Information & Security: An International Journal

Abdullatif Shikfa, vol.37, 2017, 11-27 https://doi.org/10.11610/isij.3701

GARBLED CIRCUITS: OPTIMIZATIONS AND IMPLEMENTATIONS

Abdullatif SHIKFA

Abstract: Garbled Circuits were first introduced by Yao in 1984 as a generic approach to perform secure two-party computation between two semi-honest participants. While the result already has a great theoretical significance, it was believed to have very limited applicability due to performance aspects. In the last ten-fifteen years, though, many researchers revived this approach by optimizing one aspect after the other, which results in total in several orders of magnitude of speed-up. In this article, we start by describing the original garbled circuits construction through a simple example. We then review the optimizations of garbled circuits, from point-and-permute to half-gates, going through garbled row reduction, oblivious transfer extensions, and free XOR. Finally, we present several projects that implemented garbled circuits with some of these optimizations, starting from fairplay to the more recent approaches of OblivC and ObliVM.

Keywords: Cryptography, Garbled Circuits, Secure Multi-Party Computation, Optimizations, Implementations

Introduction

Assume that two competing companies want to perform analytics on their respective databases. If each company computes the analytics on its own database, they get some result, but if they perform the analytics on the concatenation of both databases, they will get more accurate results (because the sample space will be larger). However, they do not want to reveal their respective database to the other party as this is a precious and private asset. Their goal is, therefore, to be able to perform the analytics on the concatenation of their private databases: both companies agree to learn the analytics result, but they do not want to leak information about their database to the other entity. This is a typical application scenario of secure multi-party computation.

This problem was addressed in the early eighties by Andrew Yao[1,2] who proposed garbled circuits as a generic secure two-party computation protocol. The high-level idea is that any function represented as a circuit can be garbled by one party and evaluated by the other to enable computation on private data without leaking any additional information on the data beyond the result of the computation itself. While this seminal work was of great theoretical significance, it was regarded as unpractical for years. However, in the past fifteen years many researchers worked on the concept again and proposed optimizations that improved the efficiency of garbled circuits by several orders of magnitude. All these optimizations paved the way to the development of libraries that use garbled circuits as a core building block to perform secure computation.

In this article, we start by explaining through an example the fundamentals on garbled circuits in section 1. We then present several optimizations that are key to improve the performance of garbled circuits in section 2. Finally, we present some of the recent libraries that implement garbled circuits and beyond in section 3.

1. Garbled circuits

Garbled circuits are a generic secure two-party computation protocol, which were introduced by Yao[1,2] first and then improved in many subsequent works. We adopt a description by example to better explain how garbled circuits work. Assume we have two players Alice and Bob who choose two bits each and want to check whether some of their choices are the same or not. We already observe here that the function that the players want to compute is publicly known to both entities but the inputs are private. For the sake of simplicity we assume the output of the function should be 1 if both inputs are different and 0 if they have at least one common bit.

1.1. Standard circuits

The first step is to build a circuit that achieves this functionality. Converting a function to a circuit is out of scope of this article, but in this simple case, it is easy to check that the following circuit achieves the required functionality.

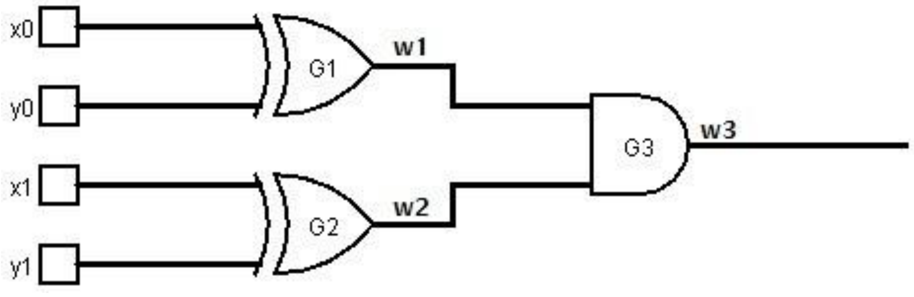

Figure 1: Sample circuit.

In this circuit x0 and x1 are the input of Alice, y0 and y1 the input of Bob, G1 and G2 are two XOR gates, G3 is an AND gate and w1, w2 w3 are the output wires of each gate, w3 being the final output of the circuit. In such a classical circuit each wire (including the input) is a bit which can take value 0 or 1, and the gate are defined by their truth table which are depicted below:

Table 1: Truth table of the XOR gate (left) and the AND gate (right).

x	y	W
0	0	0
0	1	1
1	0	1
1	1	0

x	y	w
0	0	0
0	1	0
1	0	0
1	1	1

Evaluation of standard circuits

To evaluate the circuit, one needs to go from left to right and top to bottom and get the output of each gate depending on its inputs through a lookup in the truth table. For example if x0=0, x1=0, y0=0, y1=1, then w1=0 (because G1 is an XOR gate with input 0 and 0 which corresponds to the first row of the truth table) and w2=1 (because G2 is an XOR gate with input 0 and 1 which corresponds to the second row of the truth table) and w3=0 (because G3 is an AND gate with input 0 and 1 which corresponds to the second row of the truth table).

