Constant-round aecure two-party computation from a linear number of oblivious transfer

Samuel Ranellucci and Alain Tapp

DIRO, Université de Montréal, Québec, Canada

September 18, 2013

Abstract

We construct a protocol for constant round Two-Party Secure Function Evaluation in the standard model which improves previous protocols in several ways. We are able to reduce the number of calls to Oblivious Transfer by a factor proportional to the security parameter. In addition to being more efficient than previous instantiations, our protocol only requires black box calls to OT and Commitment. This is achieved by the use of a faulty variant of the Cut-and-Choose OT. The concepts of Garbling Schemes, faulty Cut-and-Choose Oblivious Transfer and Privacy Amplification are combined using the Cut-and-Choose paradigm to obtain the final protocol.

Introduction

In the field of secure multiparty computation it is studied how to simulate a trusted third party when several players need to compute a function on their private data. Auctions, electronic voting and privacy preserving data analysis are important applications of secure multi-party computation.

The tools that we use in this paper are Oblivious Transfer [Rab81, EGL85], Commitment Schemes [Blu83, Eve83] and some variation of Yao's Garbled Circuit. In this work, we consider malicious adversaries and the strongest notion of security, called Universal Composability [Can01]. The goal of this work is to present a very efficient protocol for constant round Two-Party Secure Function Evaluation in this model based on the above primitives.

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 by using private key operations is extremely valuable. This concept is called OT extension [Bea96, IKNP03] and the fact that the protocol we present in this article is based on black box OT insures that these types of optimization apply.

Garbling schemes [Yao82] are convenient and efficient constructions and have been the basis of several interesting two-party computation protocol [FJN$^+$13, KSS12, LP11, SS11, LP07] but other approaches have been used [IPS08, LOP11, NO09, NNOB12].

Our results in perspective

In recent times, Secure Function Evaluation with malicious adversaries using Garbled Circuits have become more and more practical. Previous protocols based on garbling schemes following the

Cut-and-Choose paradigm required $\Omega(ns)$ 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 $\Omega(ns)$ commitments 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. The results of [FJN$^+$13], [Fre12], [MR13] and [KSS12] allow OT extensions but the first three are in the Random Oracle model and the third one relies on Claw-Free Collections. In contrast, our work only uses $O(n+s)$ OT, allows OT extension and does not rely on additional assumptions.

paper	OT	Commit	OT extension	Assumptions
[LP07]	ns	ns^2	yes	None
[LP11]	ns	ns	no	DDH
[SS11]	ns	ns	no	Claw-Free Collection, DDH
[Fre12, FN12]	ns	ns	yes	Random oracle
[KSS12]	ns	ns	yes	Claw-Free Collection
[FJN$^+$13]	$n+s$	$ns/log(s)$	yes	Random oracle
[MR13]	ns	ns	yes	Random oracle
[Lin13]	ns	ns	no	DDH
[HKE13]	ns	ns	no	DDH, Random oracle
this work	$n+s$	ns	yes	None

Notation and convention

We will often denote the adversary's view of a player's input as a random variable. We will refer to 1-out-of-2 Oblivious String Transfer for string of length l as OT. The security parameter will be denoted by s. A circuit's depth will be denoted by d and its input size by n. Uniformly random and independent choices for x in a set X is denoted $x \in_R X$. The notation $x \in_R A$ describes the random selection of an element x according to random Variable A. The range of an integer starting from i and ending with j is denoted $[i, j]$. As usual, we define secure function evaluation $\mathcal{F}_{SFE}$ as a two-party computation where Bob learns the output.

Structure of the paper

We first present our notation for the **Garbling schemes**. This leads to a very good generic protocol for **two-party computation** using perfect Cut-and-Chose OT. We then present a **Faulty Cut-and-Choose OT**. The drawback of using this faulty primitive is formalized in the **Selective failure attack** section. This is followed by our **Main protocol** where we propose a very efficient and secure protocol based on the Faulty Cut-and-Choose OT. The security proof in the universal composability paradigm follows (**sender simulation** and **receiver simulation**).

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 $f : \{0,1\}^n \to \{0,1\}^m$ and its representation $\bar{f}$ (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 $\bar{f}$, the algorithm GB outputs an encoding function $En(\bar{e}, .)$ which 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 which maps each garbled input X into a garbled output Y, and a string $\bar{d}$ describing a decoding function $DE(\bar{d}, .)$ which maps each garbled output y into a a final output $y = De(\bar{d}, Y)$.

