Secure Two-Party Computation via Leaky Generalized Oblivious Transfer

Samuel Ranellucci and Alain Tapp

DIRO, Université de Montréal, Québec, Canada

February 21, 2013

Abstract

We construct a very efficient protocol for constant round Two-Party Secure Function Evaluation based on general assumptions. We define and instantiate a leaky variant of Generalized Oblivious Transfer based on Oblivious Transfer and Commitment Schemes. The concepts of Garbling Schemes, Leaky Generalized Oblivious Transfer and Privacy Amplification are combined using the Cut-and-Choose paradigm to obtain the final protocol. Our solution is proven secure in the Universal Composability Paradigm.

Introduction

The tools that we use in this paper are Oblivious Transfer [Rab81, EGL85], Commitment Schemes [Blu83, Eve83] and some variation on Yao's Garbled Circuits. The notion of security that we will use in this work is Universal Composability which was introduced by Ron Canetti in [Can01]. The goal of this work is to present a very efficient protocol for constant round Two-Party Secure Function Evaluation based on these assumptions.

Important work has gone into optimizing the primitives of secure computation. Oblivious Transfer requires expensive public key operations to instantiate from scratch, thus the idea of generating a large number of them from a much smaller number of OT's by using private key operations (named OT extension) is extremely valuable [Bea96, IKNP03]. Many techniques have been used to reduce the complexity of communicating a garbled circuits such as the Free XOR technique optimization from [KS08].

There are many recent constructions for efficient secure computation. Several of them are based on Garbled Circuits such as [KSS12, LP11, SS11, LP07], but other approaches have been used [IPS08, LOP11, NO09, NNOB12].

Notation and convention

We will refer to 1-out-of-2 Oblivious String Transfer for string of length l as OT. The security parameter will be denote by s. A circuit's depth will be denoted by d and its input size by n. The notation $x \in_R A$ describes the random selection of an element x according to the uniform distribution from a set A. As usual, we define secure fonction evaluation $\mathcal{F}_{SFE}$ as a two party computation where Bob learns the output.

Our results in perspective

In recent times, Secure Function Evaluation with malicious adversaries using Garbled Circuits have become more and more practical. Garbling scheme based protocol following the Cut-and-Choose paradigm require $\Omega(ns)$ calls to OT. Lindell and Pinkas started the trend with their work in [LP07] that required $\Omega(ns)$ OT but $\Omega(ns^2)$ Commitments. The work of [LP11] and [SS11] only require $\Omega(ns)$ comitment but require stronger notions of OT such as committing OT and Batch Single Choice Cut and Choose OT. These protocols do not admit OT extension. Both the result of [Fre12] and [KSS12] allow OT extensions but the first one relies on a Random Oracle and the second one on Claw-Free Collections.

In contrast, our work allows OT extension, does not rely on additional assumptions and only uses $O(ns)$ OT and Commitments. The structure of our protocol is based on [LP11].

paper	OT	Commit	OT extension	Assumptions
[LP07]	$O(ns)$	$O(ns^2)$	yes	None
[LP11]	$O(ns)$	$O(ns)$	no	DDH
[SS11]	$O(ns)$	$O(ns)$	no	Claw-Free Collection, DDH
[Fre12, FN12]	$O(ns)$	$O(ns)$	yes	Random oracle
[KSS12]	$O(ns)$	$O(ns)$	yes	Claw-Free Collection
this work	$O(ns)$	$O(ns)$	yes	None

Leaky Generalized Oblivious Transfer

Generalized Oblivious Transfer is a very natural primitive. In OT, the receiver either learns the first or the second message. In a context where the sender transmits n messages, the receiver can learn 1-out-of-n or even k-out-of-n. In fact, there is no reason not to considered arbitrary legitimate sets of messages. The sender and receiver could define a list of accessible subsets of all the positions that are considered legitimate. At the end of the protocol, the receiver will learn the messages whose indices belongs to a legitimate set of his choice. As usual, the sender will not learn the receiver's choice. This primitive was introduced in [IK97] and a clever protocol was presented in [SSR08]. Unfortunately, the protocol that was presented is only secure against a passive adversary. A protocol based on verifiable oblivious transfer and secure against active adversary was proposed in [BNRT12].

In this section, we will present a protocol for Generalized Oblivious Transfer that we know is imperfect. This will not prevent the protocol from being useful. The advantage of the protocol described here is that it is based directly on Oblivious Transfer and Commitment Schemes. This means that the protocol is more efficient then the one proposed in [BNRT12]. In the next section, this imperfect implementation of GOT will be used to instantiate Two-Party Computation in a way that improves previous protocols. We will not present other applications of Leaky GOT; we are nevertheless confident that it could be used in other contexts.