Performance Analysis

In terms of performance, we observe that each gate is represented by a table which can be represented by four bits (assuming the rows always follow the same order). So the communication complexity is simply 4g+n, where g is the number of gates and n is the number of input bits. In terms of computation complexity, we have to perform a simple lookup per gate. Evaluating a circuit is thus extremely efficient.

1.2. Construction of garbled circuits

Now we would like to do the same while hiding the input. The idea of Yao is the following. We assume a general security parameter κ. In the current state of the art κ would typically be 128 bits, but for the sake of simplicity here, we will make it only 16 bits for illustration. Remember that we will consider one entity as the garbler (Alice, for example) and the second as the evaluator.

Alice starts by choosing two random numbers of size κ for each input and each wire. These random numbers will correspond to the garbling of the 0 and 1 values of these wires. The important security properties achieved by this step are:

- The evaluator will learn one of these two values, but he will not be able to learn the other one (because the other one is unrelated to the first one contrary to the deterministic values 0 and 1).
- When the evaluator learns one of the values, he does not know whether this value corresponds to 0 or to 1.

Note that the garbler is choosing these random values both for himself and for the input of the evaluator. An instantiation of this phase is represented in **Error! Reference source not found.**, where the top number represents the garbling corresponding to the value 0 and the below one the garbling corresponding to the value 1 for each wire (these are 4 hexadecimal digits corresponding to 16 bits).

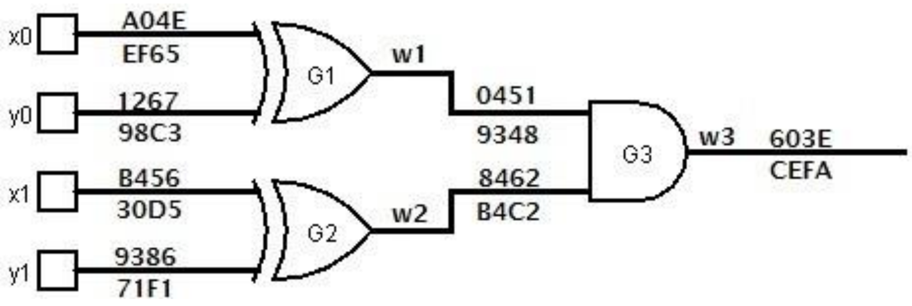

Figure 2: Sample circuit with garblings for the wires.

The truth table of the gates should now be represented with the garbled wires instead of the 0 and 1 values. For example, the table of G1 would look as in **Table 2**.

Table 2: Truth table of G1 with garbled values.

x0	y0	w1
A04E	1267	0451
A04E	98C3	9348
EF65	1267	9348
EF65	98C3	0451

However, this is just the first step of computing the garbled truth table. Indeed the current form still leaks a lot of information and one can easily deduce that this table corresponds to XOR or its complement. If one knows on top of that that this is the truth table of an XOR (as the circuit itself is not supposed to be secret) then one can deduce relations among the garbled wires. Finally, this directly reveals the different possible values of the output wires.

To solve these issues, the second step is to hide the values of the input and to encrypt the output with the input. Let us denote by E_k and encryption algorithm under key k, then each row of the table is encrypted twice with the keys being the garblings of the input. This is best seen visually in table 3.

Table 3: Garbled truth table of G1 after step 2.

w1
$E_{A04E}(E_{1267}(0451))$
$E_{A04E}(E_{98C3}(9348))$
$E_{EF65}(E_{1267}(9348))$
$E_{EF65}(E_{98C3}(0451))$

Finally to avoid revealing the information about the order of the inputs (and whether they correspond to a 0 or a 1), the garbled truth table needs to be randomly shuffled.

An example of final garbled truth tables for the circuit is depicted in **Table 4**.

