How to Circumvent the Two-Ciphertext Lower Bound for Linear Garbling Schemes

Carmen Kempka, Ryo Kikuchi, and Koutarou Suzuki

NTT Secure Platform Laboratories

Abstract. At EUROCRYPT 2015, Zahur et al. argued that all linear, and thus, efficient, garbling schemes need at least two k-bit elements to garble an AND gate with security parameter k. We show how to circumvent this lower bound, and propose an efficient garbling scheme which requires less than two k-bit elements per AND gate for most circuit layouts. Our construction slightly deviates from the linear garbling model, and constitutes no contradiction to any claims in the lower-bound proof. With our proof of concept construction, we hope to spur new ideas for more practical garbling schemes.
Our construction can directly be applied to semi-private function evaluation by garbling XOR, XNOR, NAND, OR, NOR and AND gates in the same way, and keeping the evaluator oblivious of the gate function.

Keywords: garbled circuits, lower bound on linear garbling schemes, semi-private function evaluation

1 Introduction

Yao's *garbled circuit* technique [29], modeled as a stand-alone primitive by Bellare et al. [4], is one of the most important techniques to achieve secure two-party computation. In this technique, one of the parties, the *garbler*, creates an encrypted form of a circuit, a so-called *garbled circuit*, which the other party, the *evaluator*, can evaluate without being able to learn anything other than the output of the computed function. Malkhi et al. demonstrated practical feasibility of Yao's technique with their implementation Fairplay [21].

Continued research on Yao's technique has improved its efficiency in terms of computational as well as communication cost. After Yao's original proposal, which needed four ciphertexts to garble a single gate, several techniques have been proposed which reduce the number of ciphertexts in a garbled circuit. The most important works achieve a reduction to a factor roughly between 0.25 and 0.75. Naor et al. [22] pointed out that the number of ciphertexts needed per gate can be reduced from four to three, by setting one of them to the all-zero string. Kolesnikov and Schneider [15] showed how to garble XOR gates "for free", by setting their output keys to be the XOR of their input keys. Pinkas et al. [27] use polynomial interpolation to garble gates with only two ciphertexts per gate. Their technique is not compatible with the free XOR technique.

Recently, Zahur et al. [30] observed that all of these garbling schemes mentioned above share a structure which they model as *linear garbling schemes*. Basically, garbler and evaluator use only XOR operations in the field $GF(2^k)$, and calls to a random oracle, to process the circuit. Zahur et al. showed that garbling an AND gate in this linear structure requires at least two ciphertexts. They further proposed a garbling scheme, the *half gate* construction, which matches this lower bound, and is compatible with the free XOR technique. They concluded that to require less ciphetexts, one needs to employ non-linear, and thus, presumably inefficient techniques. This gives the impression that the optimum we can achieve concerning communication cost in the semi-honest case has already been reached. In this work, we show that this is not necessarily the case.

Our Contribution: We propose an efficient garbling scheme which requires strictly less than two k-bit ciphertexts per AND gate. Our construction is easy to understand and implement, its computational cost comparable to existing practical schemes. Evaluation looks the same for XOR, XNOR, AND, NAND, OR, and NOR gates, so our technique can be applied to secure function evaluation of semi-private functions (SPF-SFE) [25], where the evaluator knows the circuit topology, but not the gate functions. If the positions of XOR gates are known, the number of ciphertext can be further reduced. We prove that our garbling scheme achieves simulation-based privacy [4] in the random oracle model.

Our construction requires only a single k-bit ciphertext for AND gates of which at least one input wire is a circuit input wire. This already seems contradictory to the lower bound, which considers a single AND gate, rather than a whole circuit. All other (inner) AND gates need one additional k-bit ciphertext for adjustment. Thus, general circuits require $1 \leq s < 2$ k-bit ciphertexts[1] per AND gate. In circuits with fan-out one, at least half of the gates are input gates, so we require $1 \leq s \leq 1.5$ ciphertexts per gate. Even though, we do not break the lower bound. We circumvent it by slightly deviating from the linear garbling model, and we do need $5 > 2$ ciphertexts for circuit input gates, and $6 > 2$ ciphertexts for inner gates. But four of them have the length of merely 2 bit.

We demonstrate how we circumvent the lower bound, and hope that our observations sow new ideas for further improvement. We further show that there is at least one other garbling scheme which circumvents the lower bound in a very similar way: a secret-sharing based construction introduced by Kolesnikov in 2005 [13] garbles AND gates with zero ciphertexts. Kolesnikov's technique produces a large blow-up of the input key size, and is impractical for large circuits. It is nonetheless interesting to look at in order to find directions for more efficient constructions.

Idea of our Construction: The linear garbling model performs all operations in $GF(2^k)$. It allows only XOR operations (denoted by $\oplus$) and random oracle calls. In contrast, we also use $(\mathbb{Z}_{2^k}, +)$, where $+$ denotes standard addition, in cases where we need $d + d \neq 0$ for some value d.

Consider a hash function H and an AND gate with input wires A and B, to which we want to assign input wire labels K_A^0, K_A^1 and K_B^0, K_B^1, respectively,

[1] The case $s = 2$ can only happen in circuits which have no input.

as well as output wire labels K_i^0 and K_i^1. We exploit a similar relation as the free XOR technique, but in $\mathbb{Z}_{2^k}$ rather than $GF(2^k)$: if $K_A^1 = K_A^0 + d$ and $K_B^1 = K_B^0 + d$ for some $d \in \mathbb{Z}_{2^k}$, we have

$$K_A^0 + K_B^1 = K_A^1 + K_B^0 = K_A^0 + K_B^0 + d.$$

The garbler can then set the output wire label K_i^0 to either

$$K_i^0 := H(K_A^0 + K_B^0) \quad \text{or} \quad K_i^0 := H(K_A^0 + K_B^0 + d),$$

each with probability $\frac{1}{2}$, and include the single ciphertext

$$G := H(K_A^0 + K_B^0) \oplus H(K_A^0 + K_B^0 + d)$$

in the garbled circuit. If we further set

$$K_i^1 := H(K_A^0 + K_B^0 + 2d) = H(K_A^1 + K_B^1),$$

the evaluator simply needs to hash his input keys, and XOR the hash value with the ciphertext G if necessary.

Obviously, this construction is not yet secure, since the ciphertext is never used if the gate's output truth value is 1. Therefore, in the case of input $(1, 1)$, with probability $\frac{1}{2}$, we just let the evaluator use the ciphertext anyway, by setting

$$K_i^1 = H(K_A^0 + K_B^0 + 2d) \oplus b_1 G$$

for a random bit $b_1 \in \{0, 1\}$. This way, we need to provide only a single k-bit ciphertext G for security parameter k. The evaluator needs to use G with probability $\frac{1}{2}$ in any case, and learns nothing about the actual input.

Additionally, we need four 2-bit ciphertexts[2] to communicate whether the k-bit ciphertext is to be used or not. Also, the difference d is not preserved for the output wire labels, so for inner gates, we need one additional k-bit ciphertext for adjustment. For the same reason, our construction is not compatible with free XOR. However, XOR gates of which at least one input wire does not depend on the output of an AND gate, can use the free XOR technique and need 0 ciphertexts, while inner XOR gates can be garbled with only one k-bit ciphertext.

How we bypass the lower bound: In all known linear garbling schemes, the operation the evaluator needs to perform, for example, which ciphertext to use, depends on wire-specific permute bits. Changing even one permute bit assigns a differrent operation to the output truth value 1. The lower-bound proof strongly depends on this fact, and on the assumption that all ciphertexts are elements of $GF(2^k)$. However, 2-bit values can be masked with 2-bit ciphertexts[3].

Our scheme can be divided into a k-bit part *not dependent* on any permute bits, and a 2-bit part which depends on permute bits. For the k-bit part, the *same*

[2] One bit in each ciphertexts contains the actual choice bit whether to use the ciphertext, and the other contains the color bit of the output label.

[3] More precisely, these ciphertexts need only 2 bits of entropy, and can be represented with a bitstring of length 2.

operation using the *same* ciphertext might be performed by the evaluator for two different inputs, which might even lead to different output truth values. Thus, arguments of the lower-bound proof do not apply[4] to our k-bit part. However, we communicate which operation to perform via several 2-bit ciphertexts, which depend on permute bits in the standard way, and for which all arguments in the lower-bound proof hold — we do need more than two of them per AND gate.

Some Remarks: Our construction offers significant improvements for semi-private functions, where the gate function needs to be hidden and free XOR cannot be used anyway. If the gate functions are known to the evaluator, whether our construction actually performs better than the half gate construction strongly depends on the circuit layout. It might offer significant improvement for circuits with fan-out one consisting mostly of odd gates like AND, NAND, OR and NOR. However, for most interesting circuits, the actual practical improvement might be insignificant or non-existent in the non-semi-private case.

One could argue that, since each circuit input is known either by the garbler or by the evaluator, all input gates can be garbled as half gates, which require only one ciphertext. This would make the half gate construction [30] strictly better than ours in the case of known gate functions. However, this approach has several problems. When used with the cut and choose technique, check circuits would reveal part of the garbler's input if generator half gates on input level are opened. In addition, inputs need to be known at the time of garbling, which makes this approach incompatible with reactive garbling [24], and prevents proponong the garbling process to an offline phase. Compliance with simulation based privacy is unclear, since the simulator does not know the inputs. In addition, this approach seems to contradict the lower bound introduced in the same paper. Nontheless, we introduce an optimization in Appendix A, which combines our scheme with this idea, such that the first *two* gate levels require only one k-bit ciphertexts per AND gate for fortunate circuit layouts.

Other Related Work: There are at least two garbling schemes [12, 13] which do not need to communicate any k-bit ciphertexts at all, if the garbled circuit has fan-out one, by garbling the circuit backwards from output gate to input gates. Both schemes produce larger input keys, and when garbling general circuits, require additional ciphertexts. One is the information theoretically secure construction by Kolesnikov [13]. Output keys are secret-shared into the input keys, and no ciphertexts are required at all. However, the secret sharing produces a blow-up in the input key size which is quadratic in the circuit depth. The other, introduced by Kempka et al. [12], creates ciphertexts by hashing public data, sparing the need to communicate them. Fitting decryption keys are then determined by the garbler, who uses a secret trapdoor to invert the ciphertexts with an inverse trapdoor one-way permutation. Due to the asymmetric primitive, the construction requires a larger security parameter.

Huang et al. [8] garble AND gates as generator half gates with one ciphertext, to realize a permutation network. Paus et al. [25] eliminate constant inputs to

[4] As we will see, this is also the case in Kolesnikov's scheme [13], where permute bits are only assigned to one input wire per gate.

reduce the circuit size. Both techniques might be used before applying ours. However, the benefits do not necessarily add up, because they reduce the number of input wires. Compliance with simulation-based privacy [4] is unclear since the simulator does not know the garbler's input.

Secure function evaluation is called *semi-private* (SPF-SFE), if the topology of the circuit is known to the evaluator, but the gate functions are kept secret. As pointed out by Paus et al. [25], Yao's original construction [29] already hides the gate function, and can directly be used for SPF-SFE. The same holds for the three-row reduction (GRR3) [22]. Both constructions allow using free XOR in circuit parts which are known to the evaluator. Paus et al. implement circuits with privately programmable blocks by garbling several functions (sub-circuits) with Yao's construction and multiplexing their output. Their construction can easily be combined with our technique, giving up free XOR for non-private parts, but reducing the garbled circuit size significantly for the private part of the circuit. One limitation here is that we cannot realize left-or-right wire choosing (multiplexing), or constant gates within a single gate. Therefore, the multiplexer-subcircuit by Paus et al. still needs to be realized using Yao[5] circuits. The half gate approach [30] hides which odd gate (AND, NAND, OR, NOR) is evaluated. However, the positions of XOR gates need to be known to the evaluator. The same holds for the GRR2-techniques of Pinkas et al. [26] and Gueron et al. [7]. SPF-SFE is also covered by works on private function evaluation, which additionally hide the circuit topology. Naturally, hiding the topology comes with a lager overhead in the circuit size. Constructions using universal circuits require $O(l \cdot log(l))$ [28] or $O(l \cdot log^2(l))$ [16] additional gates, where l is the number of gates of the original circuit. The LEGO-like construction of Katz and Malka [11] produces less overhead, but requires asymmetric primitives, in particular, one-time homomorphic encryption.

Another line of research focuses on security against malicious adversaries [19, 1, 17, 9, 5, 10, 20, 23, 6]. This work focuses on the semi-honest case.

2 Preliminaries

2.1 Notation

We use the following notations. By $x \xleftarrow{U} X$, we denote that x is randomly selected from the set X according to uniform distribution, $x \leftarrow \mathsf{Algo}$ denotes that x is the output of a probabilistic algorithm Algo, $A := B$ denotes that A is defined by B, and $[S]_x$ denotes the x-th bit of bitstring S. Our security parameter is k.

2.2 Garbling Scheme

In this section, we recall the definition of garbling schemes and the notion of *simulation-based privacy* of Bellare et al. [4].

[5] It is easy to see that we can use Yao's garbling technique, GRR3 and our technique in the same circuit, and even adjust the difference of output wire labels in gates garbled with Yao's technique or GRR3 on the way.

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 $W = \{1, ..., n + l\}$ be the set of all wires, $W_{input} = \{1, ..., n\}$ the set of circuit input wires, $W_{output} = \{n+l-m+1, ..., n+l\}$ the set of circuit output wires, and $W_{gate} = \{n+1, ..., n+l\}$ the set of gates (and their output wires). The functions $A : W_{gate} \to W \setminus W_{output}$ and $B : W_{gate} \to W \setminus W_{output}$ specifiy the first input wire $A(i)$ and the second input wire $B(i)$ of each gate i, respectively. We require $A(i) < B(i) < i$ for all $i \in W_{gate}$. The function $g : W_{gate} \times \{0,1\}^2 \to \{0,1\}$ specifies the gate function $g(i, \cdot, \cdot) = g_i(\cdot, \cdot)$ of each gate i. We leave out the parameter i if it is clear from context. 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* $\mathsf{GC} = (\mathsf{Garble}, \mathsf{Encode}, \mathsf{Eval}, \mathsf{Decode})$ *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.,* $(F, e, d) \leftarrow \mathsf{Garble}(1^k, f)$.
- Encode *takes as input encoding information e and circuit input $x \in \{0,1\}^n$, and outputs garbled input X, i.e.,* $X \leftarrow \mathsf{Encode}(e, x)$.
- Eval *takes as input garbled circuit F and garbled input X, and outputs garbled output Y, i.e.,* $Y \leftarrow \mathsf{Eval}(F, X)$
- Decode *takes as input decoding information d and garbled output Y, and outputs circuit output y, i.e.,* $y \leftarrow \mathsf{Decode}(d, Y)$.

