A Gentle Introduction to Yao's Garbled Circuits

Sophia Yakoubov

Boston Univeristy

Abstract. This is a short, gentle introduction to the Yao's Garbled Circuits construction and recent optimizations, as well as the garbled circuit definitions.

Keywords: Yao's Garbled Circuits

Table of Contents

1 Yao's Garbled Circuits: Background

1.1 Classical Yao's Garbled Circuits

Let's say two parties, Ginny and Evan, want to figure out whether they should collaborate on a project. This is a very sensitive subject for them, so neither one wants the other to learn how they feel unless the other is in favor of the collaboration. We will represent Ginny and Evan's preferences using bits: Ginny's bit g is 1 if she wants to work with Evan and 0 otherwise, and similarly Evan's bit e is 1 if he wants to work with Ginny and 0 otherwise. What they want to figure out is then simply the conjunction (AND, denoted $\wedge$) of their two bits $g \wedge e$, without revealing anything else to one another.

This can be realized using Yao's Garbled Circuits. Ginny, playing the role of a *garbled circuit generator*, or *garbler*, can generate a garbled AND gate and send it to Evan. Evan, playing the role of an *evaluator*, can then evaluate this garbled gate.

Garbled Gate Generation Garbling is a process by means of which the boolean gate truth table is obfuscated. Ginny picks four random strings, or *labels*: W_{G}^0, W_{G}^1, W_{E}^0 and W_{E}^1. W_{G}^0 corresponds to the event that $g = 0$, and W_{G}^1 corresponds to the event that $g = 1$. Similarly, W_{E}^0 corresponds to the event that $e = 0$, and W_{E}^1 corresponds to the event that $e = 1$. Ginny then uses every pair of labels corresponding to a possible scenario ($(g = 0, e = 0), (g = 0, e = 1), (g = 1, e = 0)$ and $(g = 1, e = 1)$) to encrypt the output corresponding to that scenario, as shown in Figure 1. The two relevant labels are put through a key derivation function H to derive a symmetric encryption key, and that key is used to encrypt $g \wedge e$. The *garbled gate* consists of the four resulting ciphertexts, in a random order. The garbling of an AND gate is illustrated in Figure 1.

g	e	output $g \wedge e$
0	0	0
0	1	0
1	0	0
1	1	1

$\Rightarrow$

garbled output
$\mathsf{Enc}(H(W_{\mathsf{G}}^0, W_{\mathsf{E}}^0), 0)$
$\mathsf{Enc}(H(W_{\mathsf{G}}^0, W_{\mathsf{E}}^1), 0)$
$\mathsf{Enc}(H(W_{\mathsf{G}}^1, W_{\mathsf{E}}^0), 0)$
$\mathsf{Enc}(H(W_{\mathsf{G}}^1, W_{\mathsf{E}}^1), 1)$

$\Rightarrow$

permuted garbled output
$\mathsf{Enc}(H(W_{\mathsf{G}}^0, W_{\mathsf{E}}^1), 0)$
$\mathsf{Enc}(H(W_{\mathsf{G}}^1, W_{\mathsf{E}}^1), 1)$
$\mathsf{Enc}(H(W_{\mathsf{G}}^0, W_{\mathsf{E}}^0), 0)$
$\mathsf{Enc}(H(W_{\mathsf{G}}^1, W_{\mathsf{E}}^0), 0)$

Fig. 1. The garbling of an AND gate

Garbled Gate Evaluation Once Evan receives the garbled gate, he needs to decrypt exactly one ciphertext: the one corresponding to the real values g and e, encrypted with $H(W_{\mathsf{G}}^g, W_{\mathsf{E}}^e)$. In order to do this, he needs to receive the values W_{G}^g and W_{E}^e from Ginny. Since Ginny knows g, she can send Evan W_{G}^g. The labels are all random, independent and identically distributed, so Evan won't learn anything about g from W_{G}^g. However, getting W_{E}^e to Evan is harder. Ginny

can't send both W_{E}^0 and W_{E}^1 to Evan, because that will allow Evan to decrypt *two* ciphertexts in the garbled gate. Similarly, Evan can't simply ask for the one he wants, because he doesn't want Ginny to learn e. So, Ginny and Evan use oblivious transfer (OT [EGL82]), which allows Evan to learn only W_{E}^e without revealing e to Ginny.

