

# Garbling Scheme for Formulas with Constant Size of Garbled Gates

Carmen Kempka, Ryo Kikuchi, Susumu Kiyoshima, and Koutarou Suzuki

NTT Secure Platform Laboratories, Tokyo, Japan

{kempka.carmen,kikuchi.ryo,kiyoshima.susumu,suzuki.koutarou}@lab.ntt.co.jp

**Abstract.** We provide a garbling scheme which creates garbled circuits of a very small constant size (four bits per gate) for circuits with fan-out one (formulas). For arbitrary fan-out, we additionally need only two ciphertexts per additional connection of each gate output wire. We make use of a trapdoor permutation for which we define a generalized notion of correlation robustness. We show that our notion is implied by PRIV-security, a notion for deterministic (searchable) encryption. We prove our scheme secure in the programmable random oracle model.

**Keywords:** garbled circuits, constant size of garbled gates, correlation robustness, PRIV-security

## 1 Introduction

Yao’s *garbled circuit* technique [33] is one of the most important techniques on secure computation. Very roughly speaking, this technique allows a party (the *garbler*) to create an encrypted form of a circuit—a “garbled” circuit—and an encoding of input with which the other party (the *evaluator*) can evaluate the circuit on the input but cannot compute anything other than the output. Compared with other techniques on secure computation (e.g., the technique by Goldreich et al. [12]), the garbled circuit technique has a big advantage on efficiency since we can construct *constant-round* protocols by using it.

Traditionally, the garbled circuit technique was considered to be a theoretical feasibility result; however, recently many works have demonstrated that the garbled circuit technique can also be used to construct two-party computation protocols with *practical efficiency*. The first implementation of the garbled circuit technique was shown by Malkhi et al. [28]. Since then, significant efforts have been devoted toward making the technique more practical.

A major line of research on the garbled circuit technique is the reduction of the size of garbled circuits. Since the main efficiency bottleneck of garbled-circuit-based two-party computation protocols is usually network bandwidth,

---

© IACR 2015. This article appeared in the proceedings of Asiacrypt 2015, published by Springer-Verlag, and is available at [http://dx.doi.org/10.1007/978-3-662-48797-6\\_31](http://dx.doi.org/10.1007/978-3-662-48797-6_31).

reducing the size of garbled circuits typically leads to a big improvement of efficiency in practice.\*

**Reduction of Garbled Circuit Size.** Originally, the garbled circuit technique uses four ciphertexts for each gate to create “garbled truth tables”, and thus, the size of a garbled circuit is  $O(k)$  bits per gate (where  $k$  is the security parameter). In [22], Kolesnikov and Schneider proposed a technique, called *free-XOR technique*, with which we can construct a garbled circuit that contains no ciphertexts for XOR gates. In [29], Naor et al. proposed a technique that reduces the number of ciphertexts from four to three for each gate. In [30], Pinkas et al. proposed a technique that reduces the number of ciphertexts to two for each gate. Recently, Kolesnikov et al. [21] introduced the *flexOR technique*, which requires zero, one, or two ciphertexts to garble an XOR gate—thus, garbling XOR gates is not “free” in general—but is compatible with the garbled row-reduction technique of [30]. Very recently, Zahur et al. [34] introduced a technique that is compatible with the free-XOR technique and can garble each AND gate by using only two ciphertexts; thus, this technique requires two ciphertexts for each AND gate and no ciphertexts for each XOR gate. We remark that although all of these techniques do not achieve an asymptotic reduction of the size of garbled circuits—it remains to be  $O(k)$  bits per gates—they offer a significant reduction of communication cost in practice.

A different approach for reducing the size of garbled circuits is the technique of Kolesnikov [19], which is an information-theoretic variant of Yao’s garbled circuit technique. In this technique, a circuit is garbled by using secret sharing (instead of encryption), and a garbled circuit is evaluated by recovering a share assigned to the output wire of each gate from shares assigned to the input wires of that gate. The size of the garbled circuit is zero (since there is no garbled truth table) and the size of the encoded input grows with the depth of the circuit; specifically, the size of the shares is quadratic in the depth of the gate for formulas, and exponential for circuits. For shallow circuits, the technique of Kolesnikov [19] is more efficient than other techniques.

**Our Contribution.** In this paper, we propose a garbling technique for formulas (i.e., circuits with fan-out 1) such that the size of the garbled circuit is *four bits* per gate. Unlike the optimization techniques of [22, 29, 21, 34], our technique achieves asymptotic reduction of the size of the garbled circuits. Also, unlike the information-theoretic garbled circuit technique of [19], our technique encodes input in such a way that the size of the encoded input is independent of the depth of the circuit. (For detailed comparisons, see Section 3.3.)

In our technique, ciphertexts include trapdoor permutations (instead of hash functions as in most of the previous techniques). To prove the security, we extend the definition of correlation robustness (which is originally defined for hash

---

\* In the case of security against malicious adversaries, the number of circuits that are generated in the cut-and-choose technique also has an impact on efficiency [26, 1, 23, 14, 10, 15, 27].

functions [11]) to the case of trapdoor permutations, and assume that the underlying trapdoor permutation is correlation robust. We also show that our notion of correlation robustness is implied by PRIV-security as defined in [7].

**Idea of Our Garbling Scheme.** The idea of our construction is as follows: Unlike most existing techniques, our construction garbles circuits *backwards*, starting from the output gate. This allows us to reduce communication cost drastically: We can compute the ciphertexts needed for each gate as a hash of the gate-ID, so the evaluator can re-compute them by himself and they need not be included in the garbled circuit. These ciphertexts are then interpreted by the garbler as the XOR of the output key  $K_i$  and an image of a trapdoor one-way permutation of a function of the input keys, which he inverts to compute appropriate input keys corresponding to a given output key. Altogether we will have a system of four equations of the form

$$\begin{aligned} c_0 &= E(f_0(K, L)) \oplus K_C^0 \\ c_1 &= E(f_1(K, L')) \oplus K_C^0 \\ c_2 &= E(f_2(K', L)) \oplus K_C^0 \\ c_3 &= E(f_3(K', L')) \oplus K_C^1 \end{aligned}$$

in permuted order, where  $K, K', L, L'$  are the input keys,  $K_C^0$  and  $K_C^1$  the output keys, and  $f_i$  are linear functions. The garbler can solve this system of equations to compute the input keys by inverting the trapdoor permutation  $E$ . The evaluator can only go through the circuit *forward*, using the one-way permutation to obtain output keys corresponding to his input. However, lacking the trapdoor, he cannot go backwards to compute any of the other keys.

One caveat of our backwards garbling technique is that the input keys for each gate are uniquely determined given the output keys and the ciphertexts assigned to this gate. Thus, we have no freedom in choosing any keys but the circuit output keys. Therefore, our garbling scheme only allows fan-out one, i. e., formulas. Moreover, to communicate wire choice bits, we cannot use the usual technique of defining the least significant bit of the keys as choice bit, since we have no freedom in choosing the input keys or their LSBs. Therefore, we use a hash function  $H'$  with one bit output, and publish  $H'(K_a, K_b) \oplus l_i$  for input keys  $K_a, K_b$  and corresponding choice bit  $l_i$  of the output key, giving a garbled circuit with  $l$  gates an overall size of  $4l$  bits, plus the number of bits needed to communicate the key of a keyed hash function and the index of a trapdoor one-way permutation.

Somewhat surprisingly, we can use the free-XOR technique to garble XOR gates at no additional cost, by using a “local” difference per XOR-sub-tree rather than a global difference. In the case of formulas, this will only save us the 4 bit per gate for the choice bits. However in the case of general circuits, the freedom an XOR-gate gives us in choosing input keys can save us additional ciphertexts needed for dealing with arbitrary fan-out in some cases.

Since our basic construction only allows fan-out one, a problem occurs when we garble circuits which use the same input variable multiple times, such as

$(a \wedge b) \vee (a \wedge c)$ . In such cases, we can duplicate the input wire for this variable, and assign a different input key pair to each occurrence of the variable. In the semi-honest setting, this does not affect security. In the malicious case, additional care needs to be taken to ensure that the garbler provides the same input for each occurrence of a variable. We discuss this in Section 5.3.

**Related Works.** The garbled circuit technique was introduced in the seminal paper of Yao [33]. A formal analysis of the garbled circuit technique (or, more precisely, the two-party computation protocol based on it) was presented by Lindell and Pinkas [25]. Bellare et al. [8] introduced an abstraction of the garbled circuit technique, which they call *garbling schemes*.

There are a lot of works that studied the size of garbled circuits. Other than the works we mentioned above [22, 29, 30, 21, 34], Choi et al. [11] and Applebaum [2] studied what assumptions are needed by the free-XOR technique. Choi et al. showed that a circular security assumption on the underlying hash function is sufficient. Applebaum showed that the LPN assumption is sufficient. Also, Boneh et al. [9] showed that an asymptotic reduction of garbled circuit size is possible under the learning-with-errors (LWE) assumption.

Other than the technique of [19], there are several information-theoretic variants of the garbled circuit technique, e.g., [18, 32, 17, 20].

The correlation security of trapdoor permutations has also been studied by previous work in other contexts. For example, Rosen and Segev [31] introduced *correlated product security* of trapdoor permutations and used it to construct a CCA-secure encryption scheme. Also, Hemenway et al. [13] studied the relation between the decisional variant of correlated product security and the security of deterministic encryption schemes. We remark however that these notions of correlation security are different from the one we consider in this work. Roughly speaking, in correlated product security [31], correlated inputs are applied to  $k$  functions  $f_1, \dots, f_k$  that are independently chosen from a family of functions, whereas in our notion of correlation robustness, correlated inputs are applied to a *single* function  $f$ .

The size of inputs of garbled circuits has been studied in the context of randomized encoding [16, 3, 4]. Applebaum et al. [5] proposed a garbling scheme with constant online rate, i.e., they improve the online communication complexity for input keys from  $nk$  to  $n + k$ , where  $n$  is the number of inputs and  $k$  is security parameter. In contrast, our proposed scheme improves, only for formulas, communication complexity for garbled circuits from  $lk$  to  $l + k$  and has communication complexity  $nk$  for input keys, where  $l$  is the number of gates,  $n$  is the number of inputs, and  $k$  a security parameter. We can combine the scheme of [5] and our proposed scheme to realize a randomized encoding for formulas with online communication complexity  $n + k$  for input keys and communication complexity  $l + k$  for garbled circuit.

