Improved Garbled Circuit: Free XOR Gates and Applications

Vladimir Kolesnikov[1] and Thomas Schneider[2⋆]

[1] Bell Laboratories, 600 Mountain Ave. Murray Hill, NJ 07974, USA
kolesnikov@research.bell-labs.com
[2] Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany
thomas.schneider@trust.rub.de

Abstract. We present a new garbled circuit construction for two-party secure function evaluation (SFE). In our one-round protocol, XOR gates are evaluated "for free", which results in the corresponding improvement over the best garbled circuit implementations (e.g. Fairplay [19]).
We build permutation networks [26] and Universal Circuits (UC) [25] almost exclusively of XOR gates; this results in a factor of up to 4 improvement (in both computation and communication) of their SFE. We also improve integer addition and equality testing by factor of up to 2.
We rely on the Random Oracle (RO) assumption. Our constructions are proven secure in the semi-honest model.

1 Introduction

Two-party general secure function evaluation (SFE) allows two parties to evaluate any function on their respective inputs x and y, while maintaining privacy of both x and y. SFE is (justifiably) a subject of immense amount of research, e.g. [27,28,17]. Efficient SFE algorithms enable a variety of electronic transactions, previously impossible due to mutual mistrust of participants. Examples include auctions [21,6,8,4], contract signing [7], distributed database mining [12,16], etc. As computation and communication resources have increased, SFE has become truly practical for common use. Fairplay [19] is a full-fledged implementation of generic two-party SFE with malicious players. It clearly demonstrates feasibility and efficiency of SFE of many useful functions, represented as circuits of up to $\approx 10^6$ gates. Today, generic SFE is a relatively mature technology, and even improvements by a small factor are non-trivial and are most welcome.

One area of SFE that especially benefits from our work is the SFE of *private functions* (PF-SFE). It is an extension of SFE where the evaluated function is known only by one party and needs to be kept secret (i.e. everything besides the size, the number of inputs and the number of outputs is hidden from the other party). Examples of real-life private functions include airport no-fly check function, credit evaluation function, background- and medical history checking

⋆ The work was done while the author was visiting Bell Laboratories.

function, etc. Full or even partial revelation of these functions opens vulnerabilities in the corresponding process, exploitable by dishonest participants (e.g. credit applicants), and should be prevented. It is known that the problem of PF-SFE can be reduced to the "regular" SFE [24,23]. This is done by evaluating a *Universal Circuit* (UC) [25,15] instead of a circuit defining the evaluated function. UC can be thought of as a "program execution circuit", capable of simulating any circuit C of certain size, given the description of C as input. Therefore, disclosing the UC does not reveal anything about C, except its size. At the same time, the SFE computes output correctly and C remains private, since the player holding C simply treats description of C as additional (private) input to SFE. This reduction is the most common (and often the most efficient) way of securely evaluating private functions [24,23,15].

1.1 Related work

General SFE has been a subject of immense amount of research, started by Yao [27,28], which resulted in significant advances in the field [9,21,17]. Fairplay [19] is a full practical implementation of general SFE based on garbled circuits.

Information-theoretic setting of SFE has also received a large amount of attention, e.g. [13,11]. However, due to the restrictions of the model, the resulting protocols are less efficient than those in the generous RO model. We apply some of the ideas of this setting, such as the efficient XOR gate construction (e.g. Construction 4 of [14]), in the RO setting, to obtain more efficient protocols.

1.2 Our contributions

We present a new garbled circuit construction for two-party secure function evaluation (SFE) in the semi-honest model. In our one-round protocol, XOR gates are evaluated "for free" (that is, without the use of the associated garbled tables and the corresponding hashing or symmetric key operations). Our construction is as efficient as the best garbled circuit implementations (e.g. Fairplay [19]) in handling other gates.

We next show that free XOR gates bring significant benefit to many SFE settings. We show how to build permutation networks [26] and UC [25,15] almost exclusively of XOR gates; this results in a factor of up to 4 improvement (in both computation and communication) of their SFE. As discussed above, SFE of UC is the most efficient way of evaluating private functions; thus our work improves performance of PF-SFE almost fourfold. We note that other useful functions can benefit from free XOR gates. We show how to obtain a factor of up to 2 improvement of SFE of integer addition and equality testing.

We rely on the RO assumption; we discuss its (conservative) use in Sect. 3.1.

2 Setting and Preliminaries

We consider *acyclic* boolean circuits with k gates and arbitrary fan-out. That is, the (single) output of each gate can be used as input to an arbitrary number of

gates. We assume that the gates $G_1, \ldots, G_k$ of the circuit are ordered topologically. This order (which is not necessarily unique) ensures that the i-th gate G_i has no inputs that are outputs of a successive gate G_j, where $j > i$. A topological order can always be obtained on acyclic circuits, with $O(k)$ computation.