Note that in order for this to work, Evan needs to know when decryption succeeds, and when it doesn't. Otherwise, there's no way for him to know which ciphertext yields the correct answer. So, simply XOR-ing the key with the encrypted value will not work here.

From Gates to Circuits Determining whether to collaborate is a toy application of secure two-party computation. In reality, Ginny and Evan might each have multiple input bits, and they might want to compute a much more complicated function. For instance, they might want to evaluate the hamming distance of their input strings or the intersection size of their sets. In that case, instead of garbling a single gate, Ginny will garble the entire function circuit. For gates whose output serves as input to other gates, instead of encrypting the output bit, she will encrypt a label corresponding to the output bit: W_w^0 or W_w^1. That label will then be used to derive a key for the decryption of ciphertexts in other gates, etc.

1.2 Optimizations to Yao's Garbled Circuits

technique	size per gate		calls to H per gate			
			generator		evaluator	
	XOR	AND	XOR	AND	XOR	AND
classical [Yao86]	4	4	4	4	4	4
point-and-permute [BMR90]	4	4	4	4	1	1
free XOR [KS08]	0	4	0	4	0	1
GRR3 [NPS99] + free XOR	0	3	0	4	0	1
GRR2 [PSSW09]	2	2	4	4	1	1
fleXOR [KMR14]	$\{0,1,2\}$	2	$\{0,2,4\}$	4	$\{0,1,2\}$	1
half gates [ZRE15]	0	2	0	4	0	2
garbled gadgets [BMR16]	2	2	3	3	1	1

Fig. 2. Optimizations to Yao's Garbled Circuits: Efficiency for Two-Input Gates (table partially taken from Zahur *et al.* [ZRE15]). All techniques listed after point-and-permute are assumed to incorporate the point-and-permute technique.

We now summarize a few of the relevant garbled circuit optimizations.

Point and Permute The point-and-permute technique [BMR90] saves Evan from having to try decrypting all four ciphertexts. It associates each wire w with

technique	size per gate		calls to H per gate			
			generator		evaluator	
	XOR	AND	XOR	AND	XOR	AND
half gates [ZRE15]	0	$2(n-1)$	0	$4(n-1)$	0	$2(n-1)$
garbled gadgets [BMR16]	n	n	$n+1$	$n+1$	1	1

Fig. 3. Optimizations to Yao's Garbled Circuits: Efficiency for n-Input Gates. Since the half-gates techinique is strictly better than previous optimizations in most ways, we omit the previous optimizations from this table. We use half-gates to construct an n-input gate by combining $n-1$ two-input gates.

two *select bits* p^0 and p^1 in addition to labels W^0 and W^1. For $v \in \{0,1\}$, the select bit p^v is equal to $v \oplus \mathsf{r}$, where $\mathsf{r} \in \{0,1\}$ is a randomly chosen bit. So, the select bit p^v is different for the two possible underlying values v, but does not reveal anything about v. The select bit p of each wire is retrieved (via oblivious transfer if the wire is an input wire, or decryption otherwise) along with the wire label. When evaluating a gate, Evan uses the two select bits corresponding to the two input wires (say, p_i and p_j corresponding to wires w_i and w_j) to determine which ciphertext in gate k to decrypt. Using the point-and-permute optimization, Ginny always places $\mathsf{Enc}(H(W_i^{v_i}, W_j^{v_j}), W_k^{g_k(v_i,v_j)} || p_k^{g_k(v_i,v_j)})$ in the $(2p_i^{v_i}+p_j^{v_j})$th spot in the table.