**Outline of This Work.** The rest of this paper is organized as follows. We explain preliminaries and notation in Section 2, where we also recap the formal

definition of garbling schemes, and introduce our notion of correlation robustness. We describe our basic garbling scheme for formulas in the semi-honest setting in Section 3, and prove its security in Section 4. In Section 5, we discuss possible extensions like arbitrary fan-out, incorporation of the free-XOR technique as well as extending our construction to the case of active adversaries. We discuss the instantiation of our correlation robust trapdoor one-way permutation with a PRIV-secure deterministic encryption scheme in Section 6.

## 2 Preliminaries

### 2.1 Notation

We use the following notations. By  $x \stackrel{U}{\leftarrow} X$ , we denote that  $x$  is randomly selected from set  $X$  according to the uniform distribution. By  $x \leftarrow \text{Algo}$ , we denote that probabilistic algorithm  $\text{Algo}$  outputs  $x$ . By  $A := B$ , we denote that  $A$  is defined by  $B$ . By  $[S]_x$ , we denote the  $x$ -th bit of bitstring  $S$ .

### 2.2 Garbling Scheme

In this section, we recall the definition of a garbling scheme and the notion of *simulation-based privacy* of Bellare et al. [8].

A circuit is described as  $f = (n, m, l, A, B, G)$ . Here,  $n \geq 2$  is the number of circuit input wires,  $m \geq 1$  is the number of circuit output wires, and  $l \geq 1$  is the number of gates (and their output wires). Let  $W = \{1, \dots, n+l\}$  be the set of all wires,  $W_{input} = \{1, \dots, n\}$  the set of circuit input wires,  $W_{output} = \{n+l-m+1, \dots, n+l\}$  the set of circuit output wires, and  $W_{gate} = \{n+1, \dots, n+l\}$  the set of gates (and their output wires).  $A : W_{gate} \rightarrow W \setminus W_{output}$  is a function to specify the first input wire  $A(i)$  of each gate  $i$ .  $B : W_{gate} \rightarrow W \setminus W_{output}$  is a function to specify the second input wire  $B(i)$  of each gate  $i$ . We require  $A(i) < B(i) < i$  for all  $i \in W_{gate}$ .  $G : W_{gate} \times \{0, 1\}^2 \rightarrow \{0, 1\}$  is a function to specify the gate function  $G(i, \cdot, \cdot)$  of each gate  $i$ . We will later in our garbling scheme assign to each wire  $i$  two keys  $K_{i,0}$  and  $K_{i,1}$ , representing the truth values 0 and 1 on this wire. To each wire  $i$ , we assign a permute bit  $\lambda_i$ , and to each key  $K_{i,a}$  representing truth value  $a \in \{0, 1\}$ , we assign a choice bit  $l_i^a = \lambda_i \oplus a$ .

We define the notion of garbling schemes as follows.

**Definition 1 (Garbling Scheme).** *A garbling scheme for a family of circuits  $\mathcal{F} = \{\mathcal{F}_n\}_{n \in \mathbb{N}}$ , where  $n$  is a polynomial in a security parameter  $k$ , consists of probabilistic polynomial-time algorithms  $\text{GC} = (\text{Garble}, \text{Encode}, \text{Eval}, \text{Decode})$  defined as follows.*

- **Garble** takes as input security parameter  $1^k$  and circuit  $f \in \mathcal{F}_n$ , and outputs garbled circuit  $F$ , encoding information  $e$ , and decoding information  $d$ , i.e.,  $(F, e, d) \leftarrow \text{Garble}(1^k, f)$ .
- **Encode** takes as input encoding information  $e$  and circuit input  $x \in \{0, 1\}^n$ , and outputs garbled input  $X$ , i.e.,  $X \leftarrow \text{Encode}(e, x)$ .

- **Eval** takes as input garbled circuit  $F$  and garbled input  $X$ , and outputs garbled output  $Y$ , i.e.,  $Y \leftarrow \text{Eval}(F, X)$
- **Decode** takes as input decoding information  $d$  and garbled output  $Y$ , and outputs circuit output  $y$ , i.e.,  $y \leftarrow \text{Decode}(d, Y)$ .

A garbling scheme should have the following correctness property: for all security parameters  $k$ , circuits  $f \in \mathcal{F}_n$ , and input values  $x \in \{0, 1\}^n$ ,  $(F, e, d) \leftarrow \text{Garble}(1^k, f)$ ,  $X \leftarrow \text{Encode}(e, x)$ ,  $Y \leftarrow \text{Eval}(F, X)$ ,  $y \leftarrow \text{Decode}(d, Y)$ , it holds that  $y = f(x)$ .

We then define the security notion of garbling schemes called *simulation-based privacy* as follows. We adapt the notion of Bellare et al. [8] slightly to allow the adversary access to a random oracle  $H$ . We denote by  $\Phi(f)$  the information about circuit  $f$  that is allowed to be leaked by the garbling scheme, e.g., size  $\Phi_{\text{size}}(f) = (n, m, l)$ , topology  $\Phi_{\text{topo}}(f) = (n, m, l, A, B)$ , or the entire information  $\Phi_{\text{circ}}(f) = (n, m, l, A, B, G)$  of circuit  $f = (n, m, l, A, B, G)$ .

**Definition 2 (Simulation-based Privacy).** For a garbling scheme  $\text{GC} = (\text{Garble}, \text{Encode}, \text{Eval}, \text{Decode})$ , function  $f \in \mathcal{F}_n$ , input values  $x \in \{0, 1\}^n$ , simulator  $\text{Sim}$ , adversary  $\mathcal{A}$ , and random oracle  $H$ , we define the advantage

$$\text{Adv}_{\text{GC}, \text{Sim}, \Phi, \mathcal{A}}^{\text{prv.sim}}(k) := \left| \Pr \left[ \begin{array}{l} st \leftarrow \mathcal{A}^H(1^k), (F, e, d) \leftarrow \text{Garble}(1^k, f), \\ X \leftarrow \text{Encode}(e, x) \end{array} : \mathcal{A}^H(st, F, X, d) = 1 \right] - \Pr \left[ \begin{array}{l} st \leftarrow \mathcal{A}^H(1^k), \\ (F, X, d) \leftarrow \text{Sim}(1^k, f(x), \Phi(f)) \end{array} : \mathcal{A}^H(st, F, X, d) = 1 \right] \right|.$$

A garbling scheme  $\text{GC} = (\text{Garble}, \text{Encode}, \text{Eval}, \text{Decode})$  is private, if there exists a probabilistic polynomial-time simulator  $\text{Sim}$ , such that for any function  $f \in \mathcal{F}_n$ , input values  $x \in \{0, 1\}^n$ , and probabilistic polynomial-time adversary  $\mathcal{A}$ , the advantage  $\text{Adv}_{\text{GC}, \text{Sim}, \Phi, \mathcal{A}}^{\text{prv.sim}}(k)$  is negligible.

### 2.3 Generalized Correlation Robustness

We define a generalized notion of correlation robustness for trapdoor one-way permutations, in which we extend correlation robustness as defined by Choi et al. [11]. Choi et al. considered ciphertexts of the form  $H(K \oplus a \Delta || L \oplus b \Delta || i) \oplus m$ , where  $i$  is a gate-ID,  $H$  is a hash function,  $K$  and  $L$  are input keys,  $a, b \in \{0, 1\}$  and  $\Delta$  is a global difference as needed for the free-XOR technique, meaning  $K_{i,1} = K_{i,0} \oplus \Delta$  for each wire  $i$ . Given four such ciphertexts and two input keys  $K_{A,\alpha} = K \oplus \alpha \Delta$  and  $K_{B,\beta} = L \oplus \beta \Delta$  for  $\alpha, \beta \in \{0, 1\}$ , the evaluator should only be able to decrypt one of them. Our ciphertexts have a similar form. However, we need to extend the definition of Choi et al. in two aspects. We do not have a global difference  $\Delta$ . Instead, our definition considers general correlations defined by arbitrary functions of input keys, rather than correlations given by a global difference. Since we garble gates backwards, the garbler needs to be able to invert

H. Therefore, instead of a hash function, we use a trapdoor one-way permutation  $E_\iota$ . Thus, our notion of correlation robustness allows for a trapdoor  $\iota$ .

Before we define correlation robustness, we recall the syntax of trapdoor one-way permutations.

**Definition 3 (Family of Trapdoor one-way Permutations).** *A family of trapdoor one-way permutations  $E = \{E_\iota : D_\iota \rightarrow D_\iota\}_{\iota \in I}$  for finite index set  $I$  is defined by a tuple of PPT algorithms  $E = (\text{Gen}_E, \text{Samp}_E, \text{Eval}_E, \text{Inv}_E)$  such that:*

- $\text{Gen}_E(1^k)$  is a probabilistic algorithm that outputs a pair  $(\iota, t_\iota)$  of index  $\iota \in I$  and trapdoor  $t_\iota$ .
- $\text{Samp}_E(1^k, \iota)$  is a probabilistic algorithm that outputs a uniformly random element  $x \in D_\iota$ .
- $\text{Eval}_E(1^k, \iota, x)$  is a deterministic algorithm that outputs  $y = E_\iota(x)$  (assuming that  $\iota$  is output by  $\text{Gen}_E$  and it holds that  $x \in D_\iota$ ).
- $\text{Inv}_E(\iota, t_\iota, y)$  is a deterministic algorithm that outputs an element  $x \in D_\iota$  such that  $y = E_\iota(x)$  (assuming that  $(\iota, t_\iota)$  is output by  $\text{Gen}_E$  and it holds that  $y \in D_\iota$ ).

In abuse of notation, we write  $x \stackrel{U}{\leftarrow} D_\iota$  to denote  $x \leftarrow \text{Samp}_E(1^k, \iota)$ ,  $y = E_\iota(x)$  to denote  $y = \text{Eval}_E(1^k, \iota, x)$ , and  $x = E_\iota^{-1}(t_\iota, y)$  to denote  $x = \text{Inv}_E(\iota, t_\iota, y)$ .

We define generalized correlation robustness of trapdoor one-way permutations as follows.