We concentrate on the *semi-honest* model, where players follow the protocol, but try to learn information from the execution transcripts.

We use the following standard notation: $\in_R$ denotes uniform random sampling, $||$ denotes concatenation of bit strings. $\langle a, b\rangle$ is a vector with two components a and b, and its bit string representation is $a||b$. $W_c = g(W_a, W_b)$ denotes a 2-input gate G that computes function $g : \{0,1\}^2 \to \{0,1\}$ with input wires W_a and W_b and output wire W_c.

Let N be the security parameter. Let S be an infinite set and let $X = \{X_s\}_{s\in S}$ and $Y = \{Y_s\}_{s\in S}$ be distribution ensembles. We say that X and Y are computationally indistinguishable, denoted $X \stackrel{c}{\equiv} Y$, if for every non-uniform polynomial-time distinguisher D and all sufficiently large $s \in S$, $|Pr[D(X_s) = 1] - Pr[D(Y_s) = 1]| < 1/p(|s|)$ for every polynomial p.

Random Oracle. RO model is a useful abstraction, introduced and justified by [3]. RO is simply a randomly chosen function $\{0,1\}^* \mapsto \{0,1\}^N$ – a large object which cannot be fully stored or traversed by polytime players. RO model gives oracle access to such function to all players. In practice, ROs are modeled by hash functions, such as SHA. Although it was shown [5] that a protocol secure in the RO model may not be secure once RO is instantiated, "natural" RO protocols maintain their security in practice, and are widely used.

Oblivious Transfer (OT). The 1-out-of-2 OT is a two-party protocol. The *sender* P_1 has two secrets m_0, m_1, and the *receiver* P_2 has an selection bit $i \in \{0,1\}$. At the end of the protocol, P_2 learns m_i, but nothing about m_{1-i}, and P_1 learns nothing about i. One-round OT is a widely studied primitive in the standard model [2,1], with improved implementations in the RO model [20,3].

Yao's Garbled Circuit (GC). The GC approach, excellently presented in [17], is the most efficient method of SFE of boolean circuits. Here we summarize its idea. Player P_1 first *garbles* circuit C: for each wire W_i, he randomly chooses two secrets, w_i^0 and w_i^1, where w_i^j is a *garbled value*, or *garbling*, of the W_i's value j. (Note: w_i^j does not reveal j.) Further, for each gate G_i, P_1 creates and sends to P_2 a *garbled table* T_i, with the following property: given a set of garblings of G_i's inputs, T_i allows to recover the garbling of the corresponding G_i's output, and nothing else. Then garblings of players' inputs are (obliviously) transferred to P_2. Now, P_2 can obtain the garbled output simply by evaluating the garbled circuit gate by gate, using the tables T_i. We call W_i's garbling w_i^j *active* if W_i assumes the value j when C is evaluated on the given input. Observe that for each wire, P_2 can obtain only its active garbling. The output wires of the circuit are not garbled (or their garblings are published), thus P_2 learns (only) the output of the circuit, and no internal wire values. P_1 learns the output from (semi-honest) P_2. (This step is trivial in the semi-honest model, and is usually not considered in the analysis.) Correctness of GC follows from method of construction of tables T_i. Neither party learns any additional information from the protocol execution.

3 Our Protocol

Overview. In our construction, we combine GC with the simple information-theoretic SFE implementation of XOR-gates (e.g., Construction 4 of [14]). In all GC implementations, XOR gates cost as much as AND or OR gates (i.e. in computation and communication required for creation, transfer and evaluation of the garbled tables). The XOR gates of Kolesnikov [14] are free of these costs. However, his construction imposes a restrictive global relationship on the wire secrets, which prevents its use in previous GC schemes. In this work, we show how to overcome this restriction.

First, we show an SFE implementation of the XOR gate G, derived from one of [14]. Let G have two input wires W_a, W_b and output wire W_c. Garble the wire values as follows. Randomly choose $w_a^0, w_b^0, R \in_R \{0,1\}^N$. Set $w_c^0 = w_a^0 \oplus w_b^0$, and $\forall i \in \{a,b,c\} : w_i^1 = w_i^0 \oplus R$. It is easy to see that the garbled gate output is simply obtained by XORing garbled gate inputs:
$w_c^0 = w_a^0 \oplus w_b^0 = (w_a^0 \oplus R) \oplus (w_b^0 \oplus R) = w_a^1 \oplus w_b^1$
$w_c^1 = w_c^0 \oplus R = w_a^0 \oplus (w_b^0 \oplus R) = w_a^0 \oplus w_b^1 = (w_a^0 \oplus R) \oplus w_b^0 = w_a^1 \oplus w_b^0$.
Further, garblings w_i^j do not reveal the wire values they correspond to.