In order to prove the security of the final two party protocol under the Universal Composability framework, it is necessary to present a meaningful ideal functionality for Leaky GOT that is securely implemented by our protocol. We call this functionality Leaky Generalized Oblivious Transfer for obvious reasons. The protocol, when compared to the naturally defined GOT, will only leak one bit of information to a dishonest sender. We will then prove that our protocol implements this ideal functionality. This enable us to work in a modular way. The abstraction of an imperfect protocol, where a cheating player has a limited advantage is trickier than expected. Once a cheating possibility for the sender is made explicit in the definition of the task, it opens (in a very unexpected way) the possibility that the receiver cheats.

We will now present the intuition behind the protocol. How can one use OT to implement GOT? The main idea is taken from [SSR08]. The first important observation is that the set of accessible sets of positions is always the complement of an access structure. Each of the n messages will be sent using OT in conjunction with a decoy message. Learning the decoy message is a "proof of no knowledge" about the message paired with it in the OT. The problem with revealing this decoy is that the receiver would reveal the position he had learned. To solve this problem, the set of decoy messages will form n shares in a secret sharing scheme with an access structure that is the complement of the collection associated with the particular instance of GOT. If the receiver only learns positions from a legitimate set, he will learn enough decoys to reconstruct the secret. When the receiver reveals the secret, the sender knows that the receiver has only learned messages that are included in a legitimate set. This is what was presented in [SSR08]. Unfortunately, it is clear that an active adversary can learn a lot of information about the receiver's choice by corrupting shares and then observing the secret that was reconstructed (if any) by the receiver. Thus we modify the protocol so that the receiver commits to the reconstructed secret and then requests that the sender makes it public. Then, by proving that he knew the secret (by opening the Commitment), he only reveals one bit of information to a cheating sender. A cheating sender cannot choose to learn any bit of information about the receiver's choice but every choice for the set of shares corresponds to a specific predicate. Although in general the set of predicate available to the sender is very limited, any deviant behavior at this phase is consistent with a specific predicate. The set of available predicates is perfectly determined from the choice of the underlying secret sharing scheme used in the protocol. Although it may be hard to determine if a predicate belongs to a set, it is not important since only adversaries benefit from this knowledge.

We will now formalize the notion of legitimate sets for a GOT and its relationship with secret sharing.

Definition 1.

- *Let $I = \{1, 2, ..., n\}$ be a set of indices. A collection $\mathcal{A} \subseteq \mathcal{P}(I)$ is* **monotone** *if the fact that $\mathcal{B} \in \mathcal{A}$ and $\mathcal{B} \subseteq \mathcal{C}$ implies that $\mathcal{C} \in \mathcal{A}$ is verified.*
- *An* **access structure** *is a monotone collection $\mathcal{A}$ of non-empty sets of I. A set s is* **authorized** *if $s \in \mathcal{A}$ and a set s' is* **minimal** *if there exists no strict subset s'' of s' such that $s'' \in \mathcal{A}$.*
- *The* **complement** *of a collection $\mathcal{C}$ is defined as $\mathcal{C}^* = \{b \subseteq I \mid \exists\, c \in \mathcal{C},\ b = I - c\}$.*
- *We define* $\textbf{Closure}(\mathcal{C}) = \{c \subseteq c' \mid c' \in \mathcal{C}\}$.
- *A collection $\mathcal{C}$ is* **enclosed** *if $\mathcal{C} = \textbf{Closure}(\mathcal{C})$.*
- *An element $c \in \mathcal{C}$ is* **maximal** *if there exists no $c' \in \mathcal{C}$ such that $c \subseteq c'$ and $c \neq c'$.*

The following theorem is proven in [SSR08].

Theorem 1. *For every enclosed collection C, there exists a unique access structure $\mathcal{A}$ such that $\mathcal{C}^* = \mathcal{A}$*

Definition 2. *A* **secret sharing scheme** *is a pair of randomized algorithms (Share, Reconstruct) over a domain D with an access structure $\mathcal{A}$. Share$_{\mathcal{A}}(s)$ where $s \in D$ always outputs*

$(s_1, \ldots, s_n) \in D^n$ *such that:*
(1) for all $A \in \mathcal{A}$*,* $Reconstruct_{\mathcal{A}}(\{(i, s_i) \mid i \in A\}) = s$*,*
(2) for any $A' \notin \mathcal{A}$*,* $\{(i, s_i) \mid i \in A'\}$ *gives no information about* s*.*
(3) $s_i \notin D$ *and* $i \in A \implies Reconstruct_{\mathcal{A}}(\{(i, s_i) \mid i \in A\}) = \perp$