The side-information function is an important feature of garbling schemes. In a protocol that follows, Alice will have to garble a circuit. Bob will have to verify that part of the output is independent from part of his input. The only way this can be achieved is by looking at the underlying side-information.

Definition 1. *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 2. *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 3. *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 4. *The side-information function ψ denotes the information revealed about $\bar{f}$ from $(\bar{F}, \bar{e}, \bar{d})$.*

The side-information function is an important feature of garbling schemes. In the protocol that follows, Alice will have to garble a circuit. Bob will have to verify that part of the output is independent from part of his input. The only way this can be achieved is by looking at the side-information.

Definition 5. *A garbling scheme is private relative to a topological side information 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 6. *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 a Garbling scheme was presented by Yao in order to implement secure two-party computation. In a context where the players are semi-honest, by using OT it becomes easy for Alice and Bob to evaluate a circuit of their choice securely on their private inputs. Alice can construct the garbled circuit and send it to Bob with the keys associated to her input. Alice then transmits to Bob his keys using OT. For each of Bob's input bits, Alice will send the key associated with the value 0 and the key associated with the value 1. For each of his input bits, Bob will choose to learn the key corresponding to his input at that position. Since Bob has all the input keys as well as the circuit, he can now perform the evaluation and extract the result.

In order to make this scheme secure against an active adversary, several issues have to be addressed. In terms of correctness, Bob must know that the circuit he evaluates computes the right function. This is not sufficient since Alice could transfer keys that are correct only for certain input bit and thus learn information about Bob's input following the success or failure of the protocol. Bob must know that the protocol would have worked correctly even if he had chosen another input. These two requirement can be ensured by using what is called Batch-Single Choice Cut-and-Chose OT introduced in [LP11].

Informally, Alice will create s circuits with all of their associated input keys. Half of the circuits will be completely revealed so that Bob can verify that they are correct. It is thus unlikely that this phase succeeds and that more than 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 Alice's input private, it is important that Bob always evaluates the circuit with the same input. Combining all these requirements, we see that the generic Oblivious Transfer has to be replaced with something with more structure. The set of pairs of keys for all the circuits will be organized in a table. Each of the s columns is associated with a circuit. Each row is associated with an input variable (of Bob) and each element of the table is a pair of keys, which are respectively associated with the values 0 and 1. Learning half of the circuit (check circuit) means Bob must learn for exactly half of the columns the value of both keys. The fact that Bob always uses the same inputs bits means that for every individual row he must choose between learning the first element of each pair or learning the second element of each pair. In the following section, we will present a variant of that primitive that we call Cut-and-Chose OT.

Bob can now reveal the identity of the check circuit so that Alice sends him all the associated input keys. With all the keys he can verify the correctness of those circuits. The rest of the circuits are going to be evaluation circuits and for all of those, Alice will send the keys associated to her input. By choosing different inputs for some evaluation circuit, Alice can learn information about the sensitivity of the circuit conditioned on Bob's input. To solve this problem [Fre12, FN12] introduced a verification functions v. Instead of just generating garbled circuits for f, garbled circuits for $f'[(x,a),(y,b)] = (f[x,y], v[x,a,b])$ will be generated. The following choice of functions has been introduced.

Definition 7. *We will denote* $z_i(x,a,b) = (p^i(x) \cdot b) \oplus a_i$ *and* $v(x,a,b) = (z_0, \ldots, z_n)$ *as the verification function. Where* p^i *is the cyclic shift toward the right of* i *position and* $\cdot$ *the inner product mod 2.*

It was proven in [Fre12] that v 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.

Faulty Cut-and-Choose OT

In this section, we will introduce the Cut-and-Choose OT which is a faulty variant of the Batch-Single Choice Cut-and-Chose OT introduced by [LP11]. This is a variant of OT where more than 2 messages are sent simultaneously and where the choice of messages received by Bob has a special structure. The input from the sender is a table M of ns pairs of binary messages (m_{ij0}, m_{ij1}) of length l. Using our faulty protocol, an honest sender would learn nothing and an honest receiver would learn exactly the following. For each row, the receiver selects a y_i and gets $\{m_{ijk} \mid k = y_i\}$. For each odd index of column j, the sender selects an $a_j \in \{0,1\}$ and learns all messages in $\{m_{i(j+a_j)k}\}$.