We can now pinpoint the restriction that the above XOR construction imposes on the garbled values – the garblings of the two values of each wire in the circuit must differ by the same value, i.e. $\forall i : w_i^1 = w_i^0 \oplus R$, for some global R. In contrast, in previous GC constructions, *all* garblings w_i^j were chosen independently at random, and proofs of security relied on that property.

Our main observation is that it is not necessary to select all garblings independently. In our construction (Sect. 3.1), we choose a random R once, and garble wire values, so that $\forall i : w_i^1 = w_i^0 \oplus R$.

3.1 Our Garbled Circuit Construction

Let C be a circuit. We first note that NOT gates can be implemented "for free" by simply eliminating them and inverting the correspondence of the wires' values and garblings. We thus do not further consider NOT gates.

We implement XOR gates as discussed above in Sect. 3. Further, we replace each XOR-gate with $n > 2$ inputs with $n-1$ two-input XOR-gates.

We implement all other gates using standard garbled tables [19]. Namely, each gate with n inputs is assigned a table with 2^n randomly permuted entries. Each entry is an encrypted garbling of the output wire, and garblings of the input wires serve as keys to decrypt the "right" output value. For simplicity, we present our construction and proof for the case $n = 2$. The generalization to n-input gates ($n \geq 1$) is straightforward.

In Alg. 1 below, each garbling $w = \langle k, p \rangle$ consists of a key $k \in \{0,1\}^N$ and a permutation bit $p \in \{0,1\}$. The key is used for decryption of the table entries, and p is used to select the entry for decryption. The two garblings w_i^0, w_i^1 of each wire W_i are related as required by the XOR construction: for a chosen $R \in_R \{0,1\}^N$, $\forall i : w_i^1 = \langle k_i^1, p_i^1 \rangle = \langle k_i^0 \oplus R, p_i^0 \oplus 1 \rangle$, where $w_i^0 = \langle k_i^0, p_i^0 \rangle$. $H : \{0,1\}^* \mapsto \{0,1\}^{N+1}$ is a RO.

We now formalize the above intuition and present the GC construction (Alg. 1) and evaluation (Alg. 2). In SFE, Alg. 1 is run by P_1 and Alg. 2 is run by P_2.

Algorithm 1 *(Construction of a garbled circuit)*

1. *Randomly choose global key offset* $R \in_R \{0,1\}^N$
2. *For each input wire* W_i *of* C
   (a) *Randomly choose its garbled value* $w_i^0 = \langle k_i^0, p_i^0 \rangle \in_R \{0,1\}^{N+1}$
   (b) *Set the other garbled output value* $w_i^1 = \langle k_i^1, p_i^1 \rangle = \langle k_i^0 \oplus R, p_i^0 \oplus 1 \rangle$
3. *For each gate* G_i *of* C *in topological order*
   (a) *label* $G(i)$ *with its index: label*$(G_i) = i$
   (b) *If* G_i *is an XOR-gate* $W_c = XOR(W_a, W_b)$ *with garbled input values* $w_a^0 = \langle k_a^0, p_a^0 \rangle, w_b^0 = \langle k_b^0, p_b^0 \rangle$, $w_a^1 = \langle k_a^1, p_a^1 \rangle, w_b^1 = \langle k_b^1, p_b^1 \rangle$*:*
      i. *Set garbled output value* $w_c^0 = \langle k_a^0 \oplus k_b^0, p_a \oplus p_b \rangle$
      ii. *Set garbled output value* $w_c^1 = \langle k_a^0 \oplus k_b^0 \oplus R, p_a \oplus p_b \oplus 1 \rangle$
   (c) *If* G_i *is a 2-input gate* $W_c = g_i(W_a, W_b)$ *with garbled input values* $w_a^0 = \langle k_a^0, p_a^0 \rangle, w_b^0 = \langle k_b^0, p_b^0 \rangle, w_a^1 = \langle k_a^1, p_a^1 \rangle, w_b^1 = \langle k_b^1, p_b^1 \rangle$*:*
      i. *Randomly choose garbled output value* $w_c^0 = \langle k_c^0, p_c^0 \rangle \in_R \{0,1\}^{N+1}$
      ii. *Set garbled output value* $w_c^1 = \langle k_c^1, p_c^1 \rangle = \langle k_c^0 \oplus R, p_c^0 \oplus 1 \rangle$
      iii. *Create* G_i*'s garbled table. For each of* 2^2 *possible combinations of* G_i*'s input values* $v_a, v_b \in \{0,1\}$*, set*

$$e_{v_a,v_b} = H(k_a^{v_a} || k_b^{v_b} || i) \oplus w_c^{g_i(v_a,v_b)}$$

      Sort entries e *in the table by the input pointers, i.e. place entry* e_{v_a,v_b} *in position* $\langle p_a^{v_a}, p_b^{v_b} \rangle$
4. *For each circuit-output wire* W_i *(the output of gate* G_j*) with garblings* $w_i^0 = \langle k_i^0, p_i^0 \rangle, w_i^1 = \langle k_i^1, p_i^1 \rangle$*:*
   (a) *Create garbled output table for both possible wire values* $v \in \{0,1\}$*. Set*

$$e_v = H(k_i^v || \text{“}out\text{”} || j) \oplus v$$

   Sort entries e *in the table by the input pointers, i.e. place entry* e_v *in position* p_i^v*. (There is no conflict, since* $p_i^1 = p_i^0 \oplus 1$*.)*

