An Overview of Active Security in Garbled Circuits

Author: Cesar Pereida Garcia
Supervisor: Pille Pullonen
Department of Mathematics and Computer Science.
University of Tartu
Tartu, Estonia.

December 15, 2015

Abstract

Although original Yao's garbled circuit protocol offers a solution for general *secure function evaluation* (SFE), it is useful only in a *semi-honest* model, namely, the environment where it is used must be controlled and both parties should be trusted to follow the protocol. Unfortunately this protocol only works in a limited set of deployments since in practice is desirable to use it together with untrusted parties to achieve function evaluation. This paper offers an overview of different techniques that have been used and are currently used to provide protocol security under a *malicious* model.

1 Introduction

Secure function evaluation (SFE) tries to solve the problem of performing function evaluation with secure inputs while maintaining the privacy of the inputs of the parties involved. The classical example for this problem is the "Millionaires Problem" in which two millionaires want to know who is richer without disclosing the amount of money they have.

Exist multiple categories of SFE, one category are secure functions with certain security properties tailored according to one specific problem. Although this category is interesting, it has applicability to a limited set of problems. On the other hand, there exists a second category in which arbitrary functions are transformed into secure functions using different approaches. Homomorphic Encryption Schemes and Yao's garbled circuits are some of the approaches where function evaluation can be performed on encrypted data.

Yao's garbled circuit[15] protocol approach is to take an arbitrary function and securely evaluate that function by modeling it as a boolean circuit, hiding the inputs and outputs for each gate in such way that no inputs, intermediate

values and extra circuit information are leaked during the protocol. Yao's original protocol maintains privacy as long as both parties do not deviate from the pre-established rules in the protocol.

New research improves on the idea of maintaining the privacy and the security of the garbled circuit and parties inputs even when one or both of the parties deviate from the protocol and each party actively tries to get information from the other. The original protocol provides security in the *semi-honest* model but recent research provides methods to achieve similar levels of security in a *malicious* model.

1.1 Goals

The goal of this paper is to provide a gentle introduction to current research done around active security of garbled circuits, but in order to provide a wider picture of the topic, first a basic introduction to original Yao's garbled circuits and its security features is provided and then the main ideas of new approaches are presented. This paper assumes a general understanding of cryptography but no previous familiarity with Yao's garbled circuit protocol. Formal proofs are not provided but can be consulted on the original papers. This paper focuses on the ideas, premises and research done with Yao's garbled circuits and not in specific software or hardware implementations. Discussion on active security provides a comparison of different approaches.

2 Preliminaries

2.1 Security Definitions

As it is common in cryptography defining security for a system, protocol or primitive is not a trivial task to do and for that reason security-related terms used throughout this paper are defined here. Note that many of these terms are similar or equivalent to terms used throughout the literature in cryptography. Also, most of the writing here describes two parties *Alice* who is the sender or circuit constructor and *Bob* who is the receiver or circuit evaluator.

2.1.1 The Ideal Oracle

The ideal oracle in garbled circuits is an oracle or *trusted third party* that has the function f used to evaluate the inputs In_A, In_B and it is known only to the oracle. The oracle receives the inputs In_A from Alice and In_B from Bob and returns the output of $f(In_A, In_B)$ to both parties Alice and Bob. No one learns anything more than intended from the interaction with the oracle. A protocol is considered secure if any adversary participating in the protocol can not do more harm than he would do in this ideal oracle scenario.

2.1.2 Garbled Circuits Properties

Defining the security characteristics required for a SFE protocol to be considered secure can be hard as with many other cryptographic protocols and primitives. Instead, as usual in the world of cryptography, it is easier to list and explain the basic properties required for the protocol when it is compared to the ideal oracle mentioned in section 2.1.1. Yao suggested this approach [15] and reduced the list to three basic properties that a garbled circuit should have, thus performing identical to an ideal oracle.

2.1.3 Correctness

A garbled circuit g_c must perform indistinguishable from an ideal oracle by computing the function f as expected. In simple words, *correctness* refers to the ability of a garbled circuit g_c to produce the same result as the *non-garbled* version of the circuit c of a function f given the equivalent inputs for each version of the circuit.

2.1.4 Privacy

A garbled circuit g_c must perform indistinguishable from an ideal oracle by preventing B from learning In_A given that A followed the protocol correctly. Similarly, preventing A from learning In_B given that B followed the protocol correctly. This property does not ensure that A or B will not learn anything by examining the output of the function, for example, if the function is a simple integer addition or multiplication. Using a garbled circuit provides privacy but does not stop A or B from learning each other's inputs by examining the output.