Definition 3. *Shares* $\{s_i\}$ *are* **consistent** *if for any authorized subset of shares, the reconstruction is possible and the secret obtained is always the same.*

Definition 4. *In our context, a* predicate *is a function that takes a set as an argument and returns a boolean value. The* null predicate *is the predicate that always return FALSE and will be noted* ϕ*. A* predicate class *is a set of predicates that contains the base predicate.*

The ideal functionality for the Leaky GOT has to be defined very carefully. The problem is that it is imperfect and its behavior changes if a participant is corrupt. It is important that the ideal functionality knows which if any participant is cheating. Otherwise the simulation does not make sense. In the description, the order of the steps and the expected messages are fixed.

LEAKY GENERALIZED OBLIVIOUS TRANSFER: $\mathcal{F}_{LGOT(\mathcal{I},\mathcal{P})}$
Let $\mathcal{I}$ be an enclosed collection, $\mathcal{P}$ a predicate class, p a predicate in $\mathcal{P}$, $m_1, \ldots, m_n \in \{0,1\}^l$ and $b \in \{True, False\}$.

- Upon receiving `(predicate,p)` from the Sender, if the sender is corrupted, this value of p will be used, otherwise p is set to ϕ (the null predicate).
- Upon receiving `(choice,I)` where $I \in \mathcal{I}$ is a set of indices, if $p(I)$ the functionality forwards (`failure`) to the receiver, otherwise it forwards `(match)`.
- Upon receiving `(validation,b)` from the receiver, if $b = True$ and the receiver is corrupt, or if $p(I) = True$ the functionality forwards `(failure)` to the sender and ignores any further commands, otherwise it sends `(match)` to the sender.
- Upon receiving (`send`, $m_1, ..., m_n$) from the sender, for each $i \in I$, the functionality sends (`Reveal`,`i`,m_i) to the receiver.

The following protocol implements $\mathcal{F}_{LGOT(\mathcal{I},\mathcal{P})}$ where $\mathcal{I}$ is an enclosed collection defining the accessible sets of messages.

Protocol: $\pi_{LGOT(\mathcal{I})}$

1. The sender selects $k_1, k_2, ..., k_n \in_R \{0,1\}^l$ (one-time pads)
2. Let $\mathcal{A} = \mathcal{I}^*$, $s \in_R D$ and $(s_1, s_2, ..., s_n) = Share_{\mathcal{A}}(s)$.
3. The sender selects a set of n ids that have never been used, denote these ids as $\mathrm{sid_i}$ and sends them to the receiver. For each i, the sender sends (`send`,k_i,s_i,$\mathrm{sid_i}$) to $\mathcal{F}_{OT}$.
4. Let $I \in \mathcal{I}$ be the set of messages that the receiver wishes to receive. He selects $b_i = 0$ when $i \in I$ and $b_i = 1$ otherwise. For each i, the receiver sends (`Transfer`,b_i,sid_i) to $\mathcal{F}_{OT}$ and records the result.

5. The receiver executes the recover algorithm with the shares he received and obtains S' (if he did not successfully recover a secret he selects a random value for S' instead). The receiver sends `(commit,S')` to $\mathcal{F}_{COM}$.

6. Sender sends S to the receiver.

7. The receiver checks if $S \neq S'$, if so he selects $m =$ `Failure`, otherwise he notes $m =$`match`. He then sends m to the receiver. If $S \neq S'$, the receiver then aborts.

8. The sender checks if $m =$ `Failure` and if so, he aborts. The receiver then sends `open` to $\mathcal{F}_{COM}$. The sender verifies that $S = S'$ and if not, he aborts the protocol.

9. The sender sends $z_i = m_i \oplus k_i$ to the receiver. ($\{m_i\}$ are the messages)

Theorem 2. *For every secret sharing scheme there exists a class Predicate $\mathcal{P}$ such that $\pi_{LGOT(\mathcal{I})}$ securely realizes $\mathcal{F}_{LGOT(\mathcal{I},\mathcal{P})}$ in the F_{OT}, F_{COM} hybrid model.*

Proof. The correctness of the protocol can be verified by inspection but one has to be careful when dealing with the leakage of the protocol. Given a secret sharing scheme with associated domains D, for every value $V \in D$, and every set of shares E such that each share is in D', there exists a predicate p defined over the subsets of E such that $p_{E,V}(E') = 0$ iff *Reconstruct*$(E') = v$. We can define a predicate class $\mathcal{P}$ by taking the union $p_{E,V}$ over all possible E and V. It is important to note that if E is consistent and V is the reconstructed secret of E, then the predicate is the predicate that always returns 0. The protocol precisely realizes the functionality with this predicate class.