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

We then define *simulation-based privacy* of garbling schemes as follows. We adapt the notion of Bellare et al. [4] 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 $\Phi_{size}(f) = (n, m, l)$, topology $\Phi_{topo}(f) = (n, m, l, A, B)$, or the entire information $\Phi_{circ}(f) = (n, m, l, A, B, g)$ of circuit $f = (n, m, l, A, B, g)$.

Definition 2 (Simulation-based Privacy). *For a garbling scheme* $\mathsf{GC} = (\mathsf{Garble}, \mathsf{Encode}, \mathsf{Eval}, \mathsf{Decode})$, *function $f \in \mathcal{F}_n$, input values $x \in \{0,1\}^n$, simulator* Sim, *and random oracle H, the advantage of the adversary $\mathcal{A}$ is defined as* $Adv^{\mathrm{prv.sim}}_{\mathsf{GC},\mathsf{Sim},\Phi,\mathcal{A}}(k) :=$

$$|\Pr[s \leftarrow \mathcal{A}^H(1^k), (F, e, d) \leftarrow \mathsf{Garble}(1^k, f), X \leftarrow \mathsf{Encode}(e, x) : \mathcal{A}^H(s, F, X, d) = 1]$$

$$- \Pr[s \leftarrow \mathcal{A}^H(1^k), (F, X, d) \leftarrow \mathsf{Sim}(1^k, \Phi(f), f(x)) : \mathcal{A}^H(s, F, X, d) = 1]|.$$

A garbling scheme $\mathsf{GC} = (\mathsf{Garble}, \mathsf{Encode}, \mathsf{Eval}, \mathsf{Decode})$ *is* private*, if there exists a probabilistic polynomial-time simulator* 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* $Adv^{\mathrm{prv.sim}}_{\mathsf{GC},\mathsf{Sim},\Phi,\mathcal{A}}(k)$ *is negligible.*

3 A Garbling Scheme which Circumvents the Lower Bound

We first describe our basic garbling scheme considering only AND gates, in the semi-honest model. In Section 3.2, we describe how to garble other gate types, and application to semi-private functions. Our scheme is not compatible with free XOR, but Section 3.3 shows that we can garble XOR gates with 0 or 1 k-bit ciphertexts, and sometimes even inner AND gates can be garbled with 1 k-bit ciphertext. Section 3.4 briefly discusses the malicious case. We estimate efficiency in Section 4, and prove that our scheme achieves simulation-based privacy as defined by Bellare et al. [4] in the semi-honest setting in Section 5.

3.1 Our Construction

We use the following notation. Let k be our security parameter. With the $+$ symbol, we denote addition in $\mathbb{Z}_{2^k}$. The operation $\oplus$ performs a bitwise XOR on bitstrings. Elements in $\mathbb{Z}_{2^k}$ are interpreted as bitstrings when used with the $\oplus$ operation. The function $\mathsf{lsb}(x)$ returns the least significant bit of its input x, and the function $\mathsf{lsb}_2(x)$ returns the two least significant bits of x.

We assign to each wire i two labels $K_i^0, K_i^1 \in \mathbb{Z}_{2^k}$, where K_i^b represents the truth value $b \in \{0,1\}$ on that wire. To each wire i, we assign a random permute bit λ_i known only to the garbler. Each wire label K_i^b has assigned a bit $c_i^b = \lambda_i \oplus b$, which we call the *color bit* or the *color* of a wire label, in the style of previous work, and to avoid confusion with other choice bits which we describe below. So far, this is no different from most existing garbling schemes. However, jumping ahead, to circumvent the lower bound, the actual operation to compute a gate's output label needs to be somewhat detached from the color bits and the permute bits. To achieve this, we use three additional kinds of choice bits. Their exact role, and their relations among each other as well as to the permute and color bits, will become clear in the scheme description. We provide a brief overview here. In the garbling process, the garbler chooses two random bits b_0 and b_1 for each gate. These bits define by which operation the gate's output labels are computed. The bits b_0 and b_1 are independent of all color bits c_i^b and permute bits λ_i. They need to remain secret, but define a single choice bit $\gamma^{(a,b)}$ for each gate input $(a,b) \in \{0,1\}^2$. The appropriate $\gamma^{(a,b)}$ needs to be communicated to the evaluator. We use the color bits c_i^a, c_i^b of the gate's input wires to point to the correct encryption of the corresponding choice bit $\gamma \in \{\gamma^{(a,b)}\}_{(a,b)\in\{0,1\}^2}$, which then points to the correct operation to compute the gate's output label.

Our garbling algorithm is described in Figure 1. Encoding of inputs and evaluation are described in Figure 2 and 3. Decoding consists of XORing the

color bits of the circuit output wires with the corresponding permute bits, as specified in Figure 4. To prevent attacks similar to the one described by Bellare et al. [3], we include a second parameter in our hash function H: a unique tweak j, incremented before each (evaluator's) call to the hash function. This is also done in the half gate construction for similar reasons. We denote this in the same way, using a stateful procedure $\mathsf{nextindex}()$, which increments an internal counter and returns it. For the sake of readability, we leave out this tweak in the following informal description of our garbling scheme.

Let i be an AND gate with input wires A and B. Similar to the free XOR technique, we exploit commutativity of the $+$ operation: if $K_A^1 = K_A^0 + d_i$ and $K_B^1 = K_B^0 + d_i$, we have

$$K_A^0 + K_B^1 = K_A^1 + K_B^0 = K_A^0 + K_B^0 + d_i.$$

We further arrange the output wire labels to be either the hash of the input keys, or the k-bit ciphertext included in the garbled gate XOR this hash value.

In more detail, to garble an AND gate, the garbler chooses two random bits $b_0, b_1 \in \{0,1\}$, sets the output wire label K_i^0 assigned to truth value 0 to

$$K_i^0 := H(K_A^0 + K_B^0 + b_0 d_i),$$

and includes in the garbled circuit the *single* ciphertext

$$G := H(K_A^0 + K_B^0) \oplus H(K_A^0 + K_B^0 + d_i).$$

The garbler further sets the output wire label K_i^1 assigned to truth value 1 to

$$K_i^1 := H(K_A^0 + K_B^0 + 2d_i) \oplus b_1 G = H(K_A^1 + K_B^1) \oplus b_1 G.$$

To evaluate an AND gate, given the input wire labels K_A and K_B, the evaluator needs to compute either $H(K_A + K_B)$ or $H(K_A + K_B) \oplus G$. We let the evaluator know whether he needs to use the ciphertext G via a choice bit γ, which he can compute using his input keys as described below. The choice bit γ does not reveal any information about the input, since for any input combination $(a,b) \in \{0,1\}^2$, the evaluator needs to use the ciphertext with probability $\frac{1}{2}$.

Before we continue our description, let us point out that so far, we have not used any permute bits or color bits. In fact, whether the evaluator needs to use the ciphertext G, only depends on the input, and the bits b_0 and b_1, which are independent of any wire-specific permute bits. This fact plays an important role in circumventing the lower bound on garbling schemes [30]. Details on this can be found in Section 6. Arguments in the lower-bound proof show us that to circumvent the lower bound, we need to avoid a direct dependency between permute bits assigned to the input wires and the choice bit γ, which implies that γ cannot be computed by the evaluator as a function of the color bits. Instead, we include in the garbled circuit the four 1-bit ciphertexts

$$b_{(a,b)}^{\gamma} := \mathsf{lsb}(H(K_A^a || K_B^b)) \oplus \gamma^{(a,b)},$$

which encrypt the correct choice bit $\gamma^{(a,b)}$ for each possible input combination $(a,b) \in \{0,1\}^2$. The choice bits $\gamma^{(a,b)}$ only depend on b_0 and b_1, we have

$$\gamma^{(0,0)} = b_0, \gamma^{(0,1)} = \gamma^{(1,0)} = 1 - b_0, \gamma^{(1,1)} = b_1.$$

However, we order the four ciphertext $b^{\gamma}_{(a,b)}$ according to the permute bits λ_A and λ_B of the input wires, so the evaluator can choose the correct ciphertext using his color bits $c^a_A = \lambda_A \oplus a$ and $c^b_B = \lambda_B \oplus b$ as usual.

We still need to describe how the evaluator learns the color bits. For the circuit input wires, we can use the least significant bit of the input wire labels as usual. However, we have little freedom in choosing output wire labels, and thus cannot guarantee their least significant bits to be different. Instead, as for γ, we include in the garbled circuit four additional 1-bit ciphertexts,

$$b^c_{(a,b)} := \mathsf{lsb}(H(K^a_A || K^b_B)) \oplus g_i(a,b) \oplus \lambda_i,$$

among which the evaluator chooses using the color bits of the input wire labels, so together with the four ciphertexts encrypting γ, we would have eight 1-bit ciphertexts in total. To reduce the number of oracle calls, we use the two least significant bits of the hash output, denoted by $\mathsf{lsb}_2(H(.))$, and create the four 2-bit ciphertexts

$$b^c_{(a,b)} || b^{\gamma}_{(a,b)} := \mathsf{lsb}_2(H(K^a_A || K^b_B)) \oplus ((g_i(a,b) \oplus \lambda_i) || \gamma^{(a,b)})$$

instead. This way we avoid having to evaluate the hash function twice on the same input values but with different tweaks.

Unfortunately, we cannot have a global difference d such that $K^1_i = K^0_i + d$ for each wire i. Since the labels of circuit input wires can be chosen freely, they can be given the same difference. However, this difference is not preserved and cannot be controlled in non-input wires. In the next circuit level, gate i's input wires A and B will thus have wire labels (K^0_A, K^1_A) and (K^0_B, K^1_B) with $K^1_A - K^0_A \neq K^1_B - K^0_B$ with high probability. We provide one additional ciphertext to adjust the difference: let λ_B the permute bit on wire B, and the difference d' used for this gate $d' := K^1_A - K^0_A$. Then we set $K^{\lambda_B}_{B'} := K^{\lambda_B}_B$, $K^{1-\lambda_B}_{B'} := K^{\lambda_B}_{B'} + d'$ and include a second k-bit ciphertext $E := K^{1-\lambda_B}_{B'} + K^{1-\lambda_B}_B$ in the garbled circuit. This is why we need two ciphertexts for inner[6] AND gates. The complete garbling algorithm is described in Figure 1. For better readability, we only describe AND gates in the main algorithm. A discussion about arbitrary gates and semi-private function evaluation can be found in Section 3.2.

We cannot use a field with characteristic two to compute addition, since we require $2d \neq 0$ for all differences d occurring in the garbled circuit. Therefore, we perform addition in $\mathbb{Z}_{2^k}$, which gives us a small error probability: there is one element $d_0 \in \mathbb{Z}_{2^k}$ with order 2. Since $K + 2d_0 = K$ for all $K \in \mathbb{Z}_{2^k}$, garbling a

[6] The situation changes when an inner AND gate has only XOR gates as predecessors. In this case, we can use the freedom of key choices in the XOR gates to adjust the difference of the AND gate's input keys.

Garbling algorithm $\mathsf{Garble}(1^k, f)$

Input: Security parameter k, Circuit $f = (n, m, l, A, B, g)$
Algorithm: 1. Initialize empty arrays $F[]$, $e[]$, $d[]$ with $|F| = l$, $|e| = n$ and $|d| = m$.
2. Garbling the gates: For $i := n + 1$ to $l + n$ do:
(a) Set $A := A(i)$ and $B := B(i)$
(b) If undefined, choose permute bits $\lambda_A, \lambda_B \in \{0, 1\}$ at random.
For all $(a, b) \in \{0, 1\}^2$, if undefined, set $c_A^a := \lambda_A \oplus a$, $c_B^b := \lambda_B \oplus b$.
(c) Input keys:
– If A's and B's labels are defined:
Set $d_i := K_A^1 - K_A^0$,
$K_{B'}^{\lambda_B} := K_B^{\lambda_B}$, and
$K_{B'}^{1-\lambda_B} := K_{B'}^{\lambda_B} + (-1)^{\lambda_B} d_i$.
Set $j_E := \mathsf{nextindex}()$,
$E := H(K_B^{1-\lambda_B}, j_E) \oplus K_{B'}^{1-\lambda_B}$.
– If A's labels are defined and B's labels are undefined (vice versa analog), set $d_i := K_A^1 - K_A^0$, choose K_B^0 at random and set $K_B^1 := K_B^0 + d_i$, $K_{B'}^0 := K_B^0$, $K_{B'}^1 := K_B^1$.
– If A's and B's labels are undefined,
choose K_A^0, K_B^0 and d_i at random and
set $K_A^1 := K_A^0 + d_i$, $K_B^1 := K_B^0 + d_i$, $K_{B'}^0 := K_B^0$, $K_{B'}^1 := K_B^1$.
(d) $\mathsf{GarbleAND}$:
Set $j_L := \mathsf{nextindex}()$.
Set $G := H(K_A^0 + K_{B'}^0, j_L) \oplus H(K_A^0 + K_{B'}^0 + d_i, j_L)$.
Choose random bits $b_0, b_1 \in \{0, 1\}$.
Set $K_i^0 := H(K_A^0 + K_{B'}^0 + b_0 d_i, j_L)$.
Set $K_i^1 := H(K_A^0 + K_{B'}^0 + 2d_i, j_L) \oplus b_1 G$.
Set $\gamma^{(0,0)} := b_0$, $\gamma^{(0,1)} := \gamma^{(1,0)} := 1 - b_0$, $\gamma^{(1,1)} := b_1$.
(e) Encrypt choice bits $\gamma^{(a,b)}$ and color bits of output wire:
Set $j_{c,\gamma} := \mathsf{nextindex}()$.
Choose random permute bit $\lambda_i \in \{0, 1\}$.
For all $(a, b) \in \{0, 1\}^2$,
$b^{c,\gamma}_{2c_A^a + c_B^b} := \mathsf{lsb}_2(H(K_A^a || K_{B'}^b, j_{c,\gamma})) \oplus ((g_i(a, b) \oplus \lambda_i) || \gamma^{(a,b)})$,
(f) Set $F[i] := (b_0^{c,\gamma}, b_1^{c,\gamma}, b_2^{c,\gamma}, b_3^{c,\gamma}, G, E)$, if E is defined.
Set $F[i] := (b_0^{c,\gamma}, b_1^{c,\gamma}, b_2^{c,\gamma}, b_3^{c,\gamma}, G)$, otherwise.
(g) If $j \in \{A, B\}$ is a circuit input wire ($j \in W_{input}$),
set $e[j] := (K_j^0 || c_j^0, K_j^1 || c_j^1)$.
If $i \in W_{output}$, set $d[i - (n + l) + m] := \lambda_i$.

