Truly Efficient String Oblivious Transfer Using Resettable Tamper-Proof Tokens

Vladimir Kolesnikov

Alcatel-Lucent Bell Laboratories, Murray Hill, NJ 07974,USA
kolesnikov@research.bell-labs.com

Abstract. SFE requires expensive public key operations for each input bit of the function. This cost can be avoided by using tamper-proof hardware. However, all known efficient techniques require the hardware to have long-term secure storage and to be resistant to reset or duplication attacks. This is due to the intrinsic use of counters or erasures. Known techniques that use resettable tokens rely on expensive primitives, such as generic concurrent ZK, and are out of reach of practice.
We propose a *truly efficient* String Oblivious Transfer (OT) technique relying on *resettable* (actually, *stateless*) tamper-proof token. Our protocols require between 6 and 27 *symmetric key* operations, depending on the model. Our OT is secure against covert sender and malicious receiver, and is sequentially composable.
If the token is semi-honest (e.g. if it is provided by a trusted entity, but adversarily initialized), then our protocol is secure against malicious adversaries in concurrent execution setting.
Only one party is required to provide the token, which makes it appropriate for typical asymmetric client-server scenarios (banking, TV, etc.)

1 Introduction

We propose efficiency improvements of two-party Secure Function Evaluation (SFE). We take advantage of tamper-proof hardware issued by one of the participants of the computation. We restrict ourselves to *stateless* (thus resettable) tokens to avoid the cost of adding long-term secure storage and to protect against a class of physical attacks on the hardware.

Two-party general (SFE) allows two parties to evaluate any function on their respective inputs x and y, while maintaining privacy of both x and y. SFE is (justifiably) a subject of an immense amount of research, e.g. [28, 29, 23]. Efficient SFE algorithms enable a variety of electronic transactions, previously impossible due to mutual mistrust of participants. Examples include auctions [27, 8, 12], contract signing [11], set intersection [18], etc. As computation and communication resources have increased, SFE of many useful functions has become practical for common use.

Still, SFE of most of today's functions of interest is either completely out of reach of practicality, or carries costs sufficient to deter would-be

adopters, who instead choose stronger trust models, entice users to give up their privacy with incentives, or use similar crypto-workarounds. We believe that truly practical efficiency is required for SFE to see more use in real-life applications.

To achieve this, in addition to improving protocols in the standard model, it is useful to "give ourselves" some help in the form of less demanding (yet acceptable) security properties, such as the recently proposed covert adversaries model [3]. When it fits the setting, we could also rely on additional assumptions that the world has been long using, such as simple tamper-proof (or tamper-resistant) tokens. We note that token-supported protocols received a lot of attention recently, e.g., [21, 7, 18, 14, 9, 10, 15]. To allow cheaper tokens and higher confidence in the security of the system, it is desired to minimize the assumptions on the hardware, while still reaping the performance benefit.

A weaker hardware model that recently received a lot of attention, e.g. [7, 17, 15], is that of *resettable* token. Here, the adversary, e.g., by interrupting power supply or applying highly targeted laser or electromagnetic radiation, is able to manipulate the token and reset its internal variables (e.g., a counter) to the initial state[1]. This realistic capability trivially breaks most of currently known protocols taking advantage of secure tokens. Similar effect is achieved if an attacker is able to obtain a clone of the card, e.g. by insider attacks during manufacturing process, etc. From another perspective, it may be convenient to allow legitimate clone cards to allow the user operate several instances of itself independently. For example, a person may sign up for a telephone, wireless, and TV services from the same provider. It may be convenient to simply provide him with several identical tokens, which he can use interchangeably in all of his devices. It is easy to see that such deployment clearly requires resettable tokens.

We consider even weaker *stateless* tokens. This gives an important advantage of avoiding the manufacturing cost of long-term secure storage.

Our setting, goals and approach. We consider two-party SFE aided by stateless tamper-proof tokens. In fact, our protocols work in the very important client-server setting, where only one party has the capability to issue tokens. In practice, this occurs in TV, phone, cellular and internet service provision, banking, etc. We aim to enable efficient computation of a variety of functions on moderate-size inputs. On the client side, for example, inputs could be viewing preferences, browsing

[1] Not all is necessarily lost with this adversarial capability. For example, it is *much* harder to reprogram the token, simply by resetting certain bits, to, e.g., output its keys. Thus the resettable token model appears reasonable.

history, etc. Server's inputs may include content or other digital rights to be transferred to the client. We stress that our solutions are general and can be used to compute any function.

We note the inherently asymmetric trust model in the client-server setting. Servers are usually established businesses who are likely unwilling to cheat, especially if there is a chance of being caught. The risk of loss of business and public embarrassment is a strong deterrent. Therefore, it is natural and sufficient to model servers as covert adversaries. Also, servers are capable of issuing tamper-proof tokens, and, in many cases, already routinely do. Clients, on the other hand, have much less credibility, and may be more willing to attempt interfering with the protocol and the provided token. Therefore, it is appropriate to view them as malicious adversaries, and to aim to reduce the trust assumptions on the token.

Oblivious Transfer (OT) is often a bottleneck in SFE, due to the high cost of the required public key operations. Our main contribution is, we believe, the first truly efficient protocol to use a resettable token to replace public key operations in OT (and SFE) with symmetric primitives.