Sender Simulation:

- The simulator awaits the input to the OT and records the k_i and s_i
- The simulator sends `committed` to the Environment.
- The simulator awaits a secret S.
- The simulator sends to the ideal functionality the command `(Predicate,p)` where p is the predicate that asks if the complement of I does not correctly reconstruct the secret S.
- The simulator awaits that the functionality outputs either (`failure`) or (`match`) and forwards that message to the environment; if (`failure`) was received, the simulator aborts.
- The simulator awaits the encrypted messages from the environment, he then decrypts them (using the k_i) and sends the command (`send`, m_i) to the ideal functionality.

Receiver Simulation

- The simulator awaits that the environment sends his choice of the b_i to be used in step 4. From those he can deduce I.
- The simulator selects random keys k_i, creates shares s_i for a random secret S and then sends the environment the appropriate output for each OT.
- The simulator awaits that the environment forwards the command `(commit,S',id)`.

- The simulator then sends the secret S to the environment.
- The simulator awaits that the environment sends his choice of m (match or failure), if `(failure)` was received, the simulator sends `(validation,True)` to the functionality and then aborts, otherwise he sends `(validation,False)`
- The simulator awaits the command `(open)` from the environment, if $S \neq S'$, the simulator aborts the protocol.
- The simulator calls the ideal functionality with I and receives the messages for that set. He chooses random values for the messages that he did not receive, encrypts all the messages with the keys (k_i) and then sends them to the environment.

□

Garbling schemes

In 1982, Yao generated a construction that came to be known as Yao's garbled circuit. The idea was to encode a circuit in such a way that by giving a party the right keys he could evaluate the circuit on a specific input and yet learn nothing about the input except what could be deduced from the circuit. This construction came to be used in many different applications each with their own divergent variant. Lindell and Pinkas were the first to prove the security of Yao's garbled circuit in the field of two party computation [LP09]. Garbling schemes proposed in [BHR12] define the notions that unify these different variants.

We will need to distinguish between a function $f : \{0,1\}^n \rightarrow \{0,1\}^m$ and the representation of a function as $\bar{f}$ (in our case we can restrain ourselves to circuits). A garbling algorithm GB is a randomized algorithm that transforms $\bar{f}$ into a triple of functions $GB(\bar{f}) = (F, e, d)$. We require that $f = d.F.e$. The encoding function e turns an initial input $x \in \{0,1\}^n$ into a garbled input $X = e(x)$. Evaluating the garbled function F on the garbled input X gives a garbled output $Y = F(X)$. The decoding function d turns Y into the final output $y = d(Y)$, which must coincide with $f(x)$. One has probabilistically factored f into $d.F.e$. Thus a garbling scheme $G = (GB, En, De, Ev, ev)$ is regarded as a five-tuple of algorithms, with strings $\bar{d}$, $\bar{e}$, $\bar{f}$, and $\bar{F}$ interpreted as functions under the auspices of functions De, En, ev, and Ev.

A circuit garbling scheme is a five-tuple of algorithms $\mathcal{G} = (GB, En, De, Ev, ev)$. The first two algorithms are probabilistic, the remaining are deterministic. The string $\bar{f}$ represents the function f that we wish to garble. On input $\bar{f}$, algorithm GB outputs an encoding function $En(\bar{e}, .)$ that maps an initial input $x \in \{0,1\}^n$ to a garbled input $X = En(\bar{e}, x)$ as well as $\bar{F}$ that describes a garbled function that maps each garbled input X into a garbled output Y and a string $\bar{d}$ describing a decoding function $DE(\bar{d}, .)$ that maps each garbled output y into a a final output $y = De(\bar{d}, Y)$.

Definition 5. *A garbling scheme is* **correct** *if for any function f the application of GB results in strings $(\bar{\mathcal{F}}, \bar{e}, \bar{d})$ such that $De(\bar{d}, Ev(\bar{\mathcal{F}}, En(\bar{e}, x))) = ev(\bar{f}, x)$.*

Definition 6. *A garbling scheme with an encoding function $En(\bar{e}, .)$ with input $x = x_1 \dots x_n$ where $x_i \in \{0,1\}$ is* **projective** *if for every $\bar{e}$ there exists a list of tokens $(T_1^0, T_1^1, \dots, T_n^0, T_n^1)$ such that $En(\bar{e}, x) = (T_1^{x_1}, \dots, T_n^{x_n})$*

Definition 7. *A projective scheme is* **transparent**, *if given $\bar{f}$, $\bar{\mathcal{F}}$,and the token $(T_1^0, T_1^1, \dots, T_n^0, T_n^1)$ in a random order for each variable, one can efficiently verify that $\bar{\mathcal{F}}$ is consistent with $GB(\bar{f})$ and furthermore the order of the tokens can be deduced.*

Definition 8. *A side-information function* ψ *denotes the information revealed about* $\bar{f}$ *from* $(\bar{F}, \bar{e}, \bar{d})$.