Table 4: Final garbled truth table of the three gates.

w1	w2	w3
$E_{A04E}(E_{1267}(0451))$	$E_{B456}(E_{71F1}(B4C2))$	$E_{9348}(E_{8462}(603E))$
$E_{EF65}(E_{1267}(9348))$	$E_{30D5}(E_{71F1}(8462))$	$E_{9348}(E_{B4C2}(CEFA))$
$E_{EF65}(E_{98C3}(0451))$	$E_{B456}(E_{9386}(8462))$	$E_{0451}(E_{B4C2}(603E))$
$E_{A04E}(E_{98C3}(9348))$	$E_{30D5}(E_{9386}(B4C2))$	$E_{0451}(E_{8462}(603E))$

Note that even though Alice is constructing the whole garbled circuit and know all the garblings of all wires, she is not able to evaluate the circuit because she doesn't know the actual values of Bob's input.

1.3. Transfer of the garbled circuit

With this, the garbler has finished the computations he needs to perform. What is remaining is for him to send the garbled circuit to the evaluator (Bob). More precisely, he sends to Bob:

- The garbled truth tables, as shown in Table 4,
- His garbled inputs. In our example, Alice's input were x0=0 and x1=0, hence the corresponding garbled inputs that he will send to Bob are x0=A04E and x1=B456.

Note that Bob cannot link the garbled inputs of Alice to their clear text counterparts. The last step before evaluation is that Bob needs to get his input. The tricky part here is that:

- Bob should only get the garbled version of his input and not the garbled version of the complement to his input (otherwise he would be able to compute the function on all possible values of his variables). Hence Alice cannot simply send to Bob both garbled values of each input of Bob.
- Alice should not learn the input of Bob as well, hence Bob cannot simply ask Alice to send the garbled values corresponding to y0=0 and y1=1 for privacy reasons.

To solve these seemingly contradictory requirements. Alice and Bob enter an oblivious transfer protocol which satisfies exactly these requirement. In a 1 out of 2 oblivious transfer protocol the sender (Alice) has two values and the receiver (Bob) gets one (and only one) of these inputs without Alice knowing which one he received. Oblivious transfer protocols are a topic of their own that we will not discuss in this paper, but the interested reader can read the seminal contributions of Rabin[3] and Even et al.[4] or read the survey by Phong.[5]

Hence, through two instances of Oblivious Transfer, Bob gets the garblings to his inputs which are y0=1267 and y1=71F1.

1.4. Evaluation of garbled circuits

Now Bob has all the required information to evaluate the circuit. He proceeds gate by gate. For each gate he has two garblings which allow him to decrypt only one of the four rows. To be more precise:

- He starts with G1, where he has the input are x0=A04E and y0=1267 that allow him to decrypt the first row and get the values w1=0451.
- For G2, he has the garblings x1=B456 and y1=71F1 that enable him to decrypt also the first row of the truth table of G2 and resulting in the garbled value of w2=B4C2.
- For G3, the garbled inputs are w1=0451 and w2=B4C2, which enable Bob to decrypt the third row of G3 and leads to the garbled value of w3=603E.

This concludes the evaluation of the garbled circuit and Bob returns the garbling 603E to Alice who is able to translate it as 0. Note that in all the evaluation steps, Bob is unable to determine whether he is evaluating a 0 or a 1 but he is still able to reach the garbled output of the circuit which he can decode to the clear text output of the function with the help of Alice. Hence, Alice and Bob are able to privately compute the function on their inputs. Of course this is an informal argument about the security of the scheme, for a formal proof please refer to the work of Lindell ad Pinkas.[6]

1.5. Performance of garbled circuits

Contrary to the case of clear text evaluation, we observe that each gate is represented by a table which contains four encrypted values each of size κ bits (16 bits in our example, but remember that in typical applications nowadays it would rather be 128 bits). The inputs are of the same size κ so the communication complexity is now (4g+n) κ bits (plus the cost of oblivious transfer, which is not negligible). In terms of computation complexity, the evaluator has to try four pairs of decryption operations (at most) for each gate hence evaluation is also much slower than in the clear text case.

We will now describe the optimizations to reduce both communication and computation complexity.

2. Garbled Circuits' Optimization

To optimize garbled circuits there are two aspects to take care of: computation and communication complexity. First note that the circuit that we took as example only included AND and XOR gates. This is was done on purpose as most optimizations focus on these two gates. And since the set {XOR, AND} is functionally complete it means that any circuit can be written with these two gates alone, hence it is enough to focus on these two gates.

Point and Permute

The first optimization was introduced by Beaver, Micali and Rogaway[7] in 1990 and is called Point and Permute. It mainly focuses on improving the computation complexity to make the circuit evaluation more efficient. Remember that in the basic Yao garbled circuits the evaluator had to try to decrypt the four rows and only succeed in one.

The idea of point and permute is to add a flag called selection bit to each garbling. For each wire, one of the garblings will have the selection bit set to 0 and the other will have the selection bit set to 1. This selection bit is chosen randomly and independently of the real value of the wire (it will correspond to the real value with probability half which gives no information to an attacker).