This enables the use of simpler, more efficient encryption schemes such as the one-time pad. Because the select bit tells Evan exactly which ciphertext to decrypt, decryption with the correct key no longer needs to be distinguishable from decryption with an incorrect key, as it was before.

For the rest of this discussion, we ignore the select bits in the interest of notational simplicity.

Free XOR [KS08] The free-XOR technique [KS08] enables the computation of XOR gates for free, as the name suggests. It does so by fixing the relationship between labels W^0 and W^1. When garbling the circuit, Ginny picks a single random string $R \leftarrow \{0,1\}^L$. Ginny then picks each label W^0 at random, and sets $W^1 = W^0 \oplus R$. If gate g_k is an XOR gate and takes wires w_i and w_j as input, the new label for wire w_k can be computed simply by taking the XOR of labels W_i and W_j. The label W_k^0 is then $W_i^0 \oplus W_j^0$, and $W_k^1 = W_k^0 \oplus R$. This works because

$$W_i^0 \oplus W_j^0 = W_k^0,$$

$$W_i^0 \oplus W_j^1 = W_i^0 \oplus (W_j^0 \oplus R) = (W_i^0 \oplus W_j^0) \oplus R = W_k^0 \oplus R = W_k^1,$$

$$W_i^1 \oplus W_j^0 = (W_i^0 \oplus R) \oplus W_j^0 = (W_i^0 \oplus W_j^0) \oplus R = W_k^0 \oplus R = W_k^1,$$

and

$$W_i^1 \oplus W_j^1 = (W_i^0 \oplus R) \oplus (W_j^0 \oplus R) = W_i^0 \oplus W_j^0 = W_k^0.$$

Garbled Row Reduction (GRR3) [NPS99] Garbled row reduction allows the elimination of one ciphertext. This is accomplished by picking one label in such a way that the corresponding ciphertext is 0. (The eliminated ciphertext will always be the top one, as determined by the select bits.)

Garbled Row Reduction (GRR2) [PSSW09] This second form of garbled row reduction allows the elimination of two ciphertexts instead of one. However, while the first form of garbled row reduction is compatible with the free-XOR technique, this form (GRR2) is not. In GRR2, instead of recovering the output label by decrypting a one time pad encryption, the evaluator uses polynomial interpolation over a quadratic curve. The output label is encoded as the y-intercept. One point on the polynomial is revealed in the usual way — as $y = H(W_i^{v_i}, W_j^{v_j})$, with the select bits determining $x \in \{1, 2, 3, 4\}$. Two more (the ones at $x = 5$ and $x = 6$) are included in the garbled gate. So, the evaluator will have three points to use, which is enough to perform interpolation.

Since there are two possible output labels, there are two different quadratic polynomials to consider. They are designed to intersect exactly in the two points included in the garbled gate. In the case of an AND gate, $H(W_i^0, W_j^0), H(W_i^0, W_j^1)$ and $H(W_i^1, W_j^0)$ will uniquely determine the quadratic polynomial corresponding to the 0 output. So, that polynomial determines the points at $x = 5$ and $x = 6$, which together with $H(W_i^1, W_j^1)$ determine the polynomial corresponding to the 1 output. Figure 4 shows a graphical depiction of the values included, and of their relationship to the output labels.

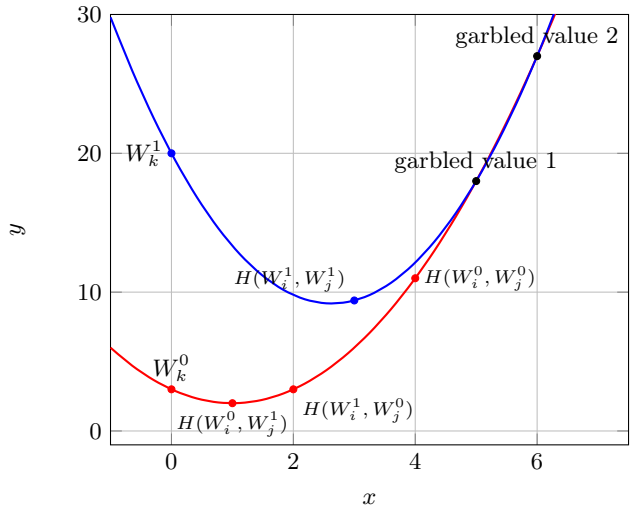

Fig. 4. GRR2 Garbled Gate Values for an AND Gate. Note that while the graph is shown over real numbers, GRR2 actually uses a finite field.