Output: Garbled circuit F, encoding e, decoding $d = (\lambda_{n+l-m+1}, \ldots, \lambda_{n+l})$

Fig. 1. The proposed garbling algorithm.

Encoding algorithm $\mathsf{Encode}(e, x)$

Inputs: Garbled input keys e, input x
Algorithm: Parse x to $x = x_1 \ldots x_n$
For $i = 1$ to n do:
Parse $e[i] = (e_0, e_1)$
$X[i] := e_{x_i}$
Return X

Fig. 2. The function Encode.

gate with input wire labels differing by d_0 produces identical output wire labels for this gate, or labels differing by G. However, the error probability is negligible, and the garbler can detect it and start over with different randomness. We need to take care of this in the malicious case, as discussed in Section 3.4.

Evaluation algorithm $\mathsf{Eval}(F, X)$

Inputs: Garbled circuit F, garbled input X
Algorithm: 1. For $j = 1$ to n do
$K_j||c_j := X[j]$
2. Compute wire labels:
For $i := n + 1$ to $l + n$ do
– Set $A := A(i)$ and $B := B(i)$.
– Set $x := 2c_A + c_B$.
– Parse $F[i] = (b_0^{c,\gamma}, b_1^{c,\gamma}, b_2^{c,\gamma}, b_3^{c,\gamma}, G, E)$
– If E is defined:
• Set $j_E :=$ nextindex().
• If $c_B = 0$, set $K_B := E \oplus H(K_B, j_E)$.
– Set $j_L :=$ nextindex(), $j_{c,\gamma} :=$ nextindex().
– Compute $c_i||\gamma := \mathsf{lsb}_2(H(K_A||K_B, j_{c,\gamma})) \oplus b_x^{c,\gamma}$.
– Set $K_i := H(K_A + K_B, j_L) \oplus \gamma G$.
3. Return $Y := (c_{n+l-m+1}, \ldots, c_{n+l})$.

Fig. 3. The evaluation algorithm.

Decoding algorithm $\mathsf{Decode}(d, Y)$

Inputs: Decoding d, evaluation output Y
Algorithm: Parse $Y = (c_1, \ldots, c_m)$
Parse $d = (\lambda_1, \ldots, \lambda_m)$
Return $f(x) := (c_1 \oplus \lambda_1, \ldots, c_m \oplus \lambda_m)$

Fig. 4. The function Decode.

Garbling other gate types

GarbleOR:
Set $j_L :=$ nextindex().
Set $G := H(K_A^0 + K_{B'}^0 + d_i, j_L) \oplus H(K_A^0 + K_{B'}^0 + 2d_i, j_L)$.
Choose random bits $b_0, b_1 \in \{0, 1\}$.
Set $K_i^0 := H(K_A^0 + K_{B'}^0, j_L) \oplus b_1 G$.
Set $K_i^1 := H(K_A^0 + K_{B'}^0 + d_i + b_0 d_i, j_L)$.
Set $\gamma^{(0,0)} := b_1$, $\gamma^{(0,1)} := \gamma^{(1,0)} := b_0$, $\gamma^{(1,1)} := 1 - b_0$.

GarbleXOR:
Set $j_L :=$ nextindex().
Set $G := H(K_A^0 + K_{B'}^0, j_L) \oplus H(K_A^0 + K_{B'}^0 + 2d_i, j_L)$.
Choose random bits $b_0, b_1 \in \{0, 1\}$.
Set $K_i^0 := H(K_A^0 + K_{B'}^0 + b_0 2d_i, j_L)$.
Set $K_i^1 := H(K_A^0 + K_{B'}^0 + d_i, j_L) \oplus b_1 G$.
Set $\gamma^{(0,0)} := 1 - b_0$, $\gamma^{(0,1)} := \gamma^{(1,0)} := b_1$, $\gamma^{(1,1)} := b_0$.

Fig. 5. Garbling OR and XOR gates. Garbling NAND, NOR and XNOR can be done by swapping K_i^0 and K_i^1 in the AND, OR and XOR description, respectively.

3.2 Arbitrary Gates and Semi-Private Function Evaluation

We can garble other odd gates like NAND, OR and NOR, as well as the even gates XOR and XNOR, in a very similar way, by substituting the GarbleAND part (Step (d) in Figure 1) with the appropriate one in Figure 5. Evaluation is the

same as for AND gates, so the evaluator only needs to know the circuit topology. This makes our construction directly applicable to semi-private functions [25]. To our knowledge, the best construction in previous work which garbles XOR and odd gates in the same way is Yao's original construction with GRR3, which needs three ciphertexts per garbled gate, while our construction needs one k-bit ciphertext for each input gate and two k-bit ciphertexts for each inner gate.

The reason we can easily garble odd and even gates in the same way is the shared additive difference d in $(\mathbb{Z}_{2^k}, +)$ of the gate input wires. In most garbling schemes, a function $F(.,.))$ is applied to the two input wire labels to compute the output labels in some way. Often F is a hash function or a key derivation function. The mapping $F(K_A, K_B)) \mapsto K_i$ has a different input/output pattern for odd and even gates in most garbling schemes (see Table 1): leaving out free-XOR, $F(K_A^a, K_B^b))$ usually has a different value for each of the four gate inputs $(a, b) \in \{0, 1\}^2$. In odd gates, three of them are mapped to a value v, and one is mapped to $1-v$, where v depends on the gate type, we call this a $3/1$ pattern. In the even gates XOR and XNOR, the two values $F(K_A^0, K_B^0))$ and $F(K_A^1, K_B^1))$ are mapped to a value v, and the other two to $1 - v$, producing the even $2/2$ pattern. In our construction, $F(K_A^a, K_B^b)) = H(K_A^a + K_B^b)$. We only have the three values $H(K_A^0 + K_B^0)$, $H(K_A^0 + K_B^0 + d)$, and $H(K_A^0 + K_B^0 + 2d)$. In each gate, two of them are mapped to a value v, and one is mapped to $1-v$, creating a $2/1$ pattern for both odd and even gates (see Table 2).

Table 1. Usual output patterns

input	odd gates		even gates
	(N)AND	(N)OR	X(N)OR
$F(K_A^0, K_B^0)$	v	$1-v$	v
$F(K_A^0, K_B^1)$	v	v	$1-v$
$F(K_A^1, K_B^0)$	v	v	$1-v$
$F(K_A^1, K_B^1)$	$1-v$	v	v
Pattern:	3/1	3/1	2/2

Table 2. Output patterns in our construction

input	odd gates		even gates
	(N)AND	(N)OR	X(N)OR
$F(K_A^0 + K_B^0)$	v	$1-v$	v
$F(K_A^0 + K_B^0 + d)$	v	v	$1-v$
$F(K_A^0 + K_B^0 + 2d)$	$1-v$	v	v
Pattern:	2/1	2/1	2/1

3.3 More Efficient Handling of XOR Gates

Our wire labels do not share a global difference Δ with $K_i^1 = K_i^0 \oplus \Delta$ for each wire i. Thus, we cannot use the free XOR technique directly. We can still incorporate its idea in our garbling scheme to save ciphertexts.

Free XOR and 1-Ciphertext-XOR Input XOR gates can be garbled with zero ciphertexts. An XOR gate with only circuit input wires as input can simply be garbled as in the free XOR technique. Now assume an XOR gate i with input wires A and B with labels $K_A^0, K_A^1, K_B^0, K_B^1$, where B is a circuit input wire, and the labels for A are already defined. We can set $\Delta_i := K_A^0 \oplus K_A^1$, choose K_B^0 at random, set $K_B^1 := K_B^0 \oplus \Delta_i$, $K_i^0 := K_A^0 \oplus K_B^0$, and $K_i^1 := K_i^0 \oplus \Delta_i$.

Inner XOR gates can be garbled using one ciphertext to adjust the difference between the input wire labels in the same way as for the AND gates, but in $GF(2^k)$ rather than $\mathbb{Z}_{2^k}$. Alternatively, one could use the FleXOR technique [14] or the technique by Gueron et al. [7] for inner XOR gates.

Backward Construction for Inner Gates with Preceding XOR Gates If all paths of an input wire of an inner gate to circuit input wires consist only of XOR gates, we can sometimes adjust these preceding XOR gates in a backward manner, such that we can garble an inner XOR gate for free, or garble an inner AND gate with one ciphertext as if it were an input gate.

As an example, consider the circuit $w_A := w_1 \oplus w_2$, $w_B := w_3 \oplus w_4$, and $w_O := w_A \wedge w_B$ where w_1, w_2, w_3, w_4 are the circuit input wires, w_A and w_B are the left and right input wires of the AND gate, and w_O is the circuit output wire. Using the following construction, we only need 0, 0, and 1 ciphertexts for the left XOR gate, right XOR gate, and AND gate, respectively.

1. Construct the left XOR gate with 0 ciphertexts as in the usual free XOR technique, using some random difference Δ_0. This defines the labels K_A^0, K_A^1 for the left input wire w_A of the AND gate.
2. Define the additive difference $d := K_A^1 - K_A^0$. Select random K_B^0, set $K_B^1 := K_B^0 + d$ for the right input wire w_B of the AND gate. The AND gate can now be garbled with 1 ciphertext.
3. Define the XOR difference $\Delta := K_B^1 \oplus K_B^0$. Select random K_3^0 and K_4^0, set $K_3^1 := K_3^0 \oplus \Delta$ and $K_4^1 := K_4^0 \oplus \Delta$ for the input wires w_3 and w_4 of the right XOR gate. No ciphertexts are needed for this gate.

Using intelligent difference adjustment like this, we can save adjustment ciphertexts for inner gates.

3.4 Security against Malicious Adversaries

To achieve security against malicious adversaries, we can combine our construction with standard cut and choose [18]. Additional care needs to be taken that a malicious garbler cannot violate correctnes by choosing input wire labels K_A^0, K_A^1 and K_B^0, K_B^1 with a difference d with order 2 in $\mathbb{Z}_{2^k}$. Otherwise, he could set the labels of the output wire to identical values $K_i^0 = K_i^1 := H(K_A^0 + K_B^0) = H(K_A^0 + K_B^0 + 2d)$, or even make the circuit output the same for any input. A standard cut and choose check as in Lindell et al. [18] can prevent this, too.

4 Efficiency

In this efficiency estimation, we focus on the communication cost of our garbling scheme. Computational cost is comparable to existing practical constructions. A comparison of the number of calls to the hash function is listed in Table 3, where we consider plain SFE and handling of XOR gates as described in Section 3.3.

Table 3. Number of oracle calls per gate in plain SFE

Technique	Garbler		Evaluator	
	XOR	AND	XOR	AND
classical [2]	4	4	4	4
row reduction (GRR3) [22]	4	4	1	1
row reduction [26]	4	4	1	1
free XOR + GRR3 [15]	0	4	1	1
fleXOR [14]	$\{0, 2, 4\}$	4	$\{0, 1, 2\}$	1
half gates [30]	0	4	0	2
this work	$\{0, 1\}$	$\{7, 8\}$	$\{0, 1\}$	$\{2, 3\}$

We estimate efficiency in three settings: Plain secure function evaluation (SFE) in which the evaluator knows all gate functions, SPF-SFE in which he only knows the circuit topology, and SFE with semi-private sub-circuits. Garbled odd gates do not differ in size, so it is sufficient to consider AND and XOR gates.

4.1 Efficiency in plain SFE

First, we estimate efficiency assuming the evaluator knows all gate functions. We call a gate with at least one circuit input wire as input wire an *input gate*, and an *inner gate* is a gate which is not an input gate. Let l_{A} denote the number of AND gates, $l_{\mathrm{A,in}}$ the number of AND gates which are input gates, and $l_{\mathrm{A,mid}} = l_{\mathrm{A}} - l_{\mathrm{A,in}}$ the number of inner AND gates. Similarly, l_{X} denotes the number of XOR gates, $l_{\mathrm{X,in}}$ the number of XOR gates which are input gates, and $l_{\mathrm{X,mid}} = l_{\mathrm{X}} - l_{\mathrm{X,in}}$ the number of inner XOR gates. We have $l = l_{\mathrm{A}} + l_{\mathrm{X}} = l_{\mathrm{A,in}} + l_{\mathrm{A,mid}} + l_{\mathrm{X,in}} + l_{\mathrm{X,mid}}$.