We implement a faulty variant of this primitive from $O(n+s)$ OT. The transfer of rows and the transfer of columns will be done independently and thus an honest Alice has to use the same information twice. Each of the n rows will incur one OT where the first string will be the first element of each pair in that row and the second string will be the second element of each pair. For each pair of columns, each column will be contained in a single string. The sender and the receiver will execute s OT with the strings associated to the columns. If the sender is honest, the receiver will get exactly what he requested. One of the problems of the protocol is that the choice of Bob can be adaptive. To prevent this, Alice will send keys (One time pads) instead of messages. Once every key is transmitted, she will send the messages encrypted with different keys. Another problem with this protocol is that the sender can choose the elements of the row inconsistent with the columns. The receiver, if he finds an inconsistency, will have to abort but this can be done at a later time. Instead, for the time being, he will select the column value as the output. This is essential to the main protocol. In any case, the success or failure of the verification step leaks information about Bob's choice and if used in the protocol described previously, it would leak information about Bob's input. This issue will be dealt with in the final protocol. The following collection denotes the sets of messages that the receiver can pick and where he learns exactly all the messages in the chosen set.

Definition 8. *Each messages is denoted as* m_{ijk} *where* $i \in [1,n], j \in [1,s], k \in \{0,1\}$.

- $\mathcal{I} = I_0 \cup I_1$
- $I_0 = \{\ \{m_{ijk} \mid k = y_i\ \} \mid y_i \in \{0,1\}\ \}$,
- $I_1 = \{\ \{m_{i(j+a_j)k} \mid a_j \in \{0,1\}\ \} \mid j \in [1,3,\dots,s-3,s-1]\}$
- $p_{ic} = p_{i1c}, \dots, p_{isc}$
- $w_j = p_{1j0}, p_{1j1}, \dots p_{nj0}, p_{nj1}$

The following protocol implements the Cut-and-Choose OT.

Protocol: π_{CCOT}

1. For each $i \in [1,n], j \in [1,s], k \in \{0,1\}$, the sender selects $p_{ijc} \in_R \{0,1\}^l$.
2. For each $i \in [1,n]$, the sender and the receiver execute $\mathcal{F}_{OT}$ with messages (p_{i0}, p_{i1}) with choice bit y_i.
3. For each odd $j \in [1,s]$, the sender and the receiver execute $\mathcal{F}_{OT}$ with messages (w_j, w_{j+1}) and choice bit a_j.

4. The receiver checks that the values he received (the p_{ic} and $w_{j'}$) are consistent. If not, he notes the inconsistency and uses the column value as the output (the $w_{j'}$).

5. The sender sends $z_{ijk} = m_{ijk} \oplus p_{ijk}$

The protocol is faulty and therefore its security will not be proven independently in this section. That being said, the problem only arises in the case of a cheating sender and therefore we can still discuss the possibility of simulating the receiver.

Theorem 1. *A simulator having access to the ideal functionality for Cut-And-Chose OT, when the environment controls the receiver, can simulate the view of the environment in the real world.*

Proof. The simulator selects $p_{ijc} \in_R \{0,1\}^l$. The simulator awaits the environment's choice (y_i and a_j) for each OT. (The simulator is able to extract the choice bits in the OTs for each row and column pair.) The simulator forwards the appropriate combination of keys as specified in the protocol. The simulator calls the ideal functionality with these choices. For each message m_{ijk} that the simulator has obtained from the ideal functionality, he uses the key p_{ijk} to encrypt those messages and sends the cyphertext $z_{ijk} = m_{ijk} \oplus p_{ijk}$ to the environment. For the messages m_{ijk} that the simulator has not extracted, he forwards a random message z_{ijk}.

The distribution of m_{ijk}, z_{ijk}, p_{ijk} in the simulation is identical to the distribution of the real world. $\square$

Selective failure attack

In order to achieve a very high level of efficiency in our final protocol, we will have to combine several ideas. One of them is to use a faulty Cut-And-Chose OT. In our protocol, Alice can gain information about Bob's input by misbehaving and waiting to see if this causes the protocol to abort. In this section, we formalize this type of attack and the impact it has on Alice's information regarding Bob's input.