Note, our encryption of table entries (Step 3(c)iii) is similar to that of Fairplay [19, Section 4.2]. Fairplay uses $e_{v_a,v_b} = H(k_a^{v_a} || i || p_a^{v_a} || p_b^{v_b}) \oplus H(k_b^{v_b} || i || p_a^{v_a} || p_b^{v_b}) \oplus w_c^{g_i(v_a,v_b)}$. This is a non-essential difference; we could use Fairplay's encryption.

Intuition for security. (A formal proof is given in Sect. 3.2.) Alg. 1 uses the output of the RO H as a one-time pad to encrypt the garbled output values in the garbled tables (Step 3(c)iii) and the garbled output tables (Step 4a). Note, any specific combination of H's inputs (keys and gate indices) is used for encryption of at most one table entry throughout our construction. (We assume that concatenation and string representation inside H is done “right”.) Further, since the evaluator of the garbled circuit only knows one garbled value per wire, he can decrypt exactly one entry of G_i's garbled table. All other entries are

encrypted with at least one key that cannot be guessed by a polytime evaluator. Therefore, one of the two of garbled values of every wire looks random to him.

We now give the corresponding GC evaluation algorithm, run by P_2. Recall, P_2 obtains all garbled tables and the garblings of P_1's input values from P_1. Garblings of input values held by P_2 are sent via OT.

Algorithm 2 *(Evaluation of a garbled circuit):*

1. *For each input wire W_i of C*
   (a) *Receive corresponding garbled value $w_i = \langle k_i, p_i \rangle$*
2. *For each gate G_i (in the topological order given by labels)*
   (a) *If G_i is an XOR-gate $W_c = XOR(W_a, W_b)$ with garbled input values $w_a = \langle k_a, p_a \rangle, w_b = \langle k_b, p_b \rangle$*
      i. *Compute garbled output value $w_c = \langle k_c, p_c \rangle = \langle k_a \oplus k_b, p_a \oplus p_b \rangle$*
   (b) *If G_i is a 2-input gate $W_c = g_i(W_a, W_b)$ with garbled input values $w_a = \langle k_a, p_a \rangle, w_b = \langle k_b, p_b \rangle$*
      i. *Decrypt garbled output value from garbled table entry e in position $\langle p_a, p_b \rangle$: $w_c = \langle k_c, p_c \rangle = H(k_a || k_b || i) \oplus e$*
3. *For each C's output wire W_i (output of gate G_j) with garbling $w_i = \langle k_i, p_i \rangle$*
   (a) *Decrypt output value f_i from garbled output table entry e in row p_i : $f_i = H(k_i || \text{“}out\text{”} || j) \oplus e$*

The GC construction and evaluation algorithms can be directly used to obtain the GC-based SFE protocol, in a standard manner. For completeness, we include the description of this protocol.

Protocol 1 *(Two-party SFE protocol):*

- **Inputs***: P_1 has private input $x = \langle x_1, .., x_{u_1} \rangle \in \{0,1\}^{u_1}$ and P_2 has private input $y = \langle y_1, .., y_{u_2} \rangle \in \{0,1\}^{u_2}$.*
- **Auxiliary input***: A boolean acyclic circuit C such that $\forall x \in \{0,1\}^{u_1}, y \in \{0,1\}^{u_2}$, it holds that $C(x,y) = f(x,y)$, where $f : \{0,1\}^{u_1} \times \{0,1\}^{u_2} \rightarrow \{0,1\}^v$. We require that C is such that if a circuit-output wire leaves some gate G, then gate G has no other wires leading from it into other gates (i.e., no circuit-output wire is also a gate-input wire). Likewise, a circuit-input wire that is also a circuit-output wire enters no gates. We also require that C is modified to contain no NOT-gates and all n-input XOR-gates with $n > 2$ replaced by 2-input XOR-gates as described in Section 3.1.*
- **The protocol***:*
  1. *P_1 constructs the garbled circuit using Algorithm 1 and sends it (i.e. the garbled tables) to P_2.*
  2. *Let $W_1, .., W_{u_1}$ be the circuit input wires corresponding to x, and let $W_{u_1+1}, .., W_{u_1+u_2}$ be the circuit input wires corresponding to y. Then,*
     (a) *P_1 sends P_2 the garbled values $w_1^{x_1}, .., w_{u_1}^{x_{u_1}}$.*
     (b) *For every $i \in \{1, .., u_2\}$, P_1 and P_2 execute a 1-out-of-2 oblivious transfer protocol, where P_1's input is $(k^0_{u_1+i}, k^1_{u_1+i})$, and P_2's input is y_i. All u_2 OT instances can be run in parallel.*
  3. *P_2 now has the garbled tables and the garblings of circuit's input wires. P_2 evaluates the garbled circuit, as described in Alg. 2, and outputs $f(x,y)$.*

It is easy to verify protocol's correctness; we do not discuss it further.