An example of instantiation is shown in **Error! Reference source not found.**.

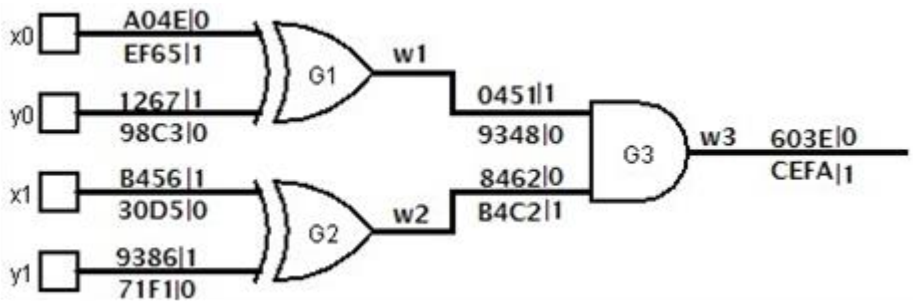

Figure 3: Garbled circuit with selection bits.

The truth tables are then ordered according to the selection bit in the classical order of rows 00, 01, 10, 11. On top of that the ciphertexts no longer need to be from a CPA-secure encryption scheme and can instead be instantiated as C⊕H(A|B) where H is a hash function. Hence both encryption and decryption are faster. The truth table thus become as shown in **Table 5**.

Note that the select bit of the output is XORed with the hash as well, hence it is not visible in clear as shown in the table.

Now the main advantage is that the evaluator knows directly which row he will be able to decrypt and he no longer need to try all the four possible rows: he just has to decrypt the row corresponding to the selection bit he sees at the end of the input wires for the gate. For example for gate 1, Bob has the input x0=A04E|0 and y0=1267|1, hence Bob directly goes to the second row of the first truth table and gets as output w1=0451|1. And the same goes for the two other gates.

Table 5: Truth table with point and permute.

w1	w2	w3
H(A04E|0|98C3|0)⊕9348|0	H(30D5|0|71F1|0)⊕8462|0	H(9348|0|8462|0)⊕603E|0
H(A04E|0|1267|1)⊕0451|1	H(30D5(|0|9386|1)⊕B4C2|1	H(9348|0|B4C2|1)⊕CEFA|1
H(EF65|1|98C3|0)⊕0451|1	H(B456|1|71F1|0)⊕B4C2|1	H(0451|1|B4C2|1)⊕603E|0
H(EF65|1|1267|1)⊕9348|0	H(B456|1|9386|1)⊕8462|0	H(0451|1|8462|0)⊕603E|0

The advantage of this technique is clear: the computation time is reduced by a factor up to four at the cost of just one more bit for each wire.

Row reduction

In 1999, Naor, Pinkas and Sumner[8] proposed a technique to reduce the communication complexity by reducing the number of rows required for each gate. The idea is the following: instead of choosing all garblings randomly it is possibly to chose one of the garblings of the output such that its encryption is 0. For example 1999: technique to reduce the number of rows per AND gate from 4 to 3. For example assume that we choose the first row of each table to be 0. This means that we will replace the value 9348|0 of w1 by H(A04E|0|98C3|0). In that case the first row of G1 will be H(A04E|0|98C3|0)⊕ H(A04E|0|98C3|0)=0.

The other value can still be chosen randomly so we will keep it as 0451|b where b is the complement of the last bit of H(A04E|0|98C3|0). Similarly 8462|0 will be replaced by H(30D5|0|71F1|0) and 603E|0 will be replaced by H(H(A04E|0|98C3|0)|H(30D5|0|71F1|0)).

An example of the full truth table of G1 is shown in **Table 6**. Hence all the first rows are now 0 and there is no need to send this row. In other words, each garbled table now contains only three rows, which reduces the communication complexity by a factor of 25%.

This technique was enhanced by Pinkas et al.[9] in 2009 to reduce further the number of rows to 2. However, this further improvement is not compatible with the free XOR technique (presented in the next section) and is of limited use for this reason. We will, therefore, not cover it in this article.

Table 6: Garbled truth table of G1 with garbled row reduction

w1
0
H(A04E|0|1267|1)⊕0451|b
H(EF65|1|98C3|0)⊕0451|b
H(EF65|1|1267|1)⊕ H(A04E|0|98C3|0)

Free XOR

Kolesnikov and Schneider[10] proposed in 2008 a technique that makes evaluation of XOR gates for free almost. They proved that Yao's garbled circuit technique remains secure if one imposes a fixed difference between the garblings of the 0 and 1 of all wires. Concretely it means that we can choose at random one of the garbling of each wire (say the one corresponding to zero) and we also choose at random a fixed difference d. Then the garbling of the 1 will be computed as the garbling of the 0 XOR the fixed difference d. The advantage now is that we can fix the garbling of the output of an XOR gate as the XOR of the inputs.

