Breaking and Fixing Garbled Circuits when a Gate has Duplicate Input Wires
(Online Version)*

Raine Nieminen(✉) and Thomas Schneider

ENCRYPTO, Technical University of Darmstadt, Germany
{nieminen,schneider}@encrypto.cs.tu-darmstadt.de

Abstract. Garbled circuits are a fundamental cryptographic primitive that allows two or more parties to securely evaluate an arbitrary Boolean circuit without revealing any information beyond the output using a constant number of communication rounds. Garbled circuits have been introduced by Yao (FOCS'86) and generalized to the multi-party setting by Beaver, Micali and Rogaway (STOC'90). Since then, several works have improved their efficiency by providing different garbling schemes and several implementations exist. Starting with the seminal Fairplay compiler (USENIX Security'04), several implementation frameworks decoupled the task of compiling the function to be evaluated into a Boolean circuit from the engine that securely evaluates that circuit, e.g., using a secure two-party computation protocol based on garbled circuits.
In this paper, we show that this decoupling of circuit generation and evaluation allows a subtle attack on several prominent garbling schemes. It occurs when violating the implicit assumption on the circuit that gates have different input wires which is most often not explicitly specified in the respective papers. The affected garbling schemes use separate calls to a deterministic encryption function for the left and right input wire of a gate to derive pseudo-random encryption pads that are XORed together. When a circuit contains a gate where the left and right input wire are the same, these two per-wire encryption pads cancel out and we demonstrate that this can result in a complete break of privacy. We show how the vulnerable garbling schemes can be fixed easily.

Keywords: Secure Multi-Party Computation, Garbled Circuits, Garbling Schemes, Circuits, Attack, Vulnerability

1 Introduction

Secure Multi-Party Computation (MPC), also knows as Secure Function Evaluation (SFE), enables two or more parties to jointly evaluate a function securely over their private inputs. In 1986, Andrew Yao introduced Garbled Circuits (GCs) [Yao86], the first general technique for two-party SFE, which was later generalized to the multi-party setting by Beaver-Micali-Rogaway (BMR) [BMR90]. GCs allow to construct constant-round MPC protocols and hence are well-suited for high latency network settings, where multi-round protocols such as the Goldreich-Micali-Wigderson (GMW) protocol [GMW87] are often less efficient [SZ13]. Research on GCs has been very active and many different constructions have been proposed that improve the concrete communication and computation complexities, e.g., Point-and-Permute [BMR90], 3-Row Reduction [NPS99], Free-XOR [KS08], Garbling via AES [KSS12], Fixed-key AES garbling [BHK+13], HalfGates [ZRE15], and Three-Halves Garbling [RR21].

Garbled circuits allow to securely evaluate a function that is represented as a Boolean circuit, e.g., consisting of AND and XOR gates. Several frameworks have been built to compile a high-level specification of the function to be evaluated securely (e.g., in Verilog, ANSI-C, or a domain specific language) into a Boolean circuit that is then evaluated using a GC engine. Some examples for such frameworks are Fairplay [MNP+04], FairplayMP [BNP08], TASTY [HKS+10], FastGC [HEK+11], VMCrypt [Mal11], PAL [MLB12], PCF [KSM+13], CBMC-GC [FHK+14], TinyGarble [SHS+15], Frigate [MGC+16], and HyCC [BDK+18].

*Please cite the journal version of this paper to appear at JoC'23 [NS23].

The basic idea of a GC is to assign to each wire w_i of the circuit two random-looking *wire labels* $\widetilde{w}_i^0, \widetilde{w}_i^1$. The length of each of these labels is the computational security parameter κ (often set to $\kappa = 128$ in implementations) such that wire label $\widetilde{w}_i^0$ corresponds to plaintext value 0 and wire label $\widetilde{w}_i^1$ corresponds to plaintext value 1. As the wire labels look random, the evaluator (who obtains exactly one label per wire) learns no information about the underlying plaintext value. For each gate in the circuit, a *garbled table* is generated by encrypting for all $2^2 = 4$ possible input combinations the corresponding output label using the corresponding input labels. To not leak any information from the position in the garbled table, its entries are permuted randomly. To preserve privacy, a crucial property required for privacy of GCs is that given a combination of two input labels only the corresponding output label can be decrypted from the garbled table.