Definition 9. *A garbling scheme is private relative to a leakage function* ψ *if for any inputs* x_0,x_1 *and description of functions* $\bar{f_0}, \bar{f_1}$ *such that* $ev(\bar{f_0}, x_0) = ev(\bar{f_1}, x_1)$ *and* $\psi(\bar{f_0}) = \psi(\bar{f_1})$*, the adversary cannot distinguish between* $(\bar{F_0}, \bar{e_0}, \bar{d_0}, X_0)$ *generated from* $(\bar{f_0}, x_0)$ *and* $(\bar{F_1}, \bar{e_1}, \bar{d_1}, \bar{X_1})$ *from* $(\bar{f_1}, x_1)$

Definition 10. *A garbling scheme is malleable if for any function* $\bar{f}$ *such that* $GB(\bar{f}) = (\bar{F}, \bar{e}, \bar{d})$ *and every* $y \in \{0,1\}^m$ *there exists a* $\bar{g}$ *such that* $\forall x, g(x) = y$ *and* $\psi(\bar{f}) = \psi(\bar{g})$.

Two party computation

The first application of Garbling scheme was presented by Yao in order to implement two party computation. In a context where the players are semi-honest, Alice and Bob can evaluate a circuit of their choice on their private inputs. Alice can construct the circuit and send it with the keys associated to her input. Now for Bob to obtain the keys associated to his own input, in a way that hides those values from Alice, Alice and Bob will execute an OT. The messages transfered by Alice will be the keys associated to Bob's input variables and Bob will only learn the keys associated to his choice of input. Since Bob has all the input keys as well as the circuit, he can perform the evaluation and extract the result.

In order to make this scheme secure against an active adversary, two things have to be verified. In term of correctness, Bob must verify that the circuit he evaluates computes the right function. For his privacy, he must also verify that the protocol would have worked even if he had chosen another input. These two problems are easily addressed by using what is called Batch-Single Choice Cut-and-Chose OT. This is an instantiation of GOT for a specific access structure [LP11].

Alice will create s circuits with all of their input keys. Half of the circuits will be completely revealed so that Bob can verify that they are correct. By virtue of contra-factual reasoning, this ensures that no more then a few of the remaining unopened circuit are incorrect. By evaluating the remaining circuits and taking the most common result as the output, correctness is guaranteed. To keep 's input private, it is important that Bob always evaluate the circuit with the same input. Combining all these requirement, we see that OT has to be replaced with a specific GOT. The set of pairs of keys for all the circuit will be organized in a table. Each of the s columns are associated with a circuit. Each row is associated with an input variable and each element of the table is a pair of keys, which are respectively associated with the value 0 and 1. Thus, Bob must learn for exactly half of the columns, for each input, the value of both keys and for the other half, for each row, he must choose between learning the first element of each pair or learning the second element of each pair. We will now formally define the enclosed collection that was just informally described. Now, $y \in \{0,1\}^n$ is the input of Bob and $J \subseteq [s], |J| = s/2$ is the set of check circuits.

Definition 11. *Each messages is denoted as* m_{ijk} *where* $i \in [1..n], j \in [1..s], k \in \{0,1\}$*. The enclosed collection* I *defining the set of messages that are authorized is the union of*

- $I_0 = \{ \{m_{ijk} \mid k = y_i \}\}$,
- $I_1 = \{ \{m_{ijk} \mid j \in J \}\}$.

For Bob to verify the correctness of the check circuits, he must obtain from Alice all the associated input keys. Then, for the evaluation circuits, Alice will only send the keys associated with her specific input. Unfortunately, Alice could decide not to use the same input for all evaluation circuits. The success of the execution of the protocol would give her information on the sensitivity of the function on her choice of input, which would leak information about Bob's input. To solve that problem [Fre12, FN12] introduced a verification functions v. Instead of just generating garbled circuits for f, garbled circuits for $f'[(x,r),(y,s)] = (f[x,y], v[x,s,r])$ will be generated. The following choice of functions have been introduced. It was proven in [Fre12] that it does not reveal information about Alice's inputs and at the same time guarantees that inconsistent inputs will be detected except with exponentially small probability.