2.1.5 Fairness

A garbled circuit protocol g_c is said to be fair of it allows to return the computed value u to both Alice and Bob. Alice should not deny the result to Bob or vise-versa. This property does not hold under the *malicious* model due to the possibility of one party denying to the other party the output of the function after evaluating the circuit by aborting the protocol before sharing the output.

2.2 Commitment Scheme and Coin Tossing

Later in this paper the ideas of *commitment* schemes and *coin-tossing* protocols are used to define phases or steps taken during the garbled circuit protocol. These two ideas are basic building blocks for cryptographic protocols and are briefly explained here.

A *commitment* scheme allows a party (Alice) to commit to a chosen value without revealing what that value is, then later on the commitment scheme, the value is decommitted to reveal what was Alice's original chosen value, the same logic can be applied for Bob. The goals of *commitment* schemes are correctness, binding and hiding; the protocol binds a value to a party so that the party

cannot change the value later during the protocol, such value is kept secret (hiding) until the decommitment phase is reached and correctness provides a way for Bob to verify the value Alice committed.

Originally[3], a *coin-tossing* protocol is a protocol that uses *commitments* to solve a dispute selecting binding values, but coin-tossing protocol has many uses. In a garbled circuit protocol, the *coin-tossing* protocol is used between Alice and Bob to get a uniformly random bit that later is used to reveal certain information. A summary of the protocol is shown in Protocol 1.

Protocol 1 Simple *Coin-Toss*.

1: Alice "calls" a coin flip, $A_c \in \{0, 1\}$.
2: Alice generates a *commitment* of her "call" and sends it to Bob.
3: Bob tosses the coin and reports the outcome to Alice.
4: Alice decommits (i.e. reveals) her value.
5: Bob verifies the value matches Alice's commitment.
6: If Alice's value matches the coin toss result reported by Bob, Alice wins.

3 Adversarial Behavior

The aim of SFE and specifically Yao's garbled circuits is to protect an honest party of dishonest behavior from an adversary. Generalizing the types of attackers into two categories makes easier to define if a garbled circuit is secure under certain adversary model. It is said that a garbled circuit protocol is secure if the properties mentioned before are achieved in that protocol.

3.1 *Semi-honest* model

Under this model, the basic assumption is that a *semi-honest* adversary follows the protocol as it is intended and he does not actively tries to cheat or cause harm, instead the adversary tries to learn as much as possible only by observing the output and transcript of the protocol.

3.2 *Malicious* model

Under this model, a *malicious* adversary actively tries to cheat and purposely deviates from the protocol to take advantage, trying to obtain input values, denying the protocol by aborting the protocol or providing wrong inputs to the protocol. It is impossible to achieve *fairness* under this model because the adversary can abort the protocol just before sending the computed output to the honest party.

4 Yao's Garbled Circuits

A general and simplified description of a Yao's protocol using garbled circuits is presented now to provide an idea how the garbled circuits are constructed and used in the protocol. Before going directly to the protocol, this section presents a tool used in the protocol to exchange one-out-of-several values, this tool is called Oblivious Transfer (OT) and it is a building block for the garbled circuits protocol. A simplified version of OT that is secure against *semi-honest* adversaries is mentioned and it is meant to be used only for understanding purposes, it is worth mentioning that a lot of research ([1], [2], [7], [4], [8]) is being done to provide security for OT against *malicious* adversaries.

4.1 Oblivious Transfer (OT)

OT is a protocol by itself and it is essential for Yao's protocol. OT is a method for two parties to exchange, in this case, *1-out-of-n* values with the sending party ignoring which value the receiving party selected and the receiving party blinded to the possible values that could have selected. The original protocol is explained in more detail in [13] but a special case *1-out-of-2* values developed by [5] is presented here as an example.

4.1.1 1-out-of-2 OT

The following protocol description in Protocol 2 defines the steps to perform *1-out-of-2* Oblivious Transfer, as mentioned before the next version is secure only under the *semi-honest* model.

Protocol 2 *Semi-honest 1-out-of-2* OT [14].