In general, selective failure is a type of attack where the adversary makes the successful completion of the protocol depend on the other player's input. We consider that an attack is a selective failure attack, if for any pairs of inputs that causes the protocol to succeed (resp. fail), from Alice's point point of view the resulting distributions over Bob's input are indistinguishable.

Theorem 2. *Any selective failure which goes undetected by Bob with probability at least 2^{-k} will reduce the min entropy for Alice about Bob's input by at most k.*

Proof. Let X be the variable representing Alice's knowledge about Bob's input and let $H_\infty(X) = m$. A selective failure attack splits the domain of X in the set in U and its complement depending whether or not the protocol aborted. Let us denote $p \geq 2^{-k}$, the probability that a randomly selected element x according to distribution X falls into U. We can see that:

$$Pr[X = x \mid x \in U, x \in_R D] \leq 2^{-m}/p \tag{1}$$

$$Pr[X = x \mid x \in U, x \in_R D] \leq 2^{-m+k} \tag{2}$$

This implies that the reduction of min entropy about X is smaller or equal to k. $\square$

Main protocol

Although the complete protocol requires many steps, the right combination of ingredients results in a very efficient protocol based on general assumptions. In the previous section, we explained a known approach to combine Cut-and-Choose OT and Garbling Schemes to obtain Two-Party Secure Computation. Our general protocol will follow this approach. Without further modification of the protocol, the privacy of the receiver's input would be jeopardized by the faultiness of the Cut-and-Chose OT. We will thus modify the general protocol to address this problem. The function to be evaluated will be modified in such a way that the leakage contains no information on Bob's input.

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 faulty Cut-and-Chose OT 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 transferred using the Cut-and-Choose OT. Intuitively, since only a limited amount of information about y' is leaked, privacy amplification ensures that y remains private.

Another important technicality has to be taken care of. We require that the topological leakage function of the garbling scheme show that for each garbled circuit that the output of the verification function is independent of (h, q, y'). If this was not verified, then a corrupted sender could simply make the consistency of the output for the verification function depend on y. This would allow the sender to learn information based on a selective failure attack.

Definition 9. *We denote $\mathcal{H}$ a family of Universal Hash Function from $\{0,1\}^{n+2s+1}$ to $\{0,1\}^n$.*

Any universal Hash Function Family can be used in our protocol but the one presented in [Tho09] is compact and efficient.

Definition 10. *Denote $f'((x, a), (y', h, q, b)) = (f(x, h(y') \oplus q), v(x, a, b))$ 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 topological side information function ψ can be used. Garbling schemes similar to Yao's garbled circuit normally meet those requirements. For explicit constructions, see [LP07, BHR12]. We will also use the following length parameters, the length of the input to the function to be evaluated is denoted as usual by n. We will denote $n_q = n$, $n_h = 2(n + 2s + 1)$, $n_x = n$, $n_a = n$, $n_y = 2n + 2s + 1$ 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 $GB(f') \to (F_j, e_j, d_j)$.

2. For each circuit j, using the encoding functions e_j, the sender extracts all of the tokens associated to both values for all inputs. Tokens associated to the sender will be denoted by $(x^0_{1j}, x^1_{1j}, \ldots, x^0_{n_x j}, x^1_{n_x j})$, $(a^0_{1j}, a^1_{1j}, \ldots, a^0_{n_a j}, a^1_{n_a j})$, those to y' by $(y^0_{1j}, y^1_{1j}, \ldots, y^0_{n_y j}, y^1_{n_y j})$, those to the hash function by $(h^0_{1j}, h^1_{1j}, \ldots, h^0_{n_h j}, h^1_{n_h j})$, those associated to q by $(q^0_{1j}, q^1_{1j}, \ldots, q^0_{n_q j}, q^1_{n_q j})$ and those to b by $(b^0_{1j}, b^1_{1j}, \ldots, b^0_{n_b j}, b^1_{n_b j})$.