On our use of RO. In previous GC work, RO's use improves efficiency in the malicious model, but is not inherent. Here, while we rely on RO, we do so conservatively. First, we use *non-programmable* RO [22], i.e. we don't allow simulator to fake RO's answers. Second, (a variant of) *correlation-robust* functions [10], a weaker notion than RO, is sufficient for our purposes. (Recall, if h is correlation-robust and $R, t_1, .., t_n$ are random, $(h(t_1 \oplus R), .., h(t_n \oplus R))$ is pseudo-random, given $t_1, .., t_n$.)

Further, concrete security of our construction is comparable to that of standard GC with RO as the encryption function. This makes even constant-factor efficiency improvements, such as those suggested in this work, meaningful. For the lack of space, we omit the detailed analysis. We only note that the main feature of our protocol, the use of the global R, has very slight impact on security (e.g., our adversary can decrypt all garbled tables, once he breaks any one of them and learns R). Further, our use of RO is not vulnerable to birthday attacks in the semi-honest model. Indeed, the circuit is small, and P_2 w.h.p. will not see RO collisions.

3.2 Proof of Security

Our protocol is secure against semi-honest adversaries, who are not allowed to deviate from the protocol. Analogously to [19,18], (w.h.p.) malicious behavior of players can be prevented by using cut-and-choose method; we don't discuss malicious players further.

We prove security in the simulation paradigm. Intuitively, a protocol π is secure if whatever is seen by its party, can be computed only from that party's input and output. The view of a party P_i, $view^{\pi}_{P_i}(x, y)$, consists of the party's own input, randomness, and all messages that P_i receives in the execution of π. Thus, a protocol is secure, if there exist *simulators* S_1, S_2, such that $\{S_1(x, f(x, y)\} \stackrel{c}{\equiv} \{view^{\pi}_{P_1}(x, y)\}$ and $\{S_2(y, f(x, y)\} \stackrel{c}{\equiv} \{view^{\pi}_{P_2}(x, y)\}$.

Case 1 - $\mathbf{P_1}$ is corrupted. P_1's view in Protocol 1 consists only of the view in the OT protocols in Step 2b. The following $S_1(x, f(x, y))$ simulates the view of P_1. Let S_1^{OT} be the simulator that is guaranteed to exist for P_1 in the secure 1-out-of-2 OT protocol. S_1 constructs a garbled circuit using Alg. 1. Then S_1 feeds the constructed garblings of the input wires corresponding to y to S_1^{OT}, and obtains the simulated transcript of the OT, which he outputs. S additionally outputs x and the randomness used in construction of GC. It is not hard to see that the output of the simulator is indistinguishable from the view of P_1.

Case 2 - $\mathbf{P_2}$ is corrupted. We construct a simulator S_2 that given input $(y, f(x, y))$ simulates the view of P_2. P_2 receives a garbled circuit (including garbled inputs), which S_2 must simulate. However, S_2 doesn't know P_1's input x. Thus, S_2 can not honestly generate the garbled circuit, since it doesn't know which of the input garblings corresponding to x to hand to P_2 in Step 2a of the protocol. Instead, S_2 generates a fake garbled circuit that always evaluates to $f(x, y)$, using a slightly modified Alg. 1. The only modification, in Step 4a, appropriately forges the output tables:

4. For each circuit-output wire W_i (the output of gate G_j) with garblings $w_i^0 = \langle k_i^0, p_i^0 \rangle, w_i^1 = \langle k_i^1, p_i^1 \rangle$:
   (a) Create **fake** garbled output table for both possible wire values $v \in \{0,1\}$ **of the same encrypted output value**. Set

$$e_v = H(k_i^v || \text{“}out\text{”} || j) \oplus \mathbf{f_i}(\mathbf{x}, \mathbf{y})$$

   Sort entries e in the table by the input pointers, i.e. place entry e_v in position p_i^v.

Let S_2^{OT} be an OT simulator for P_2. S_2 outputs y, and the fake garbled circuit (i.e. its tables). Further, for each input wire W_i held by P_2, S_2 runs and outputs $S_2^{OT}(y_i, w_i^{y_i})$. Finally, S_2 simulates the received garblings of the input wires W_j held by P_1 simply by outputting w_j^0 (fake garblings corresponding to $x = 0..0$).