We consider handling XOR gates as described in Section 3.3, without the optimization for inner gates preceded by XOR gates. We compare the size of our garbled circuits with several garbling schemes in Table 4. In our construction, an XOR gate requires 0 or 1 k-bit elements, and an AND gate requires 1 or 2 k-bit elements, depending on the gate's position in the circuit. The other constructions use the least significant bits of wire labels to communicate color bits. This reduces security by one bit, so $(k + 1)$-bit elements are needed to achieve the same security parameter k. Our construction requires 8 bits per gate in addition to the k-bit elements, $8l_{\mathrm{A}} + k(l_{\mathrm{A,in}} + 2l_{\mathrm{A,mid}} + l_{\mathrm{X,mid}})$ bits in total.

Table 4. Size of garbled circuit in the plain SFE

technique	k-bit elements/gate		total bits of garbled circuit
	XOR	AND	
classical [2]	4	4	$4(k+1)l$
row reduction (GRR3) [22]	3	3	$3(k+1)l$
row reduction (GRR2) [26]	2	2	$2(k+1)l$
free XOR + GRR3 [15]	0	3	$3(k+1)l_{\mathrm{A}}$
fleXOR [14]	$\{0, 1, 2\}$	2	x *s.t.* $2l_{\mathrm{A}}(k+1) \leq x \leq 2(k+1)l$
half gates [30]	0	2	$2l_{\mathrm{A}}(k+1)$
this work	$\{0, 1\}$	$\{1, 2\}$	$8l_{\mathrm{A}} + k(l_{\mathrm{A,in}} + 2l_{\mathrm{A,mid}} + l_{\mathrm{X,mid}})$

Our construction generates smaller garbled circuits than the half gate construction when $k(l_{\mathrm{A,in}} - l_{\mathrm{X,mid}}) - 8l_{\mathrm{A}} > 0$, i. e. when there are more input AND gates than inner XOR gates. Although our construction circumvents the lower bound and generates smaller garbled circuits in some cases, the half gate construction may still be the most efficient garbling scheme for most realistic circuits in plain SFE.

4.2 Efficiency in SPF-SFE

Second, we consider semi-private functions, where the evaluator is only allowed to learn the circuit topology. We assume that the garbler knows the function before garbling, and circuits consist of AND, NAND, OR, NOR, XOR and XNOR gates[7]. In the SPF-SFE setting, we garble XOR gates according to Fig. 5 to make them indistinguishable from other gate types. Therefore, the size of a gate does not depend on its type. Let l_{in} denote the number of input gates, l_{mid} the number of inner gates, and $l = l_{\mathrm{in}} + l_{\mathrm{mid}}$ the total number of gates. We compare our construction to other garbling schemes compatible with SPF-SFE in Table 5. We omit GRR2, free XOR + GRR3, fleXOR, and half gates since the positions of XOR gates become clear in these constructions[8]. As shown in Table 5, our construction is the most efficient one in this setting.

Table 5. Size of garbled circuit in SPF-SFE

technique	k-bit elements/gate	total bits of garbled circuit
classical [2]	4	$4(k+1)l$
GRR3 [22]	3	$3(k+1)l$
this work	$\{1, 2\}$	$8l + k(l_{\mathrm{in}} + 2l_{\mathrm{mid}})$

4.3 Efficiency in SFE with Semi-Private Sub-Circuits

Finally, we consider an evaluator who knows the gate function in some parts of the circuit, and only the topology in the other parts. Let $l^{(\mathrm{pub})}$ be the number of gates of which the evaluator knows the gate function, and $l^{(\mathrm{prv})}$ be the number of the other, "private" gates. Let $l_{\mathrm{A,in}}^{(\mathrm{pub})}$, $l_{\mathrm{X,in}}^{(\mathrm{pub})}$, and $l_{\mathrm{in}}^{(\mathrm{prv})}$ denote the public/private part of $l_{\mathrm{A,in}}$, $l_{\mathrm{X,in}}$, and l_{in}, respectively.

We observe that GRR3 and the half gate construction can be combined easily. The difference between the half gate construction and ours is $k(l_{\mathrm{A,in}}^{(\mathrm{pub})} - l_{\mathrm{X,mid}}^{(\mathrm{pub})}) -$

[7] Circuits containing multiplexers (for example to realize programmable blocks), or gates with constant output, could use GRR3 only for these gates.

[8] These constructions can be used for a circuit without XOR gates, and any circuit can be changed to that circuit by replacing an XOR gate with two odd gates. However, this replace causes the topology change and requires four ciphertexts for an XOR gate

$8l^{(\mathrm{pub})}$ in the public part. The difference between GRR3 and our technique is $2k(l^{(\mathrm{prv})}+l_{\mathrm{in}}^{(\mathrm{prv})})-5l^{(\mathrm{prv})}$ in the private part. Therefore, our construction generates smaller garbled circuits when $k(l_{\mathrm{A,in}}^{(\mathrm{pub})}+2l^{(\mathrm{prv})}+2l_{\mathrm{in}}^{(\mathrm{prv})}-l_{\mathrm{X,mid}}^{(\mathrm{pub})})-5l-3l^{(\mathrm{pub})}>0$. Which construction is the most efficient depends on how much of the circuit is private, and on the number of inner XOR gates in the public part.

5 Proof of Security

5.1 Correctness

Correctness of our garbling scheme clearly holds. In the case of AND gates, correctness follows from the following equations:

$$((g_i(a,b)\oplus\lambda_i)||\gamma^{(a,b)}) = b^{c,\gamma}_{2c_A^a+c_B^b}\oplus \mathsf{lsb}_2(H(K_A^a||K_{B'}^b,j_{c,\gamma}))$$

and

$$H(K_A^0+K_B^0,j_L)\oplus\gamma^{(0,0)}G = H(K_A^0+K_B^0,j_L)\oplus b_0G = H(K_A^0+K_B^0+b_0d_i,j_L),$$

$$H(K_A^0+K_B^1,j_L)\oplus\gamma^{(0,1)}G = H(K_A^0+K_B^0+d_i,j_L)\oplus(1-b_0)G = H(K_A^0+K_B^0+b_0d_i,j_L),$$

$$H(K_A^1+K_B^0,j_L)\oplus\gamma^{(1,0)}G = H(K_A^0+K_B^0+d_i,j_L)\oplus(1-b_0)G = H(K_A^0+K_B^0+b_0d_i,j_L),$$

$$H(K_A^1+K_B^1,j_L)\oplus\gamma^{(1,1)}G = H(K_A^0+K_B^0+2d_i,j_L)\oplus b_1G.$$

Correctness of the other gate types can be shown analogously.

5.2 Simulation-Based Privacy of Semi-Private Functions

By *active* labels we denote labels which are used in an actual evaluation. An *inactive* label is a label which is not active. For example, if the actual truth value on wire i is v_i, $K_i^{v_i}$ is an active label and $K_i^{1-v_i}$ is an inactive one.

In the proof, the simulator can obtain only active labels, and cannot obtain inactive labels and differences d_i. It means that the simulator can compute only one of $\{H(K_A^0+K_B^0+bd_A,j)\}_{b\in\{0,1,2\}}$, and one of $\{H(K_A^0+ad_A||K_B^0+bd_A,j)\}_{a,b\in\{0,1\}}$.

To simulate the hash values of inactive labels without knowledge of d_i's, the simulator uses the following 10 oracles for a given hash function $H:\{0,1\}^*\longrightarrow\mathbb{Z}_{2^k}$,

- $\mathsf{Corr}_{d_i}^{(1)}(K,b,j) = H(K+bd_i,j)$ where $K\in\mathbb{Z}_{2^k}$ and $b\in\{-2,-1,1,2\}$.
- $\mathsf{Corr}_{d_i}^{(2)}(K,b_1,b_2,j) = H(K+b_1d_i,j)\oplus H(K+b_2d_i,j)$ where $K\in\mathbb{Z}_{2^k}$ and $(b_1,b_2)\in\{(-1,-2),(1,2),(-1,1)\}$.
- For each (K,j), one can query $\mathsf{Corr}_{d_i}^{(1)}(K,b,j)$ or $\mathsf{Corr}_{d_i}^{(2)}(K,b_1,b_2,j)$ only once (one cannot query both).
- $\mathsf{Corr}_{d_i}^{(3)}(K_1,K_2,a,b,j) = H(K_1+ad_i||K_2+bd_i,j)$ where $K_1,K_2\in\mathbb{Z}_{2^k}$ and $a,b\in\{-1,0,1\}$.

– $\mathsf{Corr}^{(4)}_{d_i,d_j}(K,a,b,j') = H(K+ad_j,j') \oplus (K+bd_i)$ where $K \in \mathbb{Z}_{2^k}$ and $a,b \in \{-1,1\}$.
– $\mathsf{Corr}^{(5)}_{d_i,d_j}(i,j,a,b) = ad_i + bd_j$ where $(a,b) \in \{(1,-1),(-1,1)\}$.
– $\mathsf{Rand}^{(1)}_{d_i}(K,b,j), \mathsf{Rand}^{(2)}_{d_i}(K,j), \mathsf{Rand}^{(3)}_{d_i}(K_1,K_2,a,b,j)$ and $\mathsf{Rand}^{(4)}_{d_i,d_j}(K,a,b,j)$ output a random value in $\mathbb{Z}_{2^k}$.
– $\mathsf{Rand}^{(5)}_{d_i,d_j}(i,j,a,b)$ chooses $\hat{d}_i$ and $\hat{d}_j$ at random and outputs $a\hat{d}_i + b\hat{d}_j$.

We use $\mathsf{Corr}^{(1)}_{d_A}$, $\mathsf{Corr}^{(2)}_{d_A}$ and $\mathsf{Corr}^{(3)}_{d_A}$ for obtaining $H(K^0_A+K^0_B+bd_i,j)$, $H(K^0_A+K^0_B+b_1d_i,j) \oplus H(K^0_A+K^0_B+b_2d_i,j)$ and $H(K^0_A+ad_i||K^0_B+bd_i,j)$, which are used for simulating G and $b^{c,\gamma}$. For simulating E, we use $\mathsf{Corr}^{(4)}_{d_A,d_B}$ and $\mathsf{Corr}^{(5)}_{d_A,d_B}$ to obtain $H(K^{1-v_i}_B,j) \pm d_A$ and $\pm d_A \mp d_B$, respectively.

In the random oracle model, it is clear that $\mathsf{Corr}^{(1)}_{d_i}$, $\mathsf{Corr}^{(2)}_{d_i}$, $\mathsf{Corr}^{(3)}_{d_i}$ and $\mathsf{Corr}^{(4)}_{d_i,d_j}$ output a uniformly random distribution. In addition, each d_i is uniformly random since d_i is either chosen uniformly at random or the difference of two hash values. Therefore, $\mathsf{Corr}^{(5)}_{d_i,d_j}$ and $\mathsf{Rand}^{(5)}_{d_i,d_j}$ output an identical distribution. In our sequence of games, we replace the Corr oracles with Rand oracles to move from the real game to the simulation. The proposed garbling scheme is simulation-based private for $\Phi = \Phi_{topo}$ in the random oracle model.

Theorem 1 (Simulation-based Privacy of Semi-Private Functions). *The proposed garbling scheme described in Section 3 satisfies simulation-based privacy, for $\Phi = \Phi_{topo} = (n,m,l,A,B)$ of ciruit $f = (n,m,l,A,B,g)$, of Definition 2, if we assume that H is a random oracle. More precisely, for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{B}$ such that*

$$Adv^{\mathrm{prv.sim}}_{\mathsf{GC},\mathsf{Sim},\Phi_{topo},\mathcal{A}}(k) \leq \frac{l}{2^{k-2}} + \frac{lq}{2^k}$$

where k is the length of keys, l is the number of gates, and q is the number of random oracle queries.

Proof. We consider the simulator $\mathcal{S}$ in the simulated experiment of Definition 2, and the games $\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2^{\mathcal{O}^{(1)}_{d_i},\mathcal{O}^{(2)}_{d_i},\mathcal{O}^{(3)}_{d_i},\mathcal{O}^{(4)}_{d_i,d_j},\mathcal{O}^{(5)}_{d_i,d_j}}, \mathcal{G}_3^{\mathsf{Corr}^{(1)}_{d_i},\mathsf{Corr}^{(2)}_{d_i},\mathsf{Corr}^{(3)}_{d_i},\mathsf{Corr}^{(4)}_{d_i,d_j},\mathsf{Corr}^{(5)}_{d_i,d_j}}$, and $\mathcal{G}_{real}$. We explain the simulator and the games in the following. For simplicity, we only consider AND, OR, and XOR gates. For NAND, NOR, and XNOR gates, we can swap K^0_i and K^1_i in $\mathcal{S}$, $\mathcal{G}_0$, $\mathcal{G}_1$, $\mathcal{G}_2$, $\mathcal{G}_3$, and $\mathcal{G}_{real}$.

$\mathcal{S}(1^k, \Phi_{topo}(f), f(x))$: $\mathcal{S}$ given in Figure 6 generates the garbled circuit and garbled input (F,X,d) without knowledge of x. $\mathcal{S}$ generates only labels corresponding to truth value 0, chooses G,E uniformly at random. $\mathcal{S}$ chooses $b^c_{2c^0_A+c^0_B}$ so that Eval and Decode output $f(x)$.

$\mathcal{G}_0(1^k, \Phi_{circ}(f), f(x))$: $\mathcal{G}_0$ generates the garbled circuit and garbled input (F,X,d) as described in Figure 7. In this game, the actual value v_i for each wire i for input $x = 0$ is computed and known to the simulator. $\mathcal{G}_0$ chooses $b^c_{2c^a_A+c^b_B}$ so that Eval and Decode output $f(x)$.

$\mathcal{S}(1^k, \Phi_{topo}(f), f(x))$

Input: Security parameter k, Circuit $f = (n, m, l, A, B, g)$, and output $f(x)$.