1: Alice has a set of two values, $V \in \{v_0, v_1\}$.
2: Bob wishes to learn one of the two values and selects $i \in \{0, 1\}$.
3: Bob generates a key pair k^{pub}, k^{priv} and a random value that is indistinguishable from a public key k^{rand}.
4: Bob advertises (k_0^{pub}, k_1^{pub}) as public keys, where $k_i^{pub} = k^{pub}$ and $k_{i-1}^{pub} = k^{rand}$.
5: Alice generates $c_0 = Enc(v_0, k_0^{pub})$ and $c_1 = Enc(v_1, k_1^{pub})$.
6: Alice sends c_0 and c_1 to Bob.
7: Bob computes $v_i = Dec(c_i, k^{priv})$.

If both parties follow the protocol correctly, the output value is correct. Alice does not learn which value Bob chose and Bob only knows the value for the i he selected.

4.2 Brief Description of the Garbled Circuit Protocol

In this section the Yao's *garbled circuit* protocol is decomposed in several steps, explaining in a high level every step and finally providing a summary of the

protocol as a quick reference.

4.2.1 Function to Circuit

The function f representing the function to be evaluated is converted to a boolean *non-garbled* circuit c in which $\forall x, y \rightarrow f(x, y) = c(x, y)$. According to Goldreich et al. in [6] it is possible to map any poly-time function f with fixed size input to a boolean circuit c that returns the same result.

4.2.2 Garbling the Circuit

A circuit c is composed by n gates, in this step the truth table of each gate is garbled in order to generate the final garbled circuit g_c.
As an example for garbling a gate g_1, Alice has a circuit c composed of only one OR gate g_1^{OR}. Alice starts by generating the truth table of gate g_1^{OR} as shown in Table 1.

w_0	w_1		$w_2 = w_0 \vee w_1$
0	0		0
0	1		1
1	0		1
1	1		1

Table 1: g_1^{OR} truth table.

Alice generates a key for each possible value of every column in the truth table, 2 keys for w_0, 2 keys for w_1 and 2 keys for w_2, 6 keys in total. Every column in the truth table represents a wire in the gate g_1^{OR}. Then, Alice proceeds to encrypt every entry in w_2 using the corresponding input keys in w_0 and w_1 as shown in Table 2, then the rows are randomized to obscure the boolean values. Encryption is important because it helps the outputs to look random and additionally prevents Bob from obtaining further information. Bob gets the correct information if he uses the correct keys, otherwise the output looks completely random.

w_0	w_1		w_2		garbled value
k_0^0	k_1^0		k_2^0		$Enc(k_0^0, Enc(k_1^0, k_2^0))$
k_0^0	k_1^1		k_2^1		$Enc(k_0^0, Enc(k_1^1, k_2^1))$
k_0^1	k_1^0		k_2^1		$Enc(k_0^1, Enc(k_1^0, k_2^1))$
k_0^1	k_1^1		k_2^1		$Enc(k_0^1, Enc(k_1^1, k_2^1))$

Table 2: Garbled g_1^{OR} truth table.

Note that the output values that are not input values for other gates do not need to be garbled as these values are the output values of the whole garbled circuit g_c and are meant to be learned by both Alice and Bob.

4.2.3 Sending Alice's Garbled Values

Once that Alice has generated the garbled circuit she needs to define her own input values to send to Bob, then Alice selects the appropriate input keys for the garbled circuit g_c. Following the previous example, if Alice's first input bit is 0, she needs to send the garbled representation (k_0^0) as the first input bit to the garbled circuit. This process is repeated for all Alice's input bits, generating Alice's garbled input values and sending them together with the garbled circuit g_c to Bob.

4.2.4 Using OT for Bob's Input Values.

From previous step Bob receives the garbled circuit g_c and Alice's garbled input values, now Bob needs the garbled representation of his input bits. Alice knows the garbled representation for each possible input bit but she does not know Bob's choice of input bits, therefore they engage in a *1-out-of-2* OT protocol (Protocol 2) for each input bit from Bob. After performing OT as many times as needed, Bob ends with all the information that he needs to evaluate the circuit.

4.2.5 Evaluating the Garbled Circuit

Finally Bob can evaluate the garbled circuit using all the information from previous steps. For each input wire, Bob selects the corresponding input garbled values and decrypts the output of the whole truth table. Only one of the four values will decrypt correctly, the rest of the decryptions will produce random values. Bob continues to perform this procedure until he gets to the output wires of the garbled circuit, every output wire provides a single unencrypted bit. Bob concatenates the bits and produces a result for the function f encoded in the garbled circuit g_c. Protocol 3 summarizes Yao's garbled circuit protocol.