FleXOR [KMR14] Kolesnikov *et al.* enable the combination of the free-XOR technique with AND gate optimizations by translating wire labels to have a constant distance R on the fly. Depending on whether this translation is needed (that is, whether the inputs to the XOR gate in question are the outputs of AND gates or not), XOR garbled gates contain between 0 and 2 ciphertexts.

Half Gates [ZRE15] Zahur *et al.* introduce the first technique which only requires two ciphertexts per garbled AND gate *and* is compatible with the free-XOR optimization. They use the fact that $v_i \wedge v_j = (v_i \wedge (v_j \oplus b)) \oplus (v_i \wedge b)$ for any $b \in \{0,1\}$. In the half gates technique, b is determined to be the random value $\mathsf{r}_j \in \{0,1\}$ used to compute the select bit $p_j = v_j \oplus \mathsf{r}_j$. $b = \mathsf{r}_j$ is chosen by the garbler, and $v_j \oplus b = p_j$ is revealed to the evaluator. Using her knowledge of b, the garbler can efficiently garble the "garbler half gate" $v_i \wedge b$ using a single ciphertext. Using the fact that the evaluator knows $v_j \oplus b$, and can thus behave differently based on that value, the garbler can similarly efficiently garble the "evaluator half gate" $v_i \wedge (v_j \oplus b)$ using a single ciphertext. Taking the XOR of these two AND operations is free, so only two ciphertexts are required.

Garbled Gadgets [BMR16] Bal *et al.* extend the free XOR optimization from addition mod 2 to mod m. Let $\oplus_m$ represent the digit-wise addition operator modulo m. ($\oplus$ would then be equivalent to $\oplus_2$.) Let $\ominus_m$ be the inverse of $\oplus_m$. Because we have a higher modulus, we have m instead of 2 labels for every wire. Like Kolesnikov *et al.* [KS08], Bal *et al.* leverage a constant distance R between labels for every wire w_i. (This R will be a vector of values mod m instead of bits.) So, we will have $W^1 = W^0 \oplus_m R$, $W^2 = W^0 \oplus_m 2R$, $\ldots$, $W^{m-1} = W^0 \oplus_m (m-1)R$.

This allows us to get addition modulo m for free, in exactly the same way that free XOR allows us to get addition modulo 2 for free. Even if we are only interested in binary computations, this optimization allows us exploit the commutativity of modular addition to garble binary n-input gates using $n+1$ ciphertexts instead of 2^n ciphertexts. For any gate g_i and any vector of bits $[v_j \text{ for } j \in \mathsf{inindices}_i]$ (where $\mathsf{inindices}_i$ are the indices of the input wires to g_i) with hamming weight t,

$$M = \sum_{j \in \mathsf{inindices}_i} W_j^{v_j} = \Big(\sum_{j \in \mathsf{inindices}_i} W_j^0 \Big) + tR.$$

By applying the key derivation function H to the *modular sum* M of the gate input wire labels (instead of their concatenation), we can be oblivious to the order of the gate inputs. Figure 5 illustrates the garbling of a 6-input AND gate.

Bal *et al.* support switching moduli with ease using projection gates, which translate labels with digits modulo m_i to labels with digits modulo m_j using one ciphertext for each label.