Let us take as an example the gate G1. We keep the garblings of the 0 values of the two input as they are, namely A04E|0 and 1267|1. We choose at random a common difference say 258B|1 (it will be the same for the whole circuit). Then the garblings of the 1 value of the input will be respectively A04E|0⊕258B|1=85C5|1 and 1267|1⊕258B|1=37EC|0. For the output wire, the garbling of the 0 will exactly be the XOR of the garblings of the 0 of the input, namely A04E|0⊕1267|1=B229|1, and the garbling of the 1 will be the previous XOR the common difference namely B229|1⊕258B|1=97A2|0. Now it is easy to verify that with this set of input and output, the output is always directly the XOR of the input (we constructed it this ways for the 0 values, but it works for all values). This simply means that by adopting this construction one does not need to send a garbled truth table for XOR gates, and that evaluation of an XOR gate consists simply of XORing the value of the input; hence it is essentially free (compared to the need to decrypt in the case of other gates). On top of that, this technique is compatible with the first garbled row reduction technique presented in the previous section. In summary, we have now:

- free XOR gates,
- AND gates have only three rows.

This means that to optimize the evaluation time of garbled circuits, it is interesting to reduce the number of AND gates as much as possible even at the cost of increasing the number of XOR gates. In **Figure 4**, we show how the circuit with the garblings including all the previously mentioned improvements. In this figure, we draw in red the garblings that are computed to satisfy the free XOR requirement, and in green the garbling that is computed to satisfy the garbled row requirement.

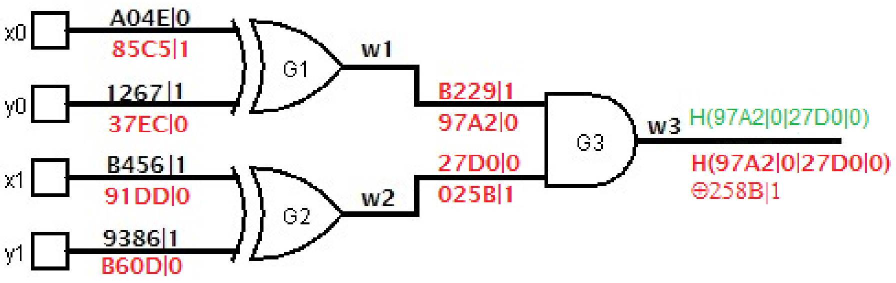

Figure 1: Garbled circuit with free XOR and garbled row reduction.

For this circuit, the only truth table that needs to be sent by Alice to Bob is the one corresponding to gate 3 and it is represented in **Table 7**.

Table 7: Garbled truth table of G3 with garbled row reduction and free XOR.

w3
0
H(97A2|0|025B|1)⊕H(97A2|0|27D0|0)⊕258B|1
H(B229|1|27D0|0)⊕H(97A2|0|27D0|0)
H(B229|1|025B|1)⊕H(97A2|0|27D0|0)

Half Gates

With the previous improvements, we already have free XOR; hence we cannot improve XOR gates further. Concerning the AND gate, we mentioned that Pinkas et al. proposed a technique to reduce the number of rows to two, but it is not compatible with the free XOR technique.

Recently, in 2015 to be more precise Zahur, Rosulek and Evans [11] managed to come up with a technique that makes free XOR compatible with only two rows per AND gate. Their technique, however, involves the introduction of specific gates that have to be evaluated in a non-standard way.

The idea is to divide each AND gate in two half gates: one generator half gate and one evaluator half gate. To illustrate this in our example, we focus on G3 only as this is the only AND gate we have. Zahur et al. made the following observation:

$$w3 = w1 \wedge w2 = w1 \wedge (r \oplus r \oplus w2) = (w1 \wedge r) \oplus (w1 \wedge (r \oplus w2))$$

where r is the select bit of the false value of w2 (0 in our case). This means that the AND gate G3 can be converted to three gates:

- An AND gate between w1 and r, and r is known to Alice (the garbler), called a generator half gate
- An AND gate between w1 and r⊕w2, and r⊕w2 is known to Bob (the evaluator) at evaluation time, because it corresponds to the select bit of the wire w2. This gate is called an evaluator half gate.
- An XOR gate between the previous two, which is implemented using the free XOR technique.

The key idea is that each half gate can be encoded with one encryption only, hence performing this transformation is efficient, and will result in total in only two encryptions (two rows) per AND gate.