Definition 12. *We will denote $z_i(x,r) = p^i(x) \cdot r \oplus s_i$ and $v[x,s,r] = (z_0, \ldots, z_n)$ as the verification function. Where p^i is the cyclic shift toward the right of i position and $\cdot$ as the inner product mod 2.*

Main protocol

Although the complete protocol requires several steps, the right combination of its ingredients results in a very efficient protocol based on general assumptions. In the previous section, we explained a known approach to combine GOT and Garbling schemes that allows us to obtain Two-party Secure Computation. Our general protocol will follow the same approach described in the previous section but will be based on Leaky GOT instead. Privacy of the receiver's input is jeopardized by the leakage in our GOT and therefore the general protocol must be modified in a clever way. The function to be evaluated will be modified in such a way that the string that will leak contains no information on Bob's input. It will only be used after the leaky GOT has been executed, privacy amplification will then be executed by revealing the hash function, thus rendering any leakage moot.

The function of two players, which is to be evaluated, will be replaced by $f'(x, y', h, q) = f(x, h(y') \oplus q) = f(x,y)$. To extract the output, y' will be chosen uniformly at random, h will be taken randomly from a family of universal hash functions with the appropriate parameters and $q = h(y') \oplus y$. Since h and q do not contain information about the input, they can (and will) be declared publicly after the leaky GOT has been used to transfer y' among all the different circuits. Thus, Alice's input will remain the same, but Bob's input is now decomposed into (y', h, q) where only y' is transfered using the leaky GOT. Intuitively, since only one bit of information about y' is leaked, privacy amplification ensures that y remains private.

Definition 13. *We denote $\mathcal{H}$ a family of Universal Hash Function from $\{0,1\}^{n+s+1}$ to $\{0,1\}^n$.*

Any universal Hash Function Family can be used in our protocol but the one presented in [] is compact and efficient.

Definition 14. *Denote $f'((x,s),(y',h,q,r)) = (f(x, h(y') \oplus q), v(x,s,r))$ where h is in $\mathcal{H}$ and v is the verification function that was defined in the previous section.*

In our main protocol any circuit garbling scheme that is correct, private, malleable, transparent, and projective relative to a side information ψ can be used. For explicit constructions see [LP07, BHR12]. We will also use the following length parameters, the length of the input to the function to be evaluate is denoted as usual by n, $n_q = n$, $n_h = 2(n+s+1)$, $n_x = 2n$, $n_y = 2n+s+1$ and $n_t = n_q + n_x + n_y + n_h$. To clarify the role of each player, we will refer to Alice as the sender and Bob as the receiver.

Protocol: π_{SFE}

1. The sender extracts s triplets $(F_j, e_j, d_j) = GB(f')$.

2. The sender extracts from the encoding functions e_j all the $2ns$ tokens associated to both values for all inputs of all circuits. Tokens associated to the sender will be denoted by $(x_{1j}^0, x_{1j}^1, \ldots, x_{n_xj}^0, x_{n_xj}^1)$, those to y' by $(y_{1j}^0, y_{1j}^1, \ldots, y_{n_yj}^0, y_{n_yj}^1)$, those to the hash function by $(h_{1j}^0, h_{1j}^1, \ldots, h_{n_hj}^0, h_{n_hj}^1)$ and those associated to q by $(q_{1j}^0, q_{1j}^1, \ldots, q_{n_qj}^0, q_{n_qj}^1)$.

3. The receiver selects uniformly at random $y' \in \{0,1\}^{n+s+1}$, $q \in \{0,1\}^n$, $h \in \mathcal{H}$ and sets $q = h(y') \oplus y$. The receiver commits to $(H, Q) = (Com(h), Com(q))$. He also choses $J \subseteq [s], |J| = s/2$ (choice of check circuits).

4. The sender sends the command $(predicate, \psi)$ to $\mathcal{F}_{LGOT(A)}$

5. The receiver inputs to $\mathcal{F}_{LGOT(\mathcal{I})}$ his choice $I \in \mathcal{I}$ where I is obtained from J and y' as describe above. He awaits a response from $\mathcal{F}_{LGOT(\mathcal{I})}$, he sends `(validation,False)` to $\mathcal{F}_{LGOT(\mathcal{I})}$ and if he received `(Failure)`, he aborts.