Since garbling (and evaluation of the garbled circuit) requires calls to an encryption function, this is a natural point for optimizations and several encryption functions have been proposed and used in the literature (we give a summary in Sect. 3 and Tab. 1). In this paper, we show an attack and fixes on garbling schemes that use separate calls to a deterministic encryption function for the left and right input wire to derive per-wire encryption pads that are XORed together. Our attack becomes particularly devastating when Free-XOR garbling [KS08] is used, as in this case we reconstruct the global offset $R = \widetilde{w}^0 \oplus \widetilde{w}^1$ which allows to decrypt the entire GC.

2 Our Attack

In this section, we show our attack on garbling schemes that use separate calls to a deterministic encryption function to derive per-wire encryption pads that are XORed together. Later in Sect. 3, we show a set of existing constructions and summarize which ones are affected by our attack.

For a gate in the garbled circuit, let $\widetilde{w}_\ell$ be the left input wire label, $\widetilde{w}_r$ the right input wire label, and $\widetilde{w}_o$ the output wire label. The GC encryption function with per-wire encryption pads is defined as

$$E(\widetilde{w}_\ell, \widetilde{w}_r; \widetilde{w}_o) = \widetilde{w}_o \oplus F(\widetilde{w}_\ell; T) \oplus F(\widetilde{w}_r; T) \ , \tag{1}$$

where $F(k;T)$ is a Pseudo-Random Function (PRF) with key k and tweak T, often instantiated based on AES. The tweak T is set such that it is unique per gate and invocation of the GC encryption function (for the 4 possible input wire label combinations), so it is often set to the gate number concatenated with the position in the garbled table. However, the affected works omit to mention (and implement) that the tweak must also be different for the left and right call of the PRF, presumably because they make the implicit assumption that the two input wires are different, i.e., $\ell \neq r$, which however is never specified in the paper.

Our main observation is that when the tweak is the same for the left and right invocation of the PRF and the input wire labels are the same ($\ell = r$), then the encryption pads cancel out and leave both output wire labels *unencrypted* in the garbled table. Let $\widetilde{w}^0$ be the wire label corresponding to plaintext bit 0 and $\widetilde{w}^1$ be the wire label corresponding to 1. If the input wires for an AND gate are the same, then $\ell = r$ and the garbled table looks as follows (before the permutation of the entries):

$$E(\widetilde{w}_\ell^0, \widetilde{w}_r^0; \widetilde{w}_o^0) = \widetilde{w}_o^0 \oplus F(\widetilde{w}_r^0; T_{00}) \oplus F(\widetilde{w}_r^0; T_{00}) = \widetilde{w}_o^0; \tag{2}$$

$$E(\widetilde{w}_\ell^0, \widetilde{w}_r^1; \widetilde{w}_o^0) = \widetilde{w}_o^0 \oplus F(\widetilde{w}_r^0; T_{01}) \oplus F(\widetilde{w}_r^1; T_{01}); \tag{3}$$

$$E(\widetilde{w}_\ell^1, \widetilde{w}_r^0; \widetilde{w}_o^0) = \widetilde{w}_o^0 \oplus F(\widetilde{w}_r^1; T_{10}) \oplus F(\widetilde{w}_r^0; T_{10}); \tag{4}$$

$$E(\widetilde{w}_\ell^1, \widetilde{w}_r^1; \widetilde{w}_o^1) = \widetilde{w}_o^1 \oplus F(\widetilde{w}_r^1; T_{11}) \oplus F(\widetilde{w}_r^1; T_{11}) = \widetilde{w}_o^1 \ . \tag{5}$$

Hence, the per-wire encryption pads cancel out for the first and last row of the truth table and leak both output wire labels $\widetilde{w}_o^0$ and $\widetilde{w}_o^1$. Note that the second and third garbled table entries differ, because the tweaks T_{01} and T_{10} usually contain (or are explicitly concatenated with) the position in the garbled table and hence are different.

2.1 Breaking GCs with Per-Wire Encryption Pads

The implications of our attack depend on the specific garbling scheme.

General Attack. When the input wires are equal ($\ell = r$), the random pads cancel out and the garbled table of the AND gate contains *both wire labels unencrypted*: the zero output wire label $\widetilde{w}_o^0$ (see Eq. (2)) and the one output wire label $\widetilde{w}_o^1$ (see Eq. (5)). Hence, the GC evaluator obtains *both* output wire labels and can evaluate the remaining circuit following the AND gate not only for the output wire label corresponding to the current evaluation, but also for the other wire label. Depending on the circuit, this can leak crucial information as described next.