3. The receiver selects $y' \in_R \{0,1\}^{n+2s+1}$, $q \in_R \{0,1\}^n$, $h \in_R \mathcal{H}$ and $J \in_R \{0,1\}^{s/2}$. The receiver sets $q = h(y') \oplus y$ and commits to h and q. (J represent the check circuit selection)

4. The sender and the receiver execute π_{CCOT} where the sender's input is $(y^0_{1j}, y^1_{1j}, \dots, y^0_{n_y j}, y^1_{n_y j})$ and the receiver's input is the set consistent with y' and J. (as mentioned earlier, in case of inconsistency, the receiver chooses the column value)

5. For each circuit $j \in [1, s]$ the sender performs the following steps

   - send F_j, d_j to the receiver
   - for each $i \in [1, n_h]$, commit to h^0_{ij} and h^1_{ij},
   - for each $i \in [1, n_b]$, commit to b^0_{ij} and b^1_{ij},
   - for each $i \in [1, n]$, select $r_{ij} \in_R \{0,1\}$ and commit to $(X^0_{ij}, X^1_{ij}) = (x^{r_{ij}}_{ij}, x^{1 \oplus r_{ij}}_{ij})$,
   - for each $i \in [1, n_a]$, select $r'_{ij} \in_R \{0,1\}$ and commit to $(A^0_{ij}, A^1_{ij}) = (a^{r'_{ij}}_{ij}, a^{1 \oplus r'_{ij}}_{ij})$

6. For each $j \in [1, s]$, the receiver checks from F_j, d_j and ψ that the output of the verification function is independent of (h, q, y'). (ψ is the topological side information function)

7. The receiver reveals J to the sender. For each $j \in J$ and each input $i \in [1, n]$, he sends y^0_{ij}, y^1_{ij} to the sender.

8. The sender aborts if the values he received in the previous step differ from what he had sent.

9. For all $j \in J$, and i having the appropriate range, the sender opens the Commitment $(X^0_{ij}, X^1_{ij}), (A^0_{ij}, A^1_{ij}), (h^0_{ij}, h^1_{ij}), (q^0_{ij}, q^1_{ij}), (b^0_{ij}, b^1_{ij})$

10. For all $j \in J$, the receiver checks that the garbled circuit j is valid. The receiver aborts at this point if a garbled circuit is faulty or if he had noticed any inconsistency during the cut-and-choose protocol.

11. For all $j \notin J$, and all $i \in [1, n]$, the sender opens $X^{x_j \oplus r_{ij}}_{ij} = x^{x_j}_{ij}$. For all $j \notin J$, and all $i \in [1, s]$, the sender opens $X^{x_j \oplus r_{ij}}_{ij} = x^{x_j}_{ij}$ and $A^{a_j \oplus r'_{ij}}_{ij} = a^{a_j}_{ij}$.

12. The receiver opens his Commitments to h and q and declares a value b.

13. The sender opens the Commitments to the token associated to the values h, q, b.

14. 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 if the validation outputs are consistent and aborts if they are not. The receiver takes the most common value as his output.

Sender simulation

The sender does not receive any output and the receiver mostly reveals randomly chosen values in the protocol. Therefore the simulator job consists mostly of extracting the sender's input, aborting with the same distribution as in the real world and revealing random values.

- The simulator sends the messages (`committed`) which are associated to the receiver committing to h and q. (3)

- The simulator selects uniformly at random the set J (of size $s/2$) and the string y' and derives an associated set I.
- The simulator simulates an execution of π_{CCOT} and extracts the environment's input to π_{CCOT} . (4)
- The simulator awaits the Commitment command with the keys associated to the circuits as well as each (F_j, d_j) (5)
- The simulator forwards J as well as all the keys whose indices are matched to J (7)
- The simulator awaits that the environment sends the open command associated to all the keys in the check circuit. The simulator aborts using the same criteria as an honest receiver. He aborts if any of the check circuits is incorrect or if the outputs of the OTs in the π_{CCOT} are inconsistent.
- The simulator awaits that the environment sends the open command to the keys associated to his choice of input. (11)
- The simulator chooses uniformly at random h, q and b and forwards them to the environment (h,q as a reveal command from $\mathcal{F}_{COM}$). (12)
- The simulator awaits that the environment sends the open commands for Commitments to keys associated to the given h, q and b (13)
- 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 circuit and each input the value of input associated to each key. For each of the sender's input bit, he takes as input bit the value that appears in the majority of evaluation circuits. He can thus extract the input and send it to the ideal functionality. (14)

It is evident to see that if the environment does not corrupt circuits, send inconsistent shares in Faulty cut-and-choose OT or open tokens which results in different inputs for different evaluation circuits, the environment will be unable to distinguish between the real or ideal setting. The distribution of all variables is identical and neither the simulator nor the real world execution will abort.