Generator half gate

For the generator half gate, the garbler knows the value of r. If the value of r is 0, the garbler just needs to encode a unary gate that always outputs 0, if the value of r is 1 then the garbler needs to encode a unary gate that corresponds to the identity. Since the value of r is known, it will not be added in the hash and the generator will produce the two ciphertexts: H(w1)⊕C and H(w1⊕d)⊕C⊕rd where w1 is the value of the garbling of 0 of the wire w1 (B229|1 in our example), d is the common difference (258B|1 in our example) and the value of C is a garbling that will be explained in few lines. These two ciphertexts are ordered corresponding to the select bit of w1 (which is 1 in our example, hence we reverse the order of these two ciphertext). The evaluator can simply evaluate this gate by taking the hash of the garbling of w1 (which is w1 or w1⊕d) and he will get C if r=0, and C or C⊕d when r=1. We are still free to choose C, hence we will set C appropriately to make the first row always 0 (similar to the garbled row reduction technique) by setting C equal to H(w1), H(w1⊕d) or H(w1⊕d)⊕d depending on the select bits and the value of r. In our example, r=0 and the select bit of w1 is 1 hence we will choose C=H(w1⊕d)=H(97A2|0). At evaluation time, the evaluator will get the value of w1 and he will get as output H(97A2|0).

Evaluator half gate

For the evaluator half gate, the evaluator knows the values of r⊕w2 at evaluation time as it corresponds to the select bit of w2. If r⊕w2=0, Bob should always obtain the value C corresponding to false (it will be set later as in the previous case). If r⊕w2=0, Bob should get C or C⊕d depending on the value of w1. But it is enough for him to get Δ=C⊕w1 and then compute Δ⊕w1 to obtain the correct value of the output. The generator will compute the two ciphertexts H(r⊕w2)⊕C and H(r⊕w2⊕d)⊕C⊕w1. To further reduce one row we choose C= H(r⊕w2) to zero the first ciphertext. In our example, r=0, then C=H(27D0|0) and the ciphertext is therefore H(025B|1) ⊕H(27D0|0)⊕B229|1. At evaluation time,

- if Bob gets the value 27D0|0 for w2, he will compute the output of this gate as H(27D0|0)
- otherwise he will get the value 025B|1 for w2, he will hash it and XOR is with the ciphertext to obtain Δ=H(27D0|0)⊕B229|1 that he will further XOR with the value that he got for w1 to obtain either H(27D0|0) or H(27D0|0)⊕258B|1.

Finally the evaluator has to perform the XOR of the two previous gate and he will get either H(97A2|0)⊕ H(27D0|0) or H(97A2|0)⊕ H(27D0|0)⊕258B|1.

In summary, this technique allows to reduce the number of rows of AND gates by replacing each AND gate by two specific half gates, each of them requiring one row only to be evaluated, and one XOR gate, which is essentially free. Furthermore, Zahur et al. proved that this is optimal under a set of reasonable assumption.

3. Implementations

As we have seen, garbled circuits have seen tremendous theoretical improvements, both on the communication and computation complexity. We will now see how these theoretical improvements have also been implemented in programming frameworks to allow developers to perform secure multi-party computation without deep knowledge of garbled circuits. We describe some of the main tools that are open source and publicly downloadable.

Fairplay (2004)

The Fairplay project by Malkhi et al.[12] is one of the first practical implementations of garbled circuit. It is a java-based implementation of garbled circuits featuring the point-and-permute improvement. The main interest of Fairplay is that it can convert a high level program similar to java to a circuit. The circuit can be a generic one or a circuit with only binary gates. It can also be used by two parties or more. In terms of performance, the implementation does not use much optimizations and is thus not very fast, garbling less than 30 gates per seconds.

FastGC (2011)

Huang et al.[13] proposed FstGC in 2011 as a major improvement over Fairplay. Not only does FastGC implement the known extensions at that time such as free XOR, garbled row reduction and Oblivious Transfer Extensions, but it also solves an important problem of memory exhaustion. Indeed, approaches such as Fairplay were generating the whole garbled circuit first and then evaluating it. As a result, only small circuits could be evaluated. FastGC on the contrary has a pipelined generation and evaluation of the circuit which improves efficiency but also scalability. In terms of programming framework, FastGC allows users to write their programs using a combination of high-level and circuit level Java code. This enables the programmers to perform circuit level optimization but requires them to have good understanding of Boolean circuits. As a result, FastGC can garble around a hundred thousand gates per seconds, and can perform a garbled AES encryption is 0.2s.