The hardware assumption: costs and security comparison with number-theoretic OT. At first, it may seem that the cost of deployment of tokens is greater than that of using standard OT based on public key primitives. We argue that this is often not the case. As noted above, tamper-resistant tokens are often already deployed in the form of cell phones' SIM cards, TV cable smart cards, etc. In these important scenarios, tokens are "free". Otherwise, they cost from $2 in retail (e.g., [1]). At the same time, using standard OT may necessitate much higher costs in terms of increased CPU requirements, much slower processing, and decreased battery life (in mobile devices). Further, evaluating garbled circuits (our main application) involves transferring them (megabytes or gigabytes of data, especially in the malicious model) to the client. In large-scale deployment of SFE (e.g., by banks and service providers), these communication costs cannot be afforded; fortunately, they can be avoided by generating circuits from a seed by the server-issued token [20]. This solution requires reliance on tamper-proof/tamper-resistant tokens, forces their use, fits well with our setting, and further justifies it.

One may also question the hardware assumption and rightfully presume that a sufficiently strong attacker can always break into the token. We argue that in many applications we envision, the cost of the break far exceeds the gain (e.g., free cable TV for one user). Therefore, the barrier raised by even weak tamper resistance is high enough for typical applications, and constitutes a reasonable assumption.

1.1 Our Contributions and Outline of the Work

We consider two-party SFE, where one party (Sender S) is able to issue a token T to the receiver R. T is assumed to be tamper-proof, but resettable. Our main contribution is a new efficient string OT protocol in the covert adversaries model [3], which takes advantage of T. To our knowledge, ours is the first truly efficient protocol that gets rid of public key operations in this setting. Our protocol requires a total of 6 symmetric key operations if the token is semi-honest and S and R are malicious, and 27 such operations in the covert adversaries model (with deterrence $\epsilon = 1/2$). This immediately leads to corresponding improvements in 2-party SFE.

We start with the overview of related work in Sect. 2 and preliminaries in Sect. 3. For clarity of presentation, we first present the semi-honest variant of the OT scheme in Sect. 4. We then show how to achieve security against covert sender and fully malicious receiver in Sect. 4.1. In Sect. 4.2, we discuss aspects of simultaneous execution and sequential composition, and show security in this case.

Under an additional reasonable assumption that the tokens are trusted to run the specified code (e.g. when standard tokens are provided by a trusted manufacturer), our protocol is composable concurrently and secure against malicious S and R (Sect. 4.3).

1.2 Main Idea of Our Approach

Our main tool is Strong Pseudorandom Permutation Generators (SPRPG). We capitalize on the observation that it is difficult to find a collision in a SPRPG F under independent secret keys k_0, k_1. At the same time, a value in the range of F does not reveal which key was used to arrive at this value (via the evaluation of F). That is, R can know a preimage of the (random to S) value under (only one) key of R's choice. This is naturally used to construct efficient OT protocols. We use T to securely store the keys and provide R the evaluation interface to F (but not F^{-1}).

We stress that the use of SPRPG (vs PRPG) does not introduce additional assumptions, and has no performance impact (Sect. 3.2).

We give additional intuition for semi-honest and covert setting protocols in Sect. 4 and 4.1, before presenting corresponding protocols.

2 Related Work

There is a massive body of work on SFE, and, in particular, two-party SFE, of which most relevant to this work is Yao's garbled circuit [28, 29, 23] and the techniques of its implementation in the malicious setting.

The complete solutions are excellently presented in [24, 3, 16]. We work in the recently introduced security model of covert adversaries [3], which we find to help significantly in design of efficient protocols. SFE solutions of [3, 16] are presented in this model. Our work is an improvement of an important building block used by above (and many other) solutions.

There has been a recent surge of interest in SFE supported by tamper-proof tokens [21, 7, 18, 14, 10, 15], and, specifically, resettable tokens [7, 15]. Our work is different from the above, as follows.

Firstly, of the above, only [7, 15] consider resettable tokens. Work of Katz [21] was the first to formalize tamper-proof token model, and show sufficiency for Universally Composable (UC) security. This is mainly a feasibility and definitional result. Chandran et al. [7] extended the results of Katz, considered resettable tokens, and constructed UC-secure protocols. Goyal and Sahai [17] constructed protocols secure against resettable adversaries (and not just tokens). Very recently, Goyal et al. [15] considered the general question of basing cryptography on tamper-proof tokens, under minimal computational assumptions. As one of their results, they showed that stateless tokens and one-way functions are sufficient for UC-secure computation. Damgård et al. [10] consider the setting where the tamper-proof token may leak a limited amount of information back to its issuer. They show how to achieve UC-secure computation in this setting. We remark that the protocols of [21, 7, 10, 17, 15] achieve stronger UC security. Our protocols either have weaker security guarantee, or rely on a semi-honest token. In exchange, we use only a few (6 to 27) symmetric key operations and thus offer truly practical performance.

Hazay and Lindell [18] give a very practical protocol which takes advantage of secure tokens (standard smart cards). As compared to our work, they solve a specific problem, using a much stronger assumption (non-resettable semi-honest smart card) than we do.

Goldwasser et al. [14] consider one-time (or, generally, k-time) programs and proofs, and rely on tamper-proof tokens to achieve it. Their token security model is different from ours – it is non-resettable, but it is assumed that data that is ever accessed by the token's program may be leaked to the adversary. This assumption makes impossible the use of pseudorandomness in their construction, and all the random values (i.e. wire keys used by garbled circuit) ever used by the program need to be preloaded explicitly. While opening a significant new application of secure hardware, these tokens can only be used for a predetermined number of executions. The most important difference, however, is that we achieve security with resettable token.