Leveraging the commutativity of modular addition to decrease the number of ciphertexts necessary for an n-input garbled gate is the key idea behind the

t (hamming weight of the input)	output
0	0
1	0
2	0
3	0
4	0
5	0
6	1

$\Rightarrow$

garbled output
$\mathsf{Enc}(H(\sum_{j\in\mathsf{inindices}_i} W_j^0), 0)$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 R), 0)$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 2R), 0)$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 3R), 0)$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 4R), 0)$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 5R), 0)$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 6R), 1)$

$\Rightarrow$

permuted garbled output
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 4R), 0)$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 5R), 0)$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 6R), 1)$
$\mathsf{Enc}(H(\sum_{j\in\mathsf{inindices}_i} W_j^0), 0)$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 R), 0)$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 2R), 0)$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 3R), 0)$

Fig. 5. The garbling of an 6-input AND gate

t (hamming weight of the input)	output
0	k'
1	k'
2	k'
3	k'
4	k'
5	k
6	k

$\Rightarrow$

garbled output
$\mathsf{Enc}(H(\sum_{j\in\mathsf{inindices}_i} W_j^0), k')$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 R), k')$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 2R), k')$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 3R), k')$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 4R), k')$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 5R), k)$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 6R), k)$

$\Rightarrow$

permuted garbled output
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 4R), k')$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 5R), k)$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 6R), k)$
$\mathsf{Enc}(H(\sum_{j\in\mathsf{inindices}_i} W_j^0), k')$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 R), k')$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 2R), k')$
$\mathsf{Enc}(H((\sum_{j\in\mathsf{inindices}_i} W_j^0) \oplus_7 3R), k')$

Fig. 6. The garbling of an 6-input THRESHOLD gate, which returns k if the sum of the inputs is high enough, and k' otherwise.

garbled gadgets. They are also compatible with point and permute [BMR90] and with garbled row reduction [NPS99].

2 Yao's Garbled Circuit Definitions

In this section, we summarize the garbling scheme formalization of Bellare *et al.* [BHR12].

2.1 Functionality

A garbling scheme $\mathcal{G}$ consists of four polynomial-time algorithms $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$.

1. $\mathsf{Gb}(1^\lambda, f) \to (F, e, d)$
   The garbling algorithm Gb takes in the security parameter and a circuit f, and returns a garbled circuit F, encoding information e, and decoding information d.
2. $\mathsf{En}(e, x) \to X$
   The encoding algorithm En takes in the encoding information e and an input x, and returns a garbled input X.
3. $\mathsf{Ev}(F, X) \to Y$
   The evaluation algorithm Ev takes in the garbled circuit F and the garbled input X, and returns a garbled output Y.
4. $\mathsf{De}(d, Y) \to y$
   The decoding algorithm De takes in the decoding information d and the garbled output Y, and returns the plaintext output y.

A garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ is *projective* if e consists of a $2n$ wire labels, where n is the number of input bits. We denote those wire labels as $(X_i^0, X_i^1)_{i \in \mathsf{inindices}}$. $\mathsf{En}(e, x = (v_i)_{i \in \mathsf{inindices}})$ then returns $X = (X_i^{v_i})_{i \in \mathsf{inindices}}$.

Similarly, we call a garbling scheme *output-projective* if d consists of 2 labels for each output bit, one corresponding to each possible value of that bit. All of the garbling schemes discussed in this paper are projective and output-projective.

2.2 Correctness

Definition 1. *A garbling scheme* $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *is* correct *if for all sufficiently large security parameters* λ*, for all functions* f *and inputs* x*,*

$$\Pr[(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, f) : \mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, x))) = f(x)] = 1.$$

2.3 Security

Bellare et. al [BHR12] describe three security notions for garbling schemes: *obliviousness*, *privacy* and *authenticity*. Informally, a garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ is *oblivious* if a garbled function F and a garbled input X do not reveal anything

about the input x. It is *private* if additionally knowing the decoding information d reveals the output y, but does not reveal anything more about the input x. It is *authentic* if an adversary, given F and X, cannot find a garbled output $Y' \neq \mathsf{Ev}(F, X)$ which decodes without error.