Theorem 1. *The output of S_2 is indistinguishable from the real view of P_2.*

Proof. (sketch) First, observe that S_2 feeds S_2^{OT} proper inputs (i.e. y and the corresponding honestly generated garblings). Thus, simulation of Step 2b of the protocol is indistinguishable from the real execution. The crux of the proof is in showing the indistinguishability of the fake and real circuits (which include the tables and the input garblings that P_2 sees). This is addressed next.

First, observe, pointers p_i^j are independent of the parties' inputs, and thus are easily simulated by S_2. For ease of presentation, we omit the details of pointer simulation from the proof.

We now show that no polytime procedure D can distinguish simulated and real garbled circuit transcripts with non-negligible probability. We proceed inductively, gate by gate in topological order, in proving this for each partial transcript τ_i, where τ_0 includes all active secrets on the input wires, and each τ_i additionally includes the garbled tables of first i gates.

Induction base. It is easy to see that the partial transcript τ_0 – active secrets on the input wires – is distributed identically in real and simulated cases. Indeed, these secrets are uniformly random in the domain. Moreover, clearly, no distinguisher D_0 can output with non-negligible probability the global key offset $\hat{R}$ used in the construction of the (either simulated or real) transcript.

For the induction step, suppose no polytime D_{i-1} can with non-negligible advantage distinguish the τ_{i-1} transcripts (i.e. those including the active secrets on the inputs and the first $i-1$ garbled tables). Moreover, assume that no polytime D_{i-1} can output the global key offset $\hat{R}$ with non-negligible probability when given τ_{i-1}. We show that these properties hold also when additionally given the i-th garbled table.

Recall, the i-th garbled table contains (a permutation of) entries:

$H(k_a || k_b || i) \oplus v_{00}$
$H(k_a || k_b \oplus \hat{R} || i) \oplus v_{01}$
$H(k_a \oplus \hat{R} || k_b || i) \oplus v_{10}$
$H(k_a \oplus \hat{R} || k_b \oplus \hat{R} || i) \oplus v_{11}$

where $v_{00}, .., v_{11} \in \{k_c, k_c \oplus \hat{R}\}$ are the output secrets that correspond to the

four possible gate input combinations. (Garbled output tables have one input and consist of two entries. The corresponding claims hold for these cases as well, via a natural modification of the following argument addressing two-input gates.)

Without loss of generality, suppose the active gate input secrets are k_a and k_b. By the induction assumption, no polytime D_{i-1} can compute both k_a and $k_a \oplus \hat{R}$, or both k_b and $k_b \oplus \hat{R}$ (otherwise D_{i-1} can output $\hat{R}$). Thus, D_{i-1} can call functions $H(k_a||k_b \oplus \hat{R}||i), H(k_a \oplus \hat{R}||k_b||i)$, or $H(k_a \oplus \hat{R}||k_b \oplus \hat{R}||i)$ only with negligible probability. Further, because of the inclusion of the gate index i, these function calls have not been made in the construction of (real or simulated) τ_i. Therefore, due to RO properties, except with negligible probability, all the inactive entries in the i-th table are distributed identically to random strings, from the point of view of D_{i-1}, and thus do not provide help to D_{i-1} in computing $\hat{R}$. Therefore, polytime D_i cannot output $\hat{R}$ or call any of $H(k_a||k_b \oplus \hat{R}||i), H(k_a \oplus \hat{R}||k_b||i)$, or $H(k_a \oplus \hat{R}||k_b \oplus \hat{R}||i)$, except with negligible probability. Therefore, no polytime D_i can distinguish the real and simulated transcripts τ_i with non-negligible probability.

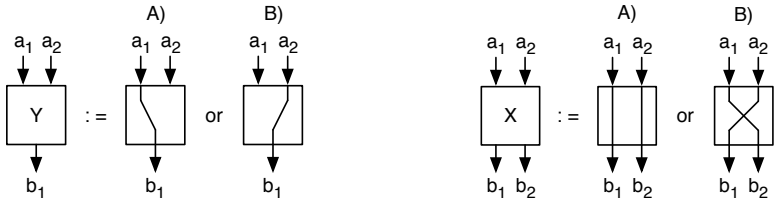
This completes the induction and the proof of the theorem. □

4 Application of our SFE constructions

We now present several motivating examples – practical functions whose SFE benefits from improvements of our construction. Universal circuit (UC) constructions [25,15] do not explicitly use many XOR gates. We show how to modify these circuits to mainly consists of XOR gates, achieving fourfold reduction of garbled circuit size. This construction may be of independent interest. Further, we show how to reduce in half the size of garbled circuits of commonly used blocks, such as integer addition and equality test.