3 Preliminaries

3.1 Notation and Security Model

We will use Pseudorandom Permutation Generators (PRPG) and Strong PRPG (SPRPG). While we don't need SPRPG properties in all uses of PRPG, for simplicity we refer to all of them as SPRPG, and denote by F. We denote the domain and range of F by D, and the key space of F by K. For simplicity, we set $K = D = \{0,1\}^n$, where n is the security parameter. We denote OT Sender by S, OT Receiver by R and the token by T. In our protocols, R will be testing the correctness of the actions of S and T. Objects associated with testing would often have subscript $_t$, for example, D_t is the subdomain of D, reserved for test queries. By "OT" we mean "string OT".

We prove security against covert adversaries. Aumann and Lindell [3] propose several definitions of security in this setting, with various guarantees. Our protocols are secure in their strongest model (strong explicit cheat formulation). Informally, this guarantees that covert adversary does not learn anything about honest party's input if he is caught.

We stress that this notion of security requires *full simulation* in the ideal world (where cheating attempts are modeled by a designated query). Further, this notion is modularly sequentially composable [3]. In particular, this means that our OT can be composed, e.g., with a garbled circuit protocol, and result in a corresponding SFE protocol in the covert model.

For simplicity of presentation, we often omit the introduction of the deterrence parameter ϵ (probability of being caught when cheating) in our calculation. Our constructions are given w.r.t. $\epsilon = 1/2$, but are readily generalized to ϵ polynomially close to 1.

We assume that tokens cannot be internally observed or tampered with; we only allow R to reset the token at will. We leave it as an interesting direction to investigate efficient security in presence of both resets and tampering (perhaps using the techniques of Gennaro et al. [13]).

3.2 Strong Pseudorandom Permutation Generators (SPRPG)

We assume reader's familiarity with PRPG. SPRPG (sometimes also referred to as *Super PRPG*) is a natural extension of the notion of PRPG which considers efficiently invertible permutations. Informally, in terms of security, the difference between the two notions is that while PRPG allows the adversary to query the "forward evaluation" oracle, SPRPG allows to query both "forward evaluation" and "inversion" oracles.

Luby and Rackoff [25] showed how to construct SPRPG from PRFG. Therefore, our use of SPRPG does not require an additional assumption.

Further, invertibility has been a design requirement of almost all block ciphers, notably, of AES. It appears that the trend will continue in the future as well. Therefore, in practical terms, our reliance on SPRPG does not incur any performance penalty. We envision instantiation of SPRPG in our constructions with AES.

We note that most of cryptographic protocols literature relies on PRFG/PRPG, and largely ignores SPRPG. The novelty of our approach is in part in using tools just outside of standard "crypto toolbox". Use of invertible PRPG seems to be promising in tamper-proof token designs.

3.3 Oblivious Transfer

Garbled Circuit (GC), excellently presented in [23], is the standard and the most efficient method of two-party SFE of boolean circuits. An important (and often the most computationally expensive) step of GC and other SFE techniques is the oblivious transfer (OT) of a secret (one of the two held by the sender), depending on the choice of the receiver.

The 1-out-of-2 OT is a two-party protocol. The *sender* S has two secrets s_0, s_1, and the *receiver* R has an selection bit $i \in \{0,1\}$. Upon completion, R learns s_i, but nothing about s_{1-i}, and S learns nothing about i. OT is a widely studied primitive in the standard model [5, 2], with improved implementations in the Random Oracle model [26, 6].

OT Using Tamper-Proof Hardware While existing OT constructions are simple, they are not very efficient due to use of several public key operations for each OT. If parties possess tamper-proof hardware tokens, such as smart cards, the use of public key operations can be avoided as follows (shown in the semi-honest model).

Suppose, S creates and gives R the following token T. Equipped with a non-resettable counter (initially set to 0), and seeded with the secret key k chosen by S, T has the following interface. Let F be a PRPG. On input i, where $i \in \{0,1\}$, T sets $counter = counter + 2$ and outputs $F_k(counter + i - 2)$. Now, to execute the j-th instance of OT, the sender sends to the receiver $(F_k(counter_j) \oplus s_0^j, F_k(counter_j + 1) \oplus s_1^j)$, where $counter_j$ is the value of the counter used for evaluation of j-th OT. Receiver obtains $F_k(counter_j + i)$ by calling T with argument i, and obtains s_i^j. Because of the properties of PRPG and since the token guarantees that F_k will not be evaluated on both $counter_j$ and $counter_j + 1$, the receiver will be able to obtain only one of s_0, s_1. Further, S does not learn the receiver's input i, since nothing is sent to S.

This efficient protocol can be naturally modified to withstand covert adversaries, but it (and other natural protocols) fails trivially if T is reset.

4 OT With Resettable Tamper-Proof Cards

We build our presentation incrementally. We find it instructive to first present the protocol for a hybrid semi-honest model, where the only allowed malicious behavior is for R to arbitrarily query the token T, and to reset T to the state T was in when it was received by R. (This additional power is necessary to exclude trivial solutions secure in the semi-honest model, such as relying on the semi-honest parties to not reset the token.) We show how to efficiently handle malicious behavior in Sect. 4.1.

In our construction, we use Strong PRPG (SPRPG), i.e. a PRPG that allows efficient inversion. As discussed in Sect. 3.2, this does not constitute an additional assumption and causes no performance penalty (we envision using AES as SPRPG).