**Definition 4 (Generalized Correlation Robustness).** *Let  $f_0, f_1, f_2, f_3$  be any two-input functions. For a family of trapdoor one-way permutations  $E = \{E_\iota : D_\iota \rightarrow D_\iota\}_{\iota \in I}$  and a probabilistic polynomial-time adversary  $\mathcal{A}$ , let us consider the following probabilistic experiment  $\text{Exp}_{E, f_a, f_b, f_c, \mathcal{A}}^{\text{corr}}(k)$  for  $a < b < c \in \{0, 1, 2, 3\}$ .*

Experiment  $\text{Exp}_{E, f_a, f_b, f_c, \mathcal{A}}^{\text{corr}}(k)$ :

1.  $\beta \stackrel{U}{\leftarrow} \{0, 1\}$ .
2.  $(K, L) \leftarrow \mathcal{A}(1^k)$ .
3.  $(\iota, t_\iota) \leftarrow \text{Gen}_E(1^k)$  and  $K' \stackrel{U}{\leftarrow} D_\iota, L' \stackrel{U}{\leftarrow} D_\iota$ .
4. If  $\beta = 0$ ,  $Z_a := E_\iota(f_a(K, L'))$ ,  $Z_b := E_\iota(f_b(K', L))$ ,  $Z_c := E_\iota(f_c(K', L'))$ , otherwise,  $Z_a \stackrel{U}{\leftarrow} D_\iota, Z_b \stackrel{U}{\leftarrow} D_\iota, Z_c \stackrel{U}{\leftarrow} D_\iota$ .
5.  $\beta' \leftarrow \mathcal{A}(\iota, Z_a, Z_b, Z_c)$ .
6. Output 1 if and only if  $\beta = \beta'$ .

Let  $\text{Adv}_{E, f_0, f_1, f_2, f_3, \mathcal{A}}^{\text{corr}}(k) := \max_{a < b < c \in \{0, 1, 2, 3\}} \{\Pr [\text{Exp}_{E, f_a, f_b, f_c, \mathcal{A}}^{\text{corr}}(k) = 1] - 1/2\}$ . Then, a family of trapdoor one-way permutations  $E$  is correlation robust w.r.t.  $f_0, f_1, f_2, f_3$ , if for any probabilistic polynomial-time adversary  $\mathcal{A}$ , the advantage  $\text{Adv}_{E, f_0, f_1, f_2, f_3, \mathcal{A}}^{\text{corr}}(k)$  is negligible.

In the proposed garbling scheme, we use the following invertible linear function  $f = (f_0, f_1, f_2, f_3)$ , with

$$\begin{aligned} f_0 &: (x, y) \mapsto x + 2y, \\ f_1, f_2, f_3 &: (x, y) \mapsto x + y. \end{aligned}$$

### 3 Garbling Scheme for Formulas

We describe our basic garbling scheme for circuits with fan-out one. An extension to general circuits is given in Section 5.1. Our garbling scheme is designed for the semi-honest case, but can be extended to the malicious case using standard techniques. A brief discussion about this can be found in Section 5.3. As mentioned in the introduction, we can use each input wire only once. Multiple occurrences of a variable are handled by duplicating the corresponding input wire and assigning a new key pair for each occurrence of the variable, i. e. we treat multiple occurrences of the same input variable as different variables. This only affects security in the malicious case (see Section 5.3).

#### 3.1 Garbling

We describe our garbling scheme informally. A formal description of our garbling algorithm and encoding function is given in Figure 1 and 2, and the evaluation algorithm is given in Figure 3. Our decoding function is defined as

$$\text{Decode} : (Y, d) \mapsto d \oplus Y.$$

Let  $q$  be a prime number such that  $(2^k - q)/2^k$  is negligible, e.g. we can use a Mersenne prime  $q = 2^k - 1$  for appropriate  $k$ .  $k$  and  $q$  are public. We can regard a random element  $a \in \{0, 1\}^k$  as an element  $a \in \mathbb{F}_q \subset \{0, 1\}^k$  with negligible error probability. Let  $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$  be a keyed hash function, modeled as a programmable random oracle. Let  $H' : \{0, 1\}^* \rightarrow \{0, 1\}$  be a hash function which outputs one bit, modeled as a (non-programmable) random oracle. Let  $E_\iota : \{0, 1\}^k \rightarrow \{0, 1\}^k$  be a trapdoor one-way permutation on  $\{0, 1\}^k$  which is correlation robust with respect to functions  $(f_0, f_1, f_2, f_3)$  on  $\{0, 1\}^k$ , with

$$f_0 : (x, y) \mapsto x + 2y \in \mathbb{F}_q,$$

$$f_1, f_2, f_3 : (x, y) \mapsto x + y \in \mathbb{F}_q.$$

Let  $l$  denote the number of gates in circuit  $f$ ; since the number of input wires is  $n$ , the circuit output wire is wire  $l + n$ . We assume that the evaluator knows the circuit topology.

During the garbling process, we assign a permute bit  $\lambda_i$  to each wire  $i$ , a key  $K_{i,0}$  with choice bit  $l_{i,0} = \lambda_i$ , and a key  $K_{i,1}$  with choice bit  $l_{i,1} = 1 - \lambda_i$ .  $K_{i,0}$  corresponds to truth value 0, and  $K_{i,1}$  to truth value 1 on this wire. To garble a circuit, the garbler first chooses a key  $R$  for the hash function  $H$  uniformly at random, and includes it in the garbled circuit. Then, he chooses a pair of output keys  $K_{l+n,0}$  and  $K_{l+n,1}$  for the circuit output wire  $l+n$  uniformly at random. He assigns to the circuit output wire the permute bit  $\lambda_{l+n} := 0$ , and sets the output key choice bits  $l_{l+n,0} := 0$  and  $l_{l+n,1} := 1$ . Then, starting from gate  $l+n$ , the garbler iteratively computes the remaining keys by computing the input keys of each gate  $i$  depending on its output key pair. Since for the input wires  $A(i)$  and  $B(i)$  of gate  $i$  we have  $A(i) < B(i) < i$  for all  $i$ , we can simply iterate over  $i$

backwards and be sure that output keys are defined before their corresponding input keys. Before the input keys of a gate  $i$  are computed, a uniformly random permute bit  $\lambda_{A(i)}$  and, respectively,  $\lambda_{B(i)}$  is chosen for its two input wires, which defines the input key choice bits  $l_{A(i)}^0 = \lambda_{A(i)}$  and  $l_{A(i)}^1 = 1 - \lambda_{A(i)}$  for wire  $A(i)$ , and analog for  $B(i)$ . To compute the input keys for each gate, the garbler computes four ciphertexts  $c_0, c_1, c_2, c_3$  by computing  $c_x := H(R, i || x)$  for  $x = 0, 1, 2, 3$ . The choice bits of the (yet undefined) input keys map each possible input combination  $(a, b) \in \{0, 1\}^2$  to a ciphertext  $c_{2l_{A(i)}^a + l_{B(i)}^b}$ . This way, the evaluator can infer which ciphertext to use when processing gate  $i$ , without knowing the actual truth values on the input wires.

Using his trapdoor  $t_\iota$ , for each of the four possible inputs  $(a, b) \in \{0, 1\}^2$ , the garbler computes  $P_{2l_A^a + l_B^b} := E_\iota^{-1}(t_\iota, c_{2l_A^a + l_B^b} \oplus K_{i, G(a, b)})$ , and solves the equation system

$$\begin{aligned} P_0 &= K_{A, \lambda_A} + 2K_{B, \lambda_B} \\ P_1 &= K_{A, \lambda_A} + K_{B, 1 - \lambda_B} \\ P_2 &= K_{A, 1 - \lambda_A} + K_{B, \lambda_B} \\ P_3 &= K_{A, 1 - \lambda_A} + K_{B, 1 - \lambda_B} \end{aligned}$$

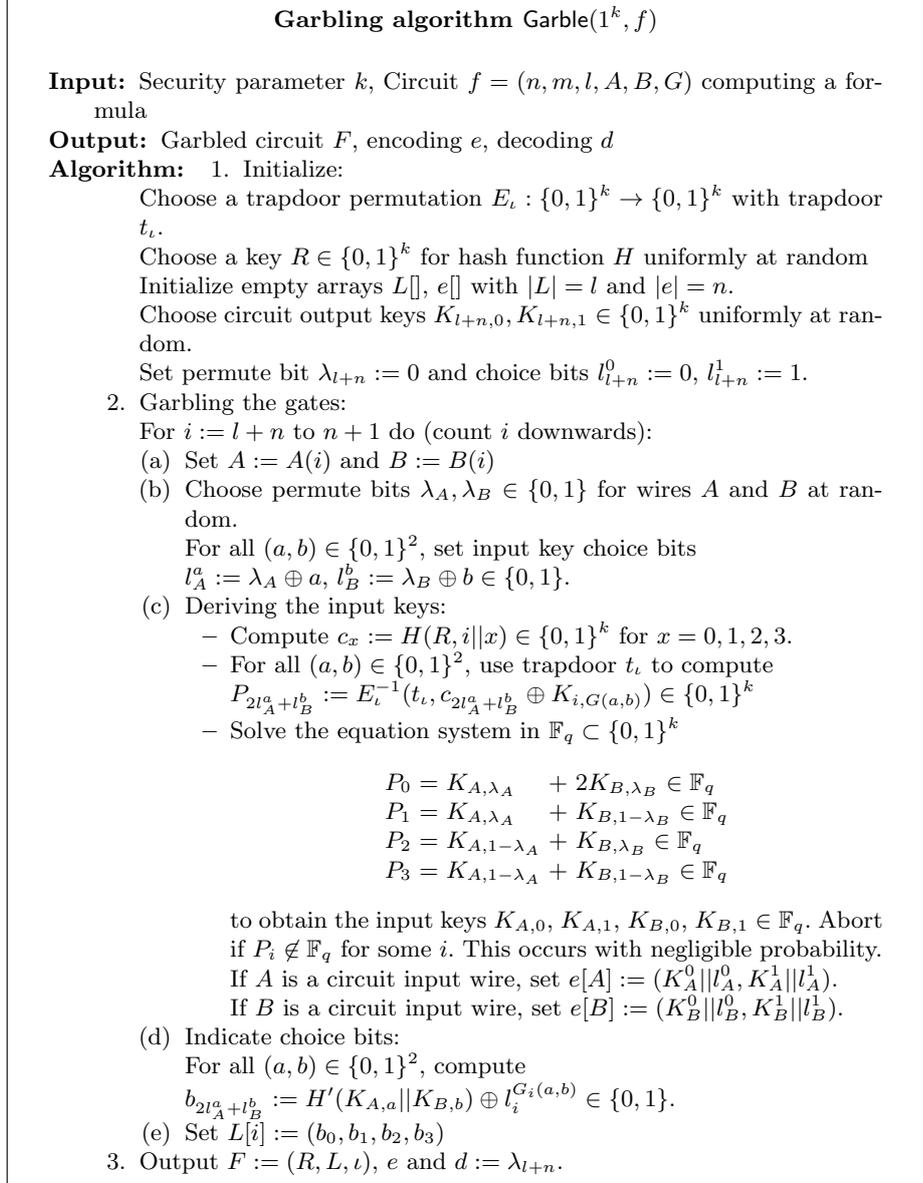
to compute the input keys  $K_{A(i), 0}, K_{A(i), 1}, K_{B(i), 0}$  and  $K_{B(i), 1}$ . To enable the evaluator to compute the choice bit of output key  $K_{i, G(a, b)}$ , for all four possible inputs  $(a, b) \in \{0, 1\}^2$ , the garbler includes in the garbled circuit  $i$  the bit  $b_{2l_A^a + l_B^b} := H'(K_{A, a} || K_{B, b}) \oplus l_i^{G_i(a, b)}$ . Since  $H'$  is a random oracle, each bit  $H'(K_{A, a} || K_{B, b})$  is random, and therefore, each bit  $b_{2l_A^a + l_B^b}$  is also random and independent of  $l_i^{G_i(a, b)}$ , so the four published bits give no information about the permute bits or the choice bits. These four bits are sorted according to the choice bits of the input keys, so the evaluator knows which one to use. The choice bits assigned to the circuit input keys are directly provided with these keys by extending them by one bit.