Universal Circuits [25,15] and Permutation Networks [26]. The size of a UC mainly comes from programmable switching networks (such as permutation network [26]) connecting the simulated gates. In turn, these networks are constructed from two types of switching blocks shown in Fig. 1, as discussed in [26,25,15]. The *Y-block* can be programmed to output one of its two inputs. The *X-block* can be programmed to either pass or cross over its two inputs to the two outputs. A natural SFE implementation of the Y-block uses a 2-input garbled gate with a garbled table with $2^2 = 4$ encrypted table entries. Similarly, X-block is implemented by two 2-input garbled gates (one for each of its two outputs), resulting in a garbled table of $2 \cdot 2^2 = 8$ entries.

We show how to take advantage of free XOR gates and implement both X- and Y-gates with only two garbled table entries each. Since permutation network [26] consists only of X-gates, this results in 75% size reduction of its SFE. UC consists *almost exclusively* of X-gates. Valiant's UC [25] for a circuit of k gates has size $\sim 19k \log k$. The $\sim 19k \log k - k$ overhead gates are X-gates that come from switching networks. A recent UC construction [15] similarly consists almost exclusively of X-gates, and of very few Y-gates and simulated gates. Thus, UC enjoys almost 75% garbled table size reduction.

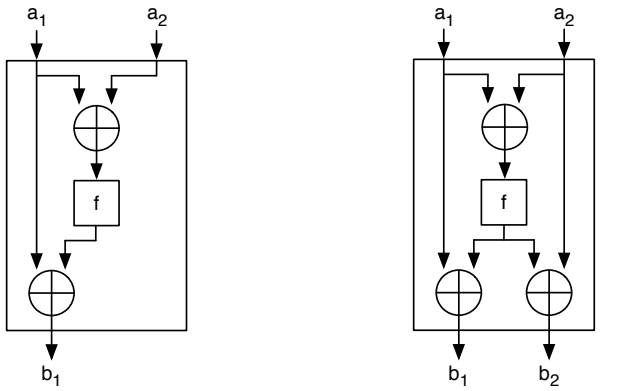


(a) Y switching block (b) X switching block

Fig. 1. Switching blocks

Let $f : \{0,1\} \mapsto \{0,1\}$ be a function (implemented with two garbled table entries). We implement X- and Y-blocks as follows (see Fig. 2). $Y(a_1, a_2) = b_1 = f(a_1 \oplus a_2) \oplus a_1$; $X(a_1, a_2) = (b_1, b_2)$, where $b_1 = f(a_1 \oplus a_2) \oplus a_1, b_2 = f(a_1 \oplus a_2) \oplus a_2$. It is easy to see that setting $f = f_0$ to the zero function results in Y choosing left input, and X passing the inputs. Further, setting $f = f_{id}$ to the identity function results in Y choosing the right input, and in X crossing its inputs:

$f = f_0$: $b_1 = 0 \oplus a_1 = a_1$; $b_2 = 0 \oplus a_2 = a_2$.
$f = f_{id}$: $b_1 = (a_1 \oplus a_2) \oplus a_1 = a_2$; $b_2 = (a_1 \oplus a_2) \oplus a_2 = a_1$.



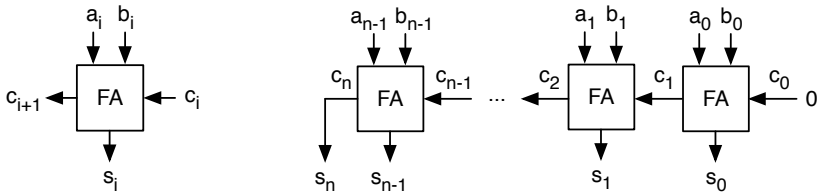
(a) Y switching block (b) X switching block

Fig. 2. Efficient implementation of switching blocks

This construction can be extended to implement programmable switching blocks X and Y, which take an additional programming input bit p. This bit determines behavior of X- (pass or cross) and Y-blocks (left or right input). The natural construction for the Y- (resp. X-) switching block uses one (resp. two) 3-input gate(s) with $2^3 = 8$ (resp. 16) encrypted table entries. In our XOR-based construction, function f is then replaced by a two-input AND-gate (with p being the second input) with $2^2 = 4$ encrypted table entries. Clearly, $p = 0$ sets $f = f_0$, and $p = 1$ sets $f = f_{id}$, allowing to program X- and Y-blocks. As above, the size of Y- and X-blocks is reduced by 50% and 75% respectively.