Protocol 3 Semi-Honest Yao's Garbled Circuit.

1: Alice generates a circuit representation c of function f.
2: Alice transforms the circuit to a garbled representation g_c by garbling every gate.
3: Alice sends g_c and garbled representation of her input values to Bob.
4: Alice and Bob perform *1-out-of-2* OTs for every input bit from Bob to receive the garbled representation.
5: Bob calculates c_g with input values and outputs the result.

5 Active Security in Garbled Circuits

Now that Yao's garbled circuit protocol has been explained in Section 4 it is easier to discuss the security limitations and considerations needed specifically when constructing the garbled circuits. The main goal of active security in garbled circuits is to ensure the garbled circuit constructed by Alice is constructed correctly even if Alice is a *malicious* adversary and tries to get Bob's input

values. The next subsections aim to provide details of the *cut-and-choose* technique and different approaches to this technique used to secure garbled circuit construction.

5.1 Standard *Cut-and-Choose* Technique

Assume that Alice is a *malicious* adversary and she constructs a garbled circuit that computes a completely different function from what she originally agreed with Bob. Yao's original garbled circuit idea only works under the *semi-honest* model, thus the *cut-and-choose* technique was first discussed and developed by Malkhi et al. in [12] to prevent this type of attacks. According to this technique, Alice constructs m garbled circuits and sends them to Bob. Bob selects $m-1$ of them and asks Alice to *open* them using the decryption keys corresponding to each circuit. Bob checks that the circuits are constructed correctly, if they are constructed correctly then Bob evaluates the remaining circuit and evaluates the output. The security behind this idea is that if a *malicious* Alice constructs even one incorrect circuit, with high probability Bob will detect the attack. In theory this technique helps detecting a *malicious* Alice constructing incorrect circuits but opens new problems and possible attacks.

5.1.1 Flaws of Standard *Cut-and-Choose* Technique

Previous *cut-and-choose* technique provides improved security against a malicious Alice when constructing the circuit but some problems are still open. Using this standard *cut-and-choose* technique, Alice can bypass it by creating two valid sets of keys for each garbled circuit, one set that produces the correct output values and another set that produces incorrect output values. When Bob asks to open the garbled circuits, Alice uses the correct set of keys, thus producing correct outputs and convincing Bob the circuits are correctly created. Then Bob asks for the garbled values to evaluate the remaining circuit, Alice sends the incorrect set of keys. Hence *malicious* Alice bypasses the *cut-and-choose* technique and produces an incorrect output. Section 5.3.2 shows how to prevent this specific attack by proving input values consistency through the use of commitments.

5.2 Improved *Cut-and-Choose* Technique

Lindell and Pinkas in [10] mention a similar but improved *cut-and-choose* technique used for their secure version of garbled circuit construction. Their approach is very similar but has a few differences that improve the original version. According to this improved technique, Alice constructs m garbled circuits and sends them to Bob. Bob selects not $m-1$ but $m/2$ (i.e. half) of them and asks Alice to *open* them using the decryption keys corresponding to each circuit. Bob checks that the circuits are constructed correctly, if they are constructed correctly then Bob evaluates the remaining $m/2$ circuits (assuming the total number of circuits is even) and evaluates the output. The security behind this

idea is that in order for a *malicious* Alice to succeed, she needs to construct more than $m/4$ incorrect garbled circuits and none of these circuits need to be in the $m/2$ circuits chosen by Bob. Hence with a high probability Bob will detect that Alice attempted to cheat. The main improvement of this variant is that Alice's probability of succeeding in the attack is lower than the standard version. In the standard version Alice has a probability of succeeding $1/m$ while in this version Lindell and Pinkas prove in [10] that Bob aborts the protocol with probability $1 - 2 \cdot 2^{-s}$ where s is the number of circuits. Also, computation for circuit evaluation is distributed between Alice and Bob, Alice only opens $m/2$ circuits for Bob and the rest $m/2$ circuits are left for Bob to evaluate.

Achieving security in a garbled circuit under the *malicious* model using only the improved *cut-and-choose* is rather complicated since it still has some issues. For that reason additional techniques are used together with the improved *cut-and-choose* to achieve security under the *malicious* model.