We set the permute bit of the circuit output wire to 0, so the choice bits of its keys correspond to the actual truth value on this wire. Apart from the keys for the circuit input wires and their choice bits, the only values communicated to the evaluator are the key  $R$  for the hash function  $H$ , the index  $\iota$  of the trapdoor one-way function, and the four bits  $b_{2l_A^a + l_B^b}$  for each gate. Altogether, our garbled circuit has size  $4l + |R| + |\iota|$ .

### 3.2 Evaluation

Evaluation (see Figure 3) is then straightforward: after obtaining the garbled circuit  $F$  and the input keys  $K_1, \dots, K_n$ , the evaluator processes the circuit *forward*. For each gate  $i$ , he computes the ciphertext  $c_{2l_A + l_B} := H(R, i || 2l_A + l_B)$ . Then he computes  $K_i := E_\iota(K_A + 2K_B) \oplus c_{2l_A + l_B}$ , if  $2l_A + l_B = 0$ , and  $K_i := E_\iota(K_A + K_B) \oplus c_{2l_A + l_B}$ , otherwise.

To obtain the choice bit  $l_i$  of the output key  $K_i$ , the evaluator computes  $l_i := H'(K_{A(i)} || K_{B(i)}) \oplus b_{2l_A + l_B}$ . The evaluator proceeds until he finally obtains the choice bit  $l_{l+n}$  of the circuit output key  $K_{l+n}$ , which equals the output  $f(x)$ .



**Fig. 1.** The proposed garbling algorithm.

### 3.3 Efficiency Comparison with Previous Schemes

**Comparison with the Half-Gates Construction.** We compare our garbling scheme with the best known result (the half-gate construction) proposed by Zahur et al. [34] on efficiency. It is difficult to compare them directly, since ours uses a public-key primitive, while the half-gate construction uses a symmetric-

<b>Encoding algorithm</b> $\text{Encode}(e, x)$
<p><b>Inputs:</b> Garbled input keys <math>e</math>, input <math>x</math></p> <p><b>Algorithm:</b> Parse <math>x</math> to <math>x = x_1 \dots x_n</math></p> <p style="padding-left: 20px;">For <math>i = 1</math> to <math>n</math> do:</p> <p style="padding-left: 40px;">Parse <math>e[i] = (e_0, e_1)</math></p> <p style="padding-left: 40px;"><math>X[i] := e_{x_i}</math></p> <p style="padding-left: 20px;">Return <math>X</math></p>

**Fig. 2.** The function Encode.

<b>Evaluation algorithm</b> $\text{Eval}(F, X)$
<p><b>Inputs:</b> Garbled circuit <math>F</math>, garbled input <math>X</math></p> <p><b>Algorithm:</b></p> <ol style="list-style-type: none"> <li>1. Parse <math>F</math> to <math>F = (R, L)</math></li> <li>2. For <math>j = 1</math> to <math>n</math> do <ul style="list-style-type: none"> <li><math>K_j    l_j := X[j]</math></li> </ul> </li> <li>3. Compute gate output keys and choice bits: <ul style="list-style-type: none"> <li>For <math>i := n + 1</math> to <math>l + n</math> do <ul style="list-style-type: none"> <li>Set <math>A := A(i)</math> and <math>B := B(i)</math>.</li> <li>Compute <math>c_{2l_A + l_B} := H(R, i    2l_A + l_B) \in \{0, 1\}^k</math>.</li> <li>If <math>x = 0</math>, set <math>K_i := E_i(K_A + 2K_B) \oplus c_x \in \{0, 1\}^k</math>,</li> <li>else set <math>K_i := E_i(K_A + K_B) \oplus c_x \in \{0, 1\}^k</math>.</li> <li>Parse <math>L[i]</math> to <math>(b_0, b_1, b_2, b_3)</math>.</li> <li>Set choice bit <math>l_i := H'(K_A    K_B) \oplus b_x \in \{0, 1\}</math>.</li> </ul> </li> </ul> </li> <li>4. Return <math>Y := l_{l+n}</math>.</li> </ol>

**Fig. 3.** The evaluation algorithm.

key one. Therefore, we evaluate the cost in an abstract way and later discuss the concrete efficiency in the circuit size.

Let  $L_E$  be the length of the domain of  $E$ ,  $L_I$  the length of  $l$ ,  $T_{\text{Eval}}$  the computation cost of  $\text{Eval}_E$ ,  $T_{\text{Inv}}$  the computation cost of  $\text{Inv}_E$ , and  $|R|$  the length of a hash key. Also let  $L_H$  be the length of the range of the correlation robust hash function used in [34],  $T_H$  the computation cost of hashing, and  $l_{\text{AND}}$  the number of AND gates. Table 1 shows the communication and computation cost of the two garbling schemes\*\*.

Regarding communication cost, the size of the garbled circuit is a constant multiple of  $l$  in our garbling scheme. Therefore, communication cost is asymptotically small when a formula is large. Regarding computation cost, our garbling scheme requires executions of  $\text{Eval}_E$  and  $\text{Inv}_E$ , which are computationally expensive compared to the computation of a hash function.

We additionally evaluate the circuit size for concrete example parameters. We assume that the trapdoor one-way permutation is instantiated by an RSA-based primitive such as RSA-DOAEP [6], the correlation robust hash function

\*\* We estimate the cost of our basic scheme here although our garbling scheme can be combined with the free-XOR technique as discussed in Subsection 5.2.

	communication cost		computation cost	
	circuit	input	garbling	evaluation
Ours	$4l +  R  + L_I$	$n \cdot L_E$	$4T_{\text{Inv}} \cdot l$	$T_{\text{Eval}} \cdot l$
[34]	$L_H \cdot l_{\text{AND}}$	$n \cdot L_H$	$2T_H \cdot l_{\text{AND}}$	$T_H \cdot l_{\text{AND}}$

**Table 1.** Comparison with the half-gates scheme [34].

is instantiated by fixed-key AES, and  $l_{\text{AND}} = 0.14l$ , since this is the case of the most significant difference among the examples in [34]. AES is regarded with 128-bit security, so we set  $L_E = 4096$ ,  $L_I = 8192$ ,  $|R| = 128$  and  $L_H = 128$ . In this case, the circuit size in our garbling scheme is smaller than the one of the half-gates scheme if  $l \geq 709$ .

Regarding total communication cost, however, our scheme cannot beat the half-gate construction in its current state. We consider our scheme a proof of concept of a new way of circuit construction. Finding more efficient instantiations for the trapdoor permutation or getting rid of the public key primitive altogether are interesting open problems.

**Comparison with Information-theoretic Garbling Scheme of [19].** We compare the efficiency of our scheme with that of the information-theoretic garbling scheme of [19], which garbles formulas more efficiently than other techniques. For simplicity, we consider the case of garbling a “balanced” formula such that all the gates connecting to the input wires have the same depth.

Regarding the communication cost, the scheme of [19] garbles a balanced formula with depth  $d$  in a way that the size of the garbled circuit is zero and the size of the encoded input is approximately  $2^{d+1} \cdot d^2$  (each of the  $2^{d+1}$  wires has a share with size approximately  $d^2$ ), and our scheme garbles such a formula in a way that the size of the garbled circuit is  $4l + |R| + L_I = 4 \cdot 2^d + |R| + L_I$  and the size of the encoded input is  $n \cdot L_E = 2^{d+1} \cdot L_E$ . Hence, in total, the communication cost of the scheme of [19] is  $2^{d+1} \cdot d^2$  whereas that of ours is  $2^{d+1}(L_E + 2) + |R| + L_I \approx 2^{d+1} \cdot L_E$  (see Table 2), and thus our scheme has smaller communication cost when  $d > \sqrt{L_E}$ .

	communication cost		
	circuit	input	total
Ours	$4 \cdot 2^d +  R  + L_I$	$2^{d+1} \cdot L_E$	$2^{d+1}(L_E + 2) +  R  + L_I$
[19]	0	$2^{d+1} \cdot d^2$	$2^{d+1} \cdot d^2$

**Table 2.** Comparison with information-theoretic garbling scheme of [19].

Regarding the computation cost, we note that since the scheme of [19] requires no cryptographic primitive whereas ours uses a public-key primitive, the

computation cost of our scheme is likely to be much bigger than that of the scheme of [19].

## 4 Security of the Proposed Scheme

The proposed garbling scheme is simulation-based private in the (programmable) random oracle model if the trapdoor permutation  $E$  is correlation robust.

**Theorem 1 (Simulation-based Privacy).** *The proposed garbling scheme described in Section 3 satisfies simulation-based privacy of Definition 2 if we assume that  $E$  satisfies correlation robustness as defined in Definition 4,  $H$  is a programmable random oracle and  $H'$  is a non-programmable random oracle. More precisely, for any adversary  $\mathcal{A}$  there exists an adversary  $\mathcal{B}$  such that*

$$Adv_{\text{GC,Sim},\Phi,\mathcal{A}}^{\text{prv.sim}}(k) \leq l \cdot Adv_{E,f_0,f_1,f_2,f_3,\mathcal{B}}^{\text{corr}}(k) + q_H \cdot 2^{-k},$$

where  $l$  is the number of gates and  $q_H$  is the number of queries  $\mathcal{A}$  makes to  $H$ .

*Proof.* We consider the following hybrid games  $H_{\text{real}}, H_0, H_1, \dots, H_l, H_{\text{sim}}$ , where  $H_{\text{real}}$  is identical to the real experiment ( $\beta = 0$ ), and  $H_{\text{sim}}$  is identical to the simulated experiment ( $\beta = 1$ ) of Definition 2. The simulator  $\text{Sim}(1^k, f(x), \Phi_{\text{topo}}(f))$  of Definition 2 is provided in Figure 6.

