H}_1, \ldots$ will be used to denote both the random variable that defines the output of environment $\mathcal{Z}$ in the experiments described, and the experiments themselves.

Hybrid $\mathcal{H}_0$**:** In this experiment, $\mathcal{Z}$ interacts with S_2^{pOTM} only. Simulator S_2^{pOTM} receives inputs $((s^1_0, s^1_1), \ldots, (s^k_0, s^k_1))$ for P_1, and inputs $(c^1, \ldots, c^k)$ for P_2 from $\mathcal{Z}$. Now S_2^{pOTM} internally simulates a real execution by running P_1 and $\mathcal{A}$ on their respective inputs, and simulating OTM. This is clearly identical to $\mathsf{REAL}_{\Pi_2,\mathcal{A},\mathcal{Z}}$.

Hybrid $\mathcal{H}_1$**:** This experiment is the same as above, except for $\mathcal{A}$'s final query to (simulated) OTM. Let the final query be $(\mathsf{run}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_j, c^j)$. Let $\{a_i \circ b_i\}_{\substack{i=1,\ldots k \\ i\neq j}}$ be OTM's responses to $\mathcal{A}$ so far. Instead of sending it to OTM, simulator S_2^{pOTM} answers the last query as follows: choose random string $\hat{a}_j \in \{0,1\}^n$. Then, for $1 \leq i \leq k$, compute $r'_i = a_i \oplus s^i_{c^i}$, and set $\hat{b}_j = \sum_{i=1, i\neq j}^{k} b_i + r'_1 \circ \ldots \circ r'_k$. Return $\hat{a}_j \circ \hat{b}_j$ to $\mathcal{A}$.

Note that $\mathcal{A}$'s view before the last query is uniformly distributed. Also, $\sum_{i=1, i\neq j}^{k} b_i + \hat{b}_j = r'_1 \circ \ldots \circ r'_k$, and for $1 \leq i \leq k$, $a_i \oplus r'_i = s^i_{c^i}$. Thus, $\mathcal{H}_0$ and $\mathcal{H}_1$ are identical.

Hybrid $\mathcal{H}_2$**:** This experiment is the same as above, except for the first and last queries by $\mathcal{A}$. Let $(\mathsf{run}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_{l_1}, c^{l_1})$ be the first query by $\mathcal{A}$ to OTM. Instead of forwarding this query to OTM, simulator S_2^{pOTM} chooses random strings $\hat{a}_{l_1} \in \{0,1\}^n$ and $\hat{b}_{l_1} \in \{0,1\}^{kn}$ and responds with $\hat{a}_{l_1} \circ \hat{b}_{l_1}$. For the last query, S_2^{pOTM} responds as in $\mathcal{H}_1$, using $\hat{a}_{l_1} \circ \hat{b}_{l_1}$ as the response to the first query. All other queries are handled honestly

As before, $\mathcal{A}$'s view before the final query is uniformly distributed. Thus, $\mathcal{H}_1$ and $\mathcal{H}_2$ are identical.

...

Hybrid $\mathcal{H}_k$: This experiment is similar to previous ones, except that S_2^{pOTM} fakes the second-last query also. Thus, in $\mathcal{H}_n$, simulator S_2^{pOTM} answers the first $k-1$ queries by responding with random strings. Then, in the last query, it uses $(s^1_{c^1}, \ldots, s^k_{c^k})$ to correct its previous responses. By the same argument as before, $\mathcal{H}_k$ is identical to $\mathcal{H}_{k-1}$.

Hybrid $\mathcal{H}_{k+1}$: This is the ideal-world experiment. The simulator S_2^{pOTM} proceeds as in $\mathcal{H}_k$ till the last query. At this point, it sends $(c^1, \ldots, c^k)$ to pOTM and obtains $(s^1_{c^1}, \ldots, s^k_{c^k})$. Then it performs corrections to previous responses as the simulator in $\mathcal{H}_k$.

Note that in $\mathcal{H}_k$, simulator S_2^{pOTM} does not need use P_1's input till after the last query. Thus, $\mathcal{H}_{k+1}$ and $\mathcal{H}_k$ are identical. This completes the proof.

□

A next natural step is to construct unconditionally secure OTPs for any polynomial-time computable function f. In Appendix B 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.

4.2.2 The Protocol

As in Section 4.1, the main ingredient in our solution is an interactive secure protocol Π for f. To explain the idea it is convenient to first assume that Π is secure in the plain model, without any oracles or setup. In such a case, we could 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. We need to guarantee that the receiver executes the tokens in the correct order, and also let the receiver pass the sender's state information from one token to the other without revealing this information to the sender. This can be done via a standard authentication mechanism: each token i outputs an (unconditionally secure) authenticated encryption of the sender's internal state in the end of Round i, and this information should be supplied by the receiver as an additional input to the token implementing Round $i+1$. If the authentication fails, the token outputs $\perp$.

The above procedure transforms 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 just using a one-time program realization of each complex token. This yields a new non-interactive protocol Π''. The main observation here is that the one-time programs are already secure against a malicious receiver, and any strategy a malicious sender may use in generating badly-formed OTPs corresponds to legitimate strategy for attacking Π' (which in turn corresponds to a legitimate strategy for attacking Π). Note that since the next message function of Π can be assumed wlog to be in NC^1 (possibly breaking each round into multiple mini-rounds), and since unconditionally secure authenticated encryption can also be realized in NC^1, we may assume that each one-time token in Π' realizes an NC^1 function. This allows us to apply the unconditional OTP construction for NC^1 described above.

From the plain model to the OT-hybrid model. Recall that 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 4.1: the sender cannot directly implement the OT calls by using OTM tokens, because this would give the receiver an advantage it does not have in Π. Namely, the receiver will be able to defer the invocation of the OTM tokens to the end of the protocol, thereby correlating some of its inputs with partial information obtained from the sender. (The same problem persists even if one applies to Π a standard transformation which guarantees that all OT calls are done in the beginning of the protocol and use random choice bits c_i which are independent of the receiver's inputs.) A natural approach to solve this problem *that does not work in our context*, which was applied in the context of OTPs in [GKR08], is to let the sender secret-share a key between the OTMs which is then used to encrypt all subsequent interaction. However,

this gives rise to correlation attacks by a malicious sender, as discussed in Section 4.1. Fortunately, we can use the (non-interactive) ExtOTM protocol from Section 4.1 to realize the approach of [GKR08] while resisting attacks by a malicious sender. The complex tokens required by the ExtOTM protocol can be themselves implemented using one-time programs, thereby eliminating the need for any tokens more complex than simple OTM tokens. We are ready to present our protocol formally now.

Protocol Π'. *(non-interactive protocol for computing $f(x, y)$):*

- ***Input:*** *P'_1 has input $x \in \{0,1\}^n$, and P'_2 has input $y \in \{0,1\}^n$.*
- ***Specified Output:*** *P'_2 should receive $f(x, y)$.*
- ***The protocol:***

1. *P'_1 constructs its first message as follows:*
   (a) *P'_1 uniformly chooses random tape r for P_1. For $1 \leq i \leq l$, P'_1 chooses keys k_1^i for message authentication scheme $MAC(\cdot)$. Also, for $1 \leq i \leq l$, P'_1 chooses random strings k_0^i, where the length of k_0^i is the length of the state output by $g_i^{x,r}(\cdot,\cdot)$.*
   (b) *P'_1 runs P_1 on x and r, and obtains the initial OT values $(r_0^1, r_1^1), \ldots, (r_0^t, r_1^t)$. P'_1 also chooses random strings $\rho_1, \ldots, \rho_t$.*
   (c) *For each $1 \leq i \leq l$, P'_1 constructs an unconditional one-time program for the following functionality G_i:*
      0. *(Only for i=1) Obtain input (m_1, s_0). If $s_0 \neq \rho_1 \circ \ldots \circ \rho_t$, output $\perp$. Else, set $s_0 = \mathsf{null}$ and proceed to step (ii) below.*
      i. *Receive input $(m_i, c_0^{i-1}, c_1^{i-1})$. Check $VF_{k_1^{i-1}}(c_0^{i-1}, c_1^{i-1}) = 1$. If not, output $\perp$. Else, set $s_{i-1} = c_0^{i-1} \oplus k_0^{i-1}$.*
      ii. *Compute $g_i^{x,r}(m_i, s_{i-1})$ to obtain P_1's i^{th} message m, and state s_i.*
      iii. *Output $(m, s_i \oplus k_0^i, MAC_{k_1^i}(s_i \oplus k_0^i))$.*
   (d) *Finally, P'_1's first message comprises of:*
      i. *Tokens for initial OTs: for $1 \leq i \leq t$, P'_1 sends $(P'_1, P'_2, id_i, ((r_0^i, r_1^i), \rho_i))$ to $\mathcal{F}^{ExtOTM}$.*
      ii. *Unconditional OTPs $G_1, \ldots, G_l$.*
2. ***Output:*** *P'_2 uniformly chooses random tape r' for P_2. Now, P'_2 runs P_2 and executes all initial OTs. Then, for each $1 \leq i \leq l$, P'_2 does the following:*
   (a) *Run P_2 and obtain its message m_i for the i^{th} round.*
   (b) *Run the i^{th} one-time program on input $(m_i, c_0^{i-1}, c_1^{i-1})$ (or (m_i, s_0) if $i = 1$), and obtain (m, c_0^i, c_1^i) as output.*
   (c) *Forward m to P_2.*

   Finally, P'_2 outputs P_2's output.

Theorem 12. *Protocol Π' UC-realizes functionality f in the $(\mathrm{OTM}, \mathcal{F}^{ExtOTM})$-hybrid model.*

Proof

Security against malicious P'_1. Let $\mathcal{A}$ be an adversary corrupting P'_1. We construct an adversary $S_{P'_1}$ such that, for every environment $\mathcal{Z}$, $\mathsf{REAL}_{\Pi',\mathcal{A},\mathcal{Z}} = \mathsf{REAL}_{\Pi,S_{P'_1},\mathcal{Z}}$, i.e., no environment can distinguish between an execution of Π' with adversary $\mathcal{A}$ and an execution of Π with adversary $S_{P'_1}$.

The adversary $S_{P'_1}$ is defined as follows: start internal simulation of adversary $\mathcal{A}$ with input y received from the environment. For each j, $1 \leq j \leq t$, obtain (r_0^j, r_1^j, ρ_j). Use (r_0^j, r_1^j) as the input for the j^{th} OT, and set $s_0 := \rho_1 \circ \ldots \circ \rho_t$. For each i, $1 \leq i \leq l$, obtain OTP for G_i. Now run protocol Π with the external P_2: in round

i, receive message m_i from P_2. Run OTP for G_i with input $(m_i, c_0^{i-1}, c_1^{i-1})$ (for $i = 1$, run OTP for G_1 with input (m_1, s_0)). Obtain G_i's response, (m^i, c_0^i, c_i^1). Store (c_0^i, c_1^i) for the next round, and forward m^i externally to P_2.

Note that in both the executions ($(S_{P_1'}, P_2)$ and $(\mathcal{A}, P_2')$), the (joint) distribution of inputs to the one time programs is identical. Thus, the outputs of the OTPs are identically distributed, and the output of P_2' in Π' and P_2 in Π are identically distributed.

Security against malicious P_2'. Let $\mathcal{A}$ be an adversary corrupting P_2', and $\mathcal{Z}$ be any environment. The adversary $S_{P_2'}$ must play the part of P_2 in Π. Adversary $S_{P_2'}$ proceeds as follows: it takes input y from the environment $\mathcal{Z}$ and internally simulates $\mathcal{A}$ on input y. For each $1 \leq i \leq l$, adversary $S_{P_2'}$ invokes the OTP simulator of Claim 10, Sim_{G_i} and sends these simulated OTP $\tilde{G}_i$ to the internal simulation of $\mathcal{A}$. To evaluate $\tilde{G}_i$, $\mathcal{A}$ issues OT queries to $S_{P_2'}$. When $\mathcal{A}$ asks the last input OT query for $\tilde{G}_i$, $\mathcal{A}$'s complete input to $\tilde{G}_i$ is determined, and Sim_{G_i} queries the functionality for the correct output. Now $S_{P_2'}$ externally forwards this to the real P_1, and obtains its response m, which it forwards to Sim_{G_i} as answer to its query.