TinyGarble (2015)

Songhori et al.[14] proposed TinyGarble, a tool that generates optimized and compressed Boolean circuits. TinyGarble views circuit generation as a logic synthesis task. Hence TinyGarble focuses primarily on the circuit component of Garbled Circuits. Hence it naturally accepts inputs as a standard hardware description language. It also accepts higher level language programs as input as long as they are compatible with existing high-level synthesis tools. TinyGarble is the most efficient for the circuit generation part but is more difficult to use by a developer, as the output of TinyGarble is a netlist (or list of gates) which is further transformed to be used with a full Garbled Circuit implementation. On top of producing circuits with the most efficient memory footprint to date, TinyGarble also performs just in-time garbling and can, therefore, securely evaluate the most practical function with a classical processor.

OblivC (2015)

Zahur and Evans[15] developed Obliv-C, which is an extension of the C language which supports the standard C features as well as extensions for data-oblivious programs. A typical Obliv-C program consists of three files: a header (.h) file, a classical .c file that takes care of the setup, networking aspects, and so on, and a specific 0oc file to take care of the private parts of the program. The programmer can therefore easily develop an application by adding the keyword obliv to the variables that should remain private and the compiler will take care of implementing garbled circuits and other techniques such as oblivious RAM to produce a securely evaluate the function. For the garbled circuits part, Obliv-C includes all the classical optimizations (free XOR, half gates, OT extensions), and complements them with Oblivious Ram and range-tracked integers. Notably, Obliv-C is the only tool in our list which is an extension of C.

ObliVM (2015)

OblivM, developed by Liu et al.,[16] is a complete framework for secure multiparty computation. It is easy to use, based on Java code and supports garbled circuits as back-end (with the project to support additional protocols such as fully homomorphic encryption). This back-end part called ObliVM-GC also includes all the aforementioned optimizations and basically builds on FastGC, improving its performance by a factor seven (garbling roughly 700 thousand gates per second). On top of this back-end part, ObliVM includes a compiler that converts the high-level java program to one or several circuits whose inputs are oblivious memory accesses. ObliVM indeed also adopts on-the-fly circuit generation. ObliVM includes several application examples with benchmarking that shows that slowdown between cleartext computation and secure computation is between one thousand and one million depending on the applications.

Conclusion

In summary, we presented Yao garbled circuits and the optimization that improve its performance. With these optimizations, the computation cost goes down from four pairs of decryption per gate to one decryption only. XOR gates are free to evaluate (actually at the cost of a traditional XOR only), and AND gates can be reduced to only two ciphertexts instead of four, their evaluation requiring two hash functions computation and three XORs. We also presented several implementations of garbled circuits and of complete secure computation frameworks that use garbled circuits as a backend. The performance of these latest implementations are still lacking compared to cleartext evaluation but they are several order of magnitudes more efficient than initial implementation. The last three tools that we presented are very recent showing that this area is a hot topic in the security community, both on the theoretical and implementation side. For instance, on the theoretical side, Wang and Malluhi[17] showed how to reduce the number of ciphertexts per AND gate even further by relaxing the assumptions of Zahur et al.[11]

Furthermore, note that we only presented garbled circuit as a two party protocol but it can also be instantiated in many different ways. For example, if only one party has secret input, garbled circuits can be used for secure delegation of computation[18]. It is also possible to implement garbled circuits with a semi-honest third party that performs the garbling[19]. That enables for example, to verify the consistency of secret input across several executions[20].

Acknowledgement

This work was made possible by the NPRP award X-063-1{014 from the Qatar National Research Fund (a member of The Qatar Foundation). The author thanks Prof Qutaibah Malluhi and Omar Abou Selo for helpful discussions about this work. The statements made herein are solely the responsibility of the author.

Endnotes:

[1] Andrew Chi-Chih Yao, "How to generate and exchange secrets," *27th Annual Symposium on Foundations of Computer Science* (sfcs 1986), *Toronto, ON, Canada*, 1986, pp. 162-167, https://doi.org/10.1109/SFCS.1986.25.

[2] Andrew Chi-Chih Yao, "Protocols for secure computations," *23rd Annual Symposium on Foundations of Computer Science* (sfcs 1982), *Chicago, IL, USA*, 1982, pp. 160-164, https://doi.org/10.1109/SFCS.1982.38.

[3] Michael O. Rabin, "How to exchange secrets with oblivious transfer," Technical Report TR-81, Aiken Computation Lab, Harvard University, 1981.

[4] Shimon Even, Oded Goldreich, Abraham Lempel, "A Randomized Protocol for Signing Contracts," *Communications of the ACM* 28, no. 6 (1985): 637–647, https://doi.org/10.11 45/3812.3818.

[5] Le Trieu Phong, "A Survey on Oblivious Transfer Protocols," *Journal of the National Institute of Information and Communications Technology* 58, no. 3/4 (2011): 181-186.