In the general attack, the evaluator obtains from the garbled AND gate with duplicate input wires four candidates for potential wire labels (taking each entry in the garbled table), two of which are the valid wire labels. He already knows one valid wire label from the evaluation of the GC, so the other wire label can be one of the remaining three candidates. Assume that the subsequent circuit contains a "normal" AND gate (i.e., with two different input wires), where w.l.o.g. the evaluator knows one valid wire label $\widetilde{w}_\ell^*$ from the evaluation of the GC, and three candidate wire labels $\widetilde{w}_\ell^1, \widetilde{w}_\ell^2, \widetilde{w}_\ell^3$ for the *left* input. He evaluates the garbled gate on the valid wire label and obtains a valid output wire label $\widetilde{w}_o^*$. Now, he evaluates the garbled gate for each of the candidate wire labels $\widetilde{w}_\ell^i$, $i \in \{1, 2, 3\}$. If the corresponding output wire label is equal to $\widetilde{w}_o^*$, then $\widetilde{w}_\ell^i$ is a valid input wire label, the plaintext output is 0 (because an AND gate evaluates to 0 in three cases), and the right plaintext input is 0 (because the AND gate evaluated to 0 for left inputs 0 and 1). Otherwise, if all output wire labels are different to $\widetilde{w}_o^*$, then the right plaintext input must be 1. Hence, the evaluator has successfully decrypted the plaintext value of the right input wire.

3-Row Reduction [NPS99]. The idea of garbled 3-Row Reduction [NPS99] is to fix one of the four entries in the table to 0^κ which then no longer needs to be sent to the evaluator. Here, we can apply the general attack described above by setting the omitted entry to 0^κ, as we explain in the following.

For a garbled 3-Row Reduction AND gate with duplicate input wires ($\ell = r$), we have two possible cases. In the first case, the first entry as in Eq. (2) (resp. the last entry as in Eq. (5)) of the garbled table is fixed to 0^κ, and therefore the actual wire label becomes all zeros, namely $\widetilde{w}_o^0 = 0^\kappa$ (resp. $\widetilde{w}_o^1 = 0^\kappa$). The other wire label is contained unencrypted in the last garbled table entry as in Eq. (5) (resp. the first garbled table entry as in Eq. (2)). In the second case, the second entry as in Eq. (3) (resp. third entry as in Eq. (4)) of the garbled table is fixed to 0^κ. In this case, the first entry (as in Eq. (2)) and the last entry (as in Eq. (5)) of the garbled table contain the unencrypted wire labels $\widetilde{w}_o^0$ and $\widetilde{w}_o^1$.

In both cases, the evaluator now has four candidate wire labels: three taken from the three garbled table entries and the fourth being the all zeros wire label 0^κ. Exactly two out of these four candidates are valid wire labels, which successfully decrypt the two output wire labels corresponding to the gate's plaintext inputs $(0, v_r)$ and $(1, v_r)$, exactly as in the general attack.

Free-XOR [KS08]. The idea of Free-XOR garbling [KS08] is that all wire labels are correlated with a global offset R that is supposed to remain hidden from the evaluator, i.e., $R = \widetilde{w}_i^0 \oplus \widetilde{w}_i^1$ for all wires i. This allows to securely evaluate XOR gates by XORing the corresponding wire labels, and therefore requires no communication for XOR gates. When Free-XOR garbling is used, our attack becomes devastating, since the garbled table of the AND gate where both input wires are the same ($\ell = r$) contains both $\widetilde{w}_o^0$ and $\widetilde{w}_o^1$, and the global offset $R = \widetilde{w}^0 \oplus \widetilde{w}^1$ is leaked. Using this, the evaluator knows *all* wire labels for *all* wires in the circuit and can evaluate it on arbitrary input combinations from which substantial information about the other party's inputs can be derived.