The main idea of our construction is to limit the functionality of the token to evaluate the SPRPG F *in the forward direction only*, to keep no state, and to have S load two random SPRPG keys on the token. Then, the simple but crucial observation is that it is infeasible for the token receiver to find two preimages (under the two keys) of any element in the range D of F. (In fact at least one of the preimages will look random to him.) At the same time, he can trivially generate a random element on the range with the preimage under either of the keys. The token creator S, who knows both keys, can invert F and compute both preimages. If he uses them to encrypt the respective input secrets, the receiver can recover the secret of his choice. We discuss the intuition for protection against malicious actions before presenting our main protocol in Sect. 4.1.

Construction 1 *(OT, stateless token, augmented semi-honest model)*

1. Initialization. *The token T is created by sender S, seeded with keys k_0, k_1, randomly chosen by S, and given to receiver R. T answers any number of queries of the form $Q(i, x) = F_{k_i}(x)$.*
2. OT Protocol execution. *S has inputs $s_0, s_1 \in D$. R has input $i \in \{0, 1\}$.*
   (a) *R chooses $x \in_R D$, and queries the token $v = Q(i, x) = F_{k_i}(x)$. R sends v to S.*
   (b) *S computes encryption keys $ek_0 = F_{k_0}^{-1}(v)$, and $ek_1 = F_{k_1}^{-1}(v)$. He then sends $(e_0, e_1) = (F_{ek_0}(s_0), F_{ek_1}(s_1))$ to R.*
   (c) *R computes and outputs $s_i = F_x^{-1}(e_i)$.*

Observation 1 *We need to be careful with the choice of the encryption scheme used to encrypt Sender's secrets s_0, s_1, since R has access to the forward evaluation oracle $F_{k_i}(x)$. For example, the random pad would not work here, since this would allow R to check the unknown secret (also*

transferred, but masked) for equality to constants of his choice. Jumping ahead, we note that to efficiently handle malicious attacks (e.g. R reusing x for different executions) and multiple executions of the protocol, we need a stronger primitive, such as semantically secure encryption (which, e.g., can be implemented simply by allocating some of the PRPG domain for randomness used for encryption).

Theorem 1. *Assume F is a SPRPG. Then the protocol of Construction 1 evaluates the String OT functionality securely in the semi-honest model, where R is additionally allowed to arbitrarily query and reset the token T to its original state (i.e. as received by R).*

Proof (Sketch.) Correctness is straightforward. To prove security, we first show the simulator Sim_S which simulates the view of the semi-honest S. It is easy to see that all S sees is a uniformly distributed element in D, which is trivial to simulate.

We now show the simulator Sim_R of the view of semi-honest R, who additionally has oracle access to T. Given input i and output s, Sim_S outputs $(i, x', (e'_0, e'_1))$, where $x' \in_R D$, $e'_i = F_{x'}(s)$, and $e'_{1-i} \in_R D$. It is easy to see that the output of Sim_S is computationally indistinguishable from the real execution. Indeed, the only "fake" part is e'_{1-i}. In the real execution, $e_{1-i} = F_{F^{-1}_{k_{1-i}}}(s_{1-i})$. A hybrid argument shows that existence of a distinguisher implies a distinguisher for F. □

4.1 Protocols for Malicious (Covert) Setting

The most practical solution to move to malicious model would be to use tokens manufactured by a third party, which are trusted to run the specified functionality (loading the keys, and answering queries as above), and be non-reprogrammable. In this semi-honest token case, Construction 1 works, with the small modification of using semantically secure encryption to transfer secrets in Step 2b (see also Observation 1). In fact, the resulting protocol is concurrently secure (see Sect. 4.3).

However, it is highly desirable to avoid this trust assumption. With the exception of Sect. 4.3, we consider tokens running arbitrary code.

Intuition. The main avenue of attack for S and T is to try to combine their views. This is done by T incorporating a side-channel in its answers to R (recall, Construction 1 provides no channel from S to T). We note that R never knows when this attack occurs, so he must continuously check on T. We handle this by a strong variant of a cut-and-choose technique, which we introduce in this work. The main avenue of attack for R

is to adversarily choose his queries to T. We handle this by using semantic encryption to encrypt secrets being passed back (see Observation 1).

We start with Construction 1, and extend it as follows. First, we embed an exponentially large number of keys in the token, by pseudorandomly generating them. This allows R to test keys of his choice at any time in the lifetime of T, a what we call *continuous cut-and-choose*. To achieve this, the keys k_0, k_1, as used in Construction 1, will be replaced with derived keys $F_{k_0}(y), F_{k_1}(y)$, where y is chosen by R. The token query function Q would now take an extra argument and return $Q(y, x) = (F_{F_{k_0}(y)}(x), F_{F_{k_1}(y)}(x))$. To test T's response, R will ask S to reveal $F_{k_0}(y), F_{k_1}(y)$. Of course this y cannot be used for "live" OT transfer. To avoid S storing large history, R and S agree, *after the token has been received by* R, on the test domain $D_t \subset D$. Now, S will only reveal keys for test queries $y_t \in D_t$, and will only use $y \in D \setminus D_t$ for "live" OT. Of course, D_t must be unpredictable to T. This is easily achieved by R defining it pseudorandomly, e.g. by R choosing k_D and setting $D_t = \{F_{k_D}(x) | x \in D, x \text{ is even}\}$.