We first address the case where the environment only corrupted some circuits. This would make the simulation fail if the probability of having the simulation abort is different from the probability in the real world execution or if the simulation does not abort but is unable to extract the relevant information. In the simulation (as in the honest protocol), each circuit is a check circuit with probability one-half. Since the receiver does not reveal any inconsistency in the cut-and-choose OT until after he has declared the choice of check circuit, the sender must corrupt circuits in such a way that more than half of evaluation circuits are corrupted and none of the check circuits are corrupted. Otherwise the simulator aborts as in the real world setting. This can only happen with exponentially small probability.

We now address the case where the sender provides inconsistent inputs to the OT in the Cut-and-Choose OT protocol.

The keys that are returned by the receiver for the check circuits only depend on the environment's choice of values in the columns (see π_{CCOT}). Since an honest receiver always knows those values by virtue of being associated to check circuits. The simulator can perfectly simulate that step by simply revealing what an honest receiver would know in the real world.

We will show that the leakage does not allow the environment to distinguish between the ideal world and the real world. If the simulator notices any inconsistency during the Cut-and-Choose protocol or that a garbled circuit was faulty, he aborts. Since the check circuits are public at this point, the environment knows that either a corrupted circuit was a check circuit (case we already dealt with) or that the receiver (simulator or honest party) has selected an input such that the row and column are inconsistent. The key factor is that the probability that the protocol aborts depends only on the choice of inconsistency and is the same for the simulation as for the real world.

It is important to note that the environment cannot distinguish between two choices of input (in π_{CCOT}) such that the protocol would not abort. As such, we can view the sender's behavior as performing a selective failure attack, which reduces the min entropy on y' by at most s. Since there is a sufficient amount of min entropy left, the application of a random element from a universal family of hash functions essentially generates a random mask. This mask hides the input and allows the simulator to generate a statistically indistinguishable distribution.

Finally, if the environment tries to send inconsistent inputs for two circuits and the circuits are valid, the validation step will result in an abort except with exponentially small probability. It is important to note that the topological side information function demonstrates independence between (h, q, y') and the output of the verification function. Otherwise the environment can use a selective failure attack to extract information on y. Thus we can see that sending inconsistent inputs will not allow the environment to distinguish between the real and ideal setting.

Receiver Simulation

- The simulator awaits the commit command associated to h and q. (3)
- The simulator awaits the receiver's input for π_{CCOT} from the environment, the simulator notes the receiver's choice of check circuit as J as well as the input y' and sets $y = h(y') \oplus q$.
- For each $j \in J$, the simulator 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 in the π_{CCOT} . He also forwards the Commitment messages to the remaining tokens as well as all the (F_j, d_j, e_j) he has constructed. (5)
- The simulator awaits the environment's choice of check circuits as well as the inputs he received in the π_{CCOT} . The simulator checks if these values are consistent with what he has sent, if it is not the case, the simulator aborts. (7)
- The simulator sends the reveal messages for all of tokens associated to check circuits. (11)
- The simulator randomly selects for each input and each circuit to send a reveal message for one of the tokens. (12)
- The simulator awaits the open commands to h and q. The simulator simulates the opening of all the tokens required by the protocol. (14)

First we note that in the case of a corrupted receiver, the simulator can simulate the faulty variant of cut-and-choose OT using a cut-and-choose OT functionality. The simulator can trivially extract the environment's input from the choice of input to the faulty variant of cut-and-choose OT, as well as his commitments to h and q. The main difference between the ideal setting and the real setting is the construction of a fake garbled circuit. Fortunately, due to the privacy of garbling schemes, these do not allow the environment to distinguish which model he is part of. The only remaining recourse for the environment is to try to lie about which circuits which are check circuits. But since the probably of guessing the keys is negligible, we can see that the real and ideal setting are indistinguishable.