However, the adversary $\mathcal{A}$ can query the OTs in any order and with arbitrary interleaving between OTs of different OTPs. This is a problem for $S_{P_2'}$. Call the i^{th} OTP *fixed* if all but one of its input OTs have been queried. Thus, when the next input OT request comes, $\mathcal{A}$'s input to this OTP will be fully specified. Now, let $k < i$, and consider the stage where the i^{th} OTP is fixed, while not all OTs for k^{th} OTP have been executed (that is, $\mathcal{A}$'s input to k^{th} OTP is still not fully specified). Now, suppose $\mathcal{A}$ queries the final input OT for i^{th} OTP, thereby fully specifying its input to this OTP. Now, $S_{P_2'}$ can not forward this message to P_1, as P_1 is waiting for a previous response.

To handle this problem, whenever $S_{P_2'}$ detects $\mathcal{A}$ attempting to execute OTP i out of order, it makes it abort; that is, when Sim_{G_i} queries the ideal functionality, $S_{P_2'}$ returns $\perp$. Details follow.

Adversary $S_{P_2'}$

1. Construction phase:
   (a) For $1 \leq i \leq l$, choose keys k_i^0, k_i^1, like P_1'.
   (b) For $1 \leq i \leq l$, run $Sim_{G_i}(\kappa)$ and send its output to $\mathcal{A}$
2. Execution phase: $S_{P_2'}$ handles $\mathcal{A}$'s queries as follows:
   (a) *Initial OTs.* On receiving (P_1', P_2', id_i, c_i) from $\mathcal{A}$ (for $1 \leq i \leq t$), forward c to the external OT, and obtain $r_{c_i}^i$. Choose a random $\hat{\rho}_i$, and return $(r_{c_i}^i, \hat{\rho}_i)$ to $\mathcal{A}$.
   (b) On receiving OT queries from $\mathcal{A}$ for unconditional OTP G_i forward the queries to Sim_{G_i}, and return the response to $\mathcal{A}$.
   (c) For Sim_{G_i}'s query to its ideal functionality, do,
      i. if i^{th} OTP is fixed, and there exists $i' < i$ such that i' is not fixed, return $\perp$.
      ii. else,
         A. if $i = 1$ and all initial OTs have not been queried, then return $\perp$. Else, let σ_1 be $\mathcal{A}$'s input to G_1. Interpret σ_1 as (m_1, s_0). If $s_0 \neq \hat{\rho}_1 \circ \ldots \circ \hat{\rho}_t$, return $\perp$. Else, forward m_1 externally to P_1, and obtain response $\tilde{m}_1$. Choose random state w, and return $(\tilde{m}_1, w \oplus k_0^1, MAC_{k_1^1}(w \oplus k_0^1))$.
         B. else, let σ_i be $\mathcal{A}$'s input to G_i. Interpret σ_i as $(m_i, c_0^{i-1}, c_1^{i-1})$. Verify $VF_{k_1^{i-1}}(c_0^{i-1}, c_1^{i-1}) = 1$. If not, return $\perp$ to Sim_{G_i}. Else, externally forward m_i to P_1, and obtain response $\tilde{m}_i$. Choose random state w, and forward $(\tilde{m}_i, w \oplus k_0^i, MAC_{k_1^i}(w \oplus k_0^i))$ to Sim_{G_i}.

Finally, output $\mathcal{A}$'s output.

We proceed to show that the random variables $\mathsf{REAL}_{\Pi',\mathcal{A},\mathcal{Z}}$ and $\mathsf{REAL}_{\Pi,S_{P_2'},\mathcal{Z}}$ are statistically close. Consider the following hybrids:

Hybrid $\mathcal{H}_0$: This distribution is the output of the following experiment: the environment $\mathcal{Z}$ interacts with adversary $S_{P_2'}$ only. $S_{P_2'}$ receives input x from $\mathcal{Z}$ for P_1' and input y from $\mathcal{Z}$ for P_2'. Now it internally simulates a real execution of honest P_1' and $\mathcal{A}$ on inputs x and y respectively. Clearly, $\mathcal{H}_0$ is identical to $\mathsf{REAL}_{\Pi',\mathcal{A},\mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: The adversary $S_{P_2'}$ proceeds as above, except for handling the first initial OT. On receiving $(P_1', P_2', id_{i_1}, c_{i_1})$ from $\mathcal{A}$, it sends obtains $(r^{i_1}_{c_{i_1}}, \rho_{i_1})$ from (simulated) $\mathcal{F}^{ExtOTM}$. Then, it chooses a random $\hat{\rho}_{i_1}$, and returns $(r^{i_1}_{c_{i_1}}, \hat{\rho_{i_1}})$ to $\mathcal{A}$.

Note that the distributions of $\hat{\rho}_{i_1}$ in $\mathcal{H}_1$ and ρ_{i_1} in $\mathcal{H}_0$ are identical. Thus, $\mathcal{H}_0$ and $\mathcal{H}_1$ are identical.

Hybrids $\mathcal{H}_2 \ldots \mathcal{H}_t$: In each hybrid $\mathcal{H}_j$, $2 \leq j \leq t$, adversary $S_{P_2'}$ replaces ρ_{i_j} with a random $\hat{\rho}_{i_j}$. As above, all these hybrids are identical.

Hybrid $\mathcal{H}_{t+1}$: Same as above, except adversary $S_{P_2'}$ replaces the l^{th} OTP with a simulated OTP. That is, $S_{P_2'}$ honestly constructs one-time programs for the first $l-1$ next message functions. But for the last one, it runs Sim_{G_l}. When $\mathcal{A}$ completely specifies its input to the last OTP by querying the last input OT for $\tilde{G}_l$, adversary $S_{P_2'}$ replies with the correct response of the l^{th} next message function.

Hybrid $\mathcal{H}_{t+2} \ldots \mathcal{H}_{t+l}$: In each hybrid $\mathcal{H}_{t+j}$, for $2 \leq j \leq l$, $S_{P_2'}$ replaces the $l-j+1$ OTP with $Sim_{G_{l-j+1}}$.

Before proceeding further, we show that the random variables $\mathcal{H}_{t+1}$ and $\mathcal{H}_{t+l}$ are statistically close. For any $1 \leq i < l$ consider any two adjacent hybrids $\mathcal{H}_{t+i}$ and $\mathcal{H}_{t+i+1}$. Observe that the only difference between the two is that in $\mathcal{H}_{t+i}$, the $(l-i)^{th}$ OTP is real, while in $\mathcal{H}_{t+i+1}$ the $(l-i)^{th}$ OTP is simulated. But by Claim 10, these distributions are statistically close. Thus, $\mathcal{H}_{t+i}$ and $\mathcal{H}_{t+i+1}$ are statistically close, for all $1 \leq i < l$.

Hybrid $\mathcal{H}_{t+l+1}$: Same as $\mathcal{H}_{t+l}$, but now if $\mathcal{A}$ queries the OTP for G_1 before executing all initial OTs, $S_{P_2'}$ returns $\perp$ to Sim_{G_1}.

Note that the OTP for G_1 takes as input $\rho_1 \circ \ldots \circ \rho_t$. Thus, if $\mathcal{A}$ tries to query G_1 OTP before executing all initial OTs, with probability negligibly close to 1, the OTP for G_1 will abort.

Hybrid $\mathcal{H}_{t+l+2}$: Same as before, but now if $\mathcal{A}$ tries to query OTPs out of order, then instead of obtaining the honest output from P_1, $S_{P_2'}$ causes the relevant OTP simulator to output $\perp$.

Note that, in $\mathcal{H}_{t+l+1}$, adversary $\mathcal{A}$ can succeed in out of order querying with probability at most the probability of generating a forged MAC, which is negligible. Thus it follows from the security of the unconditional one-time signature scheme that the two distributions are statistically close.

Hybrid $\mathcal{H}_{t+l+3}$: This experiment is the real execution of Π with $(P_1, S_{P_2'})$, where $S_{P_2'}$ interacts with the real P_1. Observe that in $\mathcal{H}_{t+l+2}$, the adversary does not use P_1's real next message function for construction of any OTPs. Instead, it only uses them to obtain P_1's responses. $\mathcal{H}_{t+l+3}$ is exactly the same except here $S_{P_2'}$ gets P_1's responses directly from P_1 rather than using its next message functions. Thus, the two distributions are identical.

□

Using bit-OT tokens. In all our constructions so far, the size of the output of our tokens is polynomially related to the input size and the security parameter. Ideally, we would only like to use hardware that handles strings of small size. For example, in the case of OT tokens, we would like to use only bit OT tokens. To this end, one can use a known *perfectly secure* reduction from string OT to bit OT [BCS96]. This reduction reduces OT on ℓ-bit strings to $O(\ell)$ parallel instances of bit OT. While in our case we need the reduction to be UC-secure against a receiver who may invoke the bit OTs in an arbitrary adaptive fashion, the reduction from [BCS96] can indeed be shown to satisfy this stronger notion of security. This is a natural consequence of the perfect security of the reduction and an efficient conditional sampling property. In the appendix, we briefly sketch an alternative construction and proof using a coding argument. Further details will be provided in the final version. Combining this final reduction with the previous results, we get an unconditionally-secure, non-interactive, UC-secure protocol for a circuit C which uses $O(|C|\cdot\text{poly}(\kappa))$ bit OT tokens. In Appendix A we sketch a simpler and self-contained alternative derivation of this reduction.

This yields the following main result of this section:

Theorem 13. (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.*

5 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 achieve interactive two-party computation protocols without requiring that the number of tokens increase with the complexity of the function being computed, as was the case in the previous section, and (2) Can we achieve two-party computation protocols using *stateless* tokens? 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. We assume only the existence of one-way functions. Since (as discussed earlier), secure protocols for any two-party task exist given OT, this suffices to achieve the claimed result. Our construction for the OT functionality (and thus for general two-party computation) is UC secure.

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[11] 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, and omitted for the sake for brevity.

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, and 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 [Bea96] (which requires only one-way functions), it suffices to implement $O(k)$ OT's, where k is the security parameter, in order to implement any polynomial number of OT's. Thus, it suffices to exchange only a linear 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.

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

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.

5.1 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 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 [BG89].

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.

Handling a Malicious Sender. 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. This has to be done in a way to enable our simulator to extract the inputs of the malicious sender. 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[12].

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. [Kil90, LP07]) 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 [HR07]) to make this probabilistic argument go through.

Now we are ready to present our protocol formally.

5.1.1 Preliminaries

Statistically Hiding Commitment Schemes We will use the statistically hiding commitment scheme of [HR07]. The receiver's transcript of commitment to bit b when the sender uses randomness r will be denoted by $\mathsf{scom}(b, r)$. The decommitment phase consists of the sender simply sending its randomness r along with bit b, and the receiver verifies if (r, b) is consistent with the transcript $\mathsf{scom}(b, r)$.

Definition 14. *(Pairwise Independent Hash Functions ([CW79])) Let $\mathcal{H}$ be a family of functions mapping strings of length $l(n)$ to strings of length $m(n)$. Then, $\mathcal{H}$ is an **efficient family of pairwise independent hash functions** if the following hold:*

Samplable. *$\mathcal{H}$ is polynomially samplable in n.*

Efficient. *There exists a polynomial time algorithm that given $x \in \{0,1\}^{l(n)}$ and a description of $h \in \mathcal{H}$ outputs $h(x)$.*

Pairwise Independence. *For every distinct $x_1, x_2 \in \{0,1\}^{l(n)}$, and every $y_1, y_2 \in \{0,1\}^{m(n)}$, we have:*

$$\Pr_{h \leftarrow \mathcal{H}}[\, h(x_1) = y_1 \wedge h(x_2) = y_2 \,] = 2^{-2m(n)}.$$

It is known ([CW79]) that there exists an efficient family of pairwise independent hash functions for every l and m whose element description size is $\mathcal{O}(\mathsf{max}(l(n), m(n)))$.

Let X be a random source with min-entropy k.

Theorem 15. *(Leftover Hash Lemma ([HILL99])) If the family $\mathcal{H}$ of hash functions $h : \{0,1\}^n \to \{0,1\}^l$ is pairwise independent, where $l = k - 2\log(1/\epsilon) - \mathcal{O}(1)$, then $Ext(X, h) := (h, h(X))$ is a strong $(k, \epsilon/2)$-extractor.*

[12] At first glance, this may not seem like a problem, since we can treat an abort as a special output string $\perp$, which the sender could have anyway provided as one his inputs to the OT. But the adaptive decision of whether or not to abort is actually a problematic additional "axis" of control that the sender has in addition to the allowed choice of strings depending on the receiver's bit. We now elaborate with an example:

A concrete "problem case" is to consider a commitment algorithm in which the first bit c_1 of the commitment to a bit b is set equal to $r \oplus b$, where r is a randomly chosen bit. Now, a dishonest sender can send a token such that when it is fed the decommitment information, it decides to abort iff $r = 1$. Let $r' = 1$ iff the honest receiver sees the token abort and thus aborts. In real life executions of the protocol, we will always have the invariant that $c_1 = r' \oplus b$. However the natural simulator for this protocol, in which the simulator (which does not know b), chooses a commitment to a random b', would lead to ideal world executions in which $c_1 \neq r' \oplus b$ with probability $1/2$.