Further, we unconditionally hide the input i of R from T by having R choose a random bit b and flipping i iff $b = 1$. S is notified of b to allow for the corresponding flip of his inputs s_0, s_1.

The above changes are sufficient. After presenting the protocol, we give additional intuition of why security holds, and then state and prove the corresponding theorem. Our construction is for deterrence factor $\epsilon = 1/2$, since the probability of catching the attempted cheat is $1/2$. This probability can be naturally modified for any ϵ polynomially close to 1, simply by performing more test queries. Namely, k test queries provide for $\epsilon = 1 - 1/k$.

Let $E = (Gen, Enc, Dec)$ be a CPA-secure encryption scheme, such that every element of the ciphertext domain is uniquely decrypted. Such schemes can be easily constructed from a SPRPG.

Construction 2 *(OT using resettable tokens, covert adversaries model, deterrence $\epsilon = 1/2$)*

1. Initialization. *Let Enc be an encryption scheme as described above. The token T is created by sender S, seeded with keys k_0, k_1, randomly chosen by S, and given to receiver R. T answers any number of queries of the form $Q(y, x) = (F_{F_{k_0}(y)}(x), F_{F_{k_1}(y)}(x))$. R chooses a random string $k_D \in_R \{0,1\}^n$ and sends to S. Parties set $D_t = \{F_{k_D}(x) | x \in D, x \text{ is even}\}$.*
2. OT Protocol execution. *S has inputs $s_0, s_1 \in D$.R has input $i \in \{0,1\}$.*
   (a) R chooses $y_t \in_R D_t$ and sends to S.

(b) *S checks that* $y_t \in D_t$ *and if so, responds with* $k_t^0 = F_{k_0}(y_t)$ *and* $k_t^1 = F_{k_1}(y_t)$. *Otherwise, S outputs* corrupted_R *and halts.*

(c) *R chooses* $b \in_R \{0,1\}, x, x_t \in_R D, y \in_R D \setminus D_t$. *Then R queries T, in random order*

$$(v_0, v_1) = Q(y, x) \qquad = (F_{F_{k_0}(y)}(x), F_{F_{k_1}(y)}(x))$$
$$(c_0, c_1) = Q(y_t, x_t) \qquad = (F_{F_{k_0}(y_t)}(x_t), F_{F_{k_1}(y_t)}(x_t))$$

R checks whether the check values c_0, c_1 *match the evaluation of F based on keys received from S. If so, R sends* $b, y, v_{i \oplus b}$ *to S. Otherwise, R outputs* corrupted_S *and halts.*

(d) *S checks that* $y \notin D_t$. *If not, S outputs* corrupted_R *and halts. If so, S computes encryption keys* $ek_0 = F^{-1}_{F_{k_0}(y)}(v_{i \oplus b})$, *and* $ek_1 = F^{-1}_{F_{k_1}(y)}(v_{i \oplus b})$, *and sends* $(e_0, e_1) = (Enc_{ek_0}(s_{0 \oplus b}), Enc_{ek_1}(s_{1 \oplus b}))$ *to R.*

(e) *R computes and outputs* $s_i = Dec_x(e_{i+b})$.

Note, only R's (selective) check is related to catching the possible covert cheating by S and T. The protocol is secure against malicious R.

Intuition for security. Observe that T only sees two Q queries with easily simulatable random-looking arguments. While S receives one message from T, T does not know which of the two queries it sees is the test one, and which is safe to attack. If T guesses the test query, then it can pass information with the other query, and S learns R's input and can set R's output. However, if T guesses incorrectly and attempts to pass information in the test query, he is caught w.h.p., and without revealing R's input. (This is because S "fixes" the test SPRPs by revealing their keys, and T must answer the query according to the fixed keys not to be caught.) Since the two queries are indistinguishable for T, the deterrence factor is $\epsilon = 1/2$. We stress that T/S cannot base their decision to cheat on R's input, since they commit to this decision (and are checked by R) before R sends any input-dependent messages.

If T behaves semi-honestly, S learns no information from seeing $v_{i \oplus b}$, since this value could have been generated with either of the k_0, k_1 since F is a permutation. Other than the above attack, S and T are limited to oblivious input substitution.

Security against malicious R's attempts to manipulate his queries follows from the use of semantically secure scheme in Step 2d. Now even if S sends back multiple encryptions under the same key (e.g., when OT is composed), R will not be able to relate them without knowing the key. As before, T keeps no state, and thus resetting T does not help.

Theorem 2. *Assume that F is a SPRPG. Then Construction 2 evaluates the String OT function securely against malicious receiver R and*

covert sender S with deterrence factor $\epsilon = 1/2$. The security against S is in the strongest covert setting (strong explicit cheat formulation).

Proof (Sketch.) We treat each corruption case separately. We give detailed description of the simulator of the protocol of Construction 2 against covert adversaries.

The OT Sender S is corrupted. We construct an ideal-model simulator Sim_S of the view of S that works with a trusted party computing the OT functionality. Let $\mathcal{A}_S$ and $\mathcal{A}_T$ be (arbitrary polytime) adversaries controlling S and T respectively. (We consider $\mathcal{A}_S$ and $\mathcal{A}_T$ in full generality. In particular, in contrast with, e.g., [18], we do not assume that $\mathcal{A}_S$ initializes $\mathcal{A}_T$, and thus do not allow Sim_S to intercept the corresponding initialization message sent by $\mathcal{A}_S$. Further, $\mathcal{A}_S$ may not even know the code of $\mathcal{A}_T$.) In its execution, Sim_S interacts with $\mathcal{A}_S$ and $\mathcal{A}_T$ in a black-box manner. We model the physical separation of S and T by the fact that $\mathcal{A}_S$ and $\mathcal{A}_T$ cannot exchange messages with each other once protocol execution begins.