Integer Adder and Multiplier. An adder for n-bit integers a, b is composed from a chain of n full adder (FA) blocks as shown in Fig. 3(b). (The last

FA block can be replaced by a smaller half-adder block.) A FA block (see Fig. 3(a)) has as inputs a carry-in c_i from the previous FA block and the two input bits a_i and b_i. It outputs two bits: carry-out $c_{i+1} = (a_i \wedge b_i) \vee (a_i \wedge c_i) \vee (b_i \wedge c_i)$ and sum $s_i = a_i \oplus b_i \oplus c_i$. The straightforward implementation of a FA uses two 3-input gates with $2 \cdot 2^3 = 16$ encrypted table entries. We can compute s_i "for free" using free XOR-gates, and use only one 3-input gate with $2^3 = 8$ encrypted table entries to compute c_{i+1}. The size of a FA block, and hence that of an n-bit adder is reduced by 50%.



(a) Full Adder (FA) (b) n-bit Adder built from n FA blocks

Fig. 3. Adder for two n-bit integers a and b

As circuits for integer multiplication consist of bit-multipliers (2-input AND-gates) and adders, the improved implementation of adders can directly be used to correspondingly improve integer-multiplication circuits.

Integer Equality Test. A similar construction is used to test equality of two n-bit integers a and b. Now, we do not compute s_i, and use carry bits as inequality flags. The carry-out bit is defined as $c_{i+1} = (a_i \neq b_i) \vee c_i = (a_i \oplus b_i) \vee c_i$. A simple implementation uses two 2-input gates or one 3-input gate (each costs 8 encrypted table entries). Free XOR gate reduces the cost to that of one 2-input OR gate (4 encrypted table entries). The size of equality test block is thus reduced by 50%.

Acknowledgments: We thank Yuval Ishai and anonymous referees for helpful comments. The second author thanks CACE project for funding his ICALP trip.

References

1. William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT '01*, LNCS, pages 119–135. Springer, 2001.
2. Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In *CRYPTO '89*, pages 547–557. Springer-Verlag, 1989.
3. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS*, pages 62–73, 1993.
4. Ian F. Blake and Vladimir Kolesnikov. Conditional encrypted mapping and comparing encrypted numbers. In *Financial Cryptography and Data Security, FC 2006*, volume 4107 of *LNCS*, pages 206–220. Springer, 2006.
5. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. In *Proc. 30th ACM Symp. on Theory of Computing*, pages 209–218, 1998.
6. Giovanni Di Crescenzo. Private selective payment protocols. In *Financial Cryptography and Data Security, FC 2000*, volume 1962 of *LNCS*. Springer, 2000.

7. Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.
8. Marc Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In *RSA Security 2001 Cryptographer's Track*, volume 2020 of *LNCS*, pages 457–471. Springer-Verlag, 2001.
9. Oded Goldreich and Ronen Vainish. How to solve any protocol problem - an efficiency improvement. In *Advances in Cryptology – CRYPTO 87*, volume 293 of *LNCS*, pages 73–86, London, UK, 1988. Springer.
10. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO 2003*, volume 2729 of *LNCS*. Springer, 2003.
11. Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP '02*, pages 244–256, 2002.
12. Murat Kantarcioglu and Chris Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'02)*, 2002.
13. Joe Kilian. Founding cryptography on oblivious transfer. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 20–31, Chicago, 1988. ACM.
14. Vladimir Kolesnikov. Gate evaluation secret sharing and secure one-round two-party computation. In *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 136–155. Springer, 2005.
15. Vladimir Kolesnikov and Thomas Schneider. A practical universal circuit construction and secure evaluation of private functions. In *Financial Cryptography and Data Security, FC 2008*, LNCS. Springer, 2008.
16. Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *LNCS*. Springer-Verlag, 2000.
17. Yehuda Lindell and Benny Pinkas. A proof of Yao's protocol for secure two-party computation. Cryptology ePrint Archive, Report 2004/175, 2004.
18. Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology – EUROCRYPT 2007*, volume 4515, pages 52–78, 2007.
19. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *USENIX*, 2004.
20. Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.
21. Moni Naor, Benny Pinkas, and Reuben Sumner. Privacy preserving auctions and mechanism design. In *1st ACM Conf. on Electronic Commerce*, 1999.
22. Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *Advances in Cryptology – CRYPTO 2002*, pages 111–126, London, UK, 2002. Springer-Verlag.
23. Benny Pinkas. Cryptographic techniques for privacy-preserving data mining. *SIGKDD Explor. Newsl.*, 4(2):12–19, 2002.
24. Tomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for NC^1. In *Proc. 40th FOCS*, pages 554–566, New York, 1999. IEEE.
25. Leslie G. Valiant. Universal circuits (preliminary report). In *Proc. 8th ACM Symp. on Theory of Computing*, pages 196–203, New York, NY, USA, 1976. ACM Press.
26. Abraham Waksman. A permutation network. *J. ACM*, 15(1):159–163, 1968.
27. Andrew C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 160–164, Chicago, 1982. IEEE.
28. Andrew C. Yao. How to generate and exchange secrets. In *Proc. 27th IEEE Symp. on Foundations of Comp. Science*, pages 162–167, Toronto, 1986. IEEE.