Circuit Optimization

We believe that the strong point of the two-party computation protocol presented in this article is its efficiency. In this section, we will explain why some recent work on optimization of protocols based on garbled circuits can be applied to our protocol. In [Lin13], a very nice optimization technique is presented for achieving secure computation while reducing the total number of circuits that have to be transmitted to s and still achieving security of 2^{-s}. Another approach was presented in [HKE13] but Lindell's technique applies more naturally to our protocol. The detailed presentation of the technique requires a full length publication and therefore we will assume in this section that the reader is familiar with Lindell's article.

The basic idea of this optimization is as follows: the protocol will be divided in two computations, an **output extraction** and a **cheating sender input extraction**. In the **output extraction** phase, the receiver will be able to extract the output of the functionality or a proof that the sender has cheated. In the **cheating sender input extraction** phase, the sender inputs the same value as in the **output extraction**, if the receiver has acquired a proof of cheating, he is able to extract the sender's input and otherwise he gets nothing. Note that having the receiver obtain the sender's input enables the protocol to be correct in a trivial way. It is necessary to weave these two computations together so that certain properties are verified. First, the protocol must ensure that the sender uses the same input in both computations. Second, it is also needed that evaluation of the garbled circuits of both computations is done before checking either of them. This is to insure that a receiver can't extract a proof of cheating from an honest sender.

A necessary building block of the construction is the modified cut-and-choose OT similar to the one in [LP11]. There are two important differences. First, the set of indices in J is no longer size restricted (instead of size exactly $s/2$). In addition, for each $j \notin J$, the receiver receives a special string which allows him to prove that $j \notin J$.

We will now describe how the main protocol can be modified to enable this optimization. Since we do not have the space to formally describe the construction, we will instead provide a high level overview of the changes required.

Instead of implementing a modified cut-and-choose directly, our protocol implements a faulty modified cut-and-choose OT. We thus have to adapt the ideas from the main protocol to deal with the faultiness and leakage. We have to tangle the computations, delay the input verification of the **output extraction** and merge it with the input consistency of **cheating sender input extraction**.

Faulty Modified Cut-and-Choose OT

The faulty version of modified cut-and-choose OT can easily be implemented by a minor modification of the faulty cut-and-choose OT protocol. Instead of doing one OT per pair of column, one simply does an OT for each column where the first element transferred in the OT is the value in the column and the second element is the secret that allows the sender to proves that $j \notin J$.

Sender input consistency

It is not to hard to realize that in the main protocol, the receiver can extract an output (even if it is an error) and then use the consistency check to verify that outputs were all consistent. It is thus possible to merge the consistency check of **output extraction** and the consistency check of **cheating sender input extraction** together, in order to verify that the sender uses the same input in both computations.

Verification of circuits

It is important to note that in our main protocol, once the faulty cut-and-choose has taken place and the sender has committed to his inputs, the circuits are fully determined. As such, we can delay the checking phase after evaluation has taken place.

Conclusion

Although our protocol is very efficient, we explain why some recent optimization work also apply to our protocol. We believe some further optimizations are possible. The inclusion of the hash function into the function to be computed could increase the size of the circuit, especially small ones. Although it is not straightforward, we believe that using the appropriate garbling scheme coupled with a family of hash function with a nice circuit structure, one could hardcode the hash function into the circuit. Another possible optimization could be to select a class of universal hash functions which mends well with the free XOR technique [KS08]. The construction of universal hash function from random binary matrices seems to be an ideal choice for these two optimizations. We believe that the idea of using faulty primitives with adapted protocols can improve secure computation also beyond two-party secure function evaluation.

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.

[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.

[FJN$^+$13] Tore Kasper Frederiksen, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. Minilego: Efficient secure two-party computation from general assumptions. In *Advances in Cryptology–EUROCRYPT 2013*, pages 537–556. Springer, 2013.

[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.

[HKE13] Yan Huang, Jonathan Katz, and Dave Evans. Efficient secure two-party computation using symmetric cut-and-choose. *IACR Cryptology ePrint Archive*, 2013:81, 2013.

[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.

[Lin13] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. *IACR Cryptology ePrint Archive*, 2013:79, 2013.

[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.

[MR13] Payman Mohassel and Ben Riva. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. *IACR Cryptology ePrint Archive*, 2013:51, 2013.

[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.

[Tho09] Mikkel Thorup. String hashing for linear probing. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 655–664. Society for Industrial and Applied Mathematics, 2009.

[Yao82] A.C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, 1982.