AuthOr: Lower Cost Privacy-Free Garbling of Arbitrary Boolean Circuits

Osman Biçer and Ali Ajorian

University of Basel, Switzerland

Abstract. Privacy-free garbling schemes of Frederiksen et al. (Eurocrypt '15) are designed to satisfy only the authenticity criterion of Bellare et al. (ACM CCS '12) and to be more efficient than full-fledged garbling schemes. In this work, we improve the state-of-the-art privacy-free version of half-gates (HG) garbling of Zahur et al. (Crypto '15) by allowing an AND gate to be garbled with zero ciphertexts whenever at least one of its input wires is freely settable by the garbler. Our full solution, AuthOr, then combines the ideas from the information-theoretic garbling of Kondi and Patra (Crypto '17) with the improved HG-based scheme we obtain. AuthOr has a lower communication cost (i.e., garbled-circuit size) than HG garbling without any additional security assumptions. Theoretically, AuthOr's garbled-circuit size reduction over HG garbling ranges from 0% to 100%, with the exact improvement depending on the circuit structure. We have implemented our scheme and evaluated it on various circuits constructed by independent researchers. Our experimental results show that, in practice, the garbled-circuit size reduction can reach roughly 98%.

Keywords: Garbled circuits · Privacy-free garbling · Verifiable computing · Zero-knowledge proofs.

1 Introduction

Garbled circuits. Introduced by Andrew Yao [Yao82], garbled circuits are a fundamental primitive in cryptography with applications to secure two-party computation [Yao86; LP07; LP11; LR15], secure multi-party computation [Lin+15; WRK17], identity-based encryption [DG17], verifiable outsourced computing [GGP10; MLR23], and zero-knowledge protocols [JKO13; FNO15; CGM16; KP17; Gan+18]. In its generic form, a garbled-circuit (GC) protocol is a two-party protocol between a garbler and an evaluator. The garbler garbles a Boolean circuit f known to both parties, and the evaluator evaluates the garbled circuit on encoded inputs to obtain encoded outputs. Classically, the garbling procedure assigns two ℓ-bit keys[1] W_i^0 and W_i^1 (for truth values 0 and 1) to each wire i and produces the garbled circuit $\hat{F}$. The evaluator receives $\hat{F}$ together with the key W_i corresponding to the true value w_i of each input wire i, then proceeds

[1] Throughout this work, $\ell \in O(\lambda)$ is the key size based on the security parameter λ.

gate-by-gate in topological order to obtain the output-wire keys. Bellare et al. [BHR12] formalized garbled circuits as a cryptographic primitive with three security notions: privacy, obliviousness, and authenticity. Privacy and obliviousness concern hiding the wire values from the evaluator, while authenticity prevents the evaluator from forging encodings of "fake" outputs that do not result from honest evaluation. The efficiency of a garbled circuit is typically measured by the computational cost for the garbler, the computational cost for the evaluator, and the size of the garbled circuit. The last of these is often the main bottleneck [ZRE15], as it directly determines the communication cost.

Privacy-free garbling. Frederiksen et al. [FNO15] showed that authenticity-oriented garbling schemes that drop privacy and obliviousness can be significantly more efficient than full-fledged schemes. The key idea is that when the evaluator already knows the true values w_a and w_b of a gate's inputs, it can exploit this knowledge during evaluation, reducing overhead. Such schemes are useful for verifiable outsourced computation [GGP10; MLR23] and for efficient zero-knowledge proof protocols [JKO13; FNO15; CGM16; KP17; Gan+18]. The latter application additionally requires a verifiability property so that a malicious verifier cannot construct a garbled circuit whose output keys would reveal the prover's witness.

There is a quest for achieving an optimal privacy-free garbling scheme. Previous works include the privacy-free schemes by [FNO15], i.e., GRR2 (compatible with FreeXOR), GRR1, and GRR1 with FleXOR. The most relevant techniques to our work are the privacy-free version of half-gates (HG) garbling [ZRE15] and the information-theoretic (IT) garbling [KP17]. If the circuit is a Boolean formula, i.e., it consists of only fan-out-1 gates, IT garbling is optimal with a garbled circuit of size zero and no cryptographic operations. Although Boolean formulas are already an important class of circuits with zero-knowledge proof-of-satisfiability applications [Gen+13], this is still quite restrictive. According to the Shannon bound [Sha49; BP23], the great majority of circuits with n input bits have size $\Theta(2^n/n)$, whereas a Boolean formula with n input bits can have size $O(n)$. For fairness, we converted an example generic circuit into a Boolean formula (see Appendix A), which resulted in a circuit of size exponential in the size of the original circuit, implying that both garbling and evaluation have exponential complexity[2]. To the best of our knowledge, before this work, HG garbling has remained the most efficient privacy-free garbling scheme applicable to all Boolean circuits.

Our contributions. We provide a novel privacy-free garbling scheme that is FreeXOR-compatible and has lower AND-gate costs than HG garbling. Our technique benefits from the incorporation of IT garbling and a more efficient version of HG garbling that we also propose here. More concretely, we achieve the following:

1. In Section 3, along with the previously developed garbling schemes that we use in our final solution, we propose an HG-based AND-gate garbling scheme

[2] The authors also acknowledge this as "not generally efficient for circuits that are not largely formulaic" in [KP17].

that is FreeXOR-compatible. This scheme allows the garbler to garble an AND gate with zero ciphertexts whenever one of the gate's input wires has not already been set by another gate that precedes it in topological order.
2. In Section 4, we combine the scheme mentioned in the previous step with IT garbling to obtain our final privacy-free garbling scheme AuthOr. The garbling algorithm Gb of AuthOr selects, for each gate, the most suitable garbling technique and benefits from IT garbling and our proposal from the previous step as much as possible, while the evaluation algorithm Ev detects for each gate which technique was used and proceeds accordingly. Theoretically, for any circuit, compared with HG garbling AuthOr achieves: (i) the same computational security guarantee based on the circular correlation robustness (CCR) of the hash function, (ii) a 0% to 100% reduction in garbled-circuit size, and (iii) a 0% to 100% reduction in the number of hash-function evaluations for both the garbler and the evaluator. The exact improvement depends on the circuit.
3. In Section 5, we provide our implementation results for both our scheme and HG garbling by testing those schemes on the Boolean circuits available from [Arc+; Jen]. While for the cryptographic circuits our efficiency improvement can be considered marginal, for other circuits we achieve up to a 98.44% reduction in garbled-circuit size without increasing the computational costs.
4. In Section B, we show the security of our scheme based on the security (i.e., authenticity of [BHR12] and verifiability of [JKO13]) of HG garbling, which in turn is based on the circular correlation robustness of the hash scheme used [ZRE15].

Comparison with the state-of-the-art. Table 1 compares our scheme with the mentioned privacy-free garbling schemes, the symmetric-key-based full-fledged fast garbling of [Gue+15], and three-halves of [RR21]. Circular correlation robustness (CCR) [Cho+12] is a standard assumption in FreeXOR- and FleXOR-compatible schemes, while the security of the non-compatible ones can be based directly on the pseudorandomness of the underlying primitive.

Comparison to other proposals. We would like to mention some other approaches than symmetric-key-based ones for fairness. The gate-ID-based garbling scheme of [Kem+15] uses trapdoor permutations (i.e., public-key operations) to garble a circuit, resulting in a garbled circuit with 4 bits per gate and an additional 2 ciphertexts for AND gates that share input wires with other gates. While in terms of garbled-circuit size $[4g, 4g + (g-1) \cdot 2ct]$ the scheme of [Kem+15] improves efficiency compared with previous schemes, it relies on public-key primitives. Other approaches include succinct garbling schemes [Can+15; AL18] and reusable garbling schemes [Gol+; Bon+14; Agr17]. However, they induce heavy computational costs and are based on strong assumptions, e.g., the existence of indistinguishability obfuscation / functional encryption / fully homomorphic encryption, the learning-with-errors (LWE) assumption, or assumptions on multilinear maps. While one recent work [Liu+24] based on the LWE assumption has a very low garbled size (g bits), its concrete computational costs are heavy, with a large number of homomorphic operations per gate.

Technique	Gb cost	Ev cost	GC size (bits)	Assump.
GRR2+FreeXOR	$3g_A s$	$g_A s$	$2g_A \ell$	CCR
GRR1	$3g_A s$	$g_A s$	$g\ell$	PRF
GRR1+FleXOR	$3g_A s$	$g_A s$	$[g_A \ell, (g_A + 2g_X)\ell]$	CCR
Fast garbling	$(4g_A + 3g_X)s$	$(2g_A + 1.5g_X)s$	$(2g_A + g_X)\ell$	PRF
Three halves	$[3g_A s, 6g_A s]$	$[1.5g_A s, 3g_A s]$	$1.5g_A \ell + 5g$	CCR
IT	$[\Omega(g), exp(g)]$	$[\Omega(g), exp(g)]$	0	None
HG	$2g_A s$	$g_A s$	$g_A \ell$	CCR
AuthOr	$\mathbf{[0, 2g_A s]}$	$\mathbf{[0, g_A s]}$	$\mathbf{[0, g_A \ell]}$	**CCR**

Table 1: Comparison of state-of-the-art garbling schemes with authenticity, evaluating garbling (Gb), evaluation (Ev), GC size, and security assumptions (CCR=circular correlation robustness and PRF=pseudo-random function). Circuit costs are parameterized by the total number g of gates, the total number g_A of AND gates, the total number g_X of XOR gates, symmetric-key operation cost s, and ciphertext size $\ell \in O(\lambda)$. Ranges reflect variations across circuit structures for f. All schemes include additional $\Theta(g)$ computation overheads from basic operations.