5.2.1 Avoid Aborting as a Counter-Measure

Following the improved *cut-and-choose* technique mentioned previously, the protocol has to define what to do in the case that Bob detects that some of the $m/2$ circuits he evaluates return different values. A logical idea is that Bob should abort the protocol because different outputs mean that Alice attempted to cheat and she constructed incorrect circuits, however aborting the protocol could result in the following attack. Assume Alice constructs most of the circuits correct except a small set, the "incorrect circuits" are a set of circuits that compute the *exclusive-or* of the function output with Bob's first bit input. If Bob complains about the output being incorrect, then Alice learns that Bob's input value was 1. Table 3 shows the information that Alice gets if Bob decides to abort the protocol.

f_0	Bob_{in_0}	$f_0 \oplus Bob_{in_0}$	Bob Returns	Alice learns
0	0	0	0	$Bob_{in_0} = 0$
0	1	1	$\perp$	$Bob_{in_0} = 1$
1	0	1	1	$Bob_{in_0} = 0$
1	1	0	$\perp$	$Bob_{in_0} = 1$

Table 3: Attack by aborting *cut-and-choose*.

In order to avoid this attack Bob must continue with the protocol instead of aborting. From the output values of all the circuits he evaluates, Bob must take the majority of the outputs, in [10] is proved that Bob gets the correct output with a high confidence by taking the majority of output values.

5.3 Achieving Correctness and Consistency

The following subsections show additional techniques used to ensure correctness of the protocol and consistency of the inputs, these techniques are needed because Alice who is the circuit constructor, and at lesser extent Bob, may try

to cheat in many different ways. For example as mentioned in Section 5.1.1 Alice can create two correct sets of keys that produce different outputs and use them at her advantage or modify the circuit to leak partial information from the circuit evaluator. The general idea of these techniques is to use checks by both parties to make sure each other behaved correctly during the protocol execution.

5.3.1 Encoding Bob's input

Once again, a *malicious* Alice wants to learn some information from Bob's inputs but now with the improved *cut-and-choose* she is forced to build correct circuits since the probability of being detected is very high and not worth it. Alice decides to build correct circuits but give inconsistent garbled inputs to Bob during the OT subprotocol. For example, Alice sends a corrupted garbled value for the first bit of the garbled circuits during the OT subprotocol, if Bob aborts the protocol because he is unable to decode the garbled tables during the circuit evaluation, then Alice knows what was Bob's input bit. Let us assume, Alice corrupts the garbled value of 0 for Bob's first input bit and if Bob's first input bit was 0, Bob will evaluate the circuit and he will receive an incorrect output value, aborting the protocol. On the other hand if Bob's input bit is 1 and the corrupted garbled value was 0, then Bob will continue with the protocol and Alice will learn Bob's input bit.

In order to avoid this attack, as shown in Figure 1, Bob can encode each of his input bits with s new inputs that are joined using an *exclusive-or* operation to form an input bit for each input gate of the garbled circuit. This approach prevents the attack mentioned at the beginning of the section because Bob has 2^{s-1} different ways to encode a 0 bit and 2^{s-1} different ways to encode a 1 bit input. This idea introduces an increase in communication overhead during the Oblivious Transfer from n to ns but Lindell and Pinkas managed to reduced the overhead to $O(max(s, n))$ using additional tricks explained in [10].

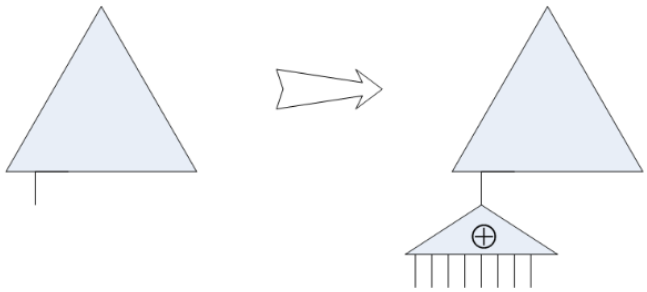

Figure 1: Securing Bob's input by replacing one input wire (left) with the *exclusive-or* of s inputs (right) [10].

5.3.2 Consistency of Constructor's Inputs

One remaining issue is that Alice should provide Bob with consistent garbled input values for all the garbled circuits Bob needs to evaluate. So far, as mentioned in Section 5.1.1, there is nothing that prevents Alice from giving different garbled input values to Bob for the garbled circuits they need to open or evaluate, respectively. Hence, in order to prevent this inconsistency, Alice has to prove to Bob that her inputs remain consistent throughout the whole protocol. The main idea is to jointly use the *cut-and-choose* technique with commitments of Alice's garbled input values, the following summary of steps describe how the consistency of garbled input values is achieved.