**Game  $H_{\text{real}}$ :** This game is identical to the real experiment of Definition 2. The garbled circuit  $(F, e, d)$  is generated by the garbling algorithm  $\text{Garble}(1^k, f)$  of the real garbling scheme, as given in Figure 1, and garbled input  $X$  is generated by the encoding algorithm  $\text{Encode}(e, x)$  (Figure 2) of the real scheme.

**Game  $H_0$ :** In this game, the garbled circuit is generated forward, from input gates to output gate, which is in contrast to the real scheme. The garbled circuit is generated by algorithm  $\text{Sim}_0(1^k, f)$  as described in Figure 4, and garbled input  $X$  is generated by encoding algorithm  $\text{Encode}(e, x)$  (Figure 2) of the real scheme. We note that in step 2-(c), we need to program the random oracle  $H$  to keep consistency.

**Game  $H_s$ :** With the Games  $H_s$ , we move from Game  $H_0$  to the full simulation with a gate-wise replacement. We incrementally replace variables not touched in an evaluation with input  $x$  with random values gate by gate, such that finally, in the full simulation, the “unused” variables of each gate are replaced by randomness. For  $s = 1, \dots, l$ , in each Game  $H_s$ , the garbled circuit is generated by algorithm  $\text{Sim}_s(1^k, x, f)$ , given in Figure 5, and garbled input  $X$  is generated by the encoding algorithm  $\text{Encode}(e, x)$  (Figure 2) of the real scheme. We say that key  $K_{i,\beta}$  is active if and only if the bit obtained on wire  $i$  is  $\beta$  when circuit  $f$  is evaluated with input  $x$ . Only in these intermediate games  $H_s$ , we give the simulator knowledge of input  $x$  and circuit  $f$ , so he can label each key  $K_{i,\beta}$  as active or inactive. We say that value  $E_i(K_{A,a} + K_{B,b})$  is active if and only if both  $K_{A,a}$  and  $K_{B,b}$  are active. We replace inactive values with random values in step 2-(c) of algorithm  $\text{Sim}_s(1^k, x, f)$  for gates

**Simulator  $\text{Sim}_0(1^k, f)$**

**Input:** Security parameter  $k$ , circuit  $f = (n, m, l, A, B, G)$  computing a formula.

**Output:** Garbled circuit  $F$ , encoding  $e$ , decoding  $d$ .

**Algorithm:** 1. Initialize:

Choose a trapdoor permutation  $E_\iota : \mathbb{F}_q \rightarrow \mathbb{F}_q$  with trapdoor  $t_\iota$ .

Choose a key  $R$  for hash function  $H$  uniformly at random

Initialize empty arrays  $L[], e[]$  with  $|L| = l$  and  $|e| = n$ .

For  $i := 1$  to  $n$  do:

(a) Choose circuit input keys  $K_{i,0}, K_{i,1} \in \mathbb{F}_q$  uniformly at random.

(b) Choose permute bit  $\lambda_i \in \{0, 1\}$  uniformly at random and set choice bits  $l_i^0 := \lambda_i \oplus 0, l_i^1 := \lambda_i \oplus 1$ .

(c) Set  $e[i] := (K_i^0 || l_i^0, K_i^1 || l_i^1)$ .

2. Garbling the gates:

For  $i := n + 1$  to  $n + l$  do (count  $i$  upwards):

(a) Set  $A := A(i)$  and  $B := B(i)$ .

(b) Choose permute bit  $\lambda_i$  for output wire  $i$  uniformly at random. Set output key choice bit  $l_i^0 := \lambda_i \oplus 0, l_i^1 := \lambda_i \oplus 1$ .

(c) 'Deriving' the output keys:

– For all  $(a, b) \in \{0, 1\}^2$ , compute

$Y_{2l_A^a + l_B^b} := E_\iota(K_{A,a} + 2K_{B,b})$ , if  $2l_A^a + l_B^b = 0$  or

$Y_{2l_A^a + l_B^b} := E_\iota(K_{A,a} + K_{B,b})$ , otherwise.

– Choose output keys  $K_{i,0}, K_{i,1}$  uniformly random

– Program random oracle such that

$c_{2l_A^a + l_B^b} := K_{i,G(a,b)} \oplus Y_{2l_A^a + l_B^b}$

(d) Indicate choice bits:

For all  $(a, b) \in \{0, 1\}^2$ , compute

$b_{2l_A^a + l_B^b} := H'(K_{A,a} || K_{B,b}) \oplus l_i^{G_i(a,b)}$ .

(e) Set  $L[i] := (b_0, b_1, b_2, b_3)$

3. Output  $F := (R, L, \iota)$ ,  $e$  and  $d := \lambda_{n+l}$ .

**Fig. 4.** The algorithm  $\text{Sim}_0$  for game  $H_0$ .

$i = n + 1, \dots, n + s$ ; gates  $i = n + s + 1, \dots, n + l$  are generated as in game  $H_0$ . We also replace the inactive output key by the active output key in this step.

**Game  $H_{sim}$ :** This game is identical to the simulated experiment in Definition 2.

The garbled circuit and garbled input  $(F, X, d)$  are generated by algorithm  $\text{Sim}(1^k, f(x), \Phi_{\text{topo}}(f))$ , given in Figure 6, without knowledge of  $x$ .

The difference between the advantage of adversary  $\mathcal{A}$  in  $H_{\text{real}}$  and his advantage in  $H_0$  is bound by  $q_H \cdot 2^{-k}$  as follows, where  $q_H$  is the number of queries that  $\mathcal{A}$  makes to  $H$ . Since  $E_\iota$  is a bijective map and  $f = (f_0, f_1, f_2, f_3)$  is an invertible linear map, the system of four linear equations

$$c_{2l_A^a + l_B^b} = K_{i,G(a,b)} \oplus E_\iota(f_{2l_A^a + l_B^b}(K_{A,a}, K_{B,b})), \quad (a, b) \in \{0, 1\}^2$$

**Simulator  $\text{Sim}_s(1^k, x, f)$**

**Input:** Security parameter  $k$ , circuit input  $x$ , circuit  $f = (n, m, l, A, B, G)$  computing a formula.

**Output:** Garbled circuit  $F$ , encoding  $e$ , decoding  $d$ .

**Algorithm:** 1. Initialize:

Same as  $\text{Sim}_0$ .

2. Garbling the gates:

For  $i := n + 1$  to  $n + s$  do (count  $i$  upwards):

(a) Set  $A := A(i)$  and  $B := B(i)$ .

(b) Choose permute bit  $\lambda_i$  for output wire  $i$  uniformly at random.

Set output key choice bit  $l_i^0 := \lambda_i \oplus 0$ ,  $l_i^1 := \lambda_i \oplus 1$ .

(c) Deriving the output keys:

– For all  $(a, b) \in \{0, 1\}^2$ ,

if  $K_{A,a}, K_{B,b}$  are active keys, compute

$Y_{2l_A^a + l_B^b} := E_\iota(K_{A,a} + 2K_{B,b})$ , if  $2l_A^a + l_B^b = 0$  or

$Y_{2l_A^a + l_B^b} := E_\iota(K_{A,a} + K_{B,b})$  if  $2l_A^a + l_B^b \neq 0$ ,

otherwise choose  $Y_{2l_A^a + l_B^b} \in \mathbb{F}_q$  uniformly at random.

– Choose output keys  $K_{i,0}, K_{i,1}$  uniformly random.

– Let  $(a^*, b^*) \in \{0, 1\}^2$  be the values such that  $K_{A,a^*}$  and  $K_{B,b^*}$  are active keys. Then, for all  $(a, b) \in \{0, 1\}^2$ , program random oracle such that  $c_{2l_A^a + l_B^b} := K_{i,G(a^*, b^*)} \oplus Y_{2l_A^a + l_B^b}$ .

(d) Indicate choice bits:

For all  $(a, b) \in \{0, 1\}^2$ , compute

$b_{2l_A^a + l_B^b} := H'(K_{A,a} || K_{B,b}) \oplus l_i^{G_i(a,b)}$ .

(e) Set  $L[i] := (b_0, b_1, b_2, b_3)$

3. Garbling the gates:

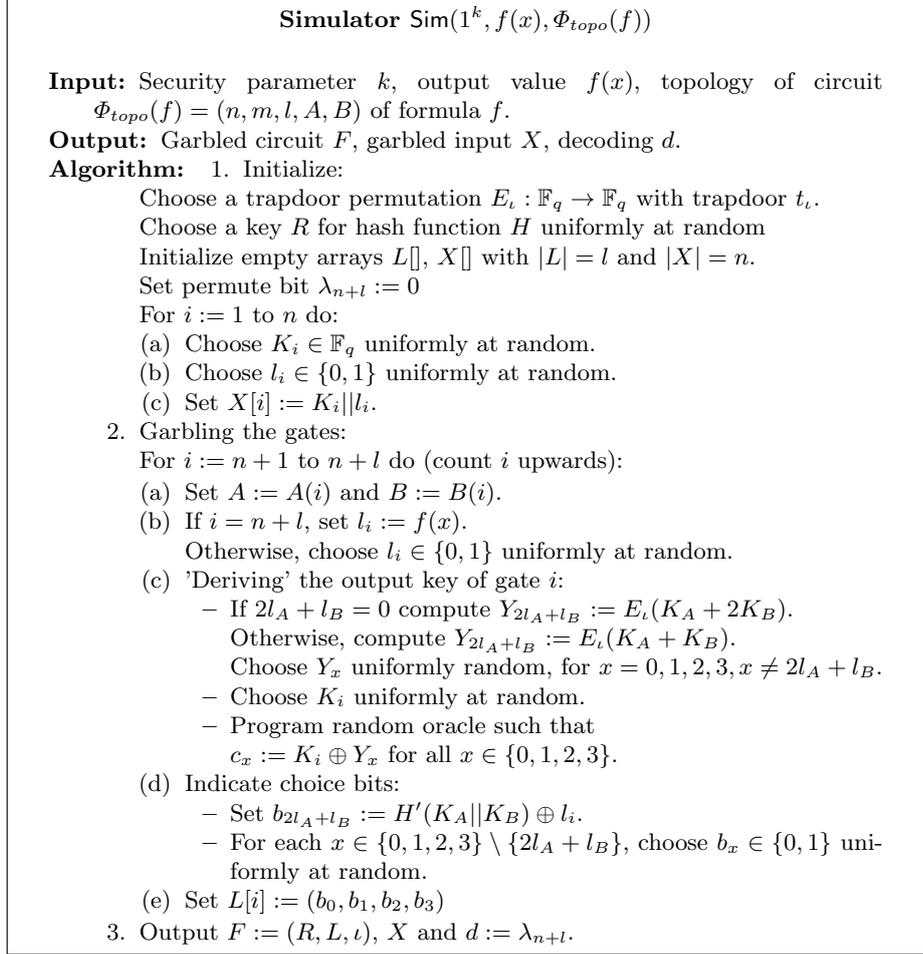
For  $i := n + s + 1$  to  $n + l$  do (count  $i$  upwards):

Same as  $\text{Sim}_0$ .