Algorithm:

1. Initialize empty arrays $F[]$, $X[]$ and d with $|F| = l$, $|X| = n$ and $d = m$.
2. Garbling the gates: For $i := n + 1$ to $l + n$ do:
   (a) Set $A := A(i)$ and $B := B(i)$
   (b) If undefined, choose permute bits $\lambda_A, \lambda_B \in \{0,1\}$ at random.
   For all $(a, b) \in \{0,1\}^2$, if undefined, set $c_A^a := \lambda_A \oplus a$, $c_B^b := \lambda_B \oplus b$.
   (c) Input keys:
   - If A's and B's labels are defined, set $j_E := \mathsf{nextindex}()$. If $0 = \lambda_B$, set $K_{B'}^0 := K_B^0$ and $E := \mathsf{Rand}_{d_A,d_B}^{(4)}(K_B^0, 1, 1, j_E)$. Otherwise, set $K_{B'}^0 := K_B^0 + \mathsf{Rand}_{d_A,d_B}^{(5)}(A, B, 1, -1)$ and $E := H(K_B^0, j_E) \oplus K_{B'}^0$.
   - If A's labels are defined and B's labels are undefined (vice versa analog), choose K_B^0 at random and set $K_{B'}^0 := K_B^0$.
   - If A's and B's labels are undefined, choose K_A^0 and K_B^0 at random and set $K_{B'}^0 := K_B^0$.

   (d) Output Keys:
   Choose random bit $b_0, b_1, \in \{0,1\}$ and set $j_L := \mathsf{nextindex}()$.
   Set $\gamma^{(0,0)} := b_0$, $\gamma^{(0,1)} := \gamma^{(1,0)} := 1 - b_0$, $\gamma^{(1,1)} := b_1$.
   Choose random G and $K_i^0 := H(K_A^0 + K_{B'}^0, j_L) \oplus \gamma^{(0,0)} G$.
   (e) Encrypt choice bit:
   Set $j_{c,\gamma} := \mathsf{nextindex}()$, choose $\lambda_i \in \{0,1\}$ at random.
   - If $(a, b) = (0, 0)$,
     $b^{c,\gamma}_{2c_A^0 + c_B^0} := \mathsf{lsb}_2(H(K_A^0 || K_{B'}^0, j_{c,\gamma})) \oplus ((g_i(0,0) \oplus \lambda_i) || \gamma^{(0,0)})$.
     Exception: if $i = l + n$, set
     $b^{c,\gamma}_{2c_A^0 + c_B^0} := \mathsf{lsb}_2(H(K_A^0 || K_{B'}^0, j_{c,\gamma})) \oplus ((g_i(0,0) \oplus \lambda_i \oplus f(x)) || \gamma^{(0,0)})$.
   - Otherwise,
     $b^{c,\gamma}_{2c_A^a + c_B^b} := \mathsf{lsb}_2(\mathsf{Rand}_{d_A}^{(3)}(K_A^0, K_{B'}^0, a, b, j_{c,\gamma})) \oplus ((g_i(a,b) \oplus \lambda_i) || \gamma^{(a,b)})$.

   (f) Set $F[i] := (b_0^{c,\gamma}, b_1^{c,\gamma}, b_2^{c,\gamma}, b_3^{c,\gamma}, G, E)$, if E is defined.
   Set $F[i] := (b_0^{c,\gamma}, b_1^{c,\gamma}, b_2^{c,\gamma}, b_3^{c,\gamma}, G)$, otherwise.
   (g) If $j \in \{A, B\}$ is a circuit input wire, set $X[j] := K_j^0 || c_j^0$.
   If $i \in W_{output}$, set $d[i - (n + l) + m] := \lambda_i$.

Output: Garbled circuit F, garbled input X, decoding d.

Fig. 6. The simulator $\mathcal{S}$.

$\mathcal{G}_1(1^k, \Phi_{circ}(f), x)$**:** $\mathcal{G}_0$ generates the garbled circuit and garbled input (F, X, d) as described in Figure 8. In this game, the actual value v_i for each wire i for input x is computed and known to the simulator. $\mathcal{G}_0$ generates the output label in one of two ways, which depends on v_i.

$\mathcal{G}_2^{\mathcal{O}_{d_i}^{(1)}, \mathcal{O}_{d_i}^{(2)}, \mathcal{O}_{d_i}^{(3)}, \mathcal{O}_{d_i,d_j}^{(4)}, \mathcal{O}_{d_i,d_j}^{(5)}}(1^k, \Phi_{circ}(f), x)$**:** In the game, (F, X, d) is generated as described in Figure 9, with three oracles without knowledge of d_i's. The oracles queried are either $(\mathsf{Rand}_{d_i}^{(1)}, \mathsf{Rand}_{d_i}^{(2)}, \mathsf{Rand}_{d_i}^{(3)}, \mathsf{Rand}_{d_i,d_j}^{(4)}, \mathsf{Rand}_{d_i,d_j}^{(5)})$ or the oracles $(\mathsf{Corr}_{d_i}^{(1)}, \mathsf{Corr}_{d_i}^{(2)}, \mathsf{Corr}_{d_i}^{(3)}, \mathsf{Corr}_{d_i,d_j}^{(4)}, \mathsf{Corr}_{d_i,d_j}^{(5)})$. In $\mathcal{G}_1$, the active labels $K_i^{v_i}$ and oracle outputs $H(K_A^0 + K_B^0 + (v_A + v_B)d, j)$ are computed similar to the real scheme.

For simulating the ciphertext G, $H(K_A^0 + K_B^0, j_L) + H(K_A^0 + K_B^0 + d, j_L)$ has to be computed. The simulator makes oracle query $\mathcal{O}_{d_A}^{(1)}(K_A^{v_A} + K_B^{v_B}, b - (v_A + v_B), j_L)$ if $(v_A, v_B) \neq (1, 1)$, and $\mathcal{O}_{d_A}^{(2)}(K_A^{v_A} + K_B^{v_B}, j_L)$ if $(v_A, v_B) = (1, 1)$.

$\mathcal{G}_0(1^k, \Phi_{circ}(f), f(x))$

Input: Security parameter k, Circuit $f = (n, m, l, A, B, g)$, and output $f(x)$.
Algorithm: 1. Initialize empty arrays $F[]$, $X[]$ and $d[]$ with $|F| = l$, $|X| = n$ and $|d| = m$.
2. Compute $f(0)$ and $v_i \in \{0,1\}$ that is the actual value on wire i for $x = 0$.
3. Garbling the gates: For $i := n+1$ to $l+n$ do:
   (a) Set $A := A(i)$ and $B := B(i)$
   (b) If undefined, choose permute bits $\lambda_A, \lambda_B \in \{0,1\}$ at random.
   For all $(a,b) \in \{0,1\}^2$, if undefined, set $c_A^a := \lambda_A \oplus a$, $c_B^b := \lambda_B \oplus b$.
   (c) Input keys:
   Same as $\mathcal{S}$.
   (d) Output Keys:
   Choose random bit $b_0, b_1, \in \{0,1\}$ and set $j_L := \mathsf{nextindex}()$.
   – AND gate case: Set $\gamma^{(0,0)} := b_0$, $\gamma^{(0,1)} := \gamma^{(1,0)} := 1-b_0$, $\gamma^{(1,1)} := b_1$.
   Choose random G and $K_i^{v_i} := H(K_A^{v_A} + K_{B'}^{v_B}, j_L) \oplus \gamma^{(v_A,v_B)}G$.
   – OR gate case: Set $\gamma^{(0,0)} := b_1$, $\gamma^{(0,1)} := \gamma^{(1,0)} := b_0$, $\gamma^{(1,1)} := 1-b_0$.
   Choose random G and $K_i^{v_i} := H(K_A^{v_A} + K_{B'}^{v_B}, j_L) \oplus \gamma^{(v_A,v_B)}G$.
   – XOR gate case: Set $\gamma^{(0,0)} := 1-b_0$, $\gamma^{(0,1)} := \gamma^{(1,0)} := b_1$, $\gamma^{(1,1)} := b_0$.
   Choose random G and $K_i^{v_i} := H(K_A^{v_A} + K_{B'}^{v_B}, j_L) \oplus \gamma^{(v_A,v_B)}G$.
   (e) Encrypt choice bit:
   Set $j_{c,\gamma} := \mathsf{nextindex}()$, choose $\lambda_i \in \{0,1\}$ at random.
   – If $(a,b) = (v_A, v_B)$,
   $b^{c,\gamma}_{2c_A^a + c_B^b} := \mathsf{lsb}_2(H(K_A^{v_A}||K_{B'}^{v_B}, j_{c,\gamma})) \oplus ((g_i(a,b) \oplus \lambda_i)||\gamma^{(a,b)})$.
   Exception: if $i = l+n$, set
   $b^{c,\gamma}_{2c_A^a + c_B^b} := \mathsf{lsb}_2(H(K_A^a||K_{B'}^b, j_{c,\gamma})) \oplus ((g_i(a,b) \oplus \lambda_i \oplus f(x))||\gamma^{(a,b)})$.
   – Otherwise,
   $b^{c,\gamma}_{2c_A^a + c_B^b} := \mathsf{lsb}_2(\mathsf{Rand}^{(3)}_{d_A}(K_A^{v_A}, K_{B'}^{v_B}, a-v_A, b-v_B, j_{c,\gamma})) \oplus ((g_i(a,b) \oplus \lambda_i)||\gamma^{(a,b)})$.
   (f) Set $F[i] := (b_0^{c,\gamma}, b_1^{c,\gamma}, b_2^{c,\gamma}, b_3^{c,\gamma}, G, E)$, if E is defined.
   Set $F[i] := (b_0^{c,\gamma}, b_1^{c,\gamma}, b_2^{c,\gamma}, b_3^{c,\gamma}, G)$, otherwise.
   (g) If $j \in \{A, B\}$ is a circuit input wire, set $X[j] := K_j^0||c_j^0$.
   If $i \in W_{output}$, set $d[i - (n+l) + m] := \lambda_i$.
Output: Garbled circuit F, garbled input X, decoding d.

Fig. 7. The game $\mathcal{G}_0$ in which the output keys are generated as in the real scheme.

For simulating the ciphertexts $b^c_{2c_A^a+c_B^b}||b^\gamma_{2c_A^a+c_B^b}$, the simulator makes oracle query $\mathcal{O}^{(3)}_{d_A}(K_A^{v_A}, K_B^{v_B}, a-v_A, b-v_B, j_{c,\gamma})$, obtains $H(K_A^0 + ad_A||K_B^0 + bd_B, j_{c,\gamma})$, and computes $b^c_{2c_A^a+c_B^b}||b^\gamma_{2c_A^a+c_B^b}$.
The ciphertext E is computed as

$$E = H(K_B^{1-\lambda_B}, j_e) + K_{B'}^{1-\lambda_B}$$
$$= \begin{cases} H(K_B^{v_B} + (1-2v_B)d_B, j_e) + K_B^{v_B} + (1-2v_B)d_A & \text{if } v_B = \lambda_B \\ H(K_B^{v_B}, j_e) + K_B^{v_B} + (2v_B - 1)d_B + (1-2v_B)d_A & \text{otherwise} \end{cases}.$$

The simulator makes oracle query $\mathcal{O}^{(4)}_{d_A,d_B}(K_B^{v_B}, 1-2v_B, 1-2v_B, j_e)$ instead of computing $H(K_B^{v_B} + (1-2v_B)d_B, j_e) + (1-2v_B)d_A$, and oracle query $\mathcal{O}^{(5)}_{d_A,d_B}(A, B, 2v_B-1, 1-2v_B)$ instead of computing $(2v_B-1)d_B + (1-2v_B)d_A$.

$\mathcal{G}_3^{\mathsf{Corr}^{(1)}_{d_i}, \mathsf{Corr}^{(2)}_{d_i}, \mathsf{Corr}^{(3)}_{d_i}, \mathsf{Corr}^{(4)}_{d_i,d_j}, \mathsf{Corr}^{(5)}_{d_i,d_j}}(1^k, \Phi_{circ}(f), x)$**:** This game, described in Figure 10, is almost identical to $\mathcal{G}_1$, except that inactive labels are defined. In this game, the simulator knows d_i's.

$\mathcal{G}_1(1^k, \Phi_{circ}(f), x)$

Input: Security parameter k, Circuit $f = (n, m, l, A, B, g)$, and input x.
Algorithm: 1. Initialize empty arrays $F[]$, $X[]$ and $d[]$ with $|F| = l$, $|X| = n$ and $|d| = m$.
2. Compute $f(x)$ and $v_i \in \{0,1\}$ that is the actual value on wire i.
3. Garbling the gates: For $i := n+1$ to $l+n$ do:
   (a) Set $A := A(i)$ and $B := B(i)$
   (b) If undefined, choose permute bits $\lambda_A, \lambda_B \in \{0,1\}$ at random.
   For all $(a,b) \in \{0,1\}^2$, if undefined, set $c_A^a := \lambda_A \oplus a$, $c_B^b := \lambda_B \oplus b$.
   (c) Input keys:
   Same as $\mathcal{S}$.
   (d) Output Keys:
   Choose random bit $b_0, b_1, \in \{0,1\}$ and set $j_L := \mathsf{nextindex}()$.
   – AND gate case: Set $\gamma^{(0,0)} := b_0$, $\gamma^{(0,1)} := \gamma^{(1,0)} := 1-b_0$, $\gamma^{(1,1)} := b_1$.
   Choose random G and $K_i^{v_i} := H(K_A^{v_A} + K_{B'}^{v_B}, j_L) \oplus \gamma^{(v_A, v_B)} G$.
   – OR gate case: Set $\gamma^{(0,0)} := b_1$, $\gamma^{(0,1)} := \gamma^{(1,0)} := b_0$, $\gamma^{(1,1)} := 1-b_0$.
   Choose random G and $K_i^{v_i} := H(K_A^{v_A} + K_{B'}^{v_B}, j_L) \oplus \gamma^{(v_A, v_B)} G$.
   – XOR gate case: Set $\gamma^{(0,0)} := 1-b_0$, $\gamma^{(0,1)} := \gamma^{(1,0)} := b_1$, $\gamma^{(1,1)} := b_0$.
   Choose random G and $K_i^{v_i} := H(K_A^{v_A} + K_{B'}^{v_B}, j_L) \oplus \gamma^{(v_A, v_B)} G$.
   (e) Encrypt choice bit:
   Set $j_{c,\gamma} := \mathsf{nextindex}()$, choose $\lambda_i \in \{0,1\}$ at random.
   – If $(a,b) = (v_A, v_B)$,
   $b^{c,\gamma}_{2c_A^a + c_B^b} := \mathsf{lsb}_2(H(K_A^{v_A} || K_{B'}^{v_B}, j_{c,\gamma})) \oplus ((g_i(a,b) \oplus \lambda_i) || \gamma^{(a,b)})$.
   – Otherwise,
   $b^{c,\gamma}_{2c_A^a + c_B^b} := \mathsf{lsb}_2(\mathsf{Rand}^{(3)}_{d_A}(K_A^{v_A}, K_{B'}^{v_B}, a - v_A, b - v_B, j_{c,\gamma})) \oplus ((g_i(a,b) \oplus \lambda_i) || \gamma^{(a,b)})$.
   (f) Set $F[i] := (b_0^{c,\gamma}, b_1^{c,\gamma}, b_2^{c,\gamma}, b_3^{c,\gamma}, G, E)$, if E is defined.
   Set $F[i] := (b_0^{c,\gamma}, b_1^{c,\gamma}, b_2^{c,\gamma}, b_3^{c,\gamma}, G)$, otherwise.
   (g) If $j \in \{A, B\}$ is a circuit input wire, set $X[j] := K_j^0 || c_j^0$.
   If $i \in W_{output}$, set $d[i - (n+l) + m] := \lambda_i$.
Output: Garbled circuit F, garbled input X, decoding d.