Step 1: Commitment creation Alice commits to her own input and Bob's input values. For Bob's inputs, she generates commitments to the input garbled values corresponding to Bob's possible input bits in each circuit, in this case the commitments are ordered pairs where the first commitment represents the 0 input bit and the second commitment the 1 input bit.

Alice builds s garbled circuits, she needs to generate commitments for each of her input bits in each of the s garbled circuits while maintaining the commitment properties.

Now, for every Alice's input wire i, Alice generates s pairs of *commitment sets*. An understandable representation of a pair of *commitment sets* is $\{W_{i,j}, W'_{i,j}\}_{j=1}^{s}$ where i is the i-th input wire and j is the j-th pair of *commitment sets*.

Let $b \in \{0,1\}$ be a random bit chosen independently for each $\{W_{i,j}, W'_{i,j}\}$ pair. A compact representation of a *commitment set* pair is given by $W_{i,j} = \{com(b), com(k_{i,1}^{b}), com(k_{i,s}^{b})\}$ and $W'_{i,j} = \{com(1-b), com(k_{i,1}^{1-b}), com(k_{i,s}^{1-b})\}$ where i represents the i-th input wire, b represents the random input bit, s represents the s-th garbled circuit. The fact that b is chosen randomly ensures that $W_{i,j}$ with probability $1/2$ will contain commitments to value 0 and with probability $1/2$ will contain commitments to value 1, same goes for $W'_{i,j}$.

Note that each set contains $s+1$ commitments, the first commitment is a commitment of the indicator bit $com(b)$ and the rest are commitments to each of the i-th input wires for every of the s circuits. Additionally, it is worth remarking that in each pair ($W_{i,s}$, $W'_{i,s}$) the values that are committed to are the same but using a different randomness for b and the construction of the commitment. A comprehensive image is shown in Figure 2.

At the end of this step, Alice sends only the s garbled circuits created and the s *commitment sets*. Assuming Alice's input is of length n, there exists sn pairs of commitment sets and and a total of $sn(2s+2) = O(s^2 n)$ commitments.

Step 2: Challenge Using the *coin-tossing* protocol (shown in Protocol 1) Alice and Bob agree on two uniformly random distributed strings of length s ($\rho, \rho' \in R\{0,1\}^s$). The 1 bits in the string ρ determine which garbled circuits must be opened, similarly 1 bits in string ρ' determine which *commitment sets*

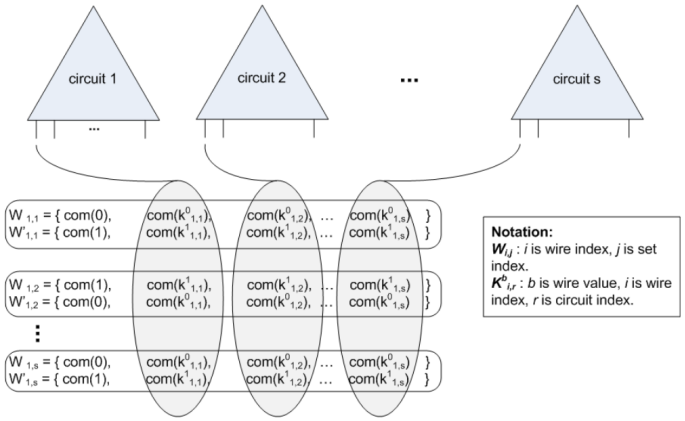

Figure 2: Alice's input commitment creation for first wire [10].

must be opened by Alice. Opened circuits are renamed to *check-circuits* and opened *commitment sets* are renamed *check-sets*.

Step 3: Opening Alice starts by opening the commitments corresponding to Bob's input values in all the *check-circuits*, then Alice opens the commitments in the *check-sets* that correspond to *check-circuits*, meaning that Alice decommits the values $com(k^0_{i,r})$ and $com(k^1_{i,r})$ for the $r-th$ circuit for each input bit i. Finally, Alice decommits the indicator bit value in each of the sets $W_{i,j}, W'_{i,j}$. Figure 3 shows the first input wire commitments to be opened for given *check-circuits*.