Definitions 2 and 3 are the simulation-based definitions of obliviousness and privacy provided by Bellare *et al.* (Though they also give indistinguishability-based definitions, we do not restate those here.) We assume that the function f is public.

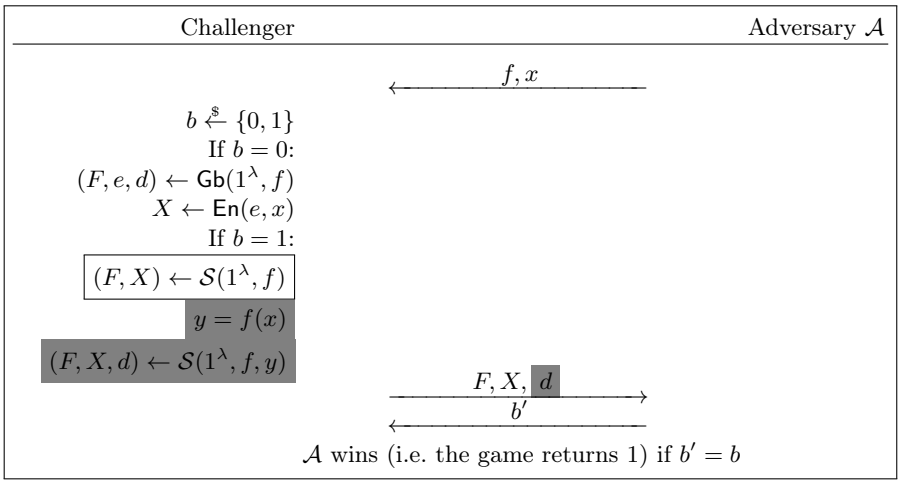

Fig. 7. The $\mathsf{OblivSim}^{\mathcal{A}}_{\mathcal{G},\mathcal{S}}(1^\lambda)$ and $\mathsf{PrivSim}^{\mathcal{A}}_{\mathcal{G},\mathcal{S}}(1^\lambda)$ Games. Steps that are present in $\mathsf{OblivSim}^{\mathcal{A}}_{\mathcal{G},\mathcal{S}}(1^\lambda)$ but not in $\mathsf{PrivSim}^{\mathcal{A}}_{\mathcal{G},\mathcal{S}}(1^\lambda)$ are in boxes, and steps that are present in $\mathsf{PrivSim}^{\mathcal{A}}_{\mathcal{G},\mathcal{S}}(1^\lambda)$ but not in $\mathsf{OblivSim}^{\mathcal{A}}_{\mathcal{G},\mathcal{S}}(1^\lambda)$ are highlighted.

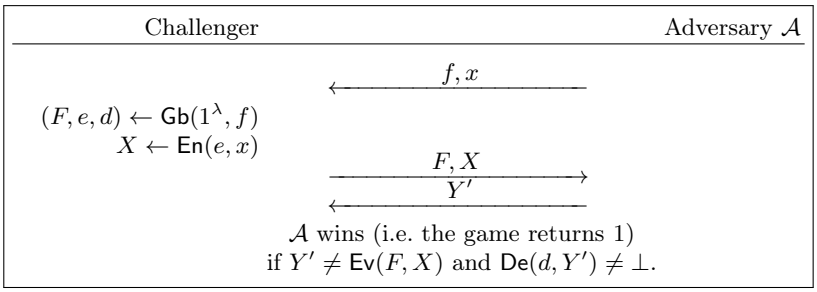

Fig. 8. The $\mathsf{Aut}^{\mathcal{A}}_{\mathcal{G}}(1^\lambda)$ Game.