Fig. 8. The game $\mathcal{G}_1$ in which the output keys are generated as in the real scheme.

$\mathcal{G}_{real}$**:** This is the real experiment of Definition 2.

Now we prove the indistinguishability between the simulators and the real protocol. We use the following chain of simulators and hybrid games.

1. $\mathcal{S} \equiv \mathcal{G}_0$: The only difference between the two games is that K^0 is used in $\mathcal{S}$ but K^v is used in $\mathcal{G}_0$ and there are 3 cases of AND, OR, and XOR in $\mathcal{G}_0$. However, the distributions of simulation are identical, since the inactive $\gamma^{(a,b)}$'s are masked by random oracle Rand.
2. $\mathcal{G}_0 \equiv \mathcal{G}_1$: The only difference between the two games is that v_i for input $x = 0$ is used and output $f(x)$ is embedded in $\mathcal{G}_0$ but v_i for input x is used in $\mathcal{G}_1$. However, the distributions of simulation are identical, since the inactive $\gamma^{(a,b)}$'s are masked by random oracle Rand and embedded $f(x)$ is masked by random λ_i.
3. $\mathcal{G}_1 \equiv \mathcal{G}_2^{\mathsf{Rand}^{(1)}_{d_i}, \mathsf{Rand}^{(2)}_{d_i}, \mathsf{Rand}^{(3)}_{d_i}, \mathsf{Rand}^{(4)}_{d_i, d_j}, \mathsf{Rand}^{(5)}_{d_i, d_j}}$: The only difference between them is how G is generated. However, G is uniformly distributed in both games and therefore these two games are indistinguishable.

$\mathcal{G}_2^{\mathcal{O}_{d_i}^{(1)},\mathcal{O}_{d_i}^{(2)},\mathcal{O}_{d_i}^{(3)},\mathcal{O}_{d_i,d_j}^{(4)},\mathcal{O}_{d_i,d_j}^{(5)}}(1^k, \Phi_{circ}(f), x)$

Input: Security parameter k, Circuit $f = (n, m, l, A, B, g)$, and input x.

Algorithm: 1. Initialize empty arrays $F[]$, $X[]$ and $d[]$ with $|F| = l$, $|X| = n$ and $|d| = m$.

2. Compute $f(x)$ and $v_i \in \{0,1\}$ that is the actual value on wire i.
3. Garbling the gates: For $i := n+1$ to $l+n$ do:
   (a) Set $A := A(i)$ and $B := B(i)$
   (b) If undefined, choose permute bits $\lambda_A, \lambda_B \in \{0,1\}$ at random.
   For all $(a,b) \in \{0,1\}^2$, if undefined, set $c_A^a := \lambda_A \oplus a$, $c_B^b := \lambda_B \oplus b$.
   (c) Input keys:
   – If A's and B's labels are defined, set $j_E := \mathsf{nextindex}()$. If $v_B = \lambda_B$, set $K_{B'}^{v_B} := K_B^{v_B}$ and $E := \mathcal{O}_{d_A,d_B}^{(4)}(K_B^{v_B}, 1-2v_B, 1-2v_B, j_E)$. Otherwise, set $K_{B'}^{v_B} := K_B^{v_B} + \mathcal{O}_{d_A,d_B}^{(5)}(A, B, 1-2v_B, 2v_B-1)$ and $E := H(K_B^{v_B}, j_E) \oplus K_{B'}^{v_B}$.
   – If A's labels are defined and B's labels are undefined (vice versa analog), choose $K_B^{v_B}$ at random and set $K_{B'}^{v_B} := K_B^{v_B}$.
   – If A's and B's labels are undefined, choose $K_A^{v_A}$ and $K_B^{v_B}$ at random and set $K_{B'}^{v_B} := K_B^{v_B}$.
   (d) Output Keys:
   Choose random bit $b_0, b_1, \in \{0,1\}$ and set $j_L := \mathsf{nextindex}()$.
   – AND gate case: Set $\gamma^{(0,0)} := b_0$, $\gamma^{(0,1)} := \gamma^{(1,0)} := 1-b_0$, $\gamma^{(1,1)} := b_1$.
   • If $(v_A, v_B) \neq (1,1)$,
   $G := H(K_A^{v_A} + K_{B'}^{v_B}, j_L) \oplus \mathcal{O}_{d_A}^{(1)}(K_A^{v_A} + K_{B'}^{v_B}, b, j_L)$ where $b = 0$ if $v_A + v_B = 1$, $b = 1$ if $v_A + v_B = 0$,
   $K_i^{v_i} := H(K_A^{v_A} + K_{B'}^{v_B}, j_L) \oplus \gamma^{(v_A, v_B)} G$.
   • If $(v_A, v_B) = (1,1)$,
   $G := \mathcal{O}_{d_A}^{(2)}(K_A^{v_A} + K_{B'}^{v_B}, -1, -2, j_L)$,
   $K_i^{v_i} := H(K_A^{v_A} + K_{B'}^{v_B}, j_L) \oplus \gamma^{(v_A, v_B)} G$.
   – OR gate case: Set $\gamma^{(0,0)} := b_1$, $\gamma^{(0,1)} := \gamma^{(1,0)} := b_0$, $\gamma^{(1,1)} := 1-b_0$.
   • If $(v_A, v_B) \neq (0,0)$,
   $G := H(K_A^{v_A} + K_{B'}^{v_B}, j_L) \oplus \mathcal{O}_{d_A}^{(1)}(K_A^{v_A} + K_{B'}^{v_B}, b, j_L)$ where $b = 1$ if $v_A + v_B = 2$, $b = 2$ if $v_A + v_B = 1$,
   $K_i^{v_i} := H(K_A^{v_A} + K_{B'}^{v_B}, j_L) \oplus \gamma^{(v_A, v_B)} G$.
   • If $(v_A, v_B) = (0,0)$,
   $G := \mathcal{O}_{d_A}^{(2)}(K_A^{v_A} + K_{B'}^{v_B}, 1, 2, j_L)$,
   $K_i^{v_i} := H(K_A^{v_A} + K_{B'}^{v_B}, j_L) \oplus \gamma^{(v_A, v_B)} G$.
   – XOR gate case: Set $\gamma^{(0,0)} := 1-b_0$, $\gamma^{(0,1)} := \gamma^{(1,0)} := b_1$, $\gamma^{(1,1)} := b_0$.
   • If $(v_A, v_B) \neq (0,1), (1,0)$,
   $G := H(K_A^{v_A} + K_{B'}^{v_B}, j_L) \oplus \mathcal{O}_{d_A}^{(1)}(K_A^{v_A} + K_{B'}^{v_B}, b, j_L)$ where $b = 0$ if $v_A + v_B = 2$, $b = 2$ if $v_A + v_B = 0$,
   $K_i^{v_i} := H(K_A^{v_A} + K_{B'}^{v_B}, j_L) \oplus \gamma^{(v_A, v_B)} G$.
   • If $(v_A, v_B) = (0,1), (1,0)$,
   $G := \mathcal{O}_{d_A}^{(2)}(K_A^{v_A} + K_{B'}^{v_B}, -1, 1, j_L)$,
   $K_i^{v_i} := H(K_A^{v_A} + K_{B'}^{v_B}, j_L) \oplus \gamma^{(v_A, v_B)} G$.
   (e) Encrypt choice bit:
   Set $j_{c,\gamma} := \mathsf{nextindex}()$, choose $\lambda_i \in \{0,1\}$ at random.
   – If $(a,b) = (v_A, v_B)$,
   $b_{2c_A^a + c_B^b}^{c,\gamma} := \mathsf{lsb}_2(H(K_A^{v_A} || K_{B'}^{v_B}, j_{c,\gamma})) \oplus ((g_i(a,b) \oplus \lambda_i) || \gamma^{(a,b)})$.
   – Otherwise,
   $b_{2c_A^a + c_B^b}^{c,\gamma} := \mathsf{lsb}_2(\mathcal{O}_{d_A}^{(3)}(K_A^{v_A}, K_{B'}^{v_B}, a - v_A, b - v_B, j_{c,\gamma})) \oplus ((g_i(a,b) \oplus \lambda_i) || \gamma^{(a,b)})$.
   (f) Set $F[i] := (b_0^{c,\gamma}, b_1^{c,\gamma}, b_2^{c,\gamma}, b_3^{c,\gamma}, G, E)$, if E is defined.
   Set $F[i] := (b_0^{c,\gamma}, b_1^{c,\gamma}, b_2^{c,\gamma}, b_3^{c,\gamma}, G)$, otherwise.
   (g) If $j \in \{A, B\}$ is a circuit input wire, set $X[j] := K_j^{v_j} || c_j^{v_j}$.
   If $i \in W_{output}$, set $d[i - (n+l) + m] := \lambda_i$.

Output: Garbled circuit F, garbled input X, decoding d.

Fig. 9. The game $\mathcal{G}_2$ in which garbling with actual values.

$\mathcal{G}_3^{\mathsf{Corr}_{d_i}^{(1)},\mathsf{Corr}_{d_i}^{(2)},\mathsf{Corr}_{d_i}^{(3)},\mathsf{Corr}_{d_i,d_j}^{(4)},\mathsf{Corr}_{d_i,d_j}^{(5)}}(1^k, \Phi_{circ}(f), x)$

Input: Security parameter k, Circuit $f = (n, m, l, A, B, g)$, and input x.
Algorithm: 1. Initialize empty arrays $F[]$, $X[]$ and $d[]$ with $|F| = l$, $|X| = n$ and $|d| = m$.
2. Compute $f(x)$ and $v_i \in \{0,1\}$ that is the actual value in wire i.
3. Garbling the gates: For $i := n+1$ to $l+n$ do:
   (a) Set $A := A(i)$ and $B := B(i)$
   (b) If undefined, choose permute bits $\lambda_A, \lambda_B \in \{0,1\}$ at random.
   For all $(a,b) \in \{0,1\}^2$, if undefined, set $c_A^a := \lambda_A \oplus a$, $c_B^b := \lambda_B \oplus b$.
   (c) Input keys:
   Same as $\mathcal{G}_2^{\mathsf{Corr}_{d_i}^{(1)},\mathsf{Corr}_{d_i}^{(2)},\mathsf{Corr}_{d_i}^{(3)},\mathsf{Corr}_{d_i,d_j}^{(4)},\mathsf{Corr}_{d_i,d_j}^{(5)}}$ except setting $K_A^{1-v_A} := (1-2v_A)d_A + K_A^{v_A}$, $K_B^{1-v_B} := (1-2v_B)d_A + K_B^{v_B}$, $K_{B'}^{1-v_B} := (1-2v_B)d_A + K_{B'}^{v_B}$ if undefined.
   (d) Output Keys:
   Same as $\mathcal{G}_2^{\mathsf{Corr}_{d_i}^{(1)},\mathsf{Corr}_{d_i}^{(2)},\mathsf{Corr}_{d_i,d_j}^{(3)},\mathsf{Corr}_{d_i,d_j}^{(4)},\mathsf{Corr}_{d_i,d_j}^{(5)}}$ except adding the following in the last step:
   - AND gate case:
     If $v_i = 0$, set $K_i^{1-v_i} := H(K_A^{v_A} + K_{B'}^{v_B} + 2d_i, j_L) \oplus b_1 G$.
     If $v_i = 1$, set $K_i^{1-v_i} := H(K_A^{v_A} + K_{B'}^{v_B} + b_0 d_i, j_L)$.
   - OR gate case:
     If $v_i = 0$, set $K_i^{1-v_i} := H(K_A^{v_A} + K_{B'}^{v_B} + d_i + b_0 d_i, j_L)$.
     If $v_i = 1$, set $K_i^{1-v_i} := H(K_A^{v_A} + K_{B'}^{v_B}, j_L) \oplus b_1 G$.
   - XOR gate case:
     If $v_i = 0$, set $K_i^{1-v_i} := H(K_A^{v_A} + K_{B'}^{v_B} + d_i, j_L) \oplus b_1 G$.
     If $v_i = 1$, set $K_i^{1-v_i} := H(K_A^{v_A} + K_{B'}^{v_B} + b_0 2d_i, j_L)$.
   (e) Encrypt choice bit:
   Same as $\mathcal{G}_2^{\mathsf{Corr}_{d_i}^{(1)},\mathsf{Corr}_{d_i}^{(2)},\mathsf{Corr}_{d_i}^{(3)},\mathsf{Corr}_{d_i,d_j}^{(4)},\mathsf{Corr}_{d_i,d_j}^{(5)}}$.
   (f) Set $F[i] := (b_0^{c,\gamma}, b_1^{c,\gamma}, b_2^{c,\gamma}, b_3^{c,\gamma}, G, E)$, if E is defined.
   Set $F[i] := (b_0^{\bar{c},\gamma}, b_1^{\bar{c},\gamma}, b_2^{\bar{c},\gamma}, b_3^{\bar{c},\gamma}, G)$, otherwise.
   (g) If $j \in \{A, B\}$ is a circuit input wire, set $X[j] := K_j^{v_j} || c_j^{v_j}$.
   If $i \in W_{output}$, set $d[i - (n+l) + m] := \lambda_i$.
Output: Garbled circuit F, garbled input X, decoding d.