6. If the sender receives `(Failure)` from the Leaky GOT, he aborts.

7. The sender executes `(send,`$(y_{1j}^0, y_{1j}^1, \ldots, y_{n_yj}^0, y_{n_yj}^1)$`)` associated to $\mathcal{F}_{LGOT(\mathcal{I})}$. For each i, j the sender commits to $(H_{ij}^0, H_{ij}^1) = h_{ij}^0, h_{ij}^1$.
   For each x_{ij}^0, x_{ij}^1, the sender selects a random $r_{ij} \in \{0,1\}$ and commits to $(X_{ij}^0, X_{ij}^1) = (Com(x_{ij}^{r_{ij}}), Com(x_{ij}^{1 \oplus r_{ij}}))$. The sender sends each F_j, d_j to the receiver.

8. The receiver reveals J to the sender and for each $j \in J$ and each input i, he sends y_{ij}^0, y_{ij}^1 to the sender.

9. The sender aborts if the values are different than what he sent.

10. For all $j \in J$, the sender opens the Commitment X_{ij}^0, X_{ij}^1, H_{ij}^0, H_{ij}^1 and Q_{ij}^0, Q_{ij}^1

11. For all $j \in J$, the receiver checks that the garbled circuit j is valid and consistent and aborts otherwise.

12. For all $j \notin J$, and all $i \in [n]$, the sender opens $X_{ij}^{c_j \oplus r_{ij}} = x_{ij}^{c_j}$.

13. The receiver opens his Commitment to h and q.

14. The sender opens the Commitments to the token associated to the values h and q

15. The receiver uses the tokens he received as well as the pairs (F_j, d_j) to do the evaluations using the garbling scheme. He checks that the validation outputs are consistent. If not he aborts. Receiver takes the most common value as his output.

Theorem 3. *For all predicate class $\mathcal{P}$, π_{SFE} securely realizes $\mathcal{F}_{SFE}$ in the $\mathcal{F}_{LGOT(\mathcal{I})}$, $\mathcal{F}_{COM}$ hybrid model.*

Proof. The correctness of the protocol can be deduced by inspection. The addition of privacy amplification in conjunction with Commitment Scheme do not compromise correctness since they are independent of the calculated function and are eventually made public. In the simulations, the number at the end of each line corresponds to the relevant step in the protocol.

Sender simulation

- The simulator sends the messages (`commited`) which are associated to the sender commiting to h and q. (3)
- The simulator awaits the command `(predicate,`p`)`. (4)
- The simulator selects uniformly at random the set J (of size $s/2$) and the string y' and derives the associated set I.
- If $p(I) = True$, then the simulator forwards the message `(failure)` to the environment and aborts. (6)
- The simulator awaits that environment's inputs for the leaky GOT, the Commitment command with the keys associated to the circuits as well as each (F_j, d_j) (7)
- The simulator forwards J as well as all the keys whose indices are matched to J (9)
- The simulator awaits that the environment send the open command associated to all the keys in the check circuit, he then checks that all check circuits are consistent and if not, he aborts. (11)
- The simulator awaits that the environment sends the open command to the keys associated to his choice of input. (11)
- The simulator choses uniformly at random h and q and forwards them to the environment. (13)
- The simulator awaits that the environment send the open commands for Commitments associated to keys associated to the given h and q (14)
- The simulator evaluates the evaluation circuits and checks that all validation outputs ($V(.)$) are consistent; if not he aborts.
- The simulator has all the tokens, because of the transparency of the garbling scheme, he can determine for each input, for each circuit the value of input associated to each key. For each input, he takes as input bit the value that appears in the majority of evaluation circuits. He can thus extract the input and sends it to the ideal functionality (15)

All messages sent from the simulator to the environment are generated using the same distribution as the protocol with an honest receiver. In the simulation, the environment could see a discrepancy at the end when the simulator extracts the implicit input from the environment. This can only happen with exponentially small probability for the simulator since the check circuit are checked and that the verification function ensures that the inputs are consistent. The leakage from the Leaky GOT, due to the privacy amplification step, will only leak a negligible amount of information about the receiver's input and thus the simulation is indistinguishable from the real protocol.