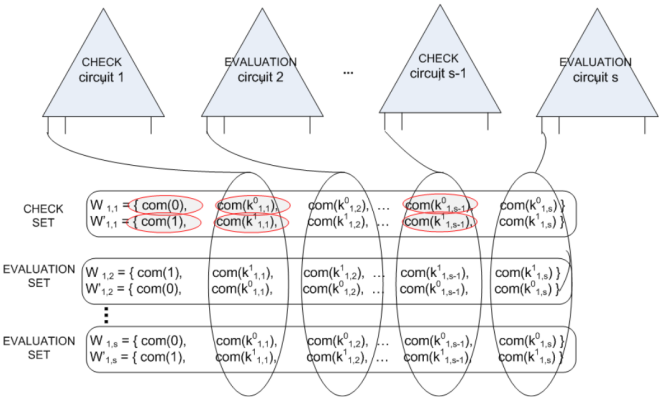

Figure 3: Alice's opening 1st input commitments for *check-circuits* [10].

Step 4: Verification Bob needs to verify the correct construction of the *check-circuits* and Alice's input commitments. For each *check-circuit* Bob receives the decommitments of his own garbled input values and the decommitments to Alice's garbled input values together with the indicator bit, then Bob proceeds to check that for each input wire i the garbled values of 0 (or 1) are equal. Once Bob checks the input garbled values, he decrypts each garbled table and checks the circuits are correctly constructed, meaning the circuits compute the agreed function f.

Step 5: Evaluation Finally, Alice reveals the garbled values corresponding to her input bits, namely, she decommits the key of the i-th input wire of the r-thf circuit with value b ($k^b_{i,r}$) in all the of the *evaluation sets*, as shown in Figure 4. Bob checks that the decommitments for each wire are equal and that opened commitments belong to only one set $W_{i,j}$ or $W'_{i,j}$ for every i, j, otherwise it implies that Alice's inputs are not consistent, giving a different garbled value 0 or 1 for the same input wire i for the *evaluation circuits*.

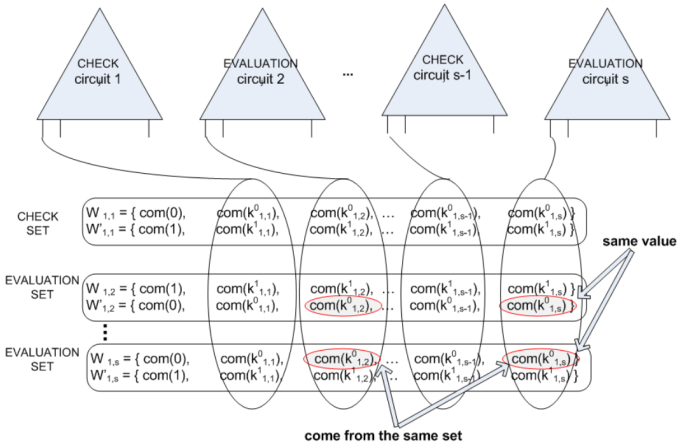

Figure 4: Bob checks commitments [10].

5.4 Secure Two-Party Computation in the Presence of *Malicious* Adversaries

Using all the techniques discussed previously is possible to finally describe a complete garbled circuit protocol that provides security under the *malicious* model. This paper presents only the protocol but the proofs of security for the protocol and the techniques used here are provided in [10]. The high level description of the protocol is as follows.

Step 0: Circuit Construction Alice and Bob want to securely compute the output of $f(In_A, In_B)$ using their respective inputs, In_A and In_B, while

preserving privacy. Both parties decide on a function f and then Alice converts this function into a garbled circuit g_c. The circuit g_c is modified in a way that permits the encoding of Bob's inputs as mentioned in 5.3.1, the resulting number of input wires increases by a factor of s.

Step 1: Commitment Construction Alice has to prove consistency of the garbled circuits and her inputs. Using the consistency technique explained in 5.3.2 Alice generates the required *commitments* for the inputs and additionally she generates commitments for the s copies of the garbled circuits. Commitments enable the protocol to use *consistency checks*.

Step 2: Oblivious Transfer For every input bit of Bob, parties engage in an actively secure *1-out-of-2* OT. After performing OT, Bob gets the garbled representation of his input bits.

Step 3: Sending Circuits and Commitments Alice sends to Bob the garbled circuits and the commitments created in step 1.

Step 4: Defining Check-Circuits and Check-Sets The step described in 5.3.2 is performed. Alice and Bob run the *coin-tossing* protocol several times to define *check-circuits* and *check-sets*.

Step 5: Decommitment Phase for Check-Circuits Alice reveals the *check-circuits* and *check-sets* according to the strings from the previous step, this step is as described in 5.3.2.