4. Output  $F := (R, L, \iota)$ ,  $e$  and  $d := \lambda_{n+l}$ .

**Fig. 5.** The algorithm  $\text{Sim}_s$  for game  $H_s$ .

is uniquely solvable with the input keys  $K_{A,0}, K_{A,1}, K_{B,0}$  and  $K_{B,1}$  as variables. Therefore, it maps uniformly random output keys  $K_{i,0}, K_{i,1}$  to uniformly random input keys  $K_{A,0}, K_{A,1}, K_{B,0}, K_{B,1}$  in  $H_{real}$ . In  $H_0$ , both the input keys  $K_{A,0}, K_{A,1}, K_{B,0}, K_{B,1}$  and the output keys  $K_{i,0}, K_{i,1}$  are chosen uniformly at random. However, the relation between the values  $c_{2l_A^a + l_B^b}$ ,  $K_{i,G(a,b)}$  and  $(K_{A,a}, K_{B,b})$ , which is given by these four equations, is preserved for all  $(a, b) \in \{0, 1\}^2$  by programming the random oracle  $H$  to output consistent values for  $c_0, c_1, c_2$  and  $c_3$ . So, all keys are uniformly random in both  $H_{real}$  and  $H_0$ , and the distributions of the garbled circuits created in  $H_{real}$  and  $H_0$  are indistinguishable to the adversary, except for the case where the adversary asks  $H(R, i || x)$  for some  $x \in \{0, 1, 2, 3\}$  before the simulator garbles gate  $i$ , where he defines this value. However, the hash key  $R$  is given to the adversary only after garbling the whole circuit. Before that,  $\mathcal{A}$  can guess  $R$  with probability  $2^{-k}$  in each query to  $H$ .



**Fig. 6.** The algorithm **Sim** to create the simulated garbled circuit.

Thus, from the union bound, the probability that  $\mathcal{A}$  makes “a bad query” to  $H$  is bound by  $q_H \cdot 2^{-k}$ .

The advantage of adversary  $\mathcal{A}$  in distinguishing  $H_s$  and  $H_{s+1}$  for  $s = 0, \dots, l-1$  is bound by the advantage  $\text{Adv}_{E, f_0, f_1, f_2, f_3, \mathcal{B}}^{\text{corr}}(k)$  of the following adversary  $\mathcal{B}$  of the correlation robustness game as follows. Let  $A$  and  $B$  denote the input wires of gate  $n+s+1$ . For simplicity, we assume that  $K_{A,0}$  and  $K_{B,1}$  are the active input keys of gate  $n+s+1$ ; the three other cases are analogous. Remember that gates are labeled after their output wires; since the first  $n$  wires are circuit input wires, in Game  $H_{s+1}$ , Gate  $n+s+1$  is the next to be replaced by a “simulated” gate. At the beginning of the correlation robustness game, the adversary  $\mathcal{B}$  selects  $K_{A,0}$  and  $K_{B,1}$  uniformly at random, and outputs  $K := K_{A,0}$  and  $L := K_{B,1}$  as

the target keys of the correlation robustness experiment. Then,  $\mathcal{B}$  receives index  $\iota$  of  $E$  and challenge  $Z_1, Z_2, Z_3$  of the correlation robustness experiment.

To provide adversary  $\mathcal{A}$  with a “challenge circuit”, adversary  $\mathcal{B}$  simulates gates  $n + 1$  to  $n + s$  as in Step 2 of algorithm  $\text{Sim}_s(1^k, x, f)$ , while setting the active keys of wire  $A$  and  $B$  to  $K_{A,0}$  and  $K_{B,1}$ , respectively. Since they were chosen uniformly at random, this does not contradict the construction in Game  $H_s$  or  $H_{s+1}$ .

The adversary  $\mathcal{B}$  then creates Gate  $n + s + 1$  as follows:

1. Compute active value  $Y_{2l_A^0+l_B^1} := E_\iota(f_{2l_A^0+l_B^1}(K, L))$ , and set the inactive values to the random values from the challenge as follows:  $Y_{2l_A^0+(1-l_B^1)} := Z_1$ ,  $Y_{2(1-l_A^0)+l_B^1} := Z_2$  and  $Y_{2(1-l_A^0)+(1-l_B^1)} := Z_3$ .
2. Choose  $K_{n+s+1,0}$  and  $K_{n+s+1,1}$  uniformly at random. Program random oracle such that  $c_{2l_A^a+l_B^b} := K_{n+s+1,G(a,b)} \oplus Y_{2l_A^a+l_B^b}$  for all  $(a, b) \in \{0, 1\}^2$ .

All remaining gates (Gate  $n + s + 2$  to Gate  $n + l$ ), if there are any, are created as in algorithm  $\text{Sim}_s(1^k, x, f)$ . Adversary  $\mathcal{B}$  gives the created garbled circuit to  $\mathcal{A}$ , and outputs whatever  $\mathcal{A}$  outputs.

If  $Z_1, Z_2$  and  $Z_3$  are correctly formed, i.e.,  $Z_1 = E_\iota(f_{2l_A^0+(1-l_B^1)}(K_{A,0}, K_{B,1}))$ ,  $Z_2 = E_\iota(f_{2(1-l_A^0)+l_B^1}(K_{A,1}, K_{B,0}))$ , and  $Z_3 = E_\iota(f_{2(1-l_A^0)+(1-l_B^1)}(K_{A,1}, K_{B,1}))$ , the simulated garbled circuit is generated as in  $H_s$ . If  $Z_1, Z_2$  and  $Z_3$  are uniformly random,  $c_{2l_A^0+l_B^1}$  is  $K_{n+s+1,G(a,b)} \oplus E_\iota(f_{2l_A^0+l_B^1}(K, L))$ , and  $c_{2l_A^0+(1-l_B^1)}$ ,  $c_{2(1-l_A^0)+l_B^1}$  and  $c_{2(1-l_A^0)+(1-l_B^1)}$  are uniformly random; thus, they are distributed identically to those in  $H_{s+1}$ . Thus, Adversary  $\mathcal{B}$  is successful in the correlation robustness game whenever Adversary  $\mathcal{A}$  is successful in distinguishing  $H_s$  and  $H_{s+1}$ . Therefore, the advantage of adversary  $\mathcal{A}$  in distinguishing  $H_s$  and  $H_{s+1}$  is bound by advantage  $\text{Adv}_{E,f_0,f_1,f_2,f_3,\mathcal{B}}^{\text{corr}}(k)$  of the correlation robustness experiment.

The distributions of the simulated garbled circuits in  $H_l$  and  $H_{sim}$  are indistinguishable as follows. In  $H_{sim}$ , the simulator “loses knowledge” of input  $x$ , i.e, he does not know which values are active, and therefore, which inputs should result in the active key. He does not need this knowledge anymore, since the inactive output keys have been replaced by the active ones in the previous games. The bits  $b_x$  for  $x = 0, 1, 2, 3$  in  $H_{sim}$  are chosen uniformly at random. Since  $H'$  is a random oracle, this is indistinguishable from setting  $b_{2l_A^a+l_B^b} := H'(K_{A,a}||K_{B,b}) \oplus l_i^{G_i(a,b)}$ .

Thus, by summing up the differences between the games, we have  $\text{Adv}_{\text{GC},\text{Sim},\Phi,\mathcal{A}}^{\text{priv.sim}}(k) \leq l \cdot \text{Adv}_{E,f_0,f_1,f_2,f_3,\mathcal{A}}^{\text{corr}}(k) + q_H \cdot 2^{-k}$ .  $\square$

## 5 Extensions

In this section, we generalize our garbling scheme to allow circuits with arbitrary fan-out. Then we show how to combine our garbling scheme with the free-XOR technique to further reduce communication cost. We also discuss the malicious case.

## 5.1 Arbitrary Fan-Out

Our scheme can handle arbitrary fan-out with a slight modification, which comes at the cost of going back to linear size of the garbled circuit in the worst case. Note that if a circuit has more than one circuit output wire, the decoding function needs to be adjusted in the obvious way. The reason our basic scheme can only be used for formulas is that our garbling algorithm leaves no degree of freedom when computing the input keys of a gate. This implies a conflict when a wire is the input wire of two different gates  $i$  and  $j$ : W.l.o.g., consider the case of a shared wire  $s := B(i) = A(j)$  and  $i < j$ . When garbling gate  $i$ , the garbler computes corresponding input keys  $K_{s,0}$  and  $K_{s,1}$  for the shared wire  $s$ . However, when garbling gate  $j$ , he obtains different input keys  $K'_{s,0} \neq K_{s,0}$  and  $K'_{s,1} \neq K_{s,1}$  for the same wire. We can solve this conflict by providing two ciphertexts for this wire by encrypting  $K'_{s,0}$  with key  $K_{s,0}$ , and  $K'_{s,1}$  with  $K_{s,1}$ . Then we can use  $K_{s,0}$  and  $K_{s,1}$  for gate  $i$ , and  $K'_{s,0}$  and  $K'_{s,1}$  for gate  $j$ . We sort the two ciphertexts according to the permute bit  $\lambda_s$  of wire  $s$ . Since  $\lambda_s$  applies to both key pairs,  $K'_{s,b}$  and  $K_{s,b}$  have the same choice bit for  $b = 0, 1$ . This way, whenever a wire is input for a second gate, the evaluator can compute the “second version” of his obtained key by decrypting the corresponding ciphertext. Our garbled circuit now has the size

$$4l + |R| + |\iota| + \sum_{d=1}^{d_{max}} (2(d-1) \cdot k \cdot l_d),$$

where  $l_d$  is the number of gates with fan-out  $d$ , and  $d_{max}$  is the maximal number of output wires of a single gate.