1. Sim_S will determine whether $\mathcal{A}_T$ wants to cheat. For this, Sim_S first obtains both opening keys from $\mathcal{A}_S$, by rewinding $\mathcal{A}_S$ and setting up different test domains. More specifically:
   (a) Sim_S chooses two random strings k'_D and k''_D, which define two testing domains $D'_t, D''_t \subset D$. Sim_S chooses $b' \in_R \{0,1\}, x', y', x'_t, y'_t \in_R D$, such that $y' \in D''_t \setminus D'_t$ and $y'_t \in D'_t \setminus D''_t$.
   (b) Sim_S gives k''_D, and then y' to $\mathcal{A}_S$, and receives two keys from $\mathcal{A}_S$.
   (c) Sim_S rewinds $\mathcal{A}_S$, gives him k'_D, then y'_t, and receives another two keys from $\mathcal{A}_S$.
2. Sim_S calls $\mathcal{A}_T$ with queries $(v'_0, v'_1) = Q(y', x'), (c'_0, c'_1) = Q(y'_t, x'_t)$, in random order. (Recall, Sim_S has obtained from $\mathcal{A}_S$ keys to verify both of $\mathcal{A}_T$'s responses.)
   (a) If neither of $\mathcal{A}_T$'s two responses are valid (where by validity we mean a response that would not cause R to output $corrupted_S$), Sim_S sends $corrupted_S$ to the trusted party, simulates the honest R aborting due to detected cheating, and outputs whatever $\mathcal{A}_S$ outputs. Since so far Sim_S only sent y'_t to $\mathcal{A}_S$, it is easy to see that the simulation is statistical.
   (b) If $\mathcal{A}_T$ sends exactly one valid response, Sim_S sends $cheat_S$ to the trusted party.
      i. If the trusted party replies with $corrupted_S$, then Sim_S rewinds $\mathcal{A}_S$, and hands it the query for which $\mathcal{A}_T$'s invalid response could legally be the test query. Sim_S then simulates honest R aborting due to detecting cheating, and outputs whatever $\mathcal{A}_S$ outputs.

ii. If the trusted party replies with $undetected_S$ and the honest R's input i, then Sim_S rewinds $\mathcal{A}_S$, and hands it the query for which $\mathcal{A}_T$'s valid response could legally be the test query. In the remainder of execution, Sim_S plays honest R with input i. At the conclusion, Sim_S outputs whatever $\mathcal{A}_S$ outputs.

(c) If both of $\mathcal{A}_S$' responses are valid, then we know that $\mathcal{A}_T$ is not passing any information to S. Then Sim_S proceeds with the simulation by playing honest R on a randomly chosen input $i \in_R \{0,1\}$. That is, Sim_S sends $b', y', v'_{i \oplus b'}$ to $\mathcal{A}_S$ (recall, $y' \notin D'_t$, so this message looks proper to $\mathcal{A}_S$). Once Sim_S receives the final message from $\mathcal{A}_S$, using the extracted keys, Sim_S correctly recovers both of $\mathcal{A}_S$'s inputs, and sends them to the trusted party. Sim_S outputs whatever $\mathcal{A}_S$ outputs. This simulation is also statistical.

This completes the description of Sim_S. It is not hard to see that Sim_S simulates the ideal view of covert S with the (ideal model) deterrent factor $\epsilon = 1/2$. In particular, ϵ is equal to $1/2$, since the probability of the honest R catching cheating in the real execution is $1/2$. The simulation is computational since, especially in a batched execution, an unbounded T can compute extra information on the test domain, and have better than $1/2$ guess of which one is the test query.

The OT Receiver R is corrupted. We construct an ideal-model simulator Sim_R that works with a trusted party computing the OT functionality. Let $\mathcal{A}_R$ be the adversary controlling R. In its execution, Sim_R plays honest S and T in their interaction with $\mathcal{A}_R$ in a black-box manner. Sim_R does not need to rewind $\mathcal{A}_R$, since $\mathcal{A}_R$'s input can be extracted from the messages sent to S and T (in real execution, S and T don't communicate, so the privacy of R's input is preserved).

1. Sim_R runs $\mathcal{A}_R$ and acts as honest S and T until the computation of e_0, e_1 in Step 2d of Construction 2. (Sim_R honestly answers as many queries to T as $\mathcal{A}_R$ requests.) At this point, Sim_R needs to provide the right answer to $\mathcal{A}_R$ in its message of Step 2d.
2. If v' output by $\mathcal{A}_R$ in simulation of Step 2c was computed as v'_i by Sim_R while answering a query $Q(y', x')$ to T, then Sim_R can compute the input i used by $\mathcal{A}_R$. Sim_R sets $i = i' \oplus b'$ (where b' was sent by $\mathcal{A}_R$ in Step 2c), calls the trusted party with input i, and receives output s. Sim_R randomly chooses encryption key r, computes random encryptions $e_{i \oplus b'} = Enc_{x'}(s)$, $e_{i \oplus b' \oplus 1} = Enc_r(0)$ and sends e_0, e_1 to $\mathcal{A}_R$. Sim_R outputs whatever $\mathcal{A}_R$ outputs.
3. If v' was never computed by Sim_R in his simulation of T, $\mathcal{A}_R$ w.h.p. cannot know its preimage. Sim_R chooses two random encryptions of 0

under random keys, sends to $\mathcal{A}_R$, and outputs whatever $\mathcal{A}_R$ outputs. □

4.2 Discussion: Protocol Composition and Practical Considerations

In this section, we discuss the issues that arise in OT protocol composition and its use in SFE.