Step 6: Decommitment Phase for Alice's Input in Evaluation-Circuits Alice reveals her input garbled values that correspond to her inputs in *evaluation-circuits* as mentioned in 5.3.2 and Figure 4.

Step 7: Correctness and Consistency Checks Bob checks the correctness of the *check-circuits*, the correctness of his inputs in the *check-circuits* (Figure 3) and finally the correctness of Alice's inputs in the *evaluation-circuits* (Figure 4).

Step 8: Circuit Evaluation If all the checks passed, Bob evaluates the *evaluation-circuits* and takes the majority value as the correct output. Otherwise Bob aborts and outputs $\perp$.

6 Further Work in *Cut-and-Choose*

Up to now, the *cut-and-choose* technique has been discussed for the *semi-honest* and *malicious* settings. This technique is very useful and as can be seen, it is the basis building technique for achieving increased security during secure

function evaluation. On the down side, the first iteration discussed in Section 5.1 increased the overhead slightly and later in Section 5.4 the computation and communication overhead increases quadratically due to the commitment creation per input wire per circuit.

In [9] Lindell proposes an asymptotically improved *cut-and-choose* technique which achieves a cheating probability of 2^{-s} where s is the number of garbled circuits, compared to the previous best cheating probability of $2^{-0.32s}$. The most important aspect of this work is that each circuit is chosen independently as *check* or *evaluation* circuit with probability $1/2$ and Alice cheats successfully if and only if all the *checked* circuits are correct and all the *evaluation* circuits are incorrect. Bob receives Alice's input x if any of the *evaluation* circuits is incorrect and allows Bob to evaluate the circuit function $f(x, y)$ where y is Bob's input values.

Moreover, in [11] Lindell and Riva improve once more the efficiency of the *cut-and-choose* technique, this time authors achieve an amortized cost of $O(\frac{s}{logN})$ when N secure computations are run, note that this approach works for multiple/batch executions. This improvement does not look significant but in the long run it can reduce the number of circuits per execution from 20 (for $N = 10$) to 4.08 (for $N = 1,048,576$) with an error of 2^{40}. The main idea proposed here is the creation of $c \cdot N$ circuits (instead of $s \cdot N$) where c is a constant number, then Alice opens a subset of the circuits and the rest are put in N small buckets of size B. One circuit from each bucket is used for each evaluation.

7 Conclusions

This paper presented a general overview of some of the current ideas used to provide security for garbled circuit protocols under the *malicious* setting. As it is most of the time, there are no *one-fits-all* solutions but a combination of primitives, classic and new ideas such as the *cut-and-choose* technique, commitments and *coin-tossing protocol* have proved to be very useful to secure the circuit construction and input consistency. Garbled circuit security and garbled circuit optimization are active fields of research that require a lot of effort and right now the priority of this research is to optimize the circuit construction while maintaining the same security because communication overhead can increase quadratically. Researchers try to find the best trade-off between the security achieved, computation overhead and communication overhead.

Lindell and Pinkas are the authors that produce more literature regarding the active security of garbled circuits and they improve their research based in the general ideas of *cut-and-choose* presented in this paper.

References

[1] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 535–548. ACM, 2013.

[2] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In *Advances in Cryptology–EUROCRYPT 2015*, pages 673–701. Springer, 2015.

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

[4] T. Chou and C. Orlandi. The simplest protocol for oblivious transfer.

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

[6] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

[7] M. Keller, E. Orsini, and P. Scholl. Actively secure ot extension with optimal overhead. In *Advances in Cryptology–CRYPTO 2015*, pages 724–741. Springer, 2015.

[8] A. Y. Lindell. Efficient fully-simulatable oblivious transfer. In *Topics in Cryptology–CT-RSA 2008*, pages 52–70. Springer, 2008.

[9] Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *Advances in Cryptology–CRYPTO 2013*, pages 1–17. Springer, 2013.

[10] 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.

[11] Y. Lindell and B. Riva. Cut-and-choose yao-based secure computation in the online/offline and batch settings. In J. Garay and R. Gennaro, editors, *Advances in Cryptology CRYPTO 2014*, volume 8617 of *Lecture Notes in Computer Science*, pages 476–494. Springer Berlin Heidelberg, 2014.

[12] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella, et al. Fairplay-secure two-party computation system. In *USENIX Security Symposium*, volume 4. San Diego, CA, USA, 2004.

[13] M. O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

[14] P. Snyder. Yaos garbled circuits: Recent directions and implementations.

[15] A. Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.