In the case that a gate has two conflicting input wires, this gate might need four ciphertexts, making our scheme seemingly inefficient. However, depending on the number of circuit input and circuit output wires, we still have an average of considerably less than four ciphertexts per gate. Consider a circuit with  $k$  gates. According to the model we use, each gate has two input wires and one output wire. So altogether we have  $2k$  input wires, of which  $n$  are circuit input wires (i.e., cannot origin from other gates), and  $k$  output wires, of which  $m$  are circuit output wires (i.e., those  $m$  wires cannot induce a conflict). So we have  $k - m$  wires which need to “go somewhere”, and  $2k - n$  places where they can go. Since the two ciphertexts are only needed for each *additional* connection of an output wire, each wire can connect to one input slot “for free”. So in the worst case, we end up with  $(2k - n) - (k - m) = k + m - n$  conflicts, meaning  $2(k - n + m)$  ciphertexts. In the case of  $m \leq n$ , we have at most two ciphertexts per gate in the worst case. If  $m > n$ , we might have more than two ciphertexts per gate in the case of an unfortunate layout, but still less than four since  $m < k$ . So in the case of arbitrary fan-out, whether or not our scheme provides an efficiency gain compared to other optimizations like the half-gate or fleXOR constructions, strongly depends on the circuit layout. As we can see in the following section, our basic scheme is compatible to the free-XOR technique. This compatibility translates to the case of arbitrary fan-out with similar conflict issues arising from conflicting wire offsets as well as conflicting keys. However, in some cases where

XOR gates are involved in a conflict, only one ciphertext is needed to solve it. This is elaborated in more detail in Section 5.2.

## 5.2 Incorporating Free-XOR

Going back to the world of formulas, our garbling scheme is compatible with a slightly adapted version of the free-XOR technique introduced in [22]. The free-XOR technique allows us to garble XOR gates “for free”, by choosing each key pair with a constant offset  $\Delta$ , such that output keys can be obtained by XORing corresponding input keys. To be compatible with our garbling scheme, XOR gates, too, have to be garbled backwards, but induce no communication cost.

For each gate  $i$ , let  $K_{i,0}$  and  $K_{i,1}$  be the already given output keys for gate  $i$ . (If  $i$  is a circuit output wire, choose the two keys at random.) If  $i$  is a non-XOR gate, the garbler does exactly what he does in our basic scheme. If  $i$  is an XOR gate, the garbler sets  $\Delta := K_{i,0} \oplus K_{i,1}$ . Then for the input wires  $A(i)$  and  $B(i)$ , the garbler chooses random keys  $K_{A(i),0}$  and  $K_{B(i),0}$  such that  $K_{A(i),0} \oplus K_{B(i),0} = K_{i,0}$  and sets  $K_{A(i),1} := K_{A(i),0} \oplus \Delta$  and  $K_{B(i),1} := K_{B(i),0} \oplus \Delta$ . The permute bits  $\lambda_{A(i)}$  and  $\lambda_{B(i)}$  of the input wires are set such that  $\lambda_{A(i)} \oplus \lambda_{B(i)} = \lambda_i$ . This way, the evaluator can obtain the choice bit of the output key by applying the XOR function to the choice bits of the input keys. Using this technique, our XOR gates are free: they induce no communication cost. This works as long as the circuit is cycle-free, which is required by the property  $A(i) < B(i) < i$  for each gate.

When incorporating free-XOR, we need our trapdoor permutation  $E_i$  to additionally achieve a circular security property similar to the one introduced by Choi et al. [11]. Our incorporation of free-XOR seems similar to the combination of the fleXOR technique with two-row-reduction in [21]. However, we have different dependencies: In the fleXOR technique, the offset  $\Delta$  is determined by the input keys of a gate, such that if there is a sub-tree in the circuit consisting only of XOR gates, there might be different offsets  $\Delta_i$  within this sub-tree. In our scheme,  $\Delta$  is determined by the output key, so there is only one  $\Delta$  within each sub-tree. Since input keys depend on output keys, we can define the input keys of an XOR sub-tree to have the same offset  $\Delta$ .

**Free-XOR and Arbitrary Fan-Out.** Incorporating the free-XOR technique allows us some freedom in choosing the input keys of XOR gates. This can reduce the number of ciphertexts needed when dealing with conflicting wires in some cases. Consider gates  $i$  and  $j$  with shared input wire  $B(i) = A(j)$  (other shared wires are analogous). There are essentially three types of conflicting wires:

- **Case 1:** The keys of the input wires of both gates  $i$  and  $j$  cannot be chosen freely:

This occurs for example when one of the gates has more than one conflicting input wire, or when both  $i$  and  $j$  are non-XOR gates. In this case, we

solve the conflict exactly as described in Section 5.1, by encrypting the keys determined by garbling gate  $j$  with those determined for gate  $i$ . Solving this conflict induces a communication cost of two ciphertexts, as before.

- **Case 2:**  $i$  and  $j$  are both XOR gates with no additional conflict:  
The problem here is that for the input wires of  $i$  and  $j$  we might have to use different values  $\Delta_i$  and  $\Delta_j$ . In this case, let  $\lambda_B$  be the permute bit of wire  $B(i)$  (i.e.,  $K_{B(i),\lambda_B}$  is the key with choice bit zero) and choose  $K_{B(i),\lambda_B} = K_{A(j),\lambda_B}$  at random (the other input wires of  $i$  and  $j$  then have to be chosen accordingly such that the correct output key is obtained). Then set  $K_{B(i),1-\lambda_B} := K_{B(i),\lambda_B} \oplus \Delta_i$  and  $K_{A(j),1-\lambda_B} := K_{A(j),\lambda_B} \oplus \Delta_j$ , and include the ciphertext  $Enc_{K_{B(i),1-\lambda_B}}(K_{A(j),1-\lambda_B})$  in the garbled circuit. No additional ciphertext is needed, so this conflict can be solved with only one ciphertext.
- **Case 3:**  $i$  is a non-XOR gate or an XOR gate with additional conflicts, and  $j$  is an XOR gate with no additional conflict:  
This can be solved similar to Case 2: let  $\lambda_B$  be the permute bit of wire  $B(i)$ . Set  $K_{A(j),\lambda_B} := K_{B(i),\lambda_B}$ . Set  $K_{A(j),1-\lambda_B} := K_{A(j),\lambda_B} \oplus \Delta_j$  and include the ciphertext  $Enc_{K_{B(i),1-\lambda_B}}(K_{A(j),1-\lambda_B})$  in the garbled circuit. Like Case 2, this conflict can be solved with only one ciphertext.

As we can see, if an XOR gate is involved in the conflict, and this is the only conflict of this XOR gate, we need to include only one ciphertext for the conflicting wire.

### 5.3 Security against Malicious Adversaries

Our scheme extends to the malicious case by using the cut-and-choose-based techniques described by Lindell and Pinkas [24].

One additional issue we need to address is that our scheme requires us to duplicate input wires if the same input variable is used multiple times. In the malicious case, we need to prove that consistent input keys are sent for each wire representing the same variable.

The case of a malicious evaluator is simply handled by sending all input keys for one variable in a single OT. The case of a malicious garbler can be solved by adapting the consistency check of the garbler’s input used by Lindell and Pinkas (see Fig. 2 in [24]) in a straightforward manner, by grouping the input wires representing the same variable.

## 6 Instantiation with PRIV-secure Deterministic Encryption

In this section, we show that we can instantiate our trapdoor one-way permutation  $E_t$  with a deterministic encryption function which achieves the notion of PRIV-security, introduced by Bellare et al. in [6]. Though not limited to deterministic encryption, PRIV-security has been introduced as the strongest security

**Experiment**  $Exp_{\mathcal{A}}^{PRIV}(k)$

**Input:** Security parameter  $k$   
 $st \leftarrow A_t(1^k)$   
 $\tilde{b} \leftarrow_R \{0, 1\}$   
 $(\mathbf{x}_0, \mathbf{x}_1) \leftarrow A_m(1^k, \tilde{b}, st)$   
 $(pk, sk) \leftarrow_R \mathcal{K}(1^k)$   
 $\mathbf{c} := E(pk, \mathbf{x}_{\tilde{b}})$   
 $b' \leftarrow A_g(1^k, pk, \mathbf{c}, st)$   
Return( $b' = \tilde{b}$ )

**Fig. 7.** The experiment  $Exp^{PRIV}$ .

notion one can achieve for deterministic encryption. It was proven equivalent to various other notions of deterministic encryption by Bellare et al. in [7].

In this section, we show that PRIV-security implies correlation robustness with respect to addition, as needed in our garbling scheme. At the end of this section, we discuss a concrete instantiation.

We briefly recall the PRIV-notion here. Though we use the term PRIV, we actually use one of its equivalents called *IND* in [7]. Let  $E(pk, m)$  denote a deterministic encryption function with key space  $\mathcal{K}(1^k)$ , domain space  $Dom(E)$  and image space  $Im(E)$ . Let  $x \leftarrow_R S$  denote the assignment of a random element in set  $S$  to variable  $x$ . As a shortcut, we write  $E(pk, \mathbf{x})$  for componentwise encryption of a vector  $\mathbf{x}$ . PRIV-security is defined via the experiment  $Exp_{\mathcal{A}}^{PRIV}$  (see Figure 7). It uses an adversary algorithm  $\mathcal{A} = (A_t, A_m, A_g)$ , which is split into three adversaries  $A_t$ ,  $A_m$  and  $A_g$ . The first one,  $A_t$ , chooses a string  $st$  which is readable (but not writable) by the other two adversaries. It does not exist in the original PRIV-definition of [6], and only used to show equivalence in [7], where  $st$  is then required to be the empty string in the actual security definition, while the most fit  $st$  is assumed hard-wired into the other two adversaries  $A_m$  and  $A_g$ . We use  $A_t$  and  $st$  for our reduction to correlation robustness in a similar way. The adversary  $A_m$  chooses vectors  $\mathbf{x}_0$  and  $\mathbf{x}_1$  of plaintexts knowing choice bit  $\tilde{b}$ , but without knowledge of the public encryption key  $pk$ . The vector  $\mathbf{x}_{\tilde{b}}$  is then encrypted componentwise. The third adversary,  $A_g$ , then tries to guess  $\tilde{b}$ , getting the public key  $pk$ , the ciphertext  $E(pk, \mathbf{x}_{\tilde{b}})$  as well as  $st$  as input.