Reuse of the token. We have argued that in many settings (especially in established client-server commercial relationships, such as TV and banking) it is realistic for a player (e.g., a service provider) to provide a token to the other party. At the same time, the same token must be sent only once, and then reused for multiple invocation of the protocol, for it to be cost-effective. In case of SFE, "multiple invocation" usually means sequential execution of simultaneous executions of OT. This is easy to achieve with our protocol, as discussed below.

Simultaneous OT. As in [3], we define simultaneous OT functionality naturally as

$$((s_0^1, s_1^1), .., (s_0^m, s_1^m), (i_1, .., i_m)) \mapsto (\perp, (s_{i_1}, .., s_{i_m})) \tag{1}$$

It is easy to see that natural self-composition of Construction 2 works. Further, efficiency may be improved, as compared to independent parallel execution, by choosing y and y_t once for the entire simultaneous OT execution (This saves about a half of computation by both S and T since $F_{k_i}(y)$ and $F_{k_i}(y_t)$ can be precomputed and reused.) The resulting protocol is a secure implementation of simultaneous OT secure against covert adversaries. The proof is almost identical to the proof of Theorem 2.

Sequential composition. Even though the token is resettable, it may still maintain state across executions. That means that if the token successfully cheats, it may be able to help S compute R's input in a *previous* OT execution. Furthermore, since T in this case is able to transfer a whole element of D to S, several bit inputs of R may be compromised. In Section 4.2, we argue that the compromise is limited. However, to simplify the claims and arguments, we "give up everything" in case of successful cheat, and allow the adversary to learn the entire previous input of R. The following discussion of sequential composition and the reuse of the token are with respect to this ideal model.

Even with the above simplification, it is not immediately clear how to prove sequential composition, since the simulator needs to rewind all the way to the initialization phase to obtain the second opening key. Therefore, an easy way to achieve sequential composition is to amend

the protocol to run the test domain selection for each (simultaneous) OT execution. Clearly, this would create a problem if the keys k_0, k_1 are reused and R chooses a different test domain. To prevent this, we instead derive k_0, k_1 for each execution. This can be done, e.g. by using $k_0^j = F_{k_0}(j), k_1^j = F_{k_1}(j)$ in the j-th batch in place of k_0, k_1. Of course, S would need to keep track of the counter, and T would then take an additional input j. It is easy to see that sequential composition holds in this case, since the simulator now only needs to rewind to the beginning of the current execution. It is also clear that T holds no state, and as such is still resettable.

Practical implications and considerations. Clearly, the ability of the token to leak a few bits of information (at the risk of being caught) on a previous execution is undesirable. This means, for example, that S and R cannot have both "high-cheat-consequence" and "low-cheat-consequence" transactions, since T could conceivably help leak high-consequence inputs in low-consequence transactions.

Still, we believe that our protocols are applicable in the majority of practical situations where the covert model is applicable, namely, where the value of the privacy of the inputs is not worth the risk associated with being caught. This is the case especially often in the settings we consider, i.e. where it is cost-effective for one party to provide a token to the other. Usually, this is a Client-Server setting, with a long-term association of parties. Examples include Client's relationships with Banks, Internet, TV, phone and cellular service providers. In these settings, there is need in protecting client's input such as browsing history, TV channel preferences, list of movies watched, and in general, the profile of user's habits and preferences. While Service Providers are interested in obtaining this information, the potential loss of business far outweighs the benefit.

Amount of the leak. Even though, for the ease of analysis, we generously "gave up" R's entire input in case of a successful cheat, this is clearly not the case in real execution. The amount of data transferred from T to S in this case is roughly the security parameter, say 128 bits. Recall, T never learns R's input, so it can only help S by transferring (compressed) parts of the history of queries T saw. Observe that T must compress random values provided by R, since this is what the query history consists of from the point of view of S. Therefore, S must spend at least "a few" bits to reasonably let S match the information and recover a bit of R's input with some degree of confidence. Therefore, with all on the attacker's side, S can potentially recover ≈ 20 input bits of R per each cheating attempt.

Leak reduction. Our protocol allows querying T out-of-synch with protocol executions. Further, T can be queried by R arbitrarily, with only a small fraction of these queries being used for the actual protocol execution. Thus, essentially for free, R can overwhelm T with queries (which are not seen by S and may not be even tested by R), so that T is likely to send information on unused queries, and thus effectively would be able to transfer little beyond a single bit.

Precomputation and OT extension. It is easy to see that much of the work of R can be precomputed. Further, as shown by Beaver [4], almost all of OT computation can be shifted to a setup phase. Additionally, the efficient technique of Ishai at al. [19] can be used to implement an arbitrary number of OT's, given a small (security parameter) number of OT's[2].

Deterrence factor. If we allow R the option to request additional test keys, this effectively increases ϵ, even if the option not used, since R can test-query T and delay the verification request.