Receiver Simulation

- The simulator awaits the commit command associated to h and q. (3)
- The simulator awaits the message `(choice,I)` from the environment, the simulator notes the receiver's choice of check circuit as J as well as the input y' and sets $y = h(y') \oplus q$. the simulator forward message `(match)`
- The simulator awaits the command `(validation,`b`)` from the environment, if $b = True$ the simulator aborts, otherwise it sends the input y to F_{SFE} and learns output z. (5)
- For each $j \in J$, the sender sets $(F_j, d_j, e_j) = GB(f')$.
- The simulator selects r uniformly at random. Let $g'[(x, y), (h, y', r)] = (z, r)$. Because of the malleability of the scheme, the simulator can produce g consistent with $GB(g')$ such that $\psi(f') = \psi(g)$. For each $j \notin J$, the sender sets $(F_j, d_j, e_j) = GB(g)$.
- The simulator sends the appropriate tokens that are associated with the environment's input to the leaky GOT. He also forwards the Commitment messages to the remaining tokens as well as all the (F_j, d_j, e_j) he constructed. (7)
- The simulator awaits the environment's choice of check circuit as well as the inputs he received from the leaky GOT. The simulator checks if these values are consistent with what he sent, if it is not the case, the simulator aborts.
- The simulator sends the reveals message for all of tokens associated to check circuits. (10)
- The simulator randomly selects for each input and each circuit to send a reveal message for one of the token. (12)
- The simulator awaits the open commands to h and q. The simulator simulates the opening of all the tokens required by the protocol. (15)

The main difference between the ideal setting and the real setting is the construction of fake garbled functions. Fortunately, due to the privacy of the garbling schemes, these do not allow the environment to distinguish which model he is part of. The Commitments hide the sender‘s input and as such leave no recourse for the environment to distinguish between the real and ideal setting. $\square$

Theorem 4. *$\mathcal{F}_{SFE}$ can be securely realized in a constant number of rounds using $O(ns)$ calls to $\mathcal{F}_{OT}$ and $O(ns)$ calls to $\mathcal{F}_{COM}$ assuming a correct, private, transparent, malleable and projective garbling scheme.*

Proof. In the protocol the functionality $\mathcal{F}_{LGOT(\mathcal{I})}$ is used only once to transmit a table containing $2ns$ keys. To do this, $O(ns)$ OT and Commitments are required. The π_{SFE} protocol does not use any additional OT but requires $O(ns)$ Commitments. It is clear that the protocol is constant round. $\square$

References

[Bea96] D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 479–488. ACM, 1996.

[BHR12] M. Bellare, V.T. Hoang, and P. Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 784–796. ACM, 2012.

[Blu83] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News*, 15(1):23–27, 1983.

[BNRT12] David Bernardo, Ryo Nishimaki, Samuel Ranellucci, and Alain Tapp. General constructions of efficient multi-party computation. *Unpublished manuscript*, 2012.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 136–145. IEEE, 2001.

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

[Eve83] Shimon Even. A protocol for signing contracts. *ACM SIGACT News*, 15(1):34–39, 1983.

[FN12] Tore Kasper Frederiksen and Jesper Buus Nielsen. Fast and maliciously secure two-party computation using the gpu. Technical report, Cryptology ePrint Archive: Report 2013/046, 2012.

[Fre12] Tore Kasper Frederisken. Optimizing actively secure two-party computations. Master's thesis, Aarhus Universit, 2012.

[IK97] Y. Ishai and E. Kushilevitz. Private simultaneous messages protocols with applications. In *Theory of Computing and Systems, 1997., Proceedings of the Fifth Israeli Symposium on*, pages 174–183. IEEE, 1997.

[IKNP03] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology-CRYPTO 2003*, pages 145–161. Springer, 2003.

[IPS08] Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer–efficiently. In *Advances in Cryptology–CRYPTO 2008*, pages 572–591. Springer, 2008.

[KS08] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free xor gates and applications. In *Automata, Languages and Programming*, pages 486–498. Springer, 2008.

[KSS12] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21st USENIX conference on Security symposium*, pages 14–14. USENIX Association, 2012.

[LOP11] Y. Lindell, E. Oxman, and B. Pinkas. The ips compiler: Optimizations, variants and concrete efficiency. In *Advances in Cryptology–CRYPTO 2011*, pages 259–276. Springer, 2011.

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

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yaos protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[LP11] Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In *Theory of Cryptography*, pages 329–346. Springer, 2011.

[NNOB12] Jesper Nielsen, Peter Nordholt, Claudio Orlandi, and Sai Burra. A new approach to practical active-secure two-party computation. *Advances in Cryptology–CRYPTO 2012*, pages 681–700, 2012.

[NO09] J. Nielsen and C. Orlandi. Lego for two-party secure computation. In *Theory of Cryptography*, pages 368–386. Springer, 2009.

[Rab81] Michael O Rabin. How to exchange secrets by oblivious transfer. Technical report, Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[SS11] A. Shelat and C.H. Shen. Two-output secure computation with malicious adversaries. In *Advances in Cryptology–EUROCRYPT 2011*, pages 386–405. Springer, 2011.

[SSR08] B. Shankar, K. Srinathan, and C.P. Rangan. Alternative protocols for generalized oblivious transfer. In *Proceedings of the 9th international conference on Distributed computing and networking*, pages 304–309. Springer-Verlag, 2008.