**Definition 5 (PRIV-security).** *An encryption function  $E$  is PRIV-secure, if the advantage  $Adv_{\mathcal{A}}^{PRIV}(k) := \Pr[Exp_{\mathcal{A}}^{PRIV}(k) = 1] - \frac{1}{2}$  is negligible in  $k$  for all security parameters  $k$  and all adversaries  $\mathcal{A} = (A_t, A_m, A_g)$ , under the following requirements:*

1.  $|\mathbf{x}_0| = |\mathbf{x}_1|$
2.  $\mathbf{x}_0[i] = \mathbf{x}_0[j]$  iff  $\mathbf{x}_1[i] = \mathbf{x}_1[j]$
3. *High min-entropy:*  $Pr[\mathbf{x}_{\beta}[i] = x : \mathbf{x} \leftarrow A_m(1^k, \tilde{b}, st)]$  is negligible in  $k$  for all  $\beta \in \{0, 1\}$ , all  $1 \leq i \leq |\mathbf{x}|$ , all  $x \in Dom(E)$ , and all  $st \in \{0, 1\}^*$ .
4.  $st$  is the empty string.

The first requirement prevents the adversary from trivially distinguishing the two ciphertexts by their length, while the second requirement addresses the fact that a deterministic encryption maps equal plaintexts to equal ciphertexts, and prevents the adversary from distinguishing ciphertexts using their equality pattern.

PRIV-security is sufficient to achieve our notion of correlation robustness with respect to the linear function we use in our garbling scheme. Therefore, we can instantiate our trapdoor permutation with a PRIV-secure deterministic encryption function. We prove this with the following theorem:

**Theorem 2 (PRIV-security implies generalized correlation robustness).**

*Let  $E$  be a PRIV-secure deterministic encryption function with key space  $\mathcal{K} = \mathcal{PK} \times \mathcal{SK}$ , plaintext space  $\mathcal{P}$  and ciphertext space  $\mathcal{C}$ . Let  $f = (f_0, f_1, f_2, f_3)$  be an invertible linear function, consisting of the four functions  $f_0, f_1, f_2, f_3 : \text{Dom}(E) \rightarrow \text{Dom}(E)$ .*

*Then the trapdoor permutation  $E_\iota : \mathcal{P} \rightarrow \mathcal{C}; m \mapsto E(\iota, m)$  is correlation robust with respect to  $(f_0, f_1, f_2, f_3)$  for all  $\iota \in \mathcal{PK}$ .*

*Proof.* We show that for each  $A^{corr}$  who breaks correlation robustness with non-negligible advantage  $p$ , there exists a PRIV-Adversary  $\mathcal{A}$  who breaks PRIV-security with advantage  $\frac{1}{2}p$ .

We first construct an adversary with non-empty state  $st$ . Let us consider an adversary  $A^{corr}$ , who can break correlation robustness of  $E_\iota$  with non-negligible advantage  $p$ . The adversary  $\mathcal{A} = (A_t, A_m, A_g)$  uses  $A^{corr}$  to break PRIV-security as follows: First, adversary  $A_t$  asks  $A^{corr}$  to output a tuple  $(K, L)$ , and sets  $st := (K, L)$ . Then,  $A_m$  chooses  $K', L', R_1, R_2$  and  $R_3$  uniformly at random. In the correlation robustness experiment, there are four ways for the challenger to pick  $a, b, c$  out of  $\{0, 1, 2, 3\}$  with  $a < b < c$ . Adversary  $A_m$  chooses such  $(a, b, c)$  uniformly at random, and sets  $x_0 := (f_a(K, L'), f_b(K', L), f_c(K', L'))$ . He sets  $x_1 := (R_1, R_2, R_3)$ . Since  $K'$  and  $L'$  are chosen uniformly at random, the plaintexts chosen by  $A_m$  all have negligible probability of occurring, and therefore the desired min-entropy. The plaintext vector elements  $K + L', K' + L$  and  $K' + L'$  of  $x_0$  are all mutually different with overwhelming probability, so  $x_0$  and  $x_1$  have the same equality pattern.

$A_g$  then obtains a public key  $pk$  and a ciphertext  $c = (c_1, c_2, c_3)$ .

If  $\tilde{b} = 0$ ,  $c$  has the form of a challenge in  $Exp_{E, f_0, f_1, f_2, f_3, \mathcal{A}}^{corr}(k)$  if  $\beta = 0$  in this experiment, while in the case  $\tilde{b} = 1$ ,  $c$  has the form of a challenge for  $\beta = 1$ . Therefore,  $A_g$  can feed the adversary  $A^{corr}$  with an appropriate challenge by handing him  $(pk, c)$ . Adversary  $A_g$  then outputs whatever  $A^{corr}$  outputs.

Now we need to modify  $\mathcal{A}$  such that  $st$  is the empty string. Without loss of generality, we can divide the random coins of  $A^{corr}$  into  $(r_1, r_2)$ , such that  $r_1$  is the randomness used to choose  $(K, L)$ . Then for each choice of  $r_1$ , there is an adversary  $A_{r_1}^{corr}$  which has  $r_1$  hard-wired and always chooses a specific  $(K_{r_1}, L_{r_1})$ . From an average argument, if  $A^{corr}$  breaks correlation-robustness with advantage  $p$ , then there is random coin  $r_p$  for which the adversary  $A_{r_p}^{corr}$  with hard-wired  $r_p$  breaks correlation-robustness with advantage  $p$ .

Consider the PRIV-Adversary  $\mathcal{A}^{r_p} = (A_t^{r_p}, A_m^{r_p}, A_g^{r_p})$  which has his random coins hard-wired, such that (1)  $A_t$  always chooses  $st$  to be the empty string, (2)  $A_m$  always chooses  $K = K_{r_p}$  and  $L = L_{r_p}$ , randomly chooses  $a, b, c \in \{1, 2, 3, 4\}$  with  $a < b < c$ , and sets  $x_0 := (f_a(K, L'), f_b(K', L), f_c(K', L'))$ , and (3) Adversary  $A_g^{r_p}$  internally executes  $A_{r_p}^{corr}$  and outputs whatever  $A_{r_p}^{corr}$  outputs. Since  $A_m^{r_p}$  has  $K$  and  $L$  hard wired,  $A_g^{r_p}$  is the only adversary who communicates with  $A_{r_p}^{corr}$ .

If  $A_{r_p}^{corr}$  breaks correlation robustness with advantage  $p$ , then  $\mathcal{A}^{r_p}$  breaks PRIV-security with advantage at least  $\frac{1}{4}p$ .  $\square$

We can instantiate our correlation robust trapdoor permutation  $E : \{0, 1\}^k \rightarrow \{0, 1\}^k$  with a PRIV-secure deterministic encryption scheme. In [6], Bellare et al. introduce a PRIV-secure length-preserving scheme *RSA-DOAEP*, which works on bitstrings. This scheme can be used to instantiate our correlation robust trapdoor permutation.

## 7 Acknowledgements

We thank the anonymous reviewers for helpful comments.

## References

1. abhi shelat and C. Shen. Two-output secure computation with malicious adversaries. In *EUROCRYPT*, pages 386–405, 2011.
2. B. Applebaum. Garbling XOR gates "for free" in the standard model. In *TCC*, pages 162–181, 2013.
3. B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in  $NC^0$ . In *FOCS 2004*, pages 166–175, 2004.
4. B. Applebaum, Y. Ishai, and E. Kushilevitz. Computationally private randomizing polynomials and their applications. In *Computational Complexity 15(2)*, pages 115–162, 2006.
5. B. Applebaum, Y. Ishai, E. Kushilevitz, and B. Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In *CRYPTO (2) 2013*, pages 166–184, 2013.
6. M. Bellare, A. Boldyreva, and A. O’Neill. Deterministic and efficiently searchable encryption. In *Proceedings of the 27th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO’07, pages 535–552, Berlin, Heidelberg, 2007. Springer-Verlag.
7. M. Bellare, M. Fischlin, A. O’Neill, and T. Ristenpart. Deterministic encryption: Definitional equivalences and constructions without random oracles. Cryptology ePrint Archive, Report 2008/267, 2008. <http://eprint.iacr.org/>.
8. M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. Cryptology ePrint Archive, Report 2012/265, 2012. <http://eprint.iacr.org/>.
9. D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, pages 533–556, 2014.

10. L. T. A. N. Brandão. Secure two-party computation with reusable bit-commitments, via a cut-and-choose with forge-and-lose technique - (extended abstract). In *ASIACRYPT*, pages 441–463, 2013.
11. S. G. Choi, J. Katz, R. Kumaresan, and H. Zhou. On the security of the "Free-XOR" technique. In *TCC*, pages 39–53, 2012.
12. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.
13. B. Hemenway, S. Lu, and R. Ostrovsky. Correlated product security from any one-way function. In *PKC*, pages 558–575, 2012.
14. Y. Huang, J. Katz, and D. Evans. Efficient secure two-party computation using symmetric cut-and-choose. In *CRYPTO*, pages 18–35, 2013.
15. Y. Huang, J. Katz, V. Kolesnikov, R. Kumaresan, and A. J. Malozemoff. Amortizing garbled circuits. In *CRYPTO*, pages 458–475, 2014.
16. Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, pages 294–304, 2000.
17. Y. Ishai and E. Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.
18. J. Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.
19. V. Kolesnikov. Gate evaluation secret sharing and secure one-round two-party computation. In *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings*, pages 136–155, 2005.
20. V. Kolesnikov and R. Kumaresan. Improved secure two-party computation via information-theoretic garbled circuits. In *SCN*, pages 205–221, 2012.
21. V. Kolesnikov, P. Mohassel, and M. Rosulek. FleXOR: Flexible garbling for xor gates that beats free-xor. In J. Garay and R. Gennaro, editors, *Advances in Cryptology CRYPTO 2014*, volume 8617 of *Lecture Notes in Computer Science*, pages 440–457. Springer Berlin Heidelberg, 2014.
22. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP*, pages 486–498, 2008.
23. Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO*, pages 1–17, 2013.
24. Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In M. Naor, editor, *Advances in Cryptology - EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78. Springer Berlin Heidelberg, 2007.
25. Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.
26. Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In *TCC*, pages 329–346, 2011.
27. Y. Lindell and B. Riva. Cut-and-choose yao-based secure computation in the online/offline and batch settings. In *CRYPTO*, pages 476–494, 2014.
28. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.
29. M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *EC*, pages 129–139, 1999.
30. B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In M. Matsui, editor, *Advances in Cryptology ASIACRYPT 2009*,

- volume 5912 of *Lecture Notes in Computer Science*, pages 250–267. Springer Berlin Heidelberg, 2009.
31. A. Rosen and G. Segev. Chosen-ciphertext security via correlated products. In *TCC*, pages 419–436, 2009.
  32. T. Sander, A. L. Young, and M. Yung. Non-interactive cryptocomputing for  $nc^1$ . In *FOCS*, pages 554–567, 1999.
  33. A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.
  34. S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *EUROCRYPT*, 2015.