Fig. 10. The game $\mathcal{G}_3$ in which including inactive keys.

4. $\mathcal{G}_2^{\mathsf{Rand}_{d_i}^{(1)},\mathsf{Rand}_{d_i}^{(2)},\mathsf{Rand}_{d_i}^{(3)},\mathsf{Rand}_{d_i,d_j}^{(4)},\mathsf{Rand}_{d_i,d_j}^{(5)}} \equiv \mathcal{G}_2^{\mathsf{Corr}_{d_i}^{(1)},\mathsf{Corr}_{d_i}^{(2)},\mathsf{Corr}_{d_i}^{(3)},\mathsf{Corr}_{d_i,d_j}^{(4)},\mathsf{Corr}_{d_i,d_j}^{(5)}}$: The Rand oracles are replaced with Corr oracles. These games are indistinguishable since the hash function is a random oracle except if $d_i = 0$ or $2d_i = 0$ for some i. However, l d_i's are chosen uniformly at random and then the probability of the event is bounded by $\Pr[\exists i \text{ s.t. } (d_i = 0) \vee (2d_i = 0)] = 1 - (1 - 2/2^k)^l \leq l/2^{k-2}$.
5. $\mathcal{G}_2^{\mathsf{Corr}_{d_i}^{(1)},\mathsf{Corr}_{d_i}^{(2)},\mathsf{Corr}_{d_i}^{(3)},\mathsf{Corr}_{d_i,d_j}^{(4)},\mathsf{Corr}_{d_i,d_j}^{(5)}} \equiv \mathcal{G}_3^{\mathsf{Corr}_{d_i}^{(1)},\mathsf{Corr}_{d_i}^{(2)},\mathsf{Corr}_{d_i}^{(3)},\mathsf{Corr}_{d_i,d_j}^{(4)},\mathsf{Corr}_{d_i,d_j}^{(5)}}$: The only difference between these two games is adding the inactive labels. However, the inactive labels are not used, so the distribution is unchanged. In $\mathcal{G}_1^{\mathsf{Corr}_{d_i}^{(1)},\mathsf{Corr}_{d_i}^{(2)},\mathsf{Corr}_{d_i}^{(3)},\mathsf{Corr}_{d_i,d_j}^{(4)},\mathsf{Corr}_{d_i,d_j}^{(5)}}$, if the adversary correctly guess one of l d_i's, and ask a key unknown to the simulator for random oracle H among q oracle queries, the simulator fails to simulate random oracle H since the

simulator does not know d_i's. The probability of this simulation failure is $lq/2^k$.

6. $\mathcal{G}_3^{\mathsf{Corr}_{d_i}^{(1)},\mathsf{Corr}_{d_i}^{(2)},\mathsf{Corr}_{d_i}^{(3)},\mathsf{Corr}_{d_i,d_j}^{(4)},\mathsf{Corr}_{d_i,d_j}^{(5)}} \equiv \mathcal{G}_{real}$: In $\mathcal{G}_2$, we first define $K_i^{v_i}$ and then define $K_i^{1-v_i}$ as $K_i^{1-v_i} := (1-2v_i)d_i + K_i^{v_i}$ if either or both input wires is a circuit input wire. In $\mathcal{G}_{real}$, we first define K_i^0 and then $K_i^1 := d_i + K_i^0$. In addition, we define $K_i^{1-v_i} := H(K_A^{v_A} + K_{B'}^{v_B} + (2-(v_A+v_B))d_A, j_L) + b_1 G$ or $K_i^{1-v_i} := H(K_A^{v_A} + K_{B'}^{v_B} + (b_0-2)d_A, j_L)$, depending on v_i and b_0 for inner wires in $\mathcal{G}_2$. In $\mathcal{G}_{real}$, we define $K_i^0 := H(K_A^0 + K_{B'}^0 + 2d_A, j_L) + b_1 G$ and $K_i^1 := H(K_A^0 + K_{B'}^0 + b_0 d_A, j_L) + b_1 G$. Although the steps to generate the labels are changed, the outputs are unchanged. Therefore, these changes do not affect the distribution.

Consequently, the simulated experiment is indistinguishable from the real experiment except negligible probability $l/2^{k-2} + lq/2^k$.

□

6 On the Lower Bound of Linear Garbling Schemes

Zahur et al. [30] observed that many practical garbling schemes share a common structure, which they formalize in their model of linear garbling schemes. They proved that in this model, garbling a single AND gate requires at least two rows. They concluded that to garble an odd gate with significantly[9] less than $2k$ bits, an inherently different, non-linear structure is needed. Our garbling scheme contradicts this, while maintaining a computational efficiency comparable to previous work. In this chapter, we first provide an intuition of how our construction circumvents the lower bound. In Section 6.2, we provide an outline of the lower-bound proof. Then, we state our garbling scheme in the linear garbling model, and show how it exploits loopholes in the lower-bound proof more formally. Since this chapter mostly discusses a single AND gate, we denote the color bits of a gate's input wires by α and β for better readability.

6.1 How we Circumvent the Lower Bound: An Intuition

Intuitively, the arguments in the lower-bound proof should also hold for our "almost linear" garbling scheme. We now show where our construction exploits loopholes. The proof relies on two assumptions which hold for most linear constructions, but are not needed for linearity in an algebraic sense:

Assumption (1): The linear operations to compute a gate's output labels *directly* depend on permute bits assigned to the gate's input wires.
Assumption (2): Each ciphertext/row consists of k bit.

[9] As they point out, one can only prove a lower-bound of at least $2k$ minus some bits, as one can always take away a few bits and maintain asymptotic security.

Let us first take a closer look at Assumption (1). The lower-bound proof strongly depends on the fact that changing the permute bits assigned to the two input wires of a gate changes the operation the evaluator needs to perform when processing this gate. This is true for most existing garbling schemes, but not for ours. In existing schemes, the evaluator usually uses two color bits α and β to choose one out of four options. In Yao's original scheme [29], when used with the point and permute technique [2], the four options are four ciphertexts. In the three-row reduction [22, 27], the options are three ciphertexts and the zero string. In the interpolation-based two-row reduction [27], the options are four x-coordinates. The half gate construction [30] has the options ciphertext or zero string for each half gate, so four possible options per garbled AND gate. The common way to let the evaluator choose the correct option is letting the options depend on permute bits, and communicating corresponding color bits to the evaluator, which keeps him oblivious of the actual input. All of the above mentioned schemes use this technique. As a side-effect, in all these constructions, changing even one permute bit inevitably changes the assignment of options to input values, in particular, which option is assigned to the input leading to output truth value 1. In our scheme, the evaluator has only two options: to use the given k-bit ciphertext or not. Neither this ciphertext itself nor its usage depends on any permute bit. In fact, it might happen that the *same* operation using this *same* ciphertext might be performed by the evaluator for two possible inputs $(x_a, x_b) \neq (x'_a, x'_b)$, with $g(x_a, x_b) \neq g(x'_a, x'_b)$. Since we have only two options, we need only one choice bit, which does not depend on permute bits.

To communicate this choice bit, we include four 2-bit ciphertexts in the garbled circuit, which do depend on permute bits. And this is where we exploit the second loophole in the lower-bound proof: Assumption (2), which says that each row has the length of k bit, is neither used nor needed[10] in any arguments in the proof. Since $M < k$ bit of information can perfectly be masked with a M-bit ciphertext, we can instead "fill the necessary rows" with our 2-bit ciphertexts.

6.2 How we Circumvent the Lower Bound: Formal Discussion

Let us first briefly summarize the linear garbling model. All elements considered in the model are in $GF(2^k)$, and the only operations allowed are XOR operations and calls to a random oracle, which outputs elements in $GF(2^k)$. The model considers garbling a *single* AND gate. Let r and h be constants, and let $\langle ., . \rangle$ denote the scalar product of two vectors. The garbler chooses $R_1, \ldots, R_r \in GF(2^k)$ at random. Using linear combinations of these as inputs to a random oracle, he obtains oracle responses $Q_1, \ldots, Q_h$. Let $S := (R_1, \ldots, R_r, Q_1, \ldots, Q_h)$. The garbler applies linear functions on S to obtain input wire labels A_0, A_1, B_0, B_1, output wire labels C_0, C_1 and ciphertexts $G_1, \ldots, G_m$. The function to obtain the ciphertexts can be written as a matrix $\mathbb{G}_{\lambda_a, \lambda_b}$ with $S \cdot \mathbb{G}_{\lambda_a, \lambda_b} = (G_1, \ldots, G_m)$, and can depend on the permute bits λ_a and λ_b of the input wires.

[10] More precisely, elements in $GF(2^k)$ or $\mathbb{Z}_{2^k}$ do not necessarily have k bits of entropy, and those with less entropy can be represented using shorter strings.

The evaluator obtains as input the wire labels $K_A \in \{A_0, A_1\}$ and $K_B \in \{B_0, B_1\}$, and color bits α and β. He makes several oracle queries using this input to obtain a vector T, which consists of his input, the oracle responses and the ciphertexts $G_1, \ldots, G_m$. He computes a linear function on T, denoted by a vector $V_{\alpha,\beta}$, to compute the output wire label $C_{(\lambda_a \oplus \alpha) \wedge (\lambda_b \oplus \beta)} = \langle V_{\alpha,\beta}, T \rangle$.

The Lower-Bound Proof We recap the parts of the lower-bound proof [30] which are important for a more formal discussion. The proof argues that the matrix $\mathbb{G}_{\lambda_a,\lambda_b}$ must have at least two rows, and thus creates at least two ciphertexts. This is based on a chain of claims, of which we circumvent the first one: it says that the $\mathbb{G}_{\lambda_a,\lambda_b}$ are all distinct. The claim is argued for as follows. The output wire label $C_{(\lambda_a \oplus \alpha) \wedge (\lambda_b \oplus \beta)}$, computed by the evaluator as $C_{(\lambda_a \oplus \alpha) \wedge (\lambda_b \oplus \beta)} = \langle V_{\alpha,\beta}, T \rangle$, can be written as

$$C_{(\lambda_a \oplus \alpha) \wedge (\lambda_b \oplus \beta)} = \left\langle V^{pub}_{\alpha,\beta}, \mathbb{M}_{\alpha,\beta} \times S^T \right\rangle + \left\langle V^{prv}_{\alpha,\beta}, \mathbb{G}_{\lambda_a,\lambda_b} \times S^T \right\rangle, \qquad (1)$$

for an appropriate matrix $\mathbb{M}_{\alpha,\beta}$, where $V_{\alpha,\beta}$ is divided into a public part $V^{pub}_{\alpha,\beta}$, independent of permute bits, and a private part $V^{prv}_{\alpha,\beta}$, which depends on λ_a and λ_b. For only *one* input $((\lambda_a \oplus \alpha), (\lambda_b \oplus \beta))$, it holds that $(\lambda_a \oplus \alpha) \wedge (\lambda_b \oplus \beta) = 1$, and thus $C_{(\lambda_a \oplus \alpha) \wedge (\lambda_b \oplus \beta)} = C_1$, the label assigned to truth value 1. When changing a permute bit, a different combination of (α, β) is assigned to C_1. However, since all other values in Equation 1 do not depend on permute bits, only $\mathbb{G}_{\lambda_a,\lambda_b}$ can change when changing λ_a or λ_b. Thus, all $\mathbb{G}_{\lambda_a,\lambda_b}$ must be distinct. Basic algebra then implies that all $\mathbb{G}_{\lambda_a,\lambda_b}$ must have at least two rows. If all rows have k bit, this implies the lower bound of $2k$ bits per gate. In our garbing scheme, we divide $\mathbb{G}_{\lambda_a,\lambda_b}$ into a k-bit(-entropy) part and a 2-bit(-entropy) part. Our k-bit part does not depend on permute bits and has only one row. The 2-bit part has four rows and thus does not contradict the arguments in the lower-bound proof.

Our Construction and the Model of Linear Garbling Schemes We now compare our scheme to the linear garbling model, and explain how it bypasses the lower bound more formally. Since the lower-bound proof only considers a single AND gate, the labels of both input wires can be chosen freely, and we can leave out the ciphertext for difference adjustment in the following discussion.

Our construction does not perfectly fit into the model in two points. The first point is that we use $\mathbb{Z}_{2^k}$ rather than $\mathbb{F}_{2^k}$, simply because we need $+$ and $\oplus$ to be different operations. The second point is that the linear garbling model considers only k-bit values. In contrast, we use oracles with k-bit output and with 2-bit output. The 2-bit oracle is implemented by using the lsb_2 function on the k-bit oracle output. Similarly, we have k-bit ciphertexts and 2-bit ciphertexts.

A garbling algorithm in the linear garbling model consists of five steps. We describe our scheme in these steps, using the same enumeration as in [30]. We omit the tweaks implemented by $\mathsf{nextindex}()$ in the calls to the random oracle H.

1. The garbler chooses several random k-bit values. The only 1-bit randomness considered in the model are the permute bits. In our scheme, the garbler

chooses the random k-bit values K_A^0, K_B^0, and d, and the random bits b_0 and b_1. So we allow 1-bit randomness here, which is only a technical issue.