[6] Yehuda Lindell and Benny Pinkas, "A proof of security of Yao's protocol for two-party computation," *Journal of Cryptology* 22, no. 2 (2009): 161-188.

[7] Donald Beaver, Silvio Micali, and Phillip Rogaway, "The round complexity of secure protocols" *Proceedings of the twenty-second annual ACM symposium on Theory of computing* (STOC '90), April 1990, pp. 503-513. https://doi.org/10.1145/100216.100287.

[8] Moni Naor, Benny Pinkas, Reuban Sumner, "Privacy preserving auctions and mechanism design," *Proceedings of the 1st ACM conference on Electronic commerce* (EC '99), November 1999, pp. 129-139, https://doi.org/10.1145/336992.337028.

[9] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams, "Secure two-party computation is practical," *Advances in Cryptology -- ASIACRYPT 2009: 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10* (Springer, 2009), 250–267, doi: 10.1007/978-3-642-10366-7_15.

[10] Vladimir Kolesnikov and Thomas Schneider, "Improved garbled circuit: Free xor gates and applications," *Automata, Languages and Programming: 35th International Colloquium, (ICALP 2008), Reykjavik, Iceland, July 7-11*, (Springer, 2008), 486–498, https://doi.org/10.1007/978-3-540-70583-3_40.

[11] Samee Zahur, Mike Rosulek, and David Evans, "Two Halves Make a Whole", *Advances in Cryptology - EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30* (Springer, 2015), 220-250, https://doi.org/10.1007/978-3-662-46803-6_8.

[12] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay—a secure two-party computation system," in *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13* (SSYM'04), USENIX Association, 2004.

[13] Yan Huang, David Evans, Jonathan Katz, and Lior Malka, “Faster secure two-party computation using garbled circuits,” in *Proceedings of the 20th USENIX conference on Security* (SEC'11), USENIX Association, 2011.

[14] Ebrahim M. Songhori, Siam U. Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar, “TinyGarble: Highly Compressed and Scalable Sequential Garbled Circuits,” in *2015 IEEE Symposium on Security and Privacy, San Jose, CA,* 2015, pp. 411-428. https://doi.org/10.1109/SP.2015.32.

[15] Samee Zahur and David Evans, “Obliv-C: A Language for Extensible Data-Oblivious Computation,” *Cryptology ePrint Archive,* Report 2015:1153, November 2015.

[16] Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi, “ObliVM: A Programming Framework for Secure Computation,” in *Proceedings of the 2015 IEEE Symposium on Security and Privacy* (SP '15). IEEE Computer Society, pp: 359-376, https://doi.org/10.1109/SP.2015.29.

[17] Yongge Wang and Qutaibah Malluhi, “Reducing Garbled Circuit Size While Preserving Circuit Gate Privacy,” in *Cryptology ePrint Archive*, Report 2017/041, 2017.

[18] Qutaibah Malluhi, Abdullatif Shikfa, and Viet Cuong Trinh, “An efficient instance hiding scheme,” in *Proceedings of the Seventh Symposium on Information and Communication Technology* (SoICT '16), December 2016, pp 388-395, https://doi.org/10.1145/3011077.30 11100.

[19] Uri Feige, Joe Killian, and Moni Naor, “A minimal model for secure computation (extended abstract),” in *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing* (STOC '94), pp. 554-563, https://doi.org/10.1145/195058.195408.

[20] ladimir Kolesnikov, Ranjit Kumaresan, and Abdullatif Shikfa, “Efficient Verification of Input Consistency in Server-Assisted Secure Function Evaluation,” *Cryptology and Network Security: 11th International Conference, CANS 2012, Darmstadt, Germany* (Springer Berlin Heidelberg, 2012), 201-217, https://doi.org/10.1007/978-3-642-35404-5_16.

About the Author

Dr. Abdullatif Shikfa is an assistant research professor at KINDI center for computing research of Qatar University. He obtained his PhD from Telecom ParisTech in 2010 and he is an alumnus of Ecole Polytechnique (MSc. 2004), of University of Nice Sophia Antipolis (MSc. 2005), and of ENST-EURECOM (MSc. 2006). Before joining Qatar University, Abdullatif held both research and industry positions: he served as research engineer at EURECOM, as scientific advisor and deputy head of the security research department at Bell Labs, Alcatel-Lucent and then he worked as technical project manager and security expert at Thales. His research interests and experience span a wide range of topics in information and communication security: he published over twenty peer-reviewed papers in international journals and conferences on various aspects of applied cryptography ranging from trust and cooperation enforcement to secure routing, through protection of users' privacy mainly by applying advanced cryptographic primitives to enable computation on encrypted data.