2 Background on Circuits and Privacy-Free Garbling

Boolean circuits and notation. A Boolean circuit f is a directed acyclic graph with gates, circuit inputs, and circuit outputs as vertices, and wires as edges. Throughout this work, we use the following notation. $i.\mathsf{GateInputs}$ denotes the input wire(s) of the gate i. $f.\mathsf{TGates}$ denotes the set of all gates with type T (e.g. AND, XOR, NOT) in the circuit f. $i.\mathsf{Type}$ denotes the type variable associated with the wire i. Each gate i is identified by the index of its output wire i.

Garbled circuits and notation. Throughout this work,

- $a \twoheadleftarrow B$ denotes that a is picked uniformly at random from the set B,
- $a \leftarrow B$ denotes that a is set as an execution of the algorithm B,
- $\ell \in O(\lambda)$ is the key length chosen based on the security parameter and the symmetric-key primitive,
- λ denotes the security parameter. $f.\mathsf{Inputs}$ denotes the input wires of the circuit f,
- $f.\mathsf{Outputs}$ denotes the output wires of the circuit f,
- $\hat{a}$ denotes a straightforward collection of the values produced as a_i,
- $\hat{F}.\mathsf{Next}$ denotes the iterator over the garbled circuit $\hat{F}$.

This work follows the garbling-scheme abstraction introduced by [BHR12]. A garbling scheme is composed of the following algorithms:

- Gb: Takes as input 1^λ and a Boolean circuit f, and outputs $\hat{F}$, $\hat{e}$, and $\hat{d}$, which denote the garbled circuit, encoding information, and decoding information, respectively.

– Ev: Takes as input f, $\hat{F}$, and $\hat{X}$, and outputs the garbled output $\hat{Y}$.
– En: Takes as input $\hat{e}$ and the plaintext input bit string $\hat{x}$ to f, and outputs the garbled input $\hat{X}$.
– De: Takes as input $\hat{d}$ and $\hat{Y}$, and outputs the plaintext output bit string $\hat{y}$ or $\perp$.
– Ve: Takes as input $\hat{e}$, $\hat{F}$, and f and outputs a single bit b (1 for verified).

Definition 1 (Correctness). *A garbling scheme is correct if for every circuit* $f : \{0,1\}^n \rightarrow \{0,1\}^m$ *and its input string* $\hat{x} \in \{0,1\}^n$, $f(x) = \mathsf{De}\Big(\hat{d}, \mathsf{Ev}\big(f, \hat{F}, \mathsf{En}(\hat{e}, \hat{x})\big)\Big)$ *holds for each execution* $(\hat{F}, \hat{e}, \hat{d}) \leftarrow \mathsf{Gb}(1^\lambda, f)$.

As we are interested in privacy-free garbling schemes, we provide the authenticity definition of [BHR12] below. The definition briefly ensures that no PPT adversary can come up with incorrect output-wire keys that are decodable by De.

Definition 2 (Authenticity). *A garbling scheme is authentic if for every PPT adversary* $\mathcal{A}$, *every circuit* $f : \{0,1\}^n \rightarrow \{0,1\}^m$ *and its input string* $\hat{x} \in \{0,1\}^n$, *there exists a negligible function* n *such that:*

$$Pr[(\hat{Y} \neq \mathsf{Ev}(f, \hat{F}, \hat{X})) \wedge (\mathsf{De}(\hat{d}, \hat{Y}) \neq \perp) : (\hat{F}, \hat{e}, \hat{d}) \leftarrow \mathsf{Gb}(1^\lambda, f),$$

$$\hat{X} \leftarrow \mathsf{En}(\hat{e}, \hat{x}), \hat{Y} \leftarrow \mathcal{A}(\hat{F}, \hat{X})] \leq \mathsf{n}(\lambda).$$

Additionally, we require the privacy-free garbling scheme to have the verifiability property of [JKO13], so that the obtained garbling scheme can easily be plugged into their given zero-knowledge proof-of-knowledge (ZKP) protocol. In this ZKP scheme, the verifier plays the garbler role during the garbled-circuit subprotocol, yet security against a malicious verifier is still achieved thanks to the verifiability of the garbled circuit. We note that the definition below is a slightly modified version of the one given in [JKO13] for arbitrary output lengths by giving y as input to Ext.

Definition 3 (Verifiability). *A garbling scheme is verifiable if for every PPT adversary* $\mathcal{A}$, *every circuit* $f : \{0,1\}^n \rightarrow \{0,1\}^m$, *there exists an expected polynomial-time algorithm* Ext *and a negligible function* n *such that:*

1. $Pr[\mathsf{Ev}(f, \hat{F}, \hat{X}_0) \neq \mathsf{Ev}(f, \hat{F}, \hat{X}_1),\ \mathsf{Ve}(\hat{e}, \hat{F}, f) = 1,\ f(\hat{x}_0) = f(\hat{x}_1) :\ (\hat{F}, \hat{e}, \hat{x}_0, \hat{x}_1) \leftarrow \mathcal{A}(1^\lambda, f),\ \hat{X}_0 \leftarrow \mathsf{En}(\hat{e}, \hat{x}_0),\ \hat{X}_1 \leftarrow \mathsf{En}(\hat{e}, \hat{x}_1)] \leq \mathsf{n}(\lambda)$, *and*
2. $Pr[\mathsf{Ext}\big(\hat{F}, \hat{e}, f(\hat{x})\big) \neq \mathsf{Ev}(f, \hat{F}, \hat{X}),\ \mathsf{Ve}(\hat{e}, \hat{F}, f) = 1 : (\hat{F}, \hat{e}, \hat{x}) \leftarrow \mathcal{A}(1^\lambda, f), \hat{X} \leftarrow \mathsf{En}(\hat{e}, \hat{x})] \leq \mathsf{n}(\lambda)$.

3 Gate Garbling Algorithms and Our Optimization on Half Gates

In this section, we elaborate on the algorithms that we use in AuthOr, i.e., privacy-free half-gates-based $\mathsf{HG2}$, $\mathsf{HG1}$, $\mathsf{HG0}$, and information-theoretic ITAND

procedure $\mathsf{GbHG2}(i, W_a^0, W_b^0, \Delta)$: $W_i^0 \leftarrow H(i, W_a^0)$ $F_i \leftarrow H(i, W_a^0) \oplus$ $\quad H(i, W_a^0 \oplus \Delta) \oplus W_b^0$ **return** (F_i, W_i^0)	**procedure** $\mathsf{GbHG1}(i, W_a^0, \Delta)$: $W_i^0 \leftarrow H(i, W_a^0)$ $W_b^0 \leftarrow H(i, W_a^0) \oplus$ $\quad H(i, W_a^0 \oplus \Delta)$ **return** (W_b^0, W_i^0)	**procedure** $\mathsf{GbHG0}(i, \Delta)$: $W_a^0 \twoheadleftarrow \{0,1\}^\ell$ $W_i^0 \leftarrow H(i, W_a^0)$ $W_b^0 \leftarrow H(i, W_a^0) \oplus H(i, W_a^0 \oplus \Delta)$ **return** (W_a^0, W_b^0, W_i^0)
procedure $\mathsf{GbFreeXOR1}(W_a^0)$: $W_b^0 \twoheadleftarrow \{0,1\}^\ell$ $W_i^0 \leftarrow W_a^0 \oplus W_b^0$ **return** (W_b^0, W_i^0)	**procedure** $\mathsf{GbFreeXOR0}()$: $W_a^0 \twoheadleftarrow \{0,1\}^\ell$ $W_b^0 \twoheadleftarrow \{0,1\}^\ell$ $W_i^0 \leftarrow W_a^0 \oplus W_b^0$ **return** (W_a^0, W_b^0, W_i^0)	**procedure** $\mathsf{GbFreeXOR2}(W_a^0, W_b^0)$: $W_i^0 \leftarrow W_a^0 \oplus W_b^0$ **return** W_i^0
procedure $\mathsf{GbITAND}(W_i^0, W_i^1)$: $W_a^0, W_b^0 \leftarrow W_i^0, W_i^0$ $W_a^1 \twoheadleftarrow \{0,1\}^\ell$ $W_b^1 \leftarrow W_a^1 \oplus W_i^1$ **return** $(W_a^0, W_a^1, W_b^0, W_b^1)$	**procedure** $\mathsf{GbITXOR}(W_i^0, W_i^1)$: $W_a^1 \twoheadleftarrow \{0,1\}^\ell$ $W_b^1 \leftarrow W_a^1 \oplus W_i^0$ $W_a^0 \leftarrow W_b^1 \oplus W_i^1$ $W_b^0 \leftarrow W_a^1 \oplus W_i^1$ **return** $(W_a^0, W_a^1, W_b^0, W_b^1)$	**procedure** $\mathsf{GbBwNOT}(W_i^0, W_i^1)$: $(W_a^0, W_a^1) \leftarrow (W_i^1, W_i^0)$ **return** (W_a^0, W_a^1) **procedure** $\mathsf{GbFwNOT}(W_a^0, \Delta)$: $W_i^0 \leftarrow W_a^0 \oplus \Delta$ **return** W_i^0

Table 2: Gate garbling algorithms that are used in AuthOr.

for garbling AND gates; FreeXOR-based FreeXOR2, FreeXOR1, FreeXOR0, and information-theoretic ITXOR for garbling XOR gates; and FwNOT and BwNOT for garbling NOT gates. Among the given gate-garbling techniques, HG1 and HG0 are our contribution. We highlight that all the given algorithms garble gates in the forward direction in topological order, except for the IT-garbling algorithms, which garble in the backward direction from outputs to inputs; we postpone how to determine which algorithm will be used for a given gate to Section 4. Tables 2, 3, and 4 show the formal garbling, evaluation, and verification algorithms, respectively.

FreeXOR (FreeXORz) [KS08]. The FreeXOR technique [KS08] requires that the keys for each wire i have the same offset Δ, so that $W_i^1 = W_i^0 \oplus \Delta$ always holds. Given the input-wire keys W_a^0 and W_b^0, FreeXOR sets the output-wire key $W_i^0 \leftarrow W_a^0 \oplus W_b^0$. Depending on the number z of inputs that have been set before garbling the XOR gate, this scheme is called FreeXORz for $z \in \{0, 1, 2\}$ and the variants differ by picking the unset wire key uniformly at random. The corresponding garbling algorithms are named GbFreeXOR2, GbFreeXOR1, and GbFreeXOR0; the evaluation algorithm for all XOR gates is EvXOR.

Privacy-free Half-Gates Garbling (HGz). Here we provide the HG2 of [ZRE15] and our contributions HG1 and HG0 obtained from it.

The HG2 algorithm [ZRE15]. The privacy-free version of the half-gates protocol [ZRE15] is compatible with FreeXOR, i.e., the keys for each wire w_i have the same offset Δ. Here, we provide a slightly modified version such that the hash call H takes as input the id i of the gate instead of the one in [ZRE15][3]. Given the input-wire keys W_a^0, W_b^0, and the offset Δ, the garbling algorithm GbHG2 for the gate with index i results in one ciphertext which is set as $F_i \leftarrow$

[3] In [ZRE15], the hash function takes as input a value that is incremented with each AND gate garbled. This modification does not affect the security at all, as their proof is based on the uniqueness of this value.

```
procedure EvHG2(i, F_i, W_a, W_b, w_a, w_b):
  if w_a = 0 then
    W_i ← H(i, W_a)
  else
    W_i ← H(i, W_a) ⊕ W_b ⊕ F_i
  return W_i

procedure EvXOR(W_a, W_b, w_a, w_b):
  W_i ← W_a ⊕ W_b
  return W_i

procedure EvNOT(W_a):
  W_i ← W_a
  return W_i
```

```
procedure EvHG0&1(i, W_a, W_b, w_a, w_b):
  if w_a = 0 then
    W_i ← H(i, W_a)
  else
    W_i ← H(i, W_a) ⊕ W_b
  return W_i

procedure EvITAND(W_a, W_b, w_a, w_b):
  if w_a = 0 then
    W_i ← W_a
  else if w_b = 0 then
    W_i ← W_b
  else
    W_i ← W_a ⊕ W_b
  return W_i
```

Table 3: Gate evaluation algorithms that are used in AuthOr.

$H(i, W_a^0) \oplus H(i, W_a^1) \oplus W_b^0$ where $H : \{0,1\}^* \to \{0,1\}^\ell$ is a hash function with circular correlation robustness [Cho+12]. The output-wire key for 0 is set as $W_i^0 = H(i, W_a^0)$. As this scheme requires that the input keys for both wires have already been fixed before the garbling, we call this gate garbling HG2. We note that among all gate-garbling schemes that we use, this one is the only one that results in a ciphertext that needs to be sent to the evaluator. If the truth value $w_a = 0$, the evaluation algorithm EvHG2 sets $W_i \leftarrow H(i, W_a)$. Otherwise, $W_i \leftarrow F_i \oplus H(i, W_a) \oplus W_b$.

Our contribution: The HG1 and HG0 algorithms. If the keys of only one input wire have not been fixed before the garbling of a gate, then the garbler can choose such that $F_i = 0^\ell$ and it does not need to be sent anymore. More concretely, given the input-wire key W_a^0 and the offset Δ, the key of the output wire i is set as $W_i^0 = H(i, W_a^0)$ by GbHG1, and the key for w_b is set as $W_b^0 \leftarrow H(i, W_a^0) \oplus H(i, W_a^1)$. If none of the wires were set before the gate being garbled, one wire key is picked randomly as $W_a^0 \twoheadleftarrow \{0,1\}^\ell$ and the algorithm GbHG0 continues in the same manner as HG1. Given the input keys W_a and W_b, if $w_a = 0$, the evaluation algorithm EvHG0&1 for both HG1 and HG0 sets $W_i \leftarrow H(i, W_a)$. Otherwise, $W_i \leftarrow H(i, W_a) \oplus W_b$.

Information-Theoretic (IT) Garbling [KP17]. Kondi and Patra [KP17] provide an elegant way of privacy-free garbling of AND and XOR gates that does not require costly hash-function calls and results in size-zero garbled circuits. However, if during the evaluation of an AND gate $w_a = 0$ and $w_b = 1$, their scheme leaks both keys W_b^0 and W_b^1 as it sets $W_a^0 = W_b^0$ during garbling. Yet, the authors prove in their work that this leakage is not a problem (as the output-wire key is never leaked) as long as there are no fan-out-2 gates in the circuit. Due to this leakage, their scheme does not have a global offset Δ. Nevertheless, the authors still show that XOR gates can be garbled with a local offset "freely". The garbling is "backward", i.e., gates are visited in inverse topological order, so that when a gate is being garbled, either its output keys are already set or the garbler picks them randomly as $W_i^0 \twoheadleftarrow \{0,1\}^\ell$ and $W_i^1 \twoheadleftarrow \{0,1\}^\ell$.

The ITAND algorithm. Given the output keys W_i^0 and W_i^1, the garbling algorithm GbITAND sets the keys for the truth value zero for both input wires

procedure $\mathsf{VeHG2}(i, W_a^0, W_a^1, W_b^0, W_b^1, F_i)$:	**procedure** $\mathsf{VeHG0\&1}(i, W_a^0, W_a^1, W_b^0, W_b^1)$:
if $W_a^0 \oplus W_a^1 \neq W_b^0 \oplus W_b^1$	**if** $W_a^0 \oplus W_a^1 \neq W_b^0 \oplus W_b^1$
return 0	**return** 0
$\Delta \leftarrow W_a^0 \oplus W_a^1$	$\Delta \leftarrow W_a^0 \oplus W_a^1$
$K_1 \leftarrow H(i, W_a^0)$	$K_1 \leftarrow H(i, W_a^0)$
$K_2 \leftarrow W_b^0 \oplus H(i, W_a^0 \oplus \Delta) \oplus F_i$	$K_2 \leftarrow W_b^0 \oplus H(i, W_a^0 \oplus \Delta)$
if $K_1 \neq K_2$	**if** $K_1 \neq K_2$
return 0	**return** 0
return $(K_1, K_1 \oplus \Delta)$	**return** $(K_1, K_1 \oplus \Delta)$
procedure $\mathsf{VeXOR}(W_a^0, W_a^1, W_b^0, W_b^1)$:	**procedure** $\mathsf{VeNOT}(W_a^0, W_a^1)$:
if $W_a^0 \oplus W_a^1 \neq W_b^0 \oplus W_b^1$	**return** (W_a^1, W_a^0)
return 0	
return $(W_a^0 \oplus W_b^0, W_a^0 \oplus W_b^1)$	**procedure** $\mathsf{VeITAND}(W_a^0, W_a^1, W_b^0, W_b^1)$:
	if $W_a^0 \neq W_b^0$
	return 0
	return $(W_a^0, W_a^1 \oplus W_b^1)$

Table 4: Gate verification algorithms that are used in AuthOr.

as $W_a^0 \leftarrow W_i^0$ and $W_b^0 \leftarrow W_i^0$. Next, it picks $W_a^1 \twoheadleftarrow \{0,1\}^\ell$ and computes $W_b^1 \leftarrow W_a^1 \oplus W_i^1$. During evaluation, $\mathsf{EvITAND}$ sets $W_i \leftarrow W_a$ if $w_a = 0$, $W_i \leftarrow W_b$ if $w_a = 1$ and $w_b = 0$, and $W_i \leftarrow W_a \oplus W_b$ otherwise.

The ITXOR algorithm. Given the output keys W_i^0 and W_i^1, the garbling algorithm $\mathsf{GbITXOR}$ sets the local offset for the XOR gate being garbled as follows. It picks $W_a^1 \twoheadleftarrow \{0,1\}^\ell$ and computes $W_b^1 \leftarrow W_a^1 \oplus W_i^0$, $W_a^0 \leftarrow W_b^1 \oplus W_i^1$, and $W_b^0 \leftarrow W_a^1 \oplus W_i^1$. During evaluation, EvXOR sets $W_i \leftarrow W_a \oplus W_b$.

Garbling NOT Gates [Kol05; KP17]. In both the forward and backward directions, NOT gates can be freely garbled by simply swapping the association of keys for 0 and 1 as in [Kol05; KP17]. That is, given a NOT gate with input wire a and output wire i, $W_a^0 = W_i^1$ and $W_a^1 = W_i^0$; based on the direction in which this is done, we name the algorithms $\mathsf{GbFwNOT}$ for the forward direction and $\mathsf{GbBwNOT}$ for the backward direction. The evaluation algorithm EvNOT simply copies the key from the input to the output in the forward direction, i.e., the key does not change while the truth value changes.

Verification Algorithms. Table 4 shows the verification algorithms for the provided gate-garbling algorithms. Given both keys for each input wire, they check whether a gate is garbled correctly. If so, they return 1 and the keys for the output wire; otherwise, they return 0. We have obtained $\mathsf{VeITAND}$ from [KP17] as the verification algorithm for the IT-garbled AND gates, which checks that the output key for 0 that could be found in all cases is equal. The verification algorithm VeXOR for XOR gates executes based on whether both input wires have the same offset. We constructed the half-gates AND verification algorithms $\mathsf{VeHG2}$ and $\mathsf{VeHG0\&1}$ based on the check that the output key for 0 that could be found in all cases is equal, similar to $\mathsf{VeITAND}$. We highlight that VeXOR, $\mathsf{VeHG2}$, and $\mathsf{VeHG0\&1}$ also check whether both input wires have the same offset.

4 Our Privacy-Free Garbling Protocol

We provide our AuthOr scheme in Table 5 that makes use of the gate-garbling algorithms in Section 3. We assume both input wires for each AND or XOR gate are distinct, as otherwise our scheme is not immune to the attack mentioned in [NS23][4].

Initialization of our Gb algorithm. The Gb algorithm starts by choosing a global offset Δ. The type of each wire is set by Gb deterministically and derived from the circuit topology. During the initialization, each wire is set as follows:

- TypeS, if that wire is input to at most a single gate.
- TypeM, otherwise.

Here, we do not propose a method for this labeling, as there are simple ways to do this in $O(g)$ that are derived from generic topological-sorting algorithms. Note that the wire labels will be updated again deterministically during the forward and backward phases of the garbling algorithm, which is discussed below.

Forward and backward phases of Gb. In our solution, we attempt to use IT garbling for as many gates as possible. To solve the wire-key leakage issue of IT garbling mentioned in Section 3, our garbling algorithm Gb has two phases: a forward phase, i.e., garbling in topological order with GbFwNOT, GbHGz, and GbFreeXORz using a global offset Δ; and a backward phase, i.e., garbling in inverse topological order with GbBwNOT, ITAND, and ITXOR.

Determination of the gate-garbling algorithms. The gate-garbling algorithm chosen for a gate is based on the types of its input wires. During the forward phase, only the gates that would result in a problem for IT garbling (if they were left to the backward phase) are garbled, and the offsets of the input and output wires of those gates are set to Δ. Gates with a TypeM input wire cannot be left to the backward phase (and should be garbled during the forward phase), as IT garbling results in the leakage of both keys for a wire, which is known to be a problem if that wire is input to more than one gate. Also, during the forward phase of Gb, when the keys of a wire are determined with offset Δ, we cannot leave that wire to the backward phase, as the leakage of both keys means leakage of Δ and the compromise of the security of all wires with the offset Δ. Hence, during the forward phase Gb replaces such wire types with TypeF and also garbles each gate that those wires are input to. Gb leaves for the backward phase only the gates whose both inputs are TypeS wires, when the time of garbling that gate comes. We highlight that the keys for these wires cannot be set later during the forward phase, due to the following theorem:

Theorem 1. *Once a wire that is input to a gate i is labeled as* TypeS *by the* Gb *algorithm, then during the forward phase its label may be changed into* TypeF *only when a gate $j \leq i$ is being garbled.*

Proof. Wires are initially labeled as either TypeS or TypeM, and the algorithm never reassigns a wire to TypeS later. Similarly, a wire labeled TypeS cannot become TypeM, since TypeM is only assigned at initialization.

[4] If this is not the case, we refer to their work for the possible countermeasures.

procedure $\mathsf{Gb}(1^\lambda, f)$:
  $\Delta \twoheadleftarrow \{0,1\}^\ell$ s.t. $\ell \in O(\lambda)$
  label each wire as TypeM or TypeS
  $\{Forward\ phase\}$
  for each gate $i \in f$ $\{in\ topo.\ order\}$ **do**
    if $i \in f.\mathsf{NOTGates}$ **then**
      $a \leftarrow i.\mathsf{GateInputs}$
      if $a.\mathsf{Type} = \mathsf{TypeM}$ **then**
        $W_a^0 \twoheadleftarrow \{0,1\}^\ell$
      if $a.\mathsf{Type} \neq \mathsf{TypeS}$ **then**
        $W_i^0 \leftarrow \mathsf{GbFwNOT}(W_a^0)$

    else if $i \in f.\mathsf{XORGates}$ **then**
      $\{a,b\} \leftarrow i.\mathsf{GateInputs}$
      if $a.\mathsf{Type} = \mathsf{TypeF}$ **and** $b.\mathsf{Type} = \mathsf{TypeF}$ **then**
        $W_i^0 \leftarrow \mathsf{GbFreeXOR2}(W_a^0, W_b^0)$

      else if $\exists p \in \{a,b\}$ $s.t.$ $p.\mathsf{Type} = \mathsf{TypeF}$ **then**
        $(W^0_{\{a,b\}\setminus p}, W_i^0) \leftarrow \mathsf{GbFreeXOR1}(W_p^0)$

      else if $\exists p \in \{a,b\}$ $s.t.$ $p.\mathsf{Type} = \mathsf{TypeM}$ **then**
        $(W_a^0, W_b^0, W_i^0) \leftarrow \mathsf{GbFreeXOR0}()$

    else
      $\{a,b\} \leftarrow i.\mathsf{GateInputs}$
      if $a.\mathsf{Type} = \mathsf{TypeF}$ **and** $b.\mathsf{Type} = \mathsf{TypeF}$ **then**
        $(\hat{F}.\mathsf{Next}, W_i^0) \leftarrow \mathsf{GbHG2}(i, W_a^0, W_b^0, \Delta)$

      else if $\exists p \in \{a,b\}$ $s.t.$ $p.\mathsf{Type} = \mathsf{TypeF}$ **then**
        $(W^0_{\{a,b\}\setminus p}, W_i^0) \leftarrow \mathsf{GbHG1}(i, W_p^0, \Delta)$

      else if $\exists p \in \{a,b\}$ $s.t.$ $p.\mathsf{Type} = \mathsf{TypeM}$ **then**
        $(W_a^0, W_b^0, W_i^0) \leftarrow \mathsf{GbHG0}(i, \Delta)$

    if gate i has TypeM or TypeF input wire **then**
      for each $p \in \{a,b,i\}$ s.t. $p.\mathsf{Type} \neq \mathsf{TypeF}$ **do**
        $W_p^1 \leftarrow W_p^0 \oplus \Delta$
        label p as TypeF

  $\{Backward\ phase\}$
  for each gate $i \in f$ $\{in\ inv.\ topo.\ order\}$ **do**
    if $i.\mathsf{Type} = \mathsf{TypeS}$ **then**
      $(W_i^0, W_i^1) \twoheadleftarrow \{0,1\}^\ell \times \{0,1\}^\ell$

    if $i \in f.\mathsf{NOTGates}$ **then**
      $a \leftarrow i.\mathsf{GateInputs}$
      if $a.\mathsf{Type} = \mathsf{TypeS}$
        $(W_a^0, W_a^1) \leftarrow \mathsf{GbBwNOT}(W_i^0, W_i^1)$

    if $i \in f.\mathsf{XORGates}$ **then**
      $(a,b) \leftarrow i.\mathsf{GateInputs}$
      if $a.\mathsf{Type} = \mathsf{TypeS}$ **then**
        $(W_a^0, W_a^1, W_b^0, W_b^1) \leftarrow \mathsf{GbITXOR}(W_i^0, W_i^1)$

    else
      $(a,b) \leftarrow \mathsf{GateInputs}(f,i)$
      if $a.\mathsf{Type} = \mathsf{TypeS}$ **then**
        $(W_a^0, W_a^1, W_b^0, W_b^1) \leftarrow \mathsf{GbITAND}(W_i^0, W_i^1)$

    label all TypeS input wires of gate i as TypeB

  $\{finalizing\ and\ return\}$
  for $i \in f.\mathsf{Inputs}$ **do**
    $e_i \leftarrow (W_i^0, W_i^1)$
  for $i \in f.\mathsf{Outputs}$ **do**
    $d_i \leftarrow (H(W_i^0, i), H(W_i^1, i))$
  return $(\hat{F}, \hat{e}, \hat{d})$

procedure $\mathsf{Ev}(f, \hat{F}, \hat{X})$:
  label each wire as TypeM or TypeS
  for each gate $i \in f$ $\{in\ topo.\ order\}$ **do**
    if $i \in f.\mathsf{NOTGates}$ **then**
      $a \leftarrow i.\mathsf{GateInputs}$
      $W_i \leftarrow \mathsf{EvNOT}(W_a)$

    else if $i \in f.\mathsf{XORGates}$ **then**
      $\{a,b\} \leftarrow i.\mathsf{GateInputs}$
      $W_i \leftarrow \mathsf{EvXOR}(W_a, W_b)$

    else
      $\{a,b\} \leftarrow i.\mathsf{GateInputs}$
      if $a.\mathsf{Type} = \mathsf{TypeF}$ **and** $b.\mathsf{Type} = \mathsf{TypeF}$ **then**
        $W_i \leftarrow \mathsf{EvHG2}(i, \hat{F}.\mathsf{Next}, W_a, W_b, w_a, w_b)$
      else if $\exists p \in \{a,b\}$ $s.t.$ $p.\mathsf{Type} = \mathsf{TypeF}$ **then**
        $W_i \leftarrow \mathsf{EvHG0\&1}(i, W_p, W_{\{a,b\}\setminus p}, w_p, w_{\{a,b\}\setminus p})$
      else if $\exists p \in \{a,b\}$ $s.t.$ $p.\mathsf{Type} = \mathsf{TypeM}$ **then**
        $W_i \leftarrow \mathsf{EvHG0\&1}(i, W_a, W_b, w_a, w_b)$
      else
        $W_i \leftarrow \mathsf{EvITAND}(W_a, W_b, w_a, w_b)$

    if gate i has TypeM or TypeF input wire **then**
      for each $p \in \{a,b,i\}$ s.t. $.\mathsf{Type} \neq \mathsf{TypeF}$ **do**
        label p as TypeF

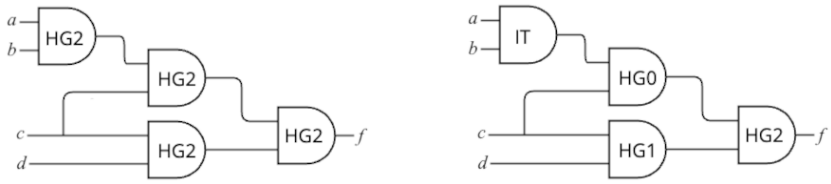
  for $i \in \hat{f}.\mathsf{Outputs}$ **do**
    $Y_i \leftarrow W_i$
  return $\hat{Y}$

procedure $\mathsf{En}(\hat{e}, \hat{x})$:
  parse $(e_i^0, e_i^1) \leftarrow e_i$
  $X_i \leftarrow e_i^{x_i}$
  return $\hat{X}$

procedure $\mathsf{De}(\hat{d}, \hat{Y})$:
  for $d_i \in \hat{d}$ **do**
    parse $(h_0, h_1) \leftarrow d_i$
    parse $(h_0, h_1) \leftarrow d_i$
    if $H(Y_i, i) = h_0$ **then** $y_i \leftarrow 0$
    else if $H(Y_i, i) = h_1$ **then** $y_i \leftarrow 1$
    else return $\perp$
  return $\hat{y}$

procedure $\mathsf{Ve}(\hat{e}, \hat{F}, f)$:
  find which garbling is used in each gate (as in Ev)
  for each gate $i \in f$ $\{in\ topo.\ order\}$ **do**
    run the related gate verification algorithm
    if it the gate verification returns 0 **then**
      return 0
    else
      assign what verification returned as (W_i^0, W_i^1)
  return 1

Table 5: Our privacy-free garbling scheme AuthOr.



(a) HG Garbled (b) AuthOr Garbled

Fig. 1: The impact of using HG0, HG1 and ITAND on the garbled circuit. The circuits are in topological order, left to right .

A TypeS wire becomes TypeF only when it is involved in the garbling of a gate, either as an input or as an output. If a is input to multiple gates, it cannot be labeled TypeS initially. If it is an output of some gate $j > i$, this violates the topological ordering (since a is also an input to gate i), and so such a gate cannot exist. Hence, only gates $j \leq i$ can change the label of a from TypeS to TypeF.

Somewhat surprisingly, even the gates whose input wires are TypeS but whose output wires are converted to TypeF by garbling another gate afterward can be left to the backward phase, as the offset of an output wire is perfectly hidden in IT garbling. We note that an AND gate that is garbled during the forward phase may still benefit from the efficiency improvements due to our HG0 or HG1 if at least one of its inputs is not TypeF before garbling this gate. Also, the garbled circuit $\hat{F}$ is determined during this phase, leaving only the complete determination of input and output keys to the backward phase. During the backward phase, all the gates that were left from the forward phase are garbled via IT garbling. Note that these are the gates whose both input wires are TypeS, so their keys can be set by GbITAND and GbITXOR without any problem. At the end, all wires become either TypeF (each with global offset Δ) or TypeB (each with arbitrary offset). We highlight that if the given circuit f is a Boolean formula, all the gates are garbled in the backward phase, and our scheme is equivalent to IT garbling.

Figure 1 compares the number of HG2 runs employed in garbling a generic circuit for both the HG garbling approach [ZRE15] and AuthOr. In the HG approach, all 4 gates of the given circuit are garbled using HG2, resulting in 4 ciphertexts. In contrast, the AuthOr garbling algorithm applies only one HG2 execution, leading to the generation of a single ciphertext.

Our Ev, En, De, and Ve algorithms. The evaluation algorithm Ev simply runs in topological order and applies the related evaluation algorithms given in Table 3. What this algorithm needs is only information on which gate-garbling scheme is used for each gate, which is determined using the same method Gb uses. All the gates garbled in the forward phase of Gb can be evaluated with the corresponding evaluation algorithm. The encoding algorithm En simply collects the keys for the circuit input wires as in [ZRE15]. The decoding algorithm De

collects the hash outputs of the keys for the circuit output wires as in [ZRE15], where the hash algorithm ensures that even with the decoding data, the evaluator is prevented from finding both truth-value keys for the output wire. The verification algorithm Ve finds the gate-garbling algorithm that is used in each gate and then, in topological order, calls the gate verification algorithms in Table 4. We highlight that there is no need for checking that the same global offset Δ is used in each gate. Instead, we check whether the incoming wires of a gate have the same offset, which is sufficient to ensure that any set of forward-garbled connected[5] gates have wires with the same local offset. Two such unconnected forward-garbled gate sets could have separate local offsets (although for simplicity, our algorithm does not allow it) and still pass the verification without sacrificing verifiability.

Theorem 2. *The AuthOr garbling scheme is a privacy-free garbling scheme with correctness, authenticity, and verifiability properties if the hash function H has circular correlation robustness (CCR).*

We show the security of our scheme in Appendix B.

5 Implementation, Experimental Results, and Efficiency

Our implementations of AuthOr as a proof-of-concept and of the state-of-the-art HG garbling [ZRE15] for comparison, both in Python 3, are given at [Ajo]. We have run tests on the arithmetic operation circuits of [Arc+], the cryptographic circuits of [Arc+], and the benchmark circuits of [Jen], all of which were constructed by independent researchers. For each circuit, we have repeated our tests 10 times on an Apple M2 MacBook Pro with 24 GB of RAM. We present the average experimental results in Table 6, which compare costs for garbling and evaluating computation times (measured as elapsed times in seconds) and the GC size (measured as the total number of ciphertexts generated). As the bottleneck of garbling schemes is GC size [ZRE15], we specifically calculate the GC size reduction of our scheme in the last columns of the given tables. We did not measure the overhead associated with the compilation and preprocessing steps required before garbling.

Preprocessing of the circuit. In our implementation, we developed a compiler to parse circuits written in both supported syntaxes and to manage this emulation process. The circuits of [Arc+] implement a buffer via an AND gate with both input pins connected to the same wire for some circuits. Since these gates do not affect the circuit's mathematical functionality or the authenticity of garbling, we removed them in a preprocessing step. Additionally, some benchmarks include pins that are set as 0 through an XOR gate with a common input. For similar reasons, we also removed these XOR gates. Additionally, we employed the "networkx" library to determine the topological order of the gates

[5] We slightly abuse the terminology as two gates can be connected via sharing any wires with each other or they are both connected to another gate.

Circuit	Half Gates [ZRE15]			**AuthOr**			
	Gb cost	Ev cost	GC Size	Gb cost	Ev Cost	GC Size	GC Size Reduction (%)
adder64	0.003	0.001	64	0.010	0.001	64	0.00
sub64	0.002	0.001	64	0.003	0.001	64	0.00
mult64	0.111	0.041	4034	0.116	0.042	3970	1.59
mult2-64	0.224	0.085	8129	0.234	0.087	4034	**50.38**
divide64	0.158	0.068	4665	0.169	0.070	4664	0.02
udivide64	0.123	0.049	4286	0.128	0.051	4225	1.42
and8	0.000	0.000	8	0.000	0.000	1	**87.50**
zero-equal	0.002	0.000	64	0.000	0.000	1	**98.44**
FP-add	0.140	0.049	5369	0.147	0.051	5348	0.39
FP-floor	0.016	0.006	645	0.017	0.006	644	0.16
FP-ceil	0.016	0.006	645	0.017	0.006	645	0.00
FP-div	2.090	0.746	80184	2.179	0.764	80163	0.03
FP-sqrt	2.350	0.844	89964	2.455	0.857	89964	0.00
FP-f2i	0.038	0.013	1464	0.042	0.014	1456	0.55
FP-mul	0.501	0.176	19426	0.523	0.180	19406	0.10
FP-eq	0.009	0.003	316	0.009	0.003	305	3.48
FP-i2f	0.064	0.022	2407	0.068	0.023	2407	0.00
cmp32-unsigned-lt	0.004	0.001	151	0.004	0.001	141	**6.62**
cmp32-signed-lteq	0.004	0.001	151	0.004	0.001	138	**8.61**
							Average: 13.67

(a)

Circuit	Half Gates [ZRE15]			**AuthOr**			
	Gb cost	Ev cost	GC Size	Gb cost	Ev Cost	GC Size	GC Size Reduction (%)
sha256	0.767	0.325	22574	0.823	0.334	22569	0.02
aes128	0.210	0.087	6401	0.227	0.090	6401	0.00
aes192	0.238	0.098	7169	0.256	0.103	7169	0.00
Keccak-f	1.211	0.474	38401	1.299	0.493	38401	0.00
sha512	1.973	0.836	57948	2.133	0.857	57943	0.01
aes256	0.292	0.121	8833	0.313	0.126	8833	0.00
							Average: 0.01

(b)

Circuit	Half Gates [ZRE15]			**AuthOr**			
	Gb cost	Ev cost	GC Size	Gb cost	Ev Cost	GC Size	GC Size Reduction (%)
c5315	0.055	0.016	2012	0.044	0.014	1229	**38.92**
c1908	0.017	0.005	576	0.014	0.004	398	**30.90**
c432	0.004	0.001	156	0.003	0.001	83	**46.79**
c6288	0.065	0.020	2385	0.056	0.017	1650	**30.82**
c7552	0.070	0.021	2491	0.060	0.019	1604	**35.61**
c17	0.000	0.000	7	0.000	0.000	1	**85.71**
c2670	0.024	0.007	876	0.019	0.006	514	**41.32**
c499	0.003	0.001	91	0.002	0.001	49	**46.15**
c880	0.009	0.003	331	0.007	0.002	171	**48.34**
c1355	0.014	0.004	507	0.010	0.003	213	**57.99**
c3540	0.033	0.010	1169	0.029	0.009	784	**32.93**
							Average: 45.05

(c)

Table 6: Experimental results on (a) arithmetic circuits, (b) cryptographic circuits, and (c) benchmark circuits of [Arc+]. Computation costs and GC sizes are in terms of seconds and the number of generated ciphertexts, respectively. GC size reduction is calculated as the percentage of GC size reduction with respect to (privacy-free version of) half-gates garbling.

in each Boolean circuit. Our implementation supports two syntaxes for representing Boolean circuits: *Bristol Fashion* [Arc+] and *Bench*. In both formats,

each circuit is represented by listing all internal Boolean gates and their pin connectivity. The set of gates discussed in Section 3 is Turing-complete, allowing us to emulate all other types of gates, including those with a fan-in greater than 2, as well as OR, NOR, NAND, and XNOR gates.

Garbling. The types of wires can be determined during the forward phase, which our implementation takes advantage of. We made this separate in Table 5 only for the sake of clarity. As suggested by [Bel+13; ZRE15], the hash function H can be implemented by using a symmetric-key encryption algorithm with a shared fixed key k. We use AES-128 encryption in CBC mode (inherited from the "Cryptography" library) to set the hash function H as:

$$H(i, W) = AES128_k(W) \oplus i \oplus W$$

GC size comparison with HG garbling [ZRE15]. As shown in Table 6, the GC size reduction is highly dependent on the architecture of the circuits. For cryptographic circuits of [Arc+], the gain is marginal because the complexity of gate connections requires our scheme to garble almost all the gates with GbHG2 during the forward phase. In contrast, for arithmetic circuits of [Arc+] and for benchmark circuits of [Arc+], the gain is up to roughly 98% and 86%, and averages roughly 14% and 45%, respectively.

```
procedure Count(f):
label each wire as TypeM or TypeS
initialize (CountHG2, CountHG0&1, CountITAND) := (0, 0, 0)
for each gate i ∈ f {in topological order} do
  if i ∈ f.ANDGates then
    {a, b} ← i.GateInputs
    if a.Type = TypeF and b.Type = TypeF then
      CountHG2 := CountHG2 + 1
    else if ∃p ∈ {a, b} s.t. p.Type = TypeF ∨ p.Type = TypeM then
      CountHG0&1 := CountHG0&1 + 1
    else
      CountITAND := CountITAND + 1
  if gate i has a TypeM or TypeF input wire then
    for each p ∈ {a, b, i} s.t. p.Type ≠ TypeF do
      label p as TypeF
return (CountHG2, CountHG0&1, CountITAND)
```

Table 7: Counting the AND gates with different garbling algorithms.

Efficiency gain based on Circuit Structure. It may be a concern to decide whether or how much would a circuit benefit from AuthOr before implementation. Generally speaking, the gain is based on the depth of the circuit as the shollow circuits benefit more from HG1 (or HG0), and on the abundance of wires with typeS at lower depths as the ones with higher number of such wires would benefit from IT garbling more. We highlight that our scheme never results in greater GC size than half gates, so for simplicity we avoid providing a formulas to calculate the gain. Instead, we propose altering the counting algorithm Count in Table 7 to determine the numbers CountHG2, CountHG0&1, and CountITAND of AND gates that HG2, HG1 (or HG0), and ITAND are applied, respectively. Then estimating the efficiency gain is a straightforward task.

References

[Agr17] S. Agrawal. "Stronger Security for Reusable Garbled Circuits, General Definitions and Attacks". In: *CRYPTO '17*. 2017.

[Ajo] A. Ajorian. *Implementation of AuthOr and Half Gates garbling schemes*. Accessible via `https://github.com/Ajorian/AuthOr/tree/main#`.

[AL18] P. Ananth and A. Lombardi. "Succinct Garbling Schemes from Functional Encryption Through a Local Simulation Paradigm". In: *TCC '18*. 2018.

[Arc+] D. Archer et al. *'Bristol Fashion' MPC Circuits*. Retrieved 2025-04-10 from `https://nigelsmart.github.io/MPC-Circuits/`.

[Bel+13] M. Bellare et al. "Efficient Garbling from a Fixed-Key Blockcipher". In: *IEEE SP '13*. 2013.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. "Foundations of garbled circuits". In: *ACM CCS '12*. 2012.

[Bon+14] D. Boneh et al. "Fully Key-Homomorphic Encryption, Arithmetic Circuit ABE and Compact Garbled Circuits". In: *EUROCRYPT '14*. 2014.

[BP23] S. Basu and L. Parida. *Quantum Analog of Shannon's Lower Bound Theorem*. 2023. URL: `https://arxiv.org/abs/2308.13091`.

[Can+15] R. Canetti et al. "Succinct Garbling and Indistinguishability Obfuscation for RAM Programs". In: *ACM STOC '15*. 2015.

[CGM16] M. Chase, C. Ganesh, and P. Mohassel. "Efficient Zero-Knowledge Proof of Algebraic and Non-Algebraic Statements with Applications to Privacy Preserving Credentials". In: *CRYPTO '16*. 2016.

[Cho+12] S. G. Choi et al. "On the Security of the "Free-XOR" Technique". In: *TCC '12*. 2012.

[DG17] N. Döttling and S. Garg. "Identity-Based Encryption from the Diffie-Hellman Assumption". In: *CRYPTO '17*. 2017.

[FNO15] T. K. Frederiksen, J. B. Nielsen, and C. Orlandi. "Privacy-Free Garbled Circuits with Applications to Efficient Zero-Knowledge". In: *EUROCRYPT '15*. 2015.

[Gan+18] C. Ganesh et al. "Efficient Adaptively Secure Zero-Knowledge from Garbled Circuits". In: *PKC '18*. 2018.

[Gen+13] R. Gennaro et al. "Quadratic Span Programs and Succinct NIZKs without PCPs". In: *EUROCRYPT '13*. 2013.

[GGP10] R. Gennaro, C. Gentry, and B. Parno. "Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers". In: *CRYPTO '10*. 2010.

[Gol+] S. Goldwasser et al. "Reusable garbled circuits and succinct functional encryption". In: *ACM STOC '2013*.

[Gue+15] S. Gueron et al. "Fast Garbling of Circuits Under Standard Assumptions". In: *ACM CCS '15*. 2015.

[Jen] M. Jenihhin. *Bench marks*. Retrieved 2025-04-10 from `https://pld.ttu.ee/~maksim/benchmarks/iscas85/bench/`.

[JKO13] M. Jawurek, F. Kerschbaum, and C. Orlandi. "Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently". In: *ACM CCS '13*. 2013.

[Kem+15] C. Kempka et al. "Garbling Scheme for Formulas with Constant Size of Garbled Gates". In: *ASIACRYPT '15*. 2015.

[Kol05] V. Kolesnikov. "Gate Evaluation Secret Sharing and Secure One-Round Two-Party Computation". In: *ASIACRYPT '05*. 2005.

[KP17] Y. Kondi and A. Patra. "Privacy-Free Garbled Circuits for Formulas: Size Zero and Information-Theoretic". In: *CRYPTO '17*. 2017.

[KS08] V. Kolesnikov and T. Schneider. "Improved Garbled Circuit: Free XOR Gates and Applications". In: *ICALP '08*. 2008.

[Lin+15] Y. Lindell et al. "Efficient Constant Round Multi-party Computation Combining BMR and SPDZ". In: *CRYPTO '15*. 2015.

[Liu+24] H. Liu et al. "Garbled Circuits with 1 Bit per Gate". In: *Cryptology ePrint Archive, Paper 2024/1988*. 2024.

[LP07] Y. Lindell and B. Pinkas. "An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries". In: *EUROCRYPT '07*. 2007.

[LP11] Y. Lindell and B. Pinkas. "Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer". In: *TCC '11*. 2011.

[LR15] Y. Lindell and B. Riva. "Blazing Fast 2PC in the Offline/Online Setting with Security for Malicious Adversaries". In: *ACM CCS '15*. 2015.

[MLR23] G. Marcadet, P. Lafourcade, and L. Robert. "RMC-PVC: A Multi-Client Reusable Verifiable Computation Protocol". In: *ACM SAC '23*. 2023.

[NS23] R. Nieminen and T. Schneider. "Breaking and Fixing Garbled Circuits When a Gate has Duplicate Input Wires". In: *Journal of Cryptology* 36 (Aug. 2023).

[RR21] M. Rosulek and L. Roy. "Three Halves Make a Whole? Beating the Half-Gates Lower Bound for Garbled Circuits". In: *CRYPTO '21*. 2021.

[Sha49] C. E. Shannon. "The synthesis of two-terminal switching circuits". In: *The Bell System Technical Journal* 28.1 (1949), pp. 59–98.

[WRK17] X. Wang, S. Ranellucci, and J. Katz. "Global-Scale Secure Multiparty Computation". In: *ACM CCS '17*. 2017.

[Yao82] A. C. Yao. "Protocols for Secure Computations". In: *IEEE FOCS '82*. 1982.

[Yao86] A. C. Yao. "How to generate and exchange secrets". In: *IEEE FOCS '86*. 1986.

[ZRE15] S. Zahur, M. Rosulek, and D. Evans. "Two Halves Make a Whole: Reducing Data Transfer in Garbled Circuits using Half Gates". In: *EUROCRYPT '15*. 2015.

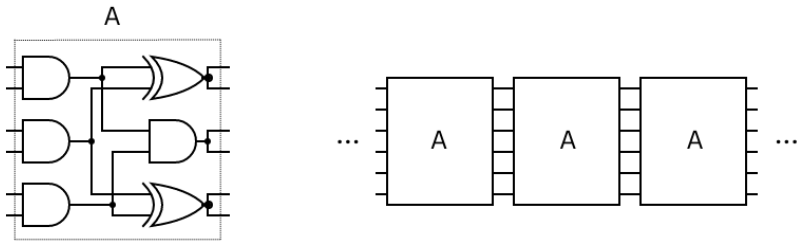


Fig. 2: An example circuit with size g obtained by repeating A-boxes $g/6$ times. The A-box construction is also shown. Circuits are left to right in topological order.

A Inefficiency of IT Garbling for Generic Circuits

For fairness of comparison with IT garbling [KP17], we convert an example generic circuit (with size g) given in Figure 2 into a Boolean formula by applying the technique mentioned in [KP17] to convert a circuit into a Boolean formula: when an input wire i of the circuit is input to n gates, we replace i by n wires. Note that each circuit layer has 3 gates, so the depth of the circuit is $g/3$. Starting from the last layer in inverse topological order, the procedure replaces each fan-out-n gate i with n fan-out-1 gates, each of which has the same input wires as the gate i had. The last layer (the $(g/3)$-th) remains the same as it does not have any fan-out-2 gates. The $(g/3-1)$-th layer has 3 fan-out-2 gates, so is replaced with 6 gates. However, now the $(g/3-2)$-th layer has 3 fan-out-4 gates, so it needs to be replaced by 12 gates. At each layer the number of the gates doubles, hence we calculate the size g' of the obtained Boolean formula as $g' = \sum_{j=0}^{g/3-1} 3 \cdot 2^j = 3 \cdot 2^{g/3} - 3$, which means $g' \in \Theta(\exp(g))$.

B Security of AuthOr

In what follows we show the authenticity and verifiability properties of AuthOr, as correctness simply follows from the correctness of the gate-garbling schemes and the use of the same wire labeling for Gb and Ev.

Theorem 3. *The AuthOr garbling scheme satisfies authenticity if the hash function H has circular correlation robustness (CCR).*

Proof. The proof that we provide here depends on the already proven security of the previous schemes: forward-garbled (FreeXOR and HG) and backward-garbled (IT). During the forward phase of Gb, if the keys for a wire i have been determined (turning that wire into TypeF), then the keys for other input and output wires of a gate that takes i as input are also determined (set as TypeF). We deduce that at the end of a Gb execution, the TypeF wires look like

"peninsulas" leaning toward the output of the circuit in a sea of TypeB wires, which is visualized in Figure 3.

The TypeB wire sea is already proven to leak no information (including XOR, AND, and NOT gates) about the other output-wire keys than the expected ones to be obtained during Ev by [KP17]. Regarding the TypeF wire peninsulas, they are a simple extension HG* of the privacy-free HG garbling scheme of [ZRE15] that we define here. HG* garbles only in the forward direction, including FwNOT, which is already shown to have authenticity by [Kol05], HG2 of [ZRE15], our proposals HG0 and HG1, and the XOR-garbling algorithms FreeXOR0, FreeXOR1, and FreeXOR2. In what follows, we show that this scheme itself has authenticity:

Lemma 1. *HG* garbling scheme (with* HGz*,* FreeXORz*, and* FwNOT *for* $z \in \{0,1,2\}$*) has authenticity if the privacy-free HG garbling scheme of [ZRE15] (with* HG2*,* FreeXORz*, and* GbFwNOT*, without* HG0*,* HG1*) has authenticity.*

Proof. We construct a PPT adversary $\mathcal{B}$ that can break the authenticity of HG garbling (played with a challenger $\mathcal{C}$), by using as a subroutine a PPT adversary $\mathcal{A}$ that breaks the authenticity of the HG* garbling scheme.

1. $\mathcal{B}$ starts the execution of $\mathcal{A}$. $\mathcal{A}$ picks a function f and an input string $\hat{x}$ and outputs them to $\mathcal{B}$, which in turn outputs them to $\mathcal{C}$.
2. $\mathcal{B}$ obtains the challenge garbled circuit $\hat{F}$ and input $\hat{X}$ from $\mathcal{C}$.
3. $\mathcal{B}$ executes the evaluation algorithm of HG garbling to obtain the key W_i of each wire i.
4. $\mathcal{B}$ finds the set $\hat{G}$ of AND gates that would be garbled with HG0 and HG1 in a normal execution of the garbling algorithm of HG*. Here, we assume the ones that would be garbled with HG1 would have the left wire a as TypeF[6].
5. $\mathcal{B}$ executes the following updates on $\hat{F}$ and $\hat{X}$.

   for each gate $i \in f$ {in topo. order} **do**
   if $i \in G$ **then**
   $\{a, b\} \leftarrow i.\mathsf{GateInputs}$
   Set $W_b^* \leftarrow W_b \oplus F_i$
   Delete F_i from $\hat{F}$
   Set each $W_b \in \hat{X}$ as the freshly generated key W_b^*

   The updated $\hat{F}$ and $\hat{X}$ are information-theoretically indistinguishable from what could be obtained in an honest execution of HG* (explained below).
6. $\mathcal{B}$ gives $\hat{F}$ and $\hat{X}$ to $\mathcal{A}$.
7. $\mathcal{A}$ gives the output $\hat{Y}$ to $\mathcal{B}$, which in turn gives it to $\mathcal{C}$.

In an honest execution of HG*, for a given W_a^0, GbHG1 sets W_b^0 as $H(i, W_a^0) \oplus H(i, W_a^1)$. Here, $\mathcal{B}$ additionally picks F_i uniformly at random and sets W_b^0 as $H(i, W_a^0) \oplus H(i, W_a^1) \oplus F_i$. On the other hand, the HG garbling picks W_b^0 uniformly at random, and then uses GbHG2 to set F_i as $H(i, W_a^0) \oplus H(i, W_a^1) \oplus W_b^0$. The obtained W_b^0 and F_i are information-theoretically indistinguishable from

[6] For other circuits, it is trivial to find an equivalent that fits into this criterion as AND gates are symmetric

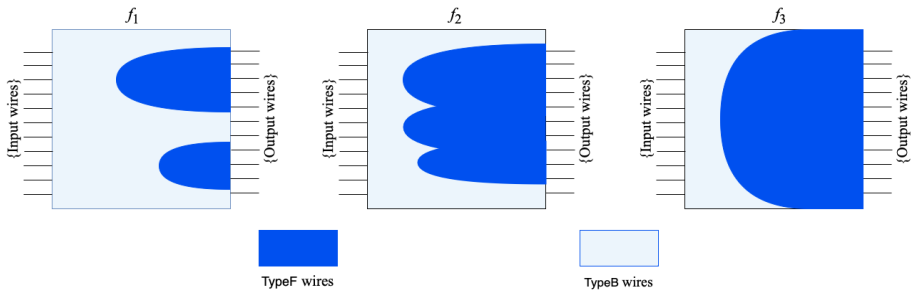


Fig. 3: Example formations of TypeF wires and TypeB wires upon Gb executions on example f_1, f_2, and f_3 (left to right in topological order).

each other. The probability that $\mathcal{A}$ comes up with an authentic $\hat{Y}$ that is not equal to what could be obtained in an honest execution of the evaluation algorithm on the given $\hat{F}$ and $\hat{X}$ is exactly the same for $\mathcal{B}$ also doing so. Hence, if it is non-negligible, $\mathcal{B}$ breaks the authenticity of HG with non-negligible probability.

So far, we have shown that the peninsulas of TypeF wires have authenticity given that the garbling scheme of [ZRE15] has authenticity (which is based on the CCR of H in their work), while [KP17] already shows that the surrounding sea of TypeB wires has information-theoretic authenticity. To conclude, we now show that the IT garbling scheme of [KP17] could be altered so that an arbitrary number of circuit output wires can have the same offset without breaking the information-theoretic authenticity. We name the altered scheme IT* garbling, which exactly has the garbling algorithms used in the backward phase of our Gb. As in the generic IT garbling, IT* uses the ITAND, ITXOR, and BwNOT algorithms, and is applicable to circuits with only TypeS wires. Additionally, it allows the adversary to choose a set of output wires that will have the same offset, i.e., the XOR of the keys for each of those wires is fixed.

Lemma 2. *The IT* garbling scheme is information-theoretically authentic.*

Proof. Any circuit f with only TypeS wires will be a set of independent subcircuits $\{f_1, \ldots, f_n\}$, each having a 1-bit output unconnected from each other. Each subcircuit f_i that the adversary does not choose to have the same offset is garbled independently, which is proven to be information-theoretically authentic by [KP17]. Hence, we will consider only the set of circuits $\{f'_1, \ldots, f'_m\}$ that the adversary chooses to have the same Δ. Also, via the reduction from [KP17], the authenticity of each of those f'_i can be reduced to the authenticity of a one-gate circuit, i.e., its output gate's authenticity. Hence, we assume each of the circuits $f'_1, \ldots, f'_m$ has only one gate (it may be any of AND, XOR, or NOT) and has the same Δ at the output. We highlight that the IT* garbling scheme still has size zero, i.e., the adversary receives only $\hat{X}$ (specifically not $\hat{F}$) upon garbling. Therefore, the information-theoretic authenticity can be proven by proving that the same $\hat{X}$ may be generated with the same probability independent of Δ for

any given circuit $f = \{f'_1, \ldots, f'_m\}$ where each f'_i is a gate and its input $\hat{x}$ whose bits are TypeS. We show this gate-by-gate for each of AND, XOR, and NOT gates.

AND gate. Given a gate i with input wires a and b, the garbling algorithm picks $W_i^0 \twoheadleftarrow \{0,1\}^\ell$ uniformly at random, sets $W_i^1 \leftarrow W_i^0 \oplus \Delta$ and then calls $\mathsf{GbITAND}$ to obtain the wire keys W_a^0, W_a^1, W_b^0, and W_b^1. Then, either W_a^0 or W_a^1 is sent as the wire key X_a and either W_b^0 or W_b^1 is sent as the wire key X_b, depending on the values of w_a and w_b. We will compare, for all combinations of w_a and w_b, the probability of obtaining the same keys X_a and X_b using Δ_0 with the one using Δ_1.

Case $w_a = 0$ *and* $w_b = 0$. The garbling algorithm picks W_i^0 uniformly at random, and sets $(W_a^0, W_b^0) \leftarrow (W_i^0, W_i^0)$ via $\mathsf{GbITAND}$. Hence, any given pair of $X_a = W_a^0$ and $X_b = W_b^0$ (the given two keys for $w_a = 0$ and $w_b = 0$) could be generated with equal probability using Δ_0 or Δ_1.

Case $w_a = 1$ *and* $w_b = 0$. The garbling algorithm picks $W_i^0 \twoheadleftarrow \{0,1\}^\ell$ uniformly at random and sets $W_b^0 \leftarrow W_i^0$ via $\mathsf{GbITAND}$. $\mathsf{GbITAND}$ also picks $W_a^1 \twoheadleftarrow \{0,1\}^\ell$. Again, due to directly picking uniformly at random, any given pair of $X_a = W_a^1$ and $X_b = W_b^0$ could be generated with equal probability using either Δ_0 or Δ_1.

Case $w_a = 0$ *and* $w_b = 1$. The garbling algorithm picks $W_i^0 \twoheadleftarrow \{0,1\}^\ell$ uniformly at random, and sets $W_a^0 \leftarrow W_i^0$ via $\mathsf{GbITAND}$. $\mathsf{GbITAND}$ also picks $W_a^1 \twoheadleftarrow \{0,1\}^\ell$ uniformly at random, and sets $W_b^1 \leftarrow W_a^1 \oplus W_i^1$ where $W_i^1 = W_i^0 \oplus \Delta$. Any given pair of X_a and X_b could be generated with equal probability using either Δ_0 or Δ_1 due to the following reasons. Due to directly picking W_i^0 uniformly at random, $X_a = W_a^0 = W_i^0$ could be generated with equal probability using either Δ_0 or Δ_1. Also, W_b^1 is generated by XORing the uniformly picked W_a^1 with some value W_i^1. Any element of $\{0,1\}^\ell$ could be generated as W_b^1 with equal probability regardless of W_i^1's value, and then set as $X_b = W_b^1$.

Case $w_a = 1$ *and* $w_b = 1$. The garbling algorithm picks $W_i^0 \twoheadleftarrow \{0,1\}^\ell$ uniformly at random, and sets $W_i^1 \leftarrow W_i^0 \oplus \Delta$, hence W_i^1 could get any value from $\{0,1\}^\ell$ with equal probability using either Δ_0 or Δ_1. $\mathsf{GbITAND}$ then picks $W_a^1 \twoheadleftarrow \{0,1\}^\ell$ uniformly at random, and sets $W_b^1 \leftarrow W_a^1 \oplus W_i^1$. We deduce $W_b^1 = W_a^1 \oplus W_i^0 \oplus \Delta$. Any given pair of X_a and X_b could be generated with equal probability using either Δ_0 or Δ_1 since X_a directly comes from the uniform picking of W_a^1 and X_b results from XORing some values with the uniformly picked W_i^0.

W_a^0 and W_b^1 are prepared independently, so their probabilities can be considered separately. Due to directly choosing W_a^0 uniformly, X_a could be generated with equal probability using either Δ_0 or Δ_1. Due to the generation of W_b^1 by XORing the uniformly randomly picked W_a^1 with some value W_i^1, regardless of $W_i^1 = W_i^0 \oplus \Delta_0$ or $W_i^1 = W_i^0 \oplus \Delta_1$, any element of $\{0,1\}^\ell$ could be generated as W_b^1 with equal probability.

XOR gate. Similar to AND-gate garbling, given a gate i with input wires a and b, the garbling algorithm picks $W_i^0 \twoheadleftarrow \{0,1\}^\ell$ uniformly at random, sets $W_i^1 \leftarrow W_i^0 \oplus \Delta$ and then calls $\mathsf{GbITXOR}$ to obtain W_a^0, W_a^1, W_b^0, and W_b^1. Then,

either W_a^0 or W_a^1 is sent as X_a and either W_b^0 or W_b^1 is sent as X_b, depending on the values of w_a and w_b. We will compare, for all combinations of w_a and w_b, the probability of obtaining the same keys X_a and X_b using Δ_0 with the one using Δ_1.

Case $w_a = 1$ *and* $w_b = 1$. The garbling algorithm picks $W_i^0 \twoheadleftarrow \{0,1\}^\ell$ uniformly at random. GbITXOR then picks $W_a^1 \twoheadleftarrow \{0,1\}^\ell$, and sets $W_b^1 \leftarrow W_a^1 \oplus W_i^0$. As W_a^1 is picked uniformly and W_b^1 is randomized by the independent choice of W_i^0, any given pair of $X_a = W_a^1$ and $X_b = W_b^1$ could be generated with equal probability using either Δ_0 or Δ_1.

Case $w_a = 1$ *and* $w_b = 0$. The garbling algorithm picks $W_i^0 \twoheadleftarrow \{0,1\}^\ell$ uniformly at random, and sets $W_i^1 \leftarrow W_i^0 \oplus \Delta$. GbITXOR then picks $W_a^1 \twoheadleftarrow \{0,1\}^\ell$, and sets $W_b^0 \leftarrow W_a^1 \oplus W_i^1$, which means $W_b^0 = W_a^1 \oplus W_i^0 \oplus \Delta$. As W_a^1 is picked uniformly at random, and W_b^0 is randomized by the independent choice of W_i^0, any given pair of $X_a = W_a^1$ and $X_b = W_b^0$ could be generated with equal probability using Δ_0 or Δ_1.

Case $w_a = 0$ *and* $w_b = 1$. The garbling algorithm picks $W_i^0 \twoheadleftarrow \{0,1\}^\ell$ uniformly at random, and sets $W_i^1 \leftarrow W_i^0 \oplus \Delta$. GbITXOR then picks $W_a^1 \twoheadleftarrow \{0,1\}^\ell$, and sets $W_b^1 \leftarrow W_a^1 \oplus W_i^0$. It also sets $W_a^0 \leftarrow W_b^1 \oplus W_i^1$, which means $W_a^0 = W_b^1 \oplus W_i^0 \oplus \Delta$. As W_b^1 is randomized by W_a^1, and W_a^0 is additionally randomized by the choice of W_i^0, any given pair of $X_a = W_a^0$ and $X_b = W_b^1$ could be generated with equal probability using either Δ_0 or Δ_1.

Case $w_a = 0$ *and* $w_b = 0$. The garbling algorithm picks $W_i^0 \twoheadleftarrow \{0,1\}^\ell$. GbITXOR then picks $W_a^1 \twoheadleftarrow \{0,1\}^\ell$. Via XOR arithmetic, it is easy to show that, upon execution of GbITXOR, $W_a^0 = W_a^1 \oplus \Delta$ and $W_b^0 = W_a^1 \oplus W_i^0 \oplus \Delta$. As W_b^0 is randomized by W_a^1, and W_a^0 is additionally randomized by the choice of W_i^0, any given pair of $X_a = W_a^0$ and $X_b = W_b^0$ could be generated with equal probability using either Δ_0 or Δ_1.

NOT gate. Given a gate i with an input wire a, the garbling algorithm picks $W_i^0 \twoheadleftarrow \{0,1\}^\ell$, sets $W_i^1 \leftarrow W_i^0 \oplus \Delta$, and then swaps the order of the generated keys to obtain W_a^0 and W_a^1. As W_a^1 is set via directly picking W_i^0 uniformly at random, in the case of $w_a = 1$, using either Δ_0 or Δ_1, X_a could be obtained with equal probability. Also, as W_a^0 is randomized by W_i^0, in the case of $w_a = 0$, again using either Δ_0 or Δ_1, X_a could be obtained with equal probability.

We conclude that each of the gates $f'_1, \ldots, f'_n$ is garbled independently with independently picked randomizations, and the shown equality of probabilities that X_a and X_b are obtained from Δ_0 and Δ_1 extends to the whole set $\hat{X}$.

Theorem 4. *The AuthOr garbling scheme satisfies verifiability if the hash function H has circular correlation robustness (CCR).*

Proof (Intuition). Our verifiability algorithm Ve calls gate verification algorithms from Table 4 in topological order based on the gate-garbling schemes used. Each gate-garbling algorithm (except for VeNOT) ensures that the output-wire key has only two possible assignments (unless the hash function H is shown to be insecure), one for truth value 0 and one for truth value 1. VeXOR does so by ensuring that the input-wire offsets are the same, which means

$W_i^0 = W_a^0 \oplus W_b^0 = W_a^1 \oplus W_b^1$ and $W_i^1 = W_a^0 \oplus W_b^1 = W_a^1 \oplus W_b^0$. The verification algorithms for HG garbling also guarantee that the input-wire offsets Δ are the same, in addition to the fact that $W_i^0 = H(W_a^0) = W_b^0 \oplus H(i, W_a^1)$ holds (F_i is added for HG2). As the inputs have the same offset Δ, the XOR arithmetic ensures $W_i^1 = W_i^0 \oplus \Delta$. We have inherited VeITAND from [KP17], which has shown that the IT garbling scheme is verifiable. Also, all of the given gate verification algorithms return the two output-wire keys of each gate until the end of the circuit. Hence, for output gates, there exist only two possible assignments. Regarding the first probability of verifiability, this ensures that the obtained wire keys will be equal, and this probability is always 0. Regarding the second probability of verifiability, we can modify Ve to build an Ext. Upon obtaining all the output-wire keys using Ve, Ext finds the corresponding output keys to the bits of y and returns them. Again, this ensures that the obtained wire keys will be equal to the ones that come from evaluation, and this probability is always 0.