2. The garbler makes several oracle queries, using the random values from Step 1 as input. The random values and oracle responses form a vector S, on which all following linear operations are performed. In our construction, we have k-bit queries and 2-bit queries. We divide S into the two vectors S_k, containing k-bit values, and S_2, containing 2-bit values. We have:
   k-bit queries:
   $Q_1 := H(K_A^0 + K_B^0)$, $Q_2 := H(K_A^0 + K_B^0 + d)$, $Q_3 := H(K_A^0 + K_B^0 + 2d)$,
   $\Rightarrow S_k = (K_A^0, K_B^0, d, Q_1, Q_2, Q_3)$.
   2-bit queries Q_{4-7}:
   $Q_{2(\lambda_a \oplus a)+(\lambda_B \oplus b)+1} = \mathsf{lsb}_2(H(K_A^0 + ad || K_B^0 + bd))$ for all $(a, b) \in \{0,1\}^2$.

3. The random permute bits λ_A, λ_B and λ_C are chosen.
4. Linear operations are performed on S to compute the input wire labels A_0, A_1, B_0, B_1 and the output wire labels C_0, C_1. The latter can be written as $C_i = \langle C_{\lambda_a,\lambda_b,i}, S\rangle$, $i \in \{0,1\}$, for appropriate vectors $C_{\lambda_a,\lambda_b,i}$, which can depend on permute bits. In our case, we have:

$$(A_0, B_0, A_1, B_1, C_0, C_1) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1-b_0 & b_0 & 0 \\ 0 & 0 & 0 & b_1 & b_1 & 1 \end{pmatrix} S_k$$

   The rows $C_{\lambda_a,\lambda_b,0} = (0,0,0,1-b_0,b_0,0)$ and $C_{\lambda_a,\lambda_b,1} = (0,0,0,b_1,b_1,1)$ define the output labels. They depend on b_0 and b_1, but not on λ_a and λ_b.
5. Ciphertexts are computed: for $i \in [m]$, $G_i = \left\langle G^{(i)}_{\lambda_a,\lambda_b}, S \right\rangle$ for appropriate $G^{(i)}_{\lambda_a,\lambda_b}$, where m is the number of ciphertexts included in the garbled circuit. In our scheme, we have k-bit and 2-bit ciphertexts:

$$G_1^{k-bit} = \left\langle G^{(1)}_{\lambda_a,\lambda_b}, S_k \right\rangle, G_i^{2-bit} = \left\langle G^{(i)}_{\lambda_a,\lambda_b}, S_2 \right\rangle, i = 2, \ldots, 5.$$

   Let $\mathbb{G}_{\lambda_a,\lambda_b}$ be the matrix consisting of the rows $G^{(i)}_{\lambda_a,\lambda_b}$ for $i \in [m]$. We divide $\mathbb{G}_{\lambda_a,\lambda_b}$ into a k-bit part and a 2-bit part. The k-bit part has only one row:

$$\mathbb{G}^{k-bit}_{\lambda_a,\lambda_b} = G^{(1)}_{\lambda_a,\lambda_b} = (0,0,0,1,1,0)$$

   for all $(\lambda_a, \lambda_b) \in \{0,1\}^2$. So our k-bit ciphertext is

$$G_1^{k-bit} := \langle (0,0,0,1,1,0), S_k \rangle = (H(K_A^0 + K_B^0) \oplus H(K_A^0 + K_B^0 + d)).$$

   As we can see, $\mathbb{G}^{k-bit}_{\lambda_a,\lambda_b}$ does not depend on λ_a and λ_b, so changing the permute bits *cannot* change $\mathbb{G}^{k-bit}_{\lambda_a,\lambda_b}$. This seems like a contradiction to the lower-bound proof, which argues that changing the permute bits must change

$\mathbb{G}_{\lambda_a,\lambda_b}$ in order to assign a different pair (α, β) to the label C_1. However, in our construction, the labels C_0 and C_1 only depend on the random bits b_0 and b_1 chosen by the garbler. Thus, the assignment of color bits to output truth values is irrelevant for the computation of the labels on the garbler's side. However, to allow the evaluator to compute the correct output label, a choice bit γ needs to be communicated using ciphertexts which do depend on α and β. These are computed by the rows of the 2-bit part $\mathbb{G}^{2-bit}_{\lambda_a,\lambda_b}$, which takes care of the dependence on permute bits this way. Thus, the k-bit part $\mathbb{G}^{k-bit}_{\lambda_a,\lambda_b}$ can stay unchanged for changing permute bits (and thus consist of only one row), without causing a contradiction to the lower-bound proof.
To enable the evaluator to compute the color bit $(\alpha \oplus \lambda_A)(\beta \oplus \lambda_B) \oplus \lambda_C$ of the output wire, we define the four rows $G^{(2)-(5)}_{\lambda_a,\lambda_b}$ of the 2-bit part such that

$$b^c_{2\alpha+\beta}||b^\gamma_{2\alpha+\beta} = \left\langle G^{(2+2\alpha+\beta)}_{\lambda_A,\lambda_B}, S_2 \right\rangle$$

$$= \mathsf{lsb}_2(H(K_A^{\alpha\oplus\lambda_A}||K_B^{\beta\oplus\lambda_B})) \oplus (((\alpha \oplus \lambda_A)(\beta \oplus \lambda_B) \oplus \lambda_C)||\gamma^{(\alpha,\beta)})$$

for all $(\alpha, \beta) \in \{0,1\}^2$, where

$$\gamma^{(\lambda_A,\lambda_B)} = b_0, \gamma^{(1\oplus\lambda_A,\lambda_B)} = \gamma^{(\lambda_A,1\oplus\lambda_B)} = 1 - b_0, \gamma^{(1\oplus\lambda_A,1\oplus\lambda_B)} = b_1.$$

The order of the rows in $\mathbb{G}^{2-bit}_{\lambda_a,\lambda_b}$ depends on λ_a and λ_b. Thus, in compliance with the lower-bound proof, $\mathbb{G}^{2-bit}_{\lambda_a,\lambda_b}$ is different for each choice of (λ_a, λ_b).

6.3 Further Analyzing the Lower Bound

In 2005, Kolesnikov [13] introduced an information-theoretically secure secret sharing based garbling scheme which requires zero ciphertexts, using only XOR operations. Kolesnikov's construction produces an exponential blow-up of the input key size, so the comparison is slightly unfair. It does not fit into the model for the simple reason that the wire labels of one input wire are not elements in $GF(2^k)$, but have the form $B_L||B_R$ for $B_L, B_R \in GF(2^k)$. Regardless, Kolesnikov's construction seems like a contradiction to the lower bound. We analyze how this "almost linear" construction circumvents the lower bound.

Kolesnikov also introduces an optimization which reduces the blow-up to approximately $\sum d_j^2$, where d_j is the depth of the jth leaf of the circuit. However, the optimization is irrelevant for our analysis.

Outline of Kolesnikov's Construction Kolesnikov's construction works by garbling circuits backwards from output gates to input gates. Consider garbling an AND gate i with yet undefined input wire labels $K^0_A, K^1_A, K^0_B, K^1_B$, and given output wire labels K^0_i and K^1_i. The labels K^0_i and K^1_i are secret-shared in the following way: Assign a *single* random permute bit λ_A to input wire A, no permute bit is assigned to the second input wire B. Choose the input labels K^0_A and K^1_A at random, append λ_A to K^0_A, and $1 - \lambda_A$ to K^1_A. The labels

K_B^0 and K_B^1 each consist of two entries, which are permuted according to λ_A: $K_B^0 := K_i^0 \oplus K_A^{\lambda_A} || K_i^0 \oplus K_A^{1-\lambda_A}$, and $K_B^1 := K_i^{\lambda_A} \oplus K_A^{\lambda_A} || K_i^{1-\lambda_A} \oplus K_A^{1-\lambda_A}$. To evaluate gate i, the evaluator XORs his label K_A with the entry of his label K_B which is indicated by the color bit appended to K_A.

Kolesnikov's Construction and the Lower Bound Kolesnikov's construction is linear in an algebraic sense. It circumvents the lower bound in a way similar to our scheme: the operations performed by the evaluator do not depend on two permute bits. Kolesnikov's construction only assigns permute bits to "A-wires" to indicate which part of the "B-wire" to use. "B-wires" are not assigned any permute or color bit. Thus we have only one bit assigned to four possible input combinations, making claim one in the lower-bound proof meaningless. And in fact, similar to the k-bit part of our construction, the same linear operation is performed for different truth values on the output wire.

6.4 Conclusion

If less than two rows imply less than two possible operations, only one or no choice bit is needed, making claim one in the lower-bound proof meaningless. It is left for future work whether our observations can be used to break the lower bound and omit even the small ciphertexts altogether without input key blow-up. It would be interesting whether garbling schemes with less than two k-bit rows can be constructed without sacrifising free XOR.

Acknowledgements

We thank the reviewers for their helpful and constructive comments.

A Combining our Construction with Half Gates

By combining our technique with the half gate construction [30], we can garble the first two gate levels in a circuit with only one ciphertext per AND gate and 0 ciphertexts per XOR gate, if the circuit layout is fortunate. Each circuit input is known either by the garbler or the evaluator, so one could argue that all input gates can be garbled as half gates. A similar technique is used by Huang et al. [8], who use generator half gates as input gates. We need to modify the half gate technique such that output wires i which are used as inputs for AND gates get an additive global difference d such that $K_i^1 = K_i^0 + d$, and output wires j which are used as input to an XOR gate get a global difference Δ such that $K_j^1 = K_j^0 \oplus \Delta$. We cannot do both at the same time, so this only saves ciphertexts when most output wires of input gates go to either an AND gate or an XOR gate, but not both. If this is the case, the next level can be garbled with our construction using one ciphertext for most AND gates, and zero ciphertexts for most XOR gates. A half gate can produce an additive difference in its output wire using

the following modification: A generator half gate with input a known to the garbler produces the ciphertexts $H(K_B) \oplus K_i^0$ and $H(K_B \oplus \Delta) \oplus (K_i^0 + ad)$, of which one is set to the zero string as in the original scheme. It is evaluated as in the original half gate construction. An evaluator half gate, where the evaluator knows input a, and gets input labels K_A^a and K_B, consists of the ciphertexts $G_1 = H(K_A^0) \oplus K_C^0$, which is set to the zero string by setting $K_C^0 = H(K_A^0)$, and $G_2 = H(K_A^1) \oplus (K_C^0 - B)$. In the evaluator half gate, we require an additive difference $K_B^1 - K_B^0 = d$ for the labels of input wire B. If $a = 0$, it is evaluated by computing $H(K_A^0) \oplus G_1$. If $a = 1$, the evaluator computes $(G_2 \oplus H(K_A^1)) + K_B$.

References

1. abhi shelat and C. Shen. Two-output secure computation with malicious adversaries. In *EUROCRYPT*, pages 386–405, 2011.
2. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513, 1990.
3. M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on Security and Privacy*, pages 478–492. IEEE Computer Society, 2013.
4. M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In *ACM CCS*, pages 784–796. ACM, 2012.
5. 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.
6. T. Frederiksen, T. Jakobsen, J. Nielsen, P. Nordholt, and C. Orlandi. MiniLEGO: Efficient secure two-party computation from general assumptions. In *EUROCRYPT*, pages 537–556. 2013.
7. S. Gueron, Y. Lindell, A. Nof, and B. Pinkas. Fast garbling of circuits under standard assumptions. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS*, pages 567–578. ACM, 2015.
8. Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*. The Internet Society, 2012.
9. Y. Huang, J. Katz, and D. Evans. Efficient secure two-party computation using symmetric cut-and-choose. In *CRYPTO*, pages 18–35, 2013.
10. Y. Huang, J. Katz, V. Kolesnikov, R. Kumaresan, and A. J. Malozemoff. Amortizing garbled circuits. In *CRYPTO*, pages 458–475, 2014.
11. J. Katz and L. Malka. Constant-round private function evaluation with linear complexity. In D. H. Lee and X. Wang, editors, *ASIACRYPT*, pages 556–571. Springer Berlin Heidelberg, 2011.
12. C. Kempka, R. Kikuchi, S. Kiyoshima, and K. Suzuki. Garbling scheme for formulas with constant size of garbled gates. In T. Iwata and J. H. Cheon, editors, *ASIACRYPT*, pages 758–782. Springer Berlin Heidelberg, 2015.
13. V. Kolesnikov. Gate evaluation secret sharing and secure one-round two-party computation. In *ASIACRYPT*, pages 136–155, 2005.
14. V. Kolesnikov, P. Mohassel, and M. Rosulek. FleXOR: Flexible garbling for xor gates that beats free-xor. In J. Garay and R. Gennaro, editors, *CRYPTO 2014*, pages 440–457. Springer Berlin Heidelberg, 2014.
15. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP*, pages 486–498, 2008.

16. V. Kolesnikov and T. Schneider. A practical universal circuit construction and secure evaluation of private functions. In G. Tsudik, editor, *Financial Cryptography and Data Security*, pages 83–97. Springer-Verlag, 2008.
17. Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO*, pages 1–17, 2013.
18. Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In M. Naor, editor, *EUROCRYPT 2007*, pages 52–78. Springer Berlin Heidelberg, 2007.
19. Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In *TCC*, pages 329–346, 2011.
20. 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.
21. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - a secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.
22. M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *ACM conference on Electronic commerce*, pages 129–139, 1999.
23. J. Nielsen and C. Orlandi. LEGO for two-party secure computation. In O. Reingold, editor, *TCC*, pages 368–386. Springer Berlin Heidelberg, 2009.
24. J. B. Nielsen and S. Ranellucci. Foundations of reactive garbling schemes. Cryptology ePrint Archive, Report 2015/693, 2015. http://eprint.iacr.org/2015/693.
25. A. Paus, A.-R. Sadeghi, and T. Schneider. Practical secure evaluation of semi-private functions. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *ACNS*, pages 89–106. Springer Berlin Heidelberg, 2009.
26. B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In M. Matsui, editor, *ASIACRYPT 2009*, pages 250–267. Springer Berlin Heidelberg, 2009.
27. B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In *ASIACRYPT*, pages 250–267, 2009.
28. L. G. Valiant. Universal circuits (preliminary report). In *STOC*, pages 196–203. ACM, 1976.
29. A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.
30. S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *EUROCRYPT*, 2015.