The attack works as follows: As for the general attack and the 3-Row Reduction [NPS99] described above, we obtain four candidate wire labels from an AND gate with duplicate input wires. The evaluator already knows one valid wire label from the GC evaluation, thus it can compute three candidate values for the global offset R, of which exactly one is the correct one. The evaluator can now determine which of the candidate global offsets is the correct one by evaluating *any* other AND gate (with two different input wires) in the circuit for different combinations of input wire labels as follows: An AND gate evalutes to 0 in three cases and to 1 in one case. Let R^* be the candidate global offset to check, and the evaluator knows the left input wire label $\widetilde{w}_\ell$ and the right input wire label $\widetilde{w}_r$. Now, he evaluates the AND gate for three combinations of input wire labels, e.g., $(\widetilde{w}_\ell, \widetilde{w}_r)$, $(\widetilde{w}_\ell \oplus R^*, \widetilde{w}_r)$, and $(\widetilde{w}_\ell, \widetilde{w}_r \oplus R^*)$. If at least two evaluations yield the same output wire label (corresponding to plaintext value 0), then R^* is the correct global offset R.

2.2 AND Gates with Duplicate Input Wires

Our attack assumes an AND gate where both input wires are the same ($\ell = r$). As $\text{AND}(X, X) = X$, this is a trivial identity gate which just outputs the input value. One would assume that such a gate would not occur in practice. We give three examples where such gates can occur naturally.

Circuits composed of NANDs only. In hardware synthesis, especially for ASICs, it is common to represent the function as NAND gates only. Here, inverters are built as $\text{NOT}(X) = \text{NAND}(X, X)$. As our attack described in Sect. 2.1 for AND gates works similarly for NAND gates, it immediately works for NAND-only functions that contain at least one NOT gate.

Agreeing on the circuit to be evaluated. SFE protocols usually assume that both parties "agree" on the circuit to be evaluated securely. When implementing this in practice, one party would suggest a circuit (or a high-level program from which the circuit is compiled) and the other party would agree to jointly evaluate it, e.g., after having tested its correctness for some inputs or inspecting the high-level program. If the party who suggests which circuit to evaluate is the one that evaluates the GC, it could suggest a circuit where one gate has duplicate input wires and then apply our attack. Overall, it is not enough for the other party to test that the circuit computes the intended function correctly.

Garbling with a hardware token [JKS⁺10]. The main bottleneck in protocols using GC is sending the garbled circuit which has size linear in the number of AND gates in the circuit. To circumvent this, the server can send to the client a hardware token (e.g., a smartcard) that generates the GC on his behalf as proposed in [JKS⁺10]. If the token has constant memory (independent of the size of the circuit), the client can send a description of the circuit gate-by-gate to the token. The token locally derives the wire labels corresponding to the gate's wire indices, generates the garbled table, and sends it to the client. Note that this does not work with garbling schemes such as 3-Row Reduction [NPS99], HalfGates [ZRE15], or Three-Halves Garbling [RR21], where the output wire label is a function of the input wire labels. The protocol adds additional measures that at the end of the protocol allow the server to verify that the client has given the "right" circuit to the token before the outputs of the function can be decrypted (for this the token locally stores a hash of the circuit description seen). Our attack can be applied to this token-based protocol as now the client can simply give an AND gate with duplicate input wires and immediately decrypt intermediate wires, hence circumventing the output decryption mechanism.

3 Affected Garbling Schemes

In this section, we list several prominent garbling schemes and denote if they are affected by our attack presented in Sect. 2. For clarity, we are presenting the encryption function constructions along the schemes using the notation from Sect. 2.

Two Parties. We summarize the garbling schemes in Tab. 1, where the *permutation bit* (for Point-and-Permute, see [BMR90]) of a wire label $\widetilde{w}$ is denoted as $p(\widetilde{w})$ and k is a fixed (and public) symmetric key. As can be seen from Tab. 1, the encryption function of several garbling schemes for the two-party setting are vulnerable to our attack [NPS99; MNP⁺04; LPS08; PSS⁺09; GLN⁺15].

Interestingly, [BHR12] explicitly states the requirement on circuits that gates must have different input wires ($\ell < r$), so their construction Garble1 with a Dual-Key Cipher (DKC) instantiated from a PRF is not affected by our attack. In fact, [BHR12] mentions in footnote 9 of their full version that [NPS99] "cannot handle a wire being used twice as an input to another gate", which might refer to the weakness discovered in our work. This construction is used in the formally verified secure two-party implementation of [ABB⁺17], where the EasyCrypt code contains the restriction that input wires are different ($\ell < r$), but this is not mentioned in the paper [ABB⁺17].