5.1.2 The Protocol

Let k be the security parameter. Let $\mathcal{H}$ be an efficient family of pairwise hash functions mapping strings of length $5k$ to k. Let $\mathcal{F}$ be a family of pseudo-random functions mapping strings of length $5k$ to k.

Protocol. *(Quasi OT.)*

- ***Input:*** *P_1 has two strings $(s_0, s_1) \in \{0,1\}^k$, P_2 has a selection bit b.*
- ***Common input:*** *An index j.*
- ***Protocol:***

  1. *P_2 chooses a random PRF key γ for family $\mathcal{F}$, and sends $(\mathsf{create}, \mathsf{sid}, P_2, P_1, \mathsf{mid}_{j,1}, M_1)$, where M_1 implements the following functionality:*
     - *On input string $x \in \{0,1\}^{5k}$, output $f_\gamma(x)$.*

     P_2 also sends to P_1 a randomly chosen string r, which serves as the first message of the (statistically binding) commitment scheme com.
  2. *P_1 chooses two strings x_0 and x_1 uniformly from $\{0,1\}^{5k}$, and a random string $\pi \in \{0,1\}^k$. P_1 sends $(\mathsf{run}, \mathsf{sid}, P_2, P_1, \mathsf{mid}_{j,1}, x_i)$ to obtain y_i, for $i \in \{0,1\}$. Now P_2 chooses a random hash function $h \in \mathcal{H}$, and sends $(y_0, y_1, h, \alpha = \mathsf{com}(\pi))$ to P_2.*
  3. *Now P_2 commits to its input bit b using the commitment scheme* scom. *That is, P_2 and P_1 run the statistically hiding commitment protocol* scom*, with P_2 acting as the sender in the commitment protocol with input bit b. Let $\hat{\alpha}$ be P_1's transcript (view) of the commitment phase, and let $\hat{r}$ be the randomness used by P_2 in the commitment protocol.*
  4. *If the commitment phase succeeds, P_1 sends $(\mathsf{create}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_{j,2}, M_2)$, where M_2 implements the following functionality:*
     - *Obtain randomness $\hat{r}$ and bit b, and verify that this is a valid decommitment (with respect to commitment transcript $\hat{\alpha}$). If so, output x_b and $\beta = \mathsf{open}(\alpha)$. Else, output $\perp$.*
  5. *P_2 sends $(\mathsf{run}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_{j,2}, (\hat{r}, b))$ and obtains x_b and β. It then checks if $f_\gamma(x_b) = y_b$. If not, it aborts. Then it checks if β is a valid opening of α. If not, it aborts. Else, let π' be the revealed string. P_2 sends π' to P_1.*
  6. *P_1 receives string π' from P_2, and checks if $\pi = \pi'$. If not, it aborts. Else, it sends $(s'_0 = s_0 \oplus h(x_0), s'_1 = s_1 \oplus h(x_1))$.*
  7. *P_2 receives (s'_0, s'_1), and outputs $s'_b \oplus h(x_b)$.*

As mentioned before, this protocol does not realize the OT functionality, as a malicious sender can selectively abort based on receiver's input. Now we present a protocol that uses QuasiOT as a subroutine and realizes OT in the $\mathcal{F}_{wrap}^{stateless}$-hybrid model.

Protocol. *(OT in $\mathcal{F}_{wrap}^{stateless}$-hybrid model.)*

- ***Input:*** *P_1 has two strings $(s_0, s_1) \in \{0,1\}^k \times \{0,1\}^k$, P_2 has selection bit b.*
- ***Output:*** *P_2 outputs s_b.*
- ***Protocol:***

  1. *P_1 chooses $z_1, \ldots, z_k, r \in \{0,1\}^k$ uniformly at random. P_2 chooses $k-1$ random bits $b_1, \ldots, b_{k-1}$, and sets b_k such that $b = \bigoplus_{j=1}^{k} b_j$. Now P_1 and P_2 execute in parallel, k copies of $QuasiOT$. For $1 \le j \le k$, the inputs to the j^{th} copy of $QuasiOT$ are $(z_j, z_j \oplus r)$ and b_j.*

2. *If all k copies of $QuasiOT$ finish without aborts, then P_1 sends to P_2 the pair $(s'_0 = s_0 \oplus \bigoplus_{j=1}^{k} z_j, s'_1 = s_1 \oplus \bigoplus_{j=1}^{k} z_j \oplus r)$.*
3. *For $1 \leq j \leq k$, let a_j be the string received by P_2 at the end of the j^{th} invocation of $QuasiOT$. Then, P_2 outputs $s'_b \oplus \bigoplus_{j=1}^{k} a_j$.*

5.1.3 Security Proof

Theorem 16. *Protocol 5.1.2 UC-realizes OT in the $\mathcal{F}_{wrap}^{stateless}$-hybrid model.*

Proof We first consider the case of malicious sender. Let $\mathcal{A}$ be an adversary corrupting P_1, and let $\mathcal{Z}$ be any environment. For each j, $1 \leq j \leq k$, we first present a sub-routine $\hat{S}_{1,j}$ that will be used by the ideal-world simulator S_1.

Subroutine $\hat{S}_{1,j}$

1. Send $(\mathsf{create}, \mathsf{sid}, P_2, P_1, \mathsf{mid}_{j,1})$ to $\mathcal{A}$. Also send a random string r.
2. For each query x to $\mathsf{mid}_{j,1}$, $\hat{S}_{1,j}$ checks if $\mathcal{A}$ queried the $\mathsf{mid}_{j,1}$ on x before. If it did, return the previous response. Else, return a randomly chosen k-bit string y. The simulator keeps a list of $\mathcal{A}$'s queries to $\mathsf{mid}_{j,1}$ and its responses.
3. When $\hat{S}_{1,j}$ receives (y_0, y_1, h, α) from $\mathcal{A}$, it checks the list of responses, and obtains the strings x_0 and x_1 to which it responded with y_0 and y_1 respectively. If for any y_i, $i \in \{0, 1\}$, no such x_i exists, set $x_i = \perp$.
4. $\hat{S}_{1,j}$ picks a random bit b, and runs the commitment protocol scom with $\mathcal{A}$, with b as its input.
5. When $\hat{S}_{1,j}$ receives $(\mathsf{create}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_{j,2}, M_{j,2})$, it honestly runs the decommitment phase of scom, and obtains output $(\hat{x}_b, \beta)$. If $\hat{x}_b \neq x_b$, it aborts. If β is not the correct opening of α, it aborts. Else, let π be the string revealed in the opening. $\hat{S}_{1,j}$ returns π to $\mathcal{A}$.
6. When $\hat{S}_{1,j}$ obtains $(s'_{j,0}, s'_{j,1})$ from $\mathcal{A}$, it returns $(s_{j,0} = s'_{j,0} \oplus h(x_0), s_{j,1} = s'_{j,1} \oplus h(x_1))$ to S_1.

Now we describe the the 'outer' simulator S_1.

Simulator S_1

1. Invoke adversary $\mathcal{A}$, and pass messages from environment to P_1 or $\mathcal{A}$, to $\mathcal{A}$ and vice versa.
2. For each j, $1 \leq j \leq k$, run $\hat{S}_{1,j}$, and pass messages between the subroutine and $\mathcal{A}$. If no subroutine $\hat{S}_{1,j}$ aborts, for each j, obtain pair $(s_{j,0}, s_{j,1})$.
3. Let (s'_0, s'_1) be the final message from $\mathcal{A}$. The simulator S_1 picks two random bit-vectors $(b_1, \ldots, b_k)$ and $(b'_1, \ldots, b'_k)$, such that $\bigoplus_{j=1}^{k} b_j = 0$ and $\bigoplus_{j=1}^{k} b'_j = 1$. Then it sets $s_0 = s'_0 \oplus \bigoplus_{j=1}^{k} s_{j,b_j}$ and $s_1 = s'_1 \oplus \bigoplus_{j=1}^{k} s_{j,b'_j}$. The simulator sends (s_0, s_1) to the ideal OT functionality.

We prove indistinguishability of real and ideal worlds via the following series of hybrids. We denote the k invocations of the QuasiOT subroutine by $QOT_1, \ldots, QOT_k$.

Hybrid $\mathcal{H}_0$: This is the real execution.

Hybrid $\mathcal{H}_{1,0}$: Same as above, except we modify the behaviour of QOT_1 as follows: instead of answering $\mathcal{A}$'s queries by running $M_{1,1}$, for each query x, we respond with a randomly chosen k-bit string. We record the queries and our responses in a list. When $\mathcal{A}$ asks query x, we first check if $\mathcal{A}$ asked x before. If this is the case, we respond with the same string as before.

Any environment that can distinguish between $\mathcal{H}_{1,0}$ and $\mathcal{H}_0$ can distinguish the PRF from a random function. Thus, as the PRF key is never revealed, by the security of the PRF, hybrids $\mathcal{H}_{1,0}$ and $\mathcal{H}_0$ are indistinguishable.

Hybrid $\mathcal{H}_{1,1}$: Same as above, except now, as we record the responses in the list, we ensure that no two of $\mathcal{A}$'s queries have the same response. If there is a collision, $\mathcal{H}_{1,1}$ aborts.

Note that the probability of abort in $\mathcal{H}_{1,1}$ is at most the probability of finding a collision in a randomly chosen function. Thus, $\mathcal{H}_{1,0}$ and $\mathcal{H}_{1,1}$ are statistically close.

Hybrid $\mathcal{H}_{1,2}$: This experiment is same as above, except the following: let (y_0, y_1, h, α) be $\mathcal{A}$'s first message to P_2. We check the list of responses to find strings x_0 and x_1 such that y_0 and y_1 were the responses to x_0 and x_1 respectively. Note that conditioned on the event that $\mathcal{H}_{1,1}$ does not abort, each y can be a response to at most a single $\mathcal{A}$ query x. If for $i \in \{0, 1\}$, y_i is not the response to any query, set $x_i = \perp$. Now, let $(\hat{x}_{b_1}, \beta)$ be $M_{1,2}$'s output. If $\hat{x}_{b_1} \neq x_{b_1}$, hybrid $\mathcal{H}_{1,2}$ outputs $\perp$. Else, let $(s'_{1,0}, s'_{1,1})$ be $\mathcal{A}$'s final message. QOT_1 returns $(s_{1,0} = s'_{1,0} \oplus h(x_0), s_{1,1} = s'_{1,1} \oplus h(x_1))$ to S_1, which forwards s_{1,b_1} to the simulated P_2.

If $x_{b_1} \neq \perp$, the the probability that the images (under a random function) of $\hat{x}_{b_1}$ and x_{b_1} would be the same, is negligible. If $x_{b_1} = \perp$, then the probability that $\mathcal{A}$ can guess the image (under a random function) of $\hat{x}_{b_1}$ is negligible. Thus, $\mathcal{H}_{1,1}$ and $\mathcal{H}_{1,2}$ are statistically close.

Hybrids $\mathcal{H}_{2,0}, \ldots, \mathcal{H}_{k,2}$: For $2 \leq j \leq k$ and $j' \in \{0, 1, 2\}$, hybrid $\mathcal{H}_{j,j'}$ modifies the behaviour of QOT_j in the same way as $\mathcal{H}_{1,j'}$ modifies the behaviour of QOT_1. By the same arguments as above, it follows that these hybrids are indistinguishable.

Hybrid $\mathcal{H}_{k+1}$: Same as above, except instead of using $\vec{v} = (b_1, \ldots, b_k)$ as the inputs to QOT_js, where $\bigoplus_{j=1}^{k} b_j = b$, we pick a random vector of bits $\vec{v}'$, and use that as inputs to the QOT_js. Let (s'_0, s'_1) be the final message from $\mathcal{A}$. Now we randomly pick two k-bit vectors $\vec{v}_0 = (b_1, \ldots, b_k)$ and $\vec{v}_1 = (b'_1, \ldots, b'_k)$ such that $\bigoplus_{j=1}^{k} b_j = 0$ and $\bigoplus_{j=1}^{k} b'_j = 1$. For each j, let $(s_{j,0}, s_{j,1})$ be the pairs of strings obtained from QOT_js. Set $s_0 = s'_0 \oplus \bigoplus_{j=1}^{k} s_{j,b_j}$ and $s_1 = s'_1 \oplus \bigoplus_{j=1}^{k} s_{j.b'_j}$. Finally, output s_b as P_2's output.