4.3 Concurrent Composition with the Semi-honest Token

We observe that a realistic change in the setup assumptions adds security in concurrent executions. This stronger security property holds if both parties trust that the token executes the prescribed code. This could be the case, e.g. if the token is supplied by a third party, and S is limited to loading the keys k_0, k_1 onto the token. This is a reasonable setting in practice, and this assumption was used, e.g., by Hazay and Lindell [18], to efficiently achieve concurrent composition.

The simpler Construction 1, modified as discussed in Observation 1, is secure against malicious S and R. Recall, the modification simply requires using a semantically secure scheme, instead of direct application of SPRPG, to encrypt messages e_0, e_1 in Step 2b of Construction 1.

Theorem 3. *Assume that F is a SPRPG. Then modified Construction 1, as described above, evaluates the String OT function securely against malicious receiver R and sender S. The security against S is statistical.*

We explicate this construction and prove Theorem 3 in the full version.

Further, security holds for concurrent composition. Intuitively, this is because the simulator will not need to rewind. Indeed, the simulator of the covert case rewinds twice – to extract the keys loaded into token T,

[2] [19] requires the use of correlation-robust functions (a weak form of random oracle). In practice, even the RO assumption seems to be much more solid than that of tamper-resistance. Thus, its use will not have effect on security.

and in relation with testing T. Now, however, the token is modeled as a separate entity from S, and S must explicitly output the keys loaded into T. Then the need for the first rewinding disappears, since Sim_S simply receives the keys from S as the first step of the simulation. Since T is semi-honest, we do not test it, and thus there is no need for second rewinding. Then, as shown in [22], this is sufficient to achieve concurrent general composition (equivalently, universal composability). In [22], this is only shown for protocols that have additional property of *initial synchronization*. Informally, a protocol is said to have initial synchronization if all parties announce that they are ready to start before they actually start. As pointed out in [18], this property always holds for two-party protocols.

Acknowledgements. I would like to thank the anonymous referees of this paper for their valuable comments.

References

1. `http://www.scdeveloper.com/cards.htm`, retrieved on Aug 28, 2009.
2. William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 119–135. Springer, 2001.
3. Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *Theory of Cryptography, TCC 2007*, volume 4392, pages 137–156, 2007.
4. Donald Beaver. Precomputing oblivious transfer. In *Advances in Cryptology – CRYPTO'95*, volume 963 of *LNCS*, pages 97–109, 1995.
5. Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In *CRYPTO '89*, pages 547–557. Springer-Verlag, 1989.
6. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS*, pages 62–73, 1993.
7. Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for uc secure computation using tamper-proof hardware. In *Advances in Cryptology – EUROCRYPT 2008*, volume 4515, pages 545–562, 2008.
8. Giovanni Di Crescenzo. Private selective payment protocols. In *Financial Cryptography and Data Security, FC 2000*, volume 1962 of *LNCS*, pages 72–89. Springer, 2000.
9. Ivan Damgård, Jesper Buus Nielsen, and Daniel Wichs. Isolated proofs of knowledge and isolated zero knowledge. In *Advances in Cryptology – EUROCRYPT 2008*, volume 4515, pages 509–526, 2008.
10. Ivan Damgård, Jesper Buus Nielsen, and Daniel Wichs. Universally composable multiparty computation with partially isolated parties. In *Theory of Cryptography, TCC 2009, To appear*, 2009.
11. Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.
12. Marc Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In *RSA Security 2001 Cryptographer's Track*, volume 2020 of *LNCS*, pages 457–471. Springer-Verlag, 2001.

13. Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In *Theory of Cryptography, TCC 2004*, volume 2951, pages 258–277, 2004.
14. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In *Advances in Cryptology – CRYPTO 2008*, volume 5157, pages 39–56, 2008.
15. Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *Theory of Cryptography, TCC 2010*, 2010.
16. Vipul Goyal, Payman Mohassel, and Adam Smith. Efficient two party and multi party computation against covert adversaries. In *Advances in Cryptology – EUROCRYPT 2008*, volume 4515, pages 289–306, 2008.
17. Vipul Goyal and Amit Sahai. Resettably secure computation. In *Advances in Cryptology – EUROCRYPT 2009*, volume 5479, pages 54–71, 2009.
18. Carmit Hazay and Yehuda Lindell. Constructions of truly practical secure protocols using standard smartcards. In *ACM Conference on Computer and Communications Security*, pages 491–500, 2008.
19. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *LNCS*. Springer-Verlag, 2003.
20. Kimmo Järvinen, Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Embedded SFE: Offloading server and network using hardware tokens. In *Financial Cryptography and Data Security, FC 2010*, 2010.
21. Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In *Advances in Cryptology – EUROCRYPT 2007*, volume 4515, pages 115–128, 2007.
22. Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. In *Proc. 38th ACM Symp. on Theory of Computing*, pages 109–118, 2006.
23. Yehuda Lindell and Benny Pinkas. A proof of Yao's protocol for secure two-party computation. Cryptology ePrint Archive, Report 2004/175, 2004. `http://eprint.iacr.org/`.
24. Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology – EUROCRYPT 2007*, volume 4515, pages 52–78, 2007.
25. Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386, 1988.
26. Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.
27. Moni Naor, Benny Pinkas, and Reuben Sumner. Privacy preserving auctions and mechanism design. In *1st ACM Conf. on Electronic Commerce*, pages 129–139, 1999.
28. Andrew C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 160–164, Chicago, 1982. IEEE.
29. Andrew C. Yao. How to generate and exchange secrets. In *Proc. 27th IEEE Symp. on Foundations of Comp. Science*, pages 162–167, Toronto, 1986. IEEE.