We note that the most recent garbling schemes [ZRE15; RR21] are not vulnerable to our attack as they go beyond encrypting single table entries. However, these schemes rely on a non-standard circularity

Table 1: Encryption functions in different garbled circuit constructions and whether the scheme is vulnerable to our attack. T: tweak, F: pseudo-random function, H: cryptographic hash function modeled as random oracle.

Scheme	Encryption Function $E(\widetilde{w}_\ell, \widetilde{w}_r; \widetilde{w}_o)$	Vulnerable
3-Row Reduction [NPS99]	$\widetilde{w}_o \oplus F(\widetilde{w}_\ell; T) \oplus F(\widetilde{w}_r; T)$	✓
LPS StM [LPS08]	$\widetilde{w}_o \oplus F(\widetilde{w}_\ell; T) \oplus F(\widetilde{w}_r; T)$	✓
PSSW StM [PSS+09]	$\widetilde{w}_o \oplus F(\widetilde{w}_\ell; T) \oplus F(\widetilde{w}_r; T)$	✓
Fairplay [MNP+04]	$\widetilde{w}_o \oplus F(\widetilde{w}_\ell; T \parallel p(\widetilde{w}_\ell) \parallel p(\widetilde{w}_r)) \oplus F(\widetilde{w}_r; T \parallel p(\widetilde{w}_\ell) \parallel p(\widetilde{w}_r))$	✓
Standard Assumptions [GLN+15]	$\widetilde{w}_o \oplus F(\widetilde{w}_\ell; T \parallel p(\widetilde{w}_\ell) \parallel p(\widetilde{w}_r)) \oplus F(\widetilde{w}_r; T \parallel p(\widetilde{w}_\ell) \parallel p(\widetilde{w}_r))$	✓
Garble1 w. DKC from PRF [BHR12]	$\widetilde{w}_o \oplus F(\widetilde{w}_\ell; T \parallel p(\widetilde{w}_\ell) \parallel p(\widetilde{w}_r)) \oplus F(\widetilde{w}_r; T \parallel p(\widetilde{w}_\ell) \parallel p(\widetilde{w}_r))$	✗
Formally Verified [ABB+17]	$\widetilde{w}_o \oplus F(\widetilde{w}_\ell; T \parallel p(\widetilde{w}_\ell) \parallel p(\widetilde{w}_r)) \oplus F(\widetilde{w}_r; T \parallel p(\widetilde{w}_\ell) \parallel p(\widetilde{w}_r))$	✓ / ✗
Free-XOR [KS08]	$\widetilde{w}_o \oplus H(\widetilde{w}_\ell \parallel \widetilde{w}_r \parallel T)$	✗
LPS ROM [LPS08]	$\widetilde{w}_o \oplus H(\widetilde{w}_\ell \parallel \widetilde{w}_r \parallel T)$	✗
PSSW ROM [PSS+09]	$\widetilde{w}_o \oplus H(\widetilde{w}_\ell \parallel \widetilde{w}_r \parallel T)$	✗
Proof of Yao [LP09]	$\widetilde{w}_o \oplus F(\widetilde{w}_\ell; r_1) \oplus F(\widetilde{w}_r; r_2), \quad r_1, r_2 \in_R \{0,1\}^\kappa$	✗
AES256 [KSS12]	$\widetilde{w}_o \oplus F(\widetilde{w}_\ell \parallel \widetilde{w}_r; T)$	✗
Fixed-key AES [BHK+13]	$\widetilde{w}_o \oplus F(k; 2\widetilde{w}_\ell \oplus 4\widetilde{w}_r \oplus T) \oplus 2\widetilde{w}_\ell \oplus 4\widetilde{w}_r \oplus T$	✗
HalfGates [ZRE15]	—	✗
Three-Halves [RR21]	—	✗

assumption [CKK+12], whereas today's best garbling scheme under standard assumptions [GLN+15] is affected by our attack.