Note that com is a statistically binding commitment scheme. Thus, with probability negligibly close to 1, α has a single opening. Next, observe that the only difference between $\mathcal{H}_{k,2}$ and $\mathcal{H}_{k+1}$ is in the inputs to the OT boxes (that is, inputs to the tokens $M_{j,2}$). Also note that in $\mathcal{H}_{k,2}$, each token $M_{j,2}$, outputs either the correct string x_{j,b_j}, or $\perp$. Thus, the only difference in the two hybrids is the probability of abort. We now show that the probabilities of abort are negligibly close to each other.

Claim 17. *Let p_1 be the probability of abort in $\mathcal{H}_{k,2}$, and p_2 be the probability of abort in $\mathcal{H}_{k+1}$. Then, $|p_1 - p_2| \leq 2^{-k+1}$.*

Proof For simplicity, we assume that the commitment scheme scom is *perfectly* hiding. In the end, we point out how to extend the proof to the case of statistical hiding schemes.

We will condition the probability of abort on $\mathcal{A}$'s transcripts of the commitments. Let $\vec{v} = (b_1, \ldots, b_k)$ be a bit vector, and let $\vec{r} = (r_1, \ldots, r_k)$ be a vector of random strings. By $\mathsf{scom}(\vec{v}, \vec{r}) = (\hat{\alpha}_1, \ldots, \hat{\alpha}_k)$ we mean a vector of transcripts, where for $1 \leq j \leq k$, $\hat{\alpha}_j$ is the transcript of the receiver in the commitment scheme scom when the sender commits bit b_j using randomness r_j (the randomness of the receiver is implicit).

Let $\vec{c} = (\hat{\alpha}_1, \ldots, \hat{\alpha}_k)$ be a transcript vector. First, note that, because of the perfect hiding property of the commitment scheme scom, the probability of occurrence of $\vec{c}$ in $\mathcal{H}_{k,2}$ and $\mathcal{H}_{k+1}$ is exactly the same. That is, the number of randomness strings $\vec{r}$ that lead to transcript $\vec{c}$ is the same in the two hybrids. For $\vec{c}$, fix a randomness

vector $\vec{r}$ that can lead to $\vec{c}$. For $\vec{c}$ and $\vec{r}$, we analyze the behaviour of the tokens $M_{j,2}$ in the following two experiments corresponding to $\mathcal{H}_{k,2}$ and $\mathcal{H}_{k+1}$ respectively: in the first, $\vec{v}$ is chosen so that $\bigoplus_{j=1}^{k} b_j = b$, and in the other, $\vec{v}$ is randomly chosen. Note that fixing $\vec{c}$ and $\vec{r}$ fixes the output of token $M_{j,2}$: on on input b_j, it either outputs x_{j,b_j} or $\perp$. Consider the following two cases:

Case 1. There exists j, $1 \leq j \leq k$ such that $M_{j,2}$ aborts neither on 0 nor on 1. In this case, as the inputs to the $M_{j',2}$s are $(k-1)$-wise independent, the probability of abort is identical in the two cases.

Case 2. For all j, $M_{j,2}$ aborts on either 0 or 1. Then, the probability that $\mathcal{H}_{k,2}$ aborts is at least $1 - 2^{-k+1}$. Thus, in this case, the difference in the probability that $\mathcal{H}_{k,2}$ aborts and the probability that $\mathcal{H}_{k+1}$ aborts is at most 2^{-k+1}.

Thus, the two hybrids are statistically close.

In the case of statistical hiding instead of perfect hiding, the probability that a transcript vector $\vec{c}$ occurs in the two hybrids are not the same, but negligibly close. Thus, we must discard some randomness vectors $\vec{r}$ from the analysis, but this changes the probabilities only be a negligible amount.

□

Hybrid $\mathcal{H}_{k+2}$: This is the ideal world. Note that we only use P_2's selection bit b in the final step to determine its output. As the ideal world P_2 is honest, it queries the ideal functionality with bit b and obtains string s_b. Thus, $\mathcal{H}_{k+2}$ and $\mathcal{H}_{k+1}$ are identical.

Next, we handle the case of a malicious receiver. Let $\mathcal{A}$ be an adversary corrupting P_2, and let $\mathcal{Z}$ be any environment. We first present $\hat{S}_{2,j}$ which will be used as a sub-routine by simulator S_2.

Subroutine $\hat{S}_{2,j}$

1. Receive as input from S_2 a pair of strings $(s_{j,0}, s_{j,1})$. Receive $(\mathsf{create}, \mathsf{sid}, P_2, P_1, \mathsf{mid}_{j,1}, M_{j,1})$ and random string r from $\mathcal{A}$. Run $M_{j,1}$ on randomly chosen $5k$-bit strings $x_{j,0}$ and $x_{j,1}$ and obtain $y_{j,0}$ and $y_{j,1}$. Choose random $\pi_j \in \{0,1\}^k$ and a random hash function $h_j \in \mathcal{H}$, and return $(y_{j,0}, y_{j,1}, h_j, \alpha_j = \mathsf{com}(\pi_j))$ to $\mathcal{A}$.
2. Run the statistical hiding commitment protocol with $\mathcal{A}$. Let $\hat{\alpha}_j$ be the receiver's transcript.
3. Send $(\mathsf{create}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_{j,2})$ to $\mathcal{A}$. If $\mathcal{A}$ correctly decommits, then let b_j be the revealed bit. Send $(x_{j,b_j}, \beta_j = \mathsf{open}(\alpha_j))$ to $\mathcal{A}$. If $\mathcal{A}$ ever decommits to both 0 and 1, output binding abort and abort.
4. Receive π' from $\mathcal{A}$. If $\pi' \neq \pi$, abort. If $\mathcal{A}$ returns the correct π without running $M_{j,2}$, output hiding abort and abort. Else, set $s'_{j,0} = s_{j,0} \oplus h_j(x_{j,0})$ and $s'_{j,1} = s_{j,1} \oplus h_j(x_{j,1})$. Send $(s'_{j,0}, s'_{,1})$ to $\mathcal{A}$, and return b_j to S_2.

Now we present the ideal-world simulator S_2.

Simulator S_2

1. Choose $z_1, \ldots, z_k, r \in \{0,1\}^k$ uniformly at random. For each $1 \leq j \leq k$, run $\hat{S}_{2,j}$ with input $(z_j, z_j \oplus r)$. If any subroutine aborts, abort. Else, let b_j be the values returned by the $\hat{S}_{2,j}$s.
2. Send $b = \bigoplus_{j=1}^{k} b_j$ to the ideal OT functionality, and obtain s_b. Choose random k-bit string $s_{1\oplus b}$. Set $s'_b = s_b \oplus \bigoplus_{j=1}^{k}(z_j \oplus b_j \cdot r)$ (where $b_j \cdot r$ is r if $b_j = 1$, else 0) and $s'_{1\oplus b} = s_{1\oplus b}$. Send (s'_0, s'_1) to $\mathcal{A}$.

We show indistinguishability of the real and ideal worlds via the following hybrids.

Hybrid $\mathcal{H}_0$: This is the real world execution.

Hybrid $\mathcal{H}_1$: Same as above, except we modify the behaviour of the QuasiOT protocols in the following way: if for any j, $\mathcal{A}$ outputs the correct value π_j without running $M_{j,2}$, then output hiding abort and abort.

We now show that the probability that $\mathcal{H}_1$ outputs hiding abort is negligible.

Claim 18. *Let ϵ be the probability that $\mathcal{H}_1$ outputs hiding abort. Then, ϵ is negligible in k.*

Proof We construct a non-uniform adversary $\mathcal{A}_{\mathsf{com}}$ that breaks the hiding property of commitment scheme com with probability ϵ. To show this, the adversary $\mathcal{A}_{\mathsf{com}}$ must succeed with probability ϵ in the following game: $\mathcal{A}_{\mathsf{com}}$ interacts with an external party, called the challenger, and acts as the receiver in commitment scheme com. It sends a random string r as the first message, and receives a commitment $\tilde{\alpha}$ to a random k bit string π. The adversary succeeds if it outputs π.

Let j be such that $\mathcal{A}$ outputs π_j without running $M_{j,2}$ with probability at least ϵ. The adversary $\mathcal{A}_{\mathsf{com}}$ works as follows: it sends a random r to the challenger, and receives a commitment $\tilde{\alpha} = \mathsf{com}(\pi)$. Now $\mathcal{A}_{\mathsf{com}}$ sets up an execution of $\mathcal{A}$ and honest P_1 with environment $\mathcal{Z}$. For the j^{th} copy of the QuasiOT subroutine, it uses $\tilde{\alpha}$ as the commitment in P_1's first message to $\mathcal{A}$. Thereafter, $\mathcal{A}_{\mathsf{com}}$ proceeds with the rest of the execution without modification. If $\mathcal{A}$ queries $M_{j,2}$ correctly, then $\mathcal{A}_{\mathsf{com}}$ aborts. Otherwise, let π' be $\mathcal{A}$'s message to P_1 in step 6 of the j^{th} QuasiOT subroutine. Adversary $\mathcal{A}_{\mathsf{com}}$ outputs π' as its guess of the string committed in $\tilde{\alpha}$.

By the hypothesis, with probability ϵ, adversary $\mathcal{A}$ outputs the correct stri ng π in step 6 of the j^{th} QuasiOT. Thus, if the commitment scheme com is computationally hiding, ϵ must be negligible.

□

Hybrid $\mathcal{H}_2$: This is the same as above, except if for any j, adversary $\mathcal{A}$ provides two different openings to $M_{j,2}$, then $\mathcal{H}_2$ outputs binding abort and aborts.

We show that the probability that $\mathcal{H}_2$ outputs binding abort is negligible.

Claim 19. *Let ϵ be the probability that for some j, adversary $\mathcal{A}$ provides different openings to $M_{j,2}$. Then, ϵ is negligible in k.*

Proof We construct a non-uniform adversary $\mathcal{A}_{\mathsf{com}}$ that breaks the binding property of commitment scheme scom with probability ϵ. To show this, the adversary $\mathcal{A}_{\mathsf{com}}$ must succeed with probability ϵ in the following game: $\mathcal{A}_{\mathsf{com}}$ interacts with an external party, called the challenger, and acts as the sender in commitment scheme scom. If the commitment phase is successful, let $\hat{\alpha}$ be the transcript of the receiver in the commitment phase. The adversary succeeds if it can produce $(r, 0)$ and $(r', 1)$, both of which are valid decommitments with respect to $\hat{\alpha}$.

Let j be such that $\mathcal{A}$ queries $M_{j,2}$ twice on two distinct bits. The adversary $\mathcal{A}_{\mathsf{com}}$ works as follows: it sets up an execution between $\mathcal{Z}$, $\mathcal{A}$ and honest P_1 as in $\mathcal{H}_2$. Then it executes the j^{th} copy of scom with the challenger. If $\mathcal{A}$ queries $M_{j,2}$ only once, abort. Else, let $(r, 0)$ and $(r', 1)$ be the two queries. $\mathcal{A}_{\mathsf{com}}$ outputs $(r, 0)$ and $(r', 1)$.

By the hypothesis, with probability at least ϵ, $\mathcal{A}_{\mathsf{com}}$ outputs two different openings. As scom is computationally binding, ϵ must be negligible in k.

□

Hybrid $\mathcal{H}_3$: Same as above, except we modify the last message of S_2. We set $s'_b = s_b \bigoplus_{j=1}^{k}(z_j \oplus b_j \cdot r)$ and set $s'_{1\oplus b}$ to a random k-bit string.

We show that in $\mathcal{H}_2$, $s'_{1\oplus b}$ is uniformly distributed in $\{0,1\}^k$ given previous messages.

We first show that for every j, if b_j is $\mathcal{A}$'s input to $M_{j,1}$, then $s'_{1\oplus b_j}$ is uniformly distributed in $\{0,1\}^k$. Observe that $M_{j,1}$ partitions the domain $\{0,1\}^{5k}$ into 2^k partitions $S_1, \ldots, S_{2^k}$, such that for all $x, x' \in S_i$, $M_{j,1}(x) = M_{j,1}(x')$. For $x \in \{0,1\}^{5k}$, let $S(x)$ be the partition that x belongs to. Call a partition S_i "good" if $|S_i| \geq 2^{3k}$. We have the following claim.

Claim 20. *If x is uniformly chosen in $\{0,1\}^{5k}$, then the probability that it belongs to a good partition is at least $1-2^{-k}$.*

Proof As there are only 2^k partitions, we have, by the union bound,

$$\Pr[\ S(x) \text{ is bad }\ ] < \frac{2^k \cdot 2^{3k}}{2^{5k}} = 2^{-k}.$$

Thus, the probability that x lies in a good partition is at least $1-2^{-k}$.

□

Thus, with probability negligibly close to 1, the random variable $\mathcal{U}_{5k}|M_{j,1}(\mathcal{U}_{5k})$ has min-entropy $3k$. Thus, by the leftover hash lemma, we have that,

$$\Delta((h_j, h_j(x_{1\oplus b_j})), (\mathcal{U}_{\mathcal{H}}, \mathcal{U}_k)) \leq 2^{-k-1}.$$

That is, $h_j(x_{1\oplus b_j})$ is distributed close to randomly in $\{0,1\}^k$. Thus, for each j, $\mathcal{A}$ learns $z_j \oplus b \cdot r$, while $z_j \oplus (1\oplus b)\cdot r$ is completely random to it. Note that as $s'_0 = s_0 \oplus \bigoplus_{j=1}^k z_j$ and $s'_1 = s_1 \oplus \bigoplus_{j=1}^k z_j \oplus r$, $\mathcal{A}$ learns only s_b, while $s_{1\oplus b}$ is complete random to it.

Hybrid $\mathcal{H}_4$: This is the ideal world. Note that $\mathcal{H}_3$ is exactly the simulator S_2. The simulator extracts b as above, sends it to the ideal OT functionality, and obtains s_b. Then it sets $s'_b = s_b \bigoplus_{j=1}^k (z_j \oplus b_j \cdot r)$, and $s'_{1\oplus b}$ to a random value, and sends (s'_0, s'_1) to $\mathcal{A}$. This hybrid is identical to $\mathcal{H}_3$.

□

Thus, we have the following main theorem for this section.

Theorem 21. (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.*

6 Oblivious Reactive Functionalities in the Non-Interactive Setting

In this section, 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. 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. For simplicity, we only focus on the standalone case and leave achieving Universal Composability for future work.

6.1 Stateless Tamper-proof Hardware Token

We define stateless reactive functionalities in terms of the real/ideal worlds.

Ideal World. The ideal world consists of a trusted party $\mathcal{F}_f$ and parties P_1 and P_2, where $f(\cdot,\cdot)$ is a two-party functionality. An execution in the ideal world proceeds as follows: party P_1 hands its input x to $\mathcal{F}_f$. Then party P_2 queries $\mathcal{F}_f$ on an inputs y_i of its choosing, to obtain $f(x, y_i)$. The functionality may maintain state from one invocation to another. The functionality may allow the receiver to query it an unbounded polynomial number of times, or might force a bound. Note that P_2 may choose its queries adaptively, based on the responses to its previous queries. For a given adversary S corrupting one of the parties in the ideal world and inputs x and y_1 to P_1 and P_2 respectively, the output of the honest party along with the output of the adversary in the above process is denoted by $\mathsf{IDEAL}_S^f(x, y_1)$.

Real World. In the real world, the honest party follows all instructions of the prescribed protocol, while an adversarial party may behave arbitrarily. Let $\mathcal{A}$ be a real world adversary. We denote the honest parties output along with the output of $\mathcal{A}$ on inputs x and y_1 to P_1 and P_2 respectively, by $\mathsf{REAL}_{\mathcal{A}}(x, y_1)$.

Definition 22. *(Secure Implementation of Stateless Reactive Functionality) A two-party protocol $\pi = (P_1, P_2)$ is said to be a secure implementation of the stateless reactive functionality f if for all probabilistic polynomial time adversaries $\mathcal{A}$ corrupting one of the parties in the real world, there exists an expected polynomial time ideal world adversary S, called the simulator, such that,*

$$\{\, \mathsf{IDEAL}_S^f(x, y) \,\}_{(x,y)} \sim \{\, \mathsf{REAL}_{\mathcal{A}}(x, y) \,\}_{(x,y)}.$$

In the following, we will often refer to P_1 as the sender, and P_2 as the receiver. A reactive functionality is *oblivious-sender*, if the sender does not get any output, and sends only one input.

We make distinction between stateful and stateless reactive functionalities: in the former, the functionality does not keep state over multiple calls, while in the latter the functionality can keep state.

6.2 Realizing Stateless Oblivious Reactive Functionalities with Semi-Honest Sender (Program Obfuscation with Stateless Tokens)

We first consider the case of stateless oblivious reactive functionalities with semi-honest senders. Recall that stateless reactive functionalities don't keep state over invocations, except the input of the sender.

We observe that in the non-interactive setting where the sender simply sends a *package* (constructed using only stateless tokens) to the receiver, the receiver can not be prevented from running this package multiple times with different inputs. Hence, we restrict our attention to only reactive functionalities where: (a) the receiver can query the ideal trusted party *an unbounded polynomial* number of times with adaptively chosen inputs of its choice, (b) the sender provides only one input and gets no output (i.e., the functionality is *sender oblivious*), and, (c) the only state maintained by the ideal trusted party between different invocation is the sender's input. We show how to construct protocol to securely realize such reactive functionalities (called stateless oblivious reactive functionality) under standard cryptographic assumptions.

Following the paradigm in Section 3, we first consider the case of semi-honest sender. We first observe that the task of constructing protocol for stateless oblivious reactive functionalities with semi-honest sender exactly coincides with the task of constructing a secure obfuscation scheme with a stateless tamper proof hardware. While the problem of constructing obfuscation scheme with tamper proof hardware has been studied before ([GO96], [GKR08], [And08]), all existing constructions crucially rely on the ability of such hardware to maintain state. These constructions do not directly extend to the case of stateless hardware.

6.2.1 Overview

We start by informally describing some intuition behind our construction and the main subtleties that result from the hardware token being stateless. To obfuscate the given the circuit C, the first step is to use a universal circuit U of appropriate size such that $U(x, C) = C(x)$. This reduces the problem of hiding the circuit to hiding the input to the universal circuit U. Next, we encrypt each bit of C (when represented as an input string to U) with a secret key

S of a non-malleable symmetric key encryption scheme. The encrypted bits, along with a description of U, can be given to the user as the obfuscated program. To run the program on input x, the user encrypt each bit of x using the hardware token (which knows the secret key K). Thus, the user now has the description of the circuit U as well as an encryption of all its input bits (i.e., the circuit C and the user input x). The user can now evaluate the circuit gate by gate: for each gate, the user would query with the encrypted values on the two input wires and get the encrypted value on the output wire in response from the token.

The above (rough) approach is problematic when the receiver is an *active* adversary. While the circuit U is being evaluated, he can change the value on any wire (by just encrypting the desired value from scratch with the public key and using that for all future queries). Furthermore, he could replace the encrypted value on one wire with the value on another, or, could launch ciphertext malleability attack to try to change the value on the wire. Fortunately, such problems can be solved by a careful usage of standard techniques. For example by, having a secret obfuscation identity (a random nonce selected while preparing the obfuscated circuit) encrypted along with each wire value (so that the adversary cannot reencrypt the desired bit from scratch), having a unique number for each wire in U (so that the adversary cannot replace value on one wire which that on another), also encrypting the description of circuit U in some way (so that the adversary cannot modify the unique wire numbers), etc.

While the "fixes" suggested above do appear in our final solution, they are not sufficient by themselves to obtain a secure construction. The main problem is an adversary using an encrypted wire value obtained in one execution of U to modify the corresponding wire value in another execution. We illustrate this problem with the following example. Consider a circuit C which has a secret signing key inbuilt inside. C takes as input a string m of some fixed size and outputs a signature on it iff m is an even number (C outputs $\perp$ otherwise). The circuit C has two parts. The first one checks whether the input is even and sets a "flag" wire to be 1 if so. The second part uses this flag value and outputs a signature iff it is 1. Now, we would like to obfuscate such a circuit C.

Clearly, in the ideal world, an adversary can produce a signature on an odd number only with negligible probability. However, in the real world, the adversary can first execute the circuit with an even number as input and store the encrypted wire values. Next, the adversary executes the circuit with an odd number. Indeed, the first part will execute and set the value on the flag wire to be 0. However at this point, the adversary can use the encrypted wire values obtained in the previous execution (where the flag was 1) to try to modify the wire values in the current execution. Thus, if the adversary is successful in changing the flag value coming from the first part, the second part of C will output a signature on the odd number. While this toy example already illustrates the problem, we remark that an adversary can even launch more sophisticated attacks in which he iteratively changes the wire values and observes the output. In fact, it can be shown that it is possible to completely recover a source code (or the description of a circuit), whose input/output behavior is consistent with the input/output behavior of C observed so far.

The above problem can be fixed with a stateful token as follows. The token will additionally insert a nonce (randomly generated once for each execution) in all the encrypted wire values for an execution. The token will answer a query iff the encrypted wire values specified in the request have the right nonce. Now, since except with negligible probability, two different executions of the obfuscated circuit will have different nonce values, the encrypted wire values obtained in one execution are not useful in another. Unfortunately, such an option is not available in our setting. Our stateless token treats all queries uniformly and is oblivious to which queries belong to which execution, how many executions are there, etc. To solve the above problem, we design and analyze a different mechanism through which the (stateless) token is able to assign a single random nonce (called execution identity) for an entire execution. We prove that our mechanism ensures that it is hard to have two "different executions" of the circuit for the same execution identity. In other words, fixing the execution identity essentially fixes the value on all the wires of the circuit being evaluated.

6.2.2 Our Construction

In this section, we present, for any two-party stateless oblivious-sender reactive functionality $f(\cdot, \cdot)$, a protocol $\Pi = (P_1, P_2)$ that securely implements $f(\cdot, \cdot)$ in the $\mathcal{F}_{wrap}^{stateless}$-hybrid model.

The Sender. Let $C := C_\kappa$ be the circuit realizing $f(\cdot,\cdot)$ for security parameter κ. P_1 creates token $\mathcal{T}$ which has the following secrets inbuilt: the key S of a non-malleable secret key encryption scheme, the secret key K of a cryptographic MAC scheme and a function G drawn from a pseudorandom function ensemble (for convenience, by "P_2 queries $\mathcal{T}$ on x", we would mean that P_2 sends a run query to $\mathcal{F}_{wrap}^{stateless}$ for token $\mathcal{T}$ with input x). The token $\mathcal{T}$ handles *five* types of queries in total. We will describe the functionality of $\mathcal{T}$ in detail later as we go along.

Given circuit C, we now construct its *obfuscation*. Without loss of generality, let C consist only of fan-in 2 NAND gates. Let C implement the function $C : \{0,1\}^m \rightarrow \{0,1\}^n$. Given the circuit C, we now construct a new circuit OTC. Let U be a universal circuit implementing the function $U : \{0,1\}^m \times \{0,1\}^{|C|} \rightarrow \{0,1\}^n$ such that $U(x, C) = C(x)$. We uniquely number all the wires (including the input and output wires) of the circuit U such that the $m + |C|$ input wires are assigned numbers from 1 to $m + |C|$ and n output wires are assigned numbers from $m + |C| + 1$ to $m + |C| + n$. The numbers assigned to the internal wires could be arbitrary as long as all the wires in the circuit have unique numbers. In other words, a wire w is an input wire iff $w \in [m + |C|]$ and an output wire iff $w \in \{m + |C| + 1, \ldots, m + |C| + n\}$. Select a random obfuscation identity O_{ID} such that $|O_{ID}|$ is at least super-logarithmic in the security parameter κ. Now for each gate g in the circuit U (defined by input wires a, b and output wire c), compute the ciphertext $C_g = Enc_S(a, b, c, O_{ID})$ where $Enc_S(m)$ represents a non-malleable encryption of message m with the key S. For each wire w such that $w \in \{m + 1, \ldots, m + |C|\}$, compute the ciphertext $C_w = Enc_S(w, O_{ID}, b_w)$ where b_w is the $(w - m)^{\text{th}}$ bit of the circuit C (when represented as an input string for the universal circuit U). Finally, compute a "header" ciphertext $C_h = Enc_S(O_{ID}, m, |C|, n)$.

The set of computed ciphertexts represent the obfuscated circuit OTC. The sender sends this circuit to the receiver and the code of token $\mathcal{T}$ to $\mathcal{F}_{wrap}^{stateless}$. This completes the description of the sender.

The Receiver. On receiving the obfuscated circuit OTC, the receiver P_2 proceeds as follow on a given input $x \in \{0,1\}^m$.s:

- **Input Encryption Phase.** For each $i \in [m]$, compute:

$$H_i = H(H_{i-1}, x_i, i)$$

Where H is a collision resistant hash function (CRHF), H_0 is an all zero string of appropriate length and x_i is the i^{th} bit of the string x. CRHFs are only assumed for simplicity. Later we sketch how to relax this assumption to use only universal one-way hash functions (UOWHF). Now, make a query of *type 1* to the token $\mathcal{T}$ sending it the computed H_m and the header ciphertext C_h. Upon receiving a query of type 1, the token $\mathcal{T}$ assigns an *execution identity* associated with this particular input x as follows. Token $\mathcal{T}$ first recovers the values $O_{ID}, m, |C|$ and n by decrypting C_h with the secret key S. $\mathcal{T}$ then selects a random execution identity E_{ID} by applying the inbuilt pseudorandom function G on the string $H_m||O_{ID}||m||(|C|)||n$ (where $||$ denotes concatenation). It computes $M_m = Enc_S(H_m, E_{ID}, O_{ID}, m, |C|, n)$ and outputs $M_m, Mac_K(M_m)$ where $Mac_K(m)$ represents a secure MAC of the message m computed with the key K.

Now for $i \in \{m, \ldots, 1\}$ the obfuscated circuit OTC makes a query of *type 2* by sending $M_i, Mac_K(M_i), H_{i-1}, x_i, i$ to the token $\mathcal{T}$. A query of type 2 is used to generate the encryption of the input wires 1 to m as follows. Upon receiving such a query, $\mathcal{T}$ first recovers $H_i, E_{ID}, O_{ID}, m, |C|, n$ from M_i and verifies that (a) $Mac_K(M_i)$ is a valid MAC of M_i with the key K, (b) $H_i = H(H_{i-1}, x_i, i)$, and, (c) $i \leq m$. $\mathcal{T}$ outputs $\perp$ if any of these checks fail. The token $\mathcal{T}$ then computes $E_i = Enc_S(i, O_{ID}, E_{ID}, x_i)$ and outputs $E_i, Mac_K(E_i)$. If $i > 1$, $\mathcal{T}$ further computes $M_{i-1} = Enc_S(H_{i-1}, E_{ID}, O_{ID}, m, |C|, n)$ and outputs $M_{i-1}, Mac_K(M_{i-1})$.

For $i \in \{m + 1, \ldots, m + |C|\}$ the circuit OTC makes a query of *type 3* by sending $(M_m, Mac_K(M_m))$ and C_i. A query of type 3 is used to generate the encryption of inputs wires from $m + 1$ to $m + |C|$. Upon receiving such a query, $\mathcal{T}$ recovers E_{ID}, O_{ID}, b_i from M_m and C_i and ensures that the associated MAC is valid and that $i \in \{m + 1, \ldots, m + |C|\}$. The token $\mathcal{T}$ then computes $E_i = Enc_S(i, O_{ID}, E_{ID}, b_i)$ and outputs $E_i, Mac_K(E_i)$.

Recall that the input to the universal circuit U is $x' = x||C$. At the end of this phase, the circuit OTC has computed $E_i = Enc_S(i, O_{ID}, E_{ID}, x'_i)$ and $Mac_K(E_i)$ for each input wire i of the universal circuit U.

- **Circuit Evaluation Phase.** The evaluation of the universal circuit U is done gate by gate in the natural order by issuing a query of *type 4* to the token $\mathcal{T}$ for each gate g. For a gate g (defined by the incoming wires a, b and the outgoing wire c), issue a query of type 4 sending $E_a, Mac_K(E_a), E_b, Mac_K(E_b), C_g$ to $\mathcal{T}$ where $E_a = Enc_S(a, O_{ID}, E_{ID}, v_a)$ and $E_c = Enc_S(b, O_{ID}, E_{ID}, v_b)$. After appropriate checks, the token $\mathcal{T}$ replies back with $E_c = Enc_S(c, O_{ID}, E_{ID}, v_c)$ where $v_c = v_a$ **NAND** v_b.

  At the end of this phase, the circuit OTC has computed $E_i = Enc_S(i, O_{ID}, E_{ID}, v_i)$ and $Mac_K(E_i)$ for each output wire i of the universal circuit U.

- **Output Decryption Phase.** The actual output is computed from the encrypted output wires by using queries of *type 5*. For each output gate i, P_2 issues a query of type 5 sending $E_i = Enc_S(i, O_{ID}, E_{ID}, v_i), Mac_K(E_i)$ and C_h to the token $\mathcal{T}$. $\mathcal{T}$ decrypts E_i and C_h, verifies that i is indeed an output wire (i.e., $m + |C| + 1 \leq i \leq m + |C| + n$), that the obfuscation identities in E_i and C_h match, and that the given MAC is valid and finally outputs the output wire value v_i.

Throughout the above, if an obvious cheating attempt is detected in a query by the token $\mathcal{T}$, it outputs $\perp$. This completes the description of the receiver.

We now sketch the modifications required to relax the CRHF assumption. Similar to the techniques of Naor and Yung [NY89], instead of letting the user choose the value on which H is being applied, the token can supply a random string of its own to be hashed along with the user values. Thus, while constructing the hash chain, the user would query the token with (H_{i-1}, x_i, i) and get a random number to be hashed along with it to compute H_i (this random number would be checked for correctness by the token during the hash chain "unwinding" process). Using techniques from [NY89], it can be shown that the UOWHF property is sufficient for proving security of this modified construction. UOWHF, in turn, are known to exist based on any OWF [Rom90].

6.2.3 Proof of Security

The core of our security analysis is the following technical lemma which roughly states that given an obfuscated circuit and access to the token $\mathcal{T}$, it is hard to have two "different executions" of the circuit for the same execution identity E_{ID}. In other words, fixing E_{ID} (for an obfuscated circuit with obfuscation identity O_{ID}) essentially fixes the value on all the wires of the universal circuit U.

Lemma 23. *For a given circuit C, generate the corresponding obfuscated circuit OTC and token $\mathcal{T}$. Let O_{ID} denote the corresponding obfuscation identity. Then for every PPT algorithm A and every circuit C:*

$$Pr[A^{\mathcal{T}}(OTC) = (F_0, Mac_K(F_0), F_1, Mac_K(F_1))] \leq negl(\kappa)$$

where $F_0 = Enc_S(i, O_{ID}, E_{ID}, 0)$, $F_1 = Enc_S(i, O_{ID}, E_{ID}, 1)$ and i is the index number of a wire in the universal circuit U.

Proof We prove the above lemma by contradiction. Assume there exists such an i. We first consider the case when i is an input wire (i.e., $i \in [m + |C|]$). We have the following two subcases:

Case 1: $i \leq m$ Examine the queries which the adversary makes. We first show that both F_0 and F_1 must have been given out by the token in response to queries of type 2. This is because, if not, one of the following must be true:

- Either F_0 or F_1 was not given out as response to any query at all. In this case, it is easy to see that the algorithm A can be used to construct a forger for the MAC scheme.

- Either F_0 or F_1 was given out as response to a query of type 3. That query request must have included a header with obfuscation identity O_{ID}. Let that header be an encryption of $O_{ID}, m', |C|', n'$. Clearly $m' < i$ (else the token would have replied with $\perp$). Hence $m' \neq m$. In this case, it can be shown that the attacker was able to break the security of the non-malleable encryption scheme Enc (by malleating the header ciphertext and producing a ciphertext of a related message).

- Either F_0 or F_1 was given out as response to a query of any other type. We first note that only type 4 queries might give a MAC on the encryption of a message of this format. That query request (of type 4) must have included a ciphertext of the form $Enc_S(a, b, i, O_{ID})$. However as part of obfuscated circuit OTC, the attacker is only provided with ciphertexts of the form $Enc_S(a, b, c, O_{ID})$ where c is an internal wire (and hence $c \neq i$). In this case again, it can be shown that the attacker was able to break the security of the encryption scheme Enc (by malleating a ciphertext corresponding to a circuit gate and producing a ciphertext of a related message).

Hence, we have established that the attacker A must have received both F_0 and F_1 in response to queries of type 2. Now we consider the following subcases:

- The token $\mathcal{T}$ assigned the same execution identity E_{ID} for two different queries (of type 1) which included, as part of the request, two different hash chain tips H_m and H'_m. Recall that the execution identity E_{ID} is computed by applying the pseudorandom function on the received hash chain tip (and other strings). Since E_{ID} is super logarithmic in the security parameter, the probability of these different inputs to the pseudorandom function leading to the same output E_{ID} is negligible. Hence the probability of this case happening is negligible in the security parameter.

- There exists only a single hash chain tip which, when included as part of the request to a query of type 1, leads the token $\mathcal{T}$ to choose the execution identity E_{ID}. In this case, it can be shown that the attacker algorithm can either be used to construct a forger for the MAC scheme or to compute collisions in the hash function H.

Case 2: $m + |C| \geq i > m$ We first show that both F_0 and F_1 must have been given out by the token in response to queries of type 3. This is because, if not, one of the following must be true:

- Either F_0 or F_1 was not given out as response to any query at all. In this case, it is easy to see that the algorithm A can be used to construct a forger for the MAC scheme.

- Either F_0 or F_1 was given out as response to a query of type 2. We look at all the ciphertexts of the form $E_i = Enc_S(i, O_{ID}, E_{ID}, x_i)$ and the associated MAC $Mac_K(E_i)$ given out as part of the response to a query of type 2. If $i > m$, it can be shown that the attacker algorithm can either be used to construct a forger for the MAC scheme or to break the security of the non-malleable encryption scheme Enc.

- Either F_0 or F_1 was given out as response to a query of any other type. We first note that only type 4 queries might give a MAC on the encryption of a message of this format. That query request (of type 4) must have included a ciphertext of the form $Enc_S(a, b, i, O_{ID})$. However as part of obfuscated circuit OTC, the attacker is only provided with ciphertexts of the form $Enc_S(a, b, c, O_{ID})$ where c is an internal wire (and hence $c \neq i$). In this case again, it can be shown that the attacker was able to break the non-malleability of the encryption scheme Enc (by malleating a ciphertext corresponding to a circuit gate and producing a ciphertext of a related message).

Hence, we have established that the attacker A must have received both F_0 and F_1 in response to queries of type 3. This means that there should exist two queries of type 3 with requests including $Enc_S(i, O_{ID}, 0)$ and $Enc_S(i, O_{ID}, 1)$. However as part of obfuscated circuit OTC, the attacker is only provided with either $Enc_S(i, O_{ID}, 0)$ or $Enc_S(i, O_{ID}, 1)$. Hence, it can be shown that the attacker was able to break the security of the encryption scheme Enc (by malleating a ciphertext corresponding to a circuit wire and producing a ciphertext of a related message).

Now we consider the case when i is an not an input wire (i.e., i is either an internal or an output wire). We first observe that both F_0 and F_1 must have been given out by the token in response to queries of type 4 (this can be shown using techniques similar to the ones used before). Now examine all the queries of type 4 for obfuscation identity O_{ID} and execution identity E_{ID} and record the value on wires of the universal circuit U. Note that each query of type 4 that determines the value on three wires: two incoming and one outgoing. Clearly, there will be two different values recorded for the wire i.

Consider a wire c such that two different values are recorded for it. Let g be the gate in the universal circuit U such that c is its output wire. Let the input wires of g be denoted by a and b. It is easy to see that two different values must have been recorded for at least one of the wires a or b (this is because, as we have shown before, for all c such that two different values are recorded for c, the corresponding two ciphertexts must have been given out as responses to queries of type 4). Applying this argument, iteratively, we get that two different values must have been accorded for an input wire of the circuit U. This is in contradiction to the first part of the proof.

□

Theorem 24. *Assuming the existence of one-way functions, the obfuscation scheme construction in section 6.2.2 is a secure implementation of $f(\cdot,\cdot)$ in the $\mathcal{F}_{wrap}^{stateless}$-hybrid model.*

Proof We show the existence of a simulator Sim for every adversary A corrupting P_2.

Description of the simulator Sim. The simulator generates token $\mathcal{T}$ as described. Let K, S denote the usual strings generated as part of $\mathcal{T}$.

Our simulator handles obfuscation of multiple circuits at the same time, and has access to the corresponding ideal function for each of them. For each such function, as opposed to the description of the circuit C implementing it, simulator is only given access to the ideal functionality $\mathcal{C}$ implementing this function (the ideal functionality) and the size of the circuit $|C|$. To produce the simulated obfuscated code corresponding to the unknown circuit C, simulator picks a random string S such that $|S| = |C|$. Simulator then returns an honestly constructed obfuscated circuit OTS of S. However, the queries to $\mathcal{T}$ made by this obfuscated circuit are handled as follows. The simulator handles all the queries of type 1-4 honestly. However, a query of type 5 (i.e., an output decryption query) is answered in a special way as follows. Recover the values i, O_{ID}, E_{ID}, v_i from the request (and apply the usual checks).

The simulator now looks at the queries of type 4 made so far with obfuscation identity O_{ID} and execution identity E_{ID} and starts recording the value on the wires in the universal circuit U. Note that each query of type 4 determines the value on three wires: two input and one output wire. The simulator aborts if there exists a wire such that two different values are recorded for it.

Define the i-useful set to be the set of all wires in the sub-circuit (of the universal circuit U) which is useful in computing the value on the output wire i. In other words, representing U as a directed graph, i-useful set is the set of all wires whose starting vertex has a path to the starting vertex of wire i. The simulator aborts if there exists a wire in the i-useful set of circuit U such that no value is recorded for it.

After successful completion of the above, the simulator has recorded the bits of the input x which are relevant for computing the output bit i. If the simulator has not recorded the full input string x (i.e., there exist bits which are not recorded and are not relevant to the output bit i), it sets the value of non-recorded bits to random. The simulator then queries $\mathcal{C}$ with the resulting input x' and obtains the output $f(x')$. The simulator hence has the correct value v_i of the wire i and gives it as response to the received type 5 query.

This completes the description of the simulator. We now show that the view of the adversary in the real execution (with an honestly obfuscated code and honest token) is indistinguishable from that in the above simulated execution. We show this by changing each execution of each obfuscated circuit (characterized by an execution identity E_{ID} and an obfuscation identity O_{ID}) one by one from real to simulated and arguing that this does not change the view of the adversary (in a computational sense). For a particular (O_{ID}, E_{ID}) tuple, the following set of hybrids show the indistinguishability of the view of the adversary in the real and the simulated execution.

Hybrid H_0: This hybrid corresponds to the real execution. That is, the simulator has access to the description of the circuit C. The simulator returns the honestly obfuscated circuit OTC and answers all the queries honestly.

Hybrid H_1: This hybrid is identical to the previous one except for the following. The simulator looks at all the queries of type 4 with obfuscation identity O_{ID} and execution identity E_{ID} and records the value on the wires of the universal circuit U (see the description of the simulator). The simulator aborts the experiment if there exists a wire in the circuit such that two different values are recorded for it. The indistinguishability of this hybrid from the previous one follows directly from lemma 23.

Hybrid H_2: This hybrid is identical to the previous one except for the following. The simulator analyzes the value on the wires of the universal circuit U and aborts if one of these values is inconsistent with how the circuit should be evaluated. More precisely, simulator aborts if there exists a gate g in U having a, b as its input wires and c as its output wire such that $v_c \neq v_a$ **NAND** v_b (where v_a, v_b and v_c are values on a, b and c respectively). To see the indistinguishability of this hybrid from the previous one, assume that such a wire c is found. It can be seen that there can not exist a query which gave out the encryption (and the associated MAC) for the wire c in question (as a different value would have been recorded on either wire a or b). Thus, in this case, the attacker algorithm can be used to construct a forger for the MAC scheme.

Hybrid H_3: This hybrid is identical to the previous one except for the following. The simulator analyzes each query of type 5 (i.e., an output decryption query) with obfuscation identity O_{ID} and execution identity E_{ID}. Upon receiving such a query for the output wire i, the simulator aborts the experiment if there exists a wire in the i-useful set of circuit U such that no value is recorded for it. To see the indistinguishability of this hybrid from the previous one, assume that such a wire is found. Then there must exist a gate in U such that no value is recorded for at least one of its input wires and a value is recorded for its output wire. However, there can not exist a query which give out the encryption and the associated MAC for that output wire. Thus, in this case again, the attacker algorithm can be used to construct a forger for the MAC scheme.

Hybrid H_4: This hybrid is identical to the previous one except that the simulator now handles queries of type 5 for obfuscation identity O_{ID} and execution identity E_{ID} as follows. Upon receiving such a query for the output wire i, the simulator retrieves the recorded the bits of the input x which are relevant for computing the output bit i. If the simulator has not recorded the full input string x (i.e., there exist bits which are not recorded and are not relevant to the output bit i), it sets the value of non-recorded bits to random. The simulator then queries the the ideal functionality with the resulting input x' and obtains the output $f(x')$ (P_1's input is implicit). The simulator hence has the correct value v_i of the wire i and gives it as response to the received type 5 query. This hybrid is identical to the previous one since we have a guarantee that the values on all the wires of the circuit U are consistent with the values that would have resulted from a real evaluation of the circuit.

Hybrid H_5: This hybrid is identical to the previous one except for the following. The simulator now gives the simulated obfuscated circuit OTS (as opposed to giving OTC). As in the previous hybrid, the simulator keeps handling all queries of type 1 to 4 honestly and queries of type 5 using the ideal functionality. Observe that the only change in this hybrid from the previous one is in the values of some bits which are encrypted. Hence, it can be shown that the security of the encryption scheme Enc implies the indistinguishability of this hybrid from the previous one.

Observe that hybrid H_5 is the same as our actual simulator. This shows the indistinguishability of the view of the adversary in the real execution from that in the simulated execution.

□

Note that as explained earlier, the assumption that CRHF exist can be relaxed to obtain an improved version of the above theorem .

6.3 Handling Malicious Senders with Stateless Tokens

Now we handle the case of malicious senders. Our starting point is a recent result by Goyal and Sahai ([GS09]). For any two party functionality $f(\cdot,\cdot)$, they construct a *stateless protocol* $\Pi = (P_1, P_2)$, where one of the parties does not need to maintain any state information at all. In their model, in the ideal world, P_1 sends its input x to the ideal functionality for $f(\cdot,\cdot)$. Party P_2 sends its input y_1 to the ideal functionality and obtains $f(x, y_1)$. At any point, P_2 can send the signal *reset* to the ideal functionality. In that case the trusted party sends *reset* to P_1, which goes back to its initial stage. Now P_1 again sends an input to the trusted party. Now P_2 can send possibly different input y_2, and obtain $f(x, y_2)$. For inputs x and y, and an ideal-world adversary S, let $\mathsf{IDEAL}^f_S(x, y)$ denote the output of the honest parties along with the output of the adversary in the ideal world.

In the real world, P_1 is a stateless, and has a single next message function G_1^x for all rounds, and P_2 has the option of interacting with P_1 any number of times. It is shown in [GS09] that such stateless protocols exist for all funtionalities $f(\cdot,\cdot)$. For a real world adversary $\mathcal{A}$, we let $\mathsf{REAL}_{\mathcal{A}}(x, y)$ denote the output of the honest party along with the output of the adversary in the real world. The following theorem is implicit in [GS09].

Theorem 25. *(Stateless Protocol [GS09]) For any two-party functionality $f(\cdot,\cdot)$, there exists a stateless protocol $\Pi = (P_1, P_2)$ such that for every probabilistic polynomial time adversary $\mathcal{A}$, there exists an expected polynomial time ideal world adversary S, such that*

$$\{\, \mathsf{IDEAL}^f_S(x, y) \,\}_{(x,y)} \sim \{\, \mathsf{REAL}_{\mathcal{A}}(x, y) \,\}_{(x,y)}.$$

Let $f(\cdot,\cdot)$ be a stateless sender-oblivious two-party reactive functionality. Now we present a protocol $\Pi'_f = (P'_1, P'_2)$ that securely implements $f(\cdot,\cdot)$ in the $\mathcal{F}^{stateless}_{wrap}$-hybrid model. Let κ be the security parameter. For the purposes of this section only, let com be a non-interactive computationally binding and computationally hiding commitment scheme.

Protocol Π'_f.

- ***Input:*** *P'_1 gets input $x \in \{\, 0,1 \,\}^\kappa$.*
- ***The Protocol:***

  1. *P'_1 proceeds as follows:*
     (a) *P'_1 chooses random string r and computes $\alpha = \mathsf{com}(x; r)$.*
     (b) *Let $\hat{f}_\alpha(\cdot,\cdot)$ be the two-party functionality defined as follows:*
        - *on input β from the first party, and y from the second party, check if β is a valid opening of α. If not, output $\perp$. Else, let $\hat{x}$ be the revealed string. Output $f(\hat{x}, y)$.*

        Let $\pi = (P_1, P_2)$ be a stateless protocol that implements $\hat{f}_\alpha$, as guaranteed in Theorem 25. Let G_1^x be a circuit implementing P_1's next message function (recall that as P_1 is stateless, it has a single next message function for all rounds). Now P'_1 uses Theorem 24 and constructs obfuscated circuit $\hat{G}$ for G_1^x.
     (c) *Finally, P'_1 sends the obfuscated circuit $\hat{G}$ and α to P'_2.*
  2. *P'_2 internally runs party P_2 of protocol π. When P_2 outputs a message m for P_1, party P'_2 sends the message to the obfuscated circuit $\hat{G}$ and obtains response $\hat{m}$. Then it sends $\hat{m}$ to P_2 as P_1's response in π. Thus, P'_2 acts as a man-in-the-middle to $\hat{G}$ and the simulated P_2. Finally, P'_2 outputs P_2's output.*

□

6.3.1 Proof of Security

In this section, we briefly sketch the proofs of security. We handle the case of malicious sender and malicious receiver separately.

Malicious Sender. Let $\mathcal{A}$ be the adversary corrupting P_1'. We first construct a malicious sender $\mathcal{A}^*$ that corrupts P_1 in π. The adversary $\mathcal{A}^*$ runs $\mathcal{A}$ and receives the commitment α and $\hat{G}$. Now it carries out the protocol π with the real P_2, passing messages from P_2 to $\mathcal{A}$ and vice-versa. Let S_1 be the ideal world simulator for $\mathcal{A}^*$ guaranteed by the security of protocol π. Now we construct the ideal world simulator S_1' for $\mathcal{A}$. The simulator S_1' sets up an internal execution between $\mathcal{A}$, $\mathcal{A}^*$ and S_1. It passes messages from $\mathcal{A}$ to $\mathcal{A}^*$ to S_1 and back. Finally, S_1 outputs some string $\tilde{x}$. S_1' sends $\tilde{x}$ to the ideal functionality. It follows from the security of π that the ideal and real views are indistinguishable.

Malicious Receiver. Let $\mathcal{A}$ be the adversary corrupting P_2'. We first construct a malicious receiver $\mathcal{A}^*$ that corrupts P_2 in π. Let S_O be the obfuscating simulator as in Theorem 24. The adversary $\mathcal{A}^*$ picks random strings s and r, and computes $\alpha = \mathsf{com}(s; r)$, and sends α to $\mathcal{A}$. Then, it internally runs an execution between S_O and $\mathcal{A}$. When S_O queries its ideal functionality with message m, $\mathcal{A}^*$ sends m externally to P_1 and receives response $\hat{m}$. It sends $\hat{m}$ to S_O as response to its query. Let S_2 be the ideal world simulator for $\mathcal{A}^*$ guaranteed by Theorem 24. Now we construct the ideal world simulator S_2' for $\mathcal{A}$. The simulator S_2' sets up an internal execution between $\mathcal{A}$, $\mathcal{A}^*$ and S_2. It passes messages from $\mathcal{A}$ to $\mathcal{A}^*$ to S_2 and back. When S_1 outputs some string $\tilde{y}$, S_1' sends $\tilde{y}$ to the ideal functionality and returns the response to S_2. It follows from the security of π that the ideal and real views are indistinguishable.

6.4 Stateful Tamper-proof Hardware

We first observe that the construction in Section 3 (in the non-interactive setting) can be extended in straightforward way to obtain secure protocol for *bounded* reactive functionalities (where the ideal world trusted party forces an a priori fixed bound on the number of times the receiver queries) in the non-interactive setting with stateful hardware. This can also be viewed as a natural extension of the concept of t-time program ([GKR08]) to the malicious sender case. We observe that such a bound is inherent with read-once tamper proof hardware tokens.

Protocols for general reactive functionalities with more complex stateful hardware tokens can be constructed using standard techniques. Similar to before, we first focus on the semi-honest sender case. The task of constructing secure protocols for the semi-honest sender case exactly coincides with the task of constructing secure *stateful obfuscation scheme* with a stateful tamper-proof hardware token. We note that the notion of stateful obfuscation (with a stateful hardware) is implicit in previous work in the literature ([GO96]). Such an obfuscation scheme can be constructed using standard techniques in a straightforward way. Security against a malicious sender can be achieved using the same high-level idea as in Section 3. The sender takes an interactive secure computation protocol for the appropriate reactive functionality and sends the receiver an obfuscated program (with stateful tamper proof hardware) for the next message function of this protocol.

Acknowledgments. We thank Juerg Wullschleger for pointing us to the work in [BCU$^+$] and other helpful comments. We thank Guy Rothblum for useful discussions.

References

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in nc^0. *SIAM J. Comput.*, 36(4):845–888, 2006.

[And08] W. Erik Anderson. On the secure obfuscation of deterministic finite automata. Cryptology ePrint Archive, Report 2008/184, 2008.

[BCS96] Gilles Brassard, Claude Crépeau, and Miklos Santha. Oblivious transfers and intersecting codes. *IEEE Transactions on Information Theory*, 42(6):1769–1780, 1996.

[BCU$^+$] Harry Buhrman, Matthias Christandl, Falk Unger, Stephanie Wehner, and Andreas Winter. Implications of superstrong nonlocality for cryptography. Proceedings of the Royal Society A, 462(2071), pages 1919-1932.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *STOC*, pages 479–488, 1996.

[BG89] Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority (extended announcement). In *FOCS*, pages 468–473. IEEE, 1989.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[Bra93] Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In *CRYPTO*, pages 302–318, 1993.

[Can01a] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[Can01b] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

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

[CK91] Benny Chor and Eyal Kushilevitz. A zero-one law for boolean privacy. *SIAM J. Discrete Math.*, 4(1):36–47, 1991.

[CP92] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *CRYPTO*, pages 89–105, 1992.

[CP93] Ronald Cramer and Torben P. Pedersen. Improved privacy in wallets with observers (extended abstract). In *EUROCRYPT*, pages 329–343, 1993.

[CW79] Larry Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.

[DNW08] Ivan Damgård, Jesper Buus Nielsen, and Daniel Wichs. Isolated proofs of knowledge and isolated zero knowledge. In *EUROCRYPT*, pages 509–526, 2008.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *STOC*, pages 554–563, 1994.

[GHY87] Zvi Galil, Stuart Haber, and Moti Yung. Cryptographic computation: Secure faut-tolerant protocols and the public-key model. In *CRYPTO*, pages 135–155, 1987.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy Rothblum. One-time programs. In *CRYPTO*, Lecture Notes in Computer Science, pages 39–56. Springer, 2008.

[GLM+04] Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic tamper-proof (atp) security: Theoretical foundations for security against hardware tampering. In *TCC*, pages 258–277, 2004.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, Cambridge, UK, 2001.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, Cambridge, UK, 2004.

[GS09] Vipul Goyal and Amit Sahai. Resettably secure computation. In *Eurocrypt*, 2009.

[GV87] Oded Goldreich and Ronen Vainish. How to solve any protocol problem - an efficiency improvement. In *CRYPTO*, pages 73–86, 1987.

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

[HL08] Carmit Hazay and Yehuda Lindell. Constructions of truly practical secure protocols using standards-martcards. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM Conference on Computer and Communications Security*, pages 491–500. ACM, 2008.

[HMQU05a] Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. Universally composable zero-knowledge arguments and commitments from signature cards. In *5th Central European Conference on Cryptology*, page A version is available at http://homepages.cwi.nl/ hofheinz/card.pdf., 2005.

[HMqU05b] Dennis Hofheinz, Jrn Mller-quade, and Dominique Unruh. Universally composable zero-knowledge arguments and commitments from signature cards. In *In Proc. of the 5th Central European Conference on Cryptology MoraviaCrypt 2005*. Mathematical Publications, 2005.

[HR07] Iftach Haitner and Omer Reingold. Statistically-hiding commitment from any one-way function. In *STOC*, pages 1–10, 2007.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.

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

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits ii: Keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308–327, 2006.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.

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

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

[Kil90] Joe Kilian. *Uses of Randomness in Algorithms and Protocols*. MIT Press, 1990.

[Kus92] Eyal Kushilevitz. Privacy and communication complexity. *SIAM J. Discrete Math.*, 5(2):273–284, 1992.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, pages 52–78, 2007.

[MN05] Tal Moran and Moni Naor. Basing cryptographic protocols on tamper-evident seals. In *ICALP*, pages 285–297, 2005.

[MS08] Tal Moran and Gil Segev. David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In *EUROCRYPT*, pages 527–544, 2008.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.

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

[Rab81] Michael O. Rabin. How to exchange secrets with oblivious transfer, 1981.

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

[Yao86] A. Yao. How to generate and share secrets. In *FOCS*, pages 162–167, 1986.

A Reduction from large to small tokens

Here, we give a very informal but self-contained discussion of a reduction from large to small OT tokens. Note that no constants have been optimized here.

We first implement OT tokens for k-bit strings using smaller OT tokens on $\mathcal{O}(\log k)$-bit strings. Let (s^0, s^1) be the two input strings of length k. The sender picks a finite field with $11k$ points $\alpha_1, \ldots, \alpha_{11k}$. Next, the sender picks two random polynomials f_0 and f_1 of degree $10k-1$, such that f_0 matches s^0 at the first k points, and f_1 matches s^1 at the first k points (that is, $f_0(\alpha_1) = s^0_1, \ldots, f_0(\alpha_k) = s^0_k$, and same for f_1 and s^1). The sender sends $10k$ tokens, where, for $1 \leq i \leq 10k$, token M_i implements the following functionality: on input bit b, output $f_b(\alpha_{k+i})$. Let c be the receiver's selection bit. The honest receiver runs each token M_i on c, and obtains $f_c(\alpha_{k+1}), \ldots, f_c(\alpha_{10k})$. Using Lagrange interpolation, the receiver reconstructs the polynomial and evaluates it on the first k points to obtain s_c.

We briefly describe the simulators. First, assume that the receiver is malicious. Observe that if the receiver gets less than $8k$ points for a polynomial f of degree $10k$, then it learns nothing about the polynomial because of the randomization. Now, the simulator must handle the token queries from the receiver. That is, for any $1 \leq i \leq 10k$, the receiver will send a bit c_i to the simulator, and it must reply with a field element. For an initial number of queries, say $4k$, the simulator responds with random field elements. Let's say that after the first $4k$ queries, the majority of queries were for bit c. Now, the simulator sends c to the large string OT token, and obtains s^c. Next, it picks a polynomial of degree $10k$ that agrees with s^c at the first k points, and the random values that the simulator picked for queries corresponding to bit c among the first $4k$ queries. For the rest of the queries, $4k \leq i \leq 10k$, if $c_i = c$, the simulator replies with $f_c(k+i)$, else reply with a random value. A dishonest receiver can get at most $8k$ points for polynomial f_{1-c}, and thus learns nothing.

Next consider the case that the sender is malicious. The simulator queries all the tokens with bit 1, reconstructs the polynomial f_1, and computes s_1. Now, the simulator rewinds the tokens, and runs them with bit 0, reconstructs f_0, and obtains s_0. Then it can construct the large string OT token using the strings (s_0, s_1). The reduction to bit OT is similar, as the argument above can be generalized to any linear code with minimal relative distance at least $1/2$. Since the reduction above yields string OT's with logarithmic length strings, we can use a Hadamard code instead of a Reed-Solomon code as above, to complete the argument. We will give further details of this simple self-contained UC-secure reduction in the full version of this paper.

B Unconditional OTPs for Poly-Size Circuits - Alternate Construction

Here we briefly describe an alternate approach to construct unconditional OTPs for poly-size circuits. The construction is similar to Yao's garbled circuit construction ([Yao86]). In fact, our construction can be thought as an unconditional analogue of Yao's construction in the hardware token model. Let C be a polynomial size circuit, and let κ be the security parameter. The *unconditional garbled circuit* G of C is constructed as follows: for each wire w_i of C, choose key pair (gk_i^0, gk_j^1) of length κ each. Intuitively, gk_i^0 represents bit 0 and gk_i^1 represents bit 1 on wire w_i. Let $g_i : \{0,1\} \times \{0,1\} \rightarrow \{0,1\}$ be a gate in C, with input wires w_j and w_k, and output wire w_l. Given keys corresponding to the bit value of w_j and w_k, the correct key for the output wire is given by the following function:

$$f^{g_i}_{gk_j^0, gk_j^1, gk_k^0, gk_k^1}(gk_j^{\sigma_j}, gk_k^{\sigma_k}) \quad = \quad gk_l^{g_i(\sigma_i, \sigma_j)}. \tag{1}$$

This function can be implemented by log depth circuits, and thus can be realized by an unconditional one-time program, as constructed in the previous sub-section. Let G_i be the unconditional OTP for gate g_i. The unconditional garbled circuit for C consists the $|C|$ unconditional OTPs $G_1, \ldots, G_{|C|}$. Now we present our unconditional OTP for C.

Protocol. *(One-Time Program for poly size circuit C)*

- ***Input:*** *P_1 has input $x \in \{0,1\}^n$.*
- ***Output:*** *P_2 should receive $C(x, y)$, for $y \in \{0,1\}^n$.*
- ***The Protocol:***

  1. *P_1 chooses n random strings $s_1, \ldots, s_n$ of length $|C(1^n, 1^n)|$, and constructs circuit C' such that $C'(x, y) = C(x, y) \oplus_{i=1}^n s_i$. Then P_1 constructs unconditional garbled circuit G for C' as above. Let $G_1, \ldots, G_{|C'|}$ be the unconditional OTPs corresponding to gates of C'. Let $w_{\mathsf{out}_1}, \ldots, w_{\mathsf{out}_l}$ be the output wires of C'. Finally, P_1 sends $(G_1, \ldots, G_{|C'|}, (gk^0_{\mathsf{out}_1}, gk^1_{\mathsf{out}_1}), \ldots, (gk^0_{\mathsf{out}_l}, gk^1_{\mathsf{out}_l}))$ to P_2.*
  2. *Let $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ be the input wires corresponding to P_1 and P_2's inputs in C'. Then,*
     (a) P_1 sends $gk^{u_1}_{x_1}, \ldots gk^{u_n}_{x_n}$ to P_2 in the clear.
     (b) For $1 \leq i \leq n$ P_1 sends $(\mathsf{create}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_i, (gk_0^{v_i} \circ s_i, gk_1^{v_i} \circ s_i))$ to OTM.
  3. *P_2 evaluates G gate by gate. For each gate g_i of G, P_2 executes the unconditional OTP for that gate, G_i on the labels of the input wires. Let $gk^{\alpha_i}_{\mathsf{out}_i}$ be the value of w_{out_i}, for $1 \leq i \leq l$. Set $z = \alpha_1 \circ \ldots \circ \alpha_l$. Finally, P_2 outputs $z \bigoplus_{i=1}^n s_i$.*

We informally argue the security of the above protocol. For further details, see the full version of this paper. We begin by describing the construction of a *fake garbled circuit* $\tilde{G}$ for a circuit C' in the hardware-token model: for each wire w_i of C', choose pair of keys (gk_i^0, gk_i^1) as before. For each non-output gate g_i, instead of implementing the selection function as in Equation 1, construct unconditional OTP for the following function:

$$f^{g}_{gk_j^0, gk_j^1, gk_k^0, gk_k^1}(gk_j^{\sigma_j}, gk_k^{\sigma_k}) \quad = \quad gk_l^0.$$

Note that the output of the function is always gk_l^0, irrespective of the value of input wires, or the type of gate. Thus, the fake garbled circuit can always be made to output any fixed value $z = \alpha_1 \circ \ldots \circ \alpha_l$ (for $\alpha_i \in \{0,1\}$) by fixing the the output keys as above.

We now describe the simulator S. Simulator S constructs the fake garbled circuit as follows: for each gate g_i of C, simulator S runs the corresponding OTP simulator Sim_{g_i}, and sends the simulated OTP to P_2 (that is, sends the corresponding create messages). Now P_2 starts evaluating the circuit. First, P_2 queries the initial OT tokens to obtain keys corresponding to its (P_2's) input $y = y_1 \ldots y_n$. For all but the last such query, S answers

with $gk_i^0 \circ s_i$, where s_i is a random string. Let $(\mathsf{run}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_j, y_j)$ be the last query. When P_2 asks the final query, its input y is completely determined. S sends y to the ideal functionality and obtains $C(x, y)$. Now S sets $s_j = C(x, y) \oplus (\bigoplus_{i=1, i\neq j}^{n} s_i)$, and sends $gk_j^0 \circ s_j$. Party P_2 also evaluates the OTPs corresponding to the gates of C. Simulator S forwards these queries to the corresponding OTP simulator.

The proof of security follows from a hybrid argument. We define hybrids $\mathcal{H}_0, \ldots, \mathcal{H}_{|C|}$. Hybrid $\mathcal{H}_0$ is the real execution. We proceed gate by gate, and in each successive hybrid, the garbled circuit computing the gate table of the next gate is replaced by the simulated garbled circuit. Thus, two successive hybrids $\mathcal{H}_i$ and $\mathcal{H}_{i+1}$ differ only in one unconditional OTP: in $\mathcal{H}_i$, the garbled table of gate g_i is implemented by the real unconditional OTP, while in $\mathcal{H}_{i+1}$, it is implemented by a simulated OTP. It is easy to see that if any two successive hybrids are statistically far apart, then that contradicts Claim 10.