Consider the obliviousness game $\mathsf{OblivSim}^{\mathcal{A}}_{\mathcal{G},\mathcal{S}}(1^\lambda)$ and the privacy game $\mathsf{PrivSim}^{\mathcal{A}}_{\mathcal{G},\mathcal{S}}(1^\lambda)$ described in Figure 7, each of which is parametrized by a security parameter λ, a garbling scheme $\mathcal{G}$, an adversary $\mathcal{A}$ and a simulator $\mathcal{S}$. The two games have many steps in common. Steps that are present in the obliviousness game but not in the privacy game are in boxes, and steps that are present in the privacy game but not in the obliviousness game are highlighted. $\mathsf{OblivSim}^{\mathcal{A}}_{\mathcal{G},\mathcal{S}}(1^\lambda) = 1$ (similarly, $\mathsf{PrivSim}^{\mathcal{A}}_{\mathcal{G},\mathcal{S}}(1^\lambda) = 1$) if the adversary wins (that is, $b = b'$), and $\mathsf{OblivSim}^{\mathcal{A}}_{\mathcal{G},\mathcal{S}}(1^\lambda) = 0$ (similarly, $\mathsf{PrivSim}^{\mathcal{A}}_{\mathcal{G},\mathcal{S}}(1^\lambda) = 0$) otherwise.

Define the adversary $\mathcal{A}$'s advantage in the the obliviousness game as

$$\mathsf{OblivAdv}_{\mathcal{G},\mathcal{S}}(1^\lambda, \mathcal{A}) = \left| \Pr[\mathsf{OblivSim}^{\mathcal{A}}_{\mathcal{G},\mathcal{S}}(1^\lambda) = 1] - \frac{1}{2} \right|.$$

Similarly, define the adversary $\mathcal{A}$'s advantage in the the privacy game as

$$\mathsf{PrivAdv}_{\mathcal{G},\mathcal{S}}(1^\lambda, \mathcal{A}) = \left| \Pr[\mathsf{PrivSim}^{\mathcal{A}}_{\mathcal{G},\mathcal{S}}(1^\lambda) = 1] - \frac{1}{2} \right|.$$

Definition 2. *A garbling scheme* $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *is* oblivious *if for all sufficiently large security parameters* λ*, there must exist a polynomial-time simulator* $\mathcal{S}$ *such that for any polynomial time adversary* $\mathcal{A}$*,*

$$\mathsf{OblivAdv}_{\mathcal{G},\mathcal{S}}(1^\lambda, \mathcal{A}) = \mathsf{negl}\,.$$

Definition 3. *A garbling scheme* $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *is* private *if for all sufficiently large security parameters* λ*, there must exist a polynomial-time simulator* $\mathcal{S}$ *such that for any polynomial time adversary* $\mathcal{A}$*,*

$$\mathsf{PrivAdv}_{\mathcal{G},\mathcal{S}}(1^\lambda, \mathcal{A}) = \mathsf{negl}\,.$$

Definition 4. *A garbling scheme* $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *is* authentic *if for all sufficiently large security parameters* λ*, for any polynomial time adversary* $\mathcal{A}$*,*

$$\Pr[\mathcal{A} \textit{ wins } \mathsf{Aut}^{\mathcal{A}}_{\mathcal{G}}(1^\lambda)] = \mathsf{negl}\,.$$

References

BHR12. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. Cryptology ePrint Archive, Report 2012/265, 2012. http://eprint.iacr.org/2012/265. (Page 9.)

BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990. (Pages 4 and 9.)

BMR16. Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for boolean and arithmetic circuits. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 565–577. ACM Press, October 2016. (Pages 2, 4, 5, and 7.)

EGL82. Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 205–210. Plenum Press, New York, USA, 1982. (Page 4.)

KMR14. Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 440–457. Springer, Heidelberg, August 2014. (Pages 2, 4, and 7.)

KS08. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008. (Pages 2, 4, 5, and 7.)

NPS99. Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, pages 129–139, New York, NY, USA, 1999. ACM. (Pages 2, 4, 6, and 9.)

PSSW09. Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Heidelberg, December 2009. (Pages 2, 4, and 6.)

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986. (Page 4.)

ZRE15. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015. (Pages 2, 4, 5, and 7.)