Multiple Parties. In addition to the schemes in Tab. 1, Ben-Efraim et al. [BLO16] described an encryption function for multi-party garbling based on the approach of Beaver-Micali-Rogaway (BMR) [BMR90] using a *double-key pseudo-random function* as $F^2(\widetilde{w}_\ell, \widetilde{w}_r; T \parallel j)$, where j is the index of the party. Implementations of this can also be vulnerable to our attack, e.g., if $F^2(\widetilde{w}_\ell, \widetilde{w}_r; T \parallel j) = F(\widetilde{w}_\ell; T \parallel j) \oplus F(\widetilde{w}_r; T \parallel j)$ and the full construction for N parties is

$$\bigoplus_{i=1}^{N} (F(\widetilde{w}_\ell; T \parallel j) \oplus F(\widetilde{w}_r; T \parallel j)) \oplus \widetilde{w}_o \ .$$

A similar construction was presented in [BLO17] before the authors described a different construction using a key-homomorphic PRF. Later in their paper the authors wanted to "ensure that the same PRF is not queried twice with the same input" (see Definition 7 in [BLO17]). However, as in the two-party garbling schemes, the technical details are not explicitly presented and the encryption function remains vulnerable.

The focus in these works seems to be always on the case where two (or more) gates share a common input wire, hence the encryption functions are designed to include some gate identifier, i.e., the tweak T. We emphasize that this is not enough, since the same gate can also have duplicate input wires.

4 Affected MPC Frameworks

In this section, we list several MPC frameworks and denote if they are vulnerable to our attack presented in Sect. 2. We summarize the frameworks in Tab. 2 alongside with the used encryption function construction (cf. Sect. 3), and if the framework is vulnerable.

Fairplay [MNP+04] uses an affected encryption function (see Tab. 1) and is vulnerable to our attack. We note that their compiler replaces AND gates with duplicate input wires with identity gates (similarly,

Table 2: MPC Frameworks, their encryption function construction, and whether the framework is vulnerable to our attack.

MPC Framework	Encryption Function	Vulnerable
Fairplay [MNP+04]	Fairplay [MNP+04]	✓
TASTY [HKS+10]	Similar to [KS08; LPS08; PSS+09] (ROM)	✗
FastGC [HEK+11]		✗
VMCrypt [Mal11]		✗
ABY [DSZ15]		✗
TinyGarble [SHS+15]	HalfGates [ZRE15]	✗
Obliv-C [ZE15]		✗
ObliVM [LWN+15]		✗

the Frigate compiler [MGC+16] replaces them with a wire), so such gates are not generated automatically. However, one can input a circuit containing an AND gate with duplicate input wires into Fairplay's GC evaluation engine making our attack work (see Appendix A for details).

The most recent MPC frameworks such as TASTY [HKS+10], FastGC [HEK+11], VMCrypt [Mal11], ABY [DSZ15], TinyGarble [SHS+15], Obliv-C [ZE15], and ObliVM [LWN+15] implement a more recent encryption function for garbling and are not vulnerable to our attack.

5 Fixing Garbled Circuit with Per-Wire Encryption Pads

In this section, we give two countermeasures against our attack on Garbled Circuits (GCs) with per-wire encryption pads and AND gates with duplicate input wires.

1. The first countermeasure is to implement a simple validity check to make sure that the two input wires are indeed distinct, i.e., $\ell \neq r$.
2. The second countermeasure is to modify the encryption function such that the two PRFs are called on differing values even when both inputs are the same. Modern garbling schemes such as fixed-key AES garbling [BHK+13] already provide this inherently. Otherwise, we can simply append a 0 to the left PRF call and a 1 to the right PRF call. More specifically, we modify Eq. (1) into the following:

$$E(\widetilde{w}_\ell, \widetilde{w}_r; \widetilde{w}_o) = \widetilde{w}_o \oplus F(\widetilde{w}_\ell; T \parallel 0) \oplus F(\widetilde{w}_r; T \parallel 1) \ . \tag{6}$$

   So far, including the gate index in the tweak T ensured that PRF calls are unique when the same wire is used as input to multiple AND gates. Moreover, including the permutation bits (resp. position in the garbled table) ensured that the PRF calls are unique for the four garbled table entries as the input wire labels are used to encrypt multiple garbled table entries. By additionally including the left/right bit as in Eq. (6), we ensure that also within one garbled table entry the PRF calls are unique for the left and right input wire, even if this is the same wire.

Since most MPC frameworks decouple the task of compiling the function to be evaluated into a Boolean circuit from the secure evaluation of the Boolean circuit, we recommend to apply *both* countermeasures: The circuit compiler should check for all gates that the two input wires are always distinct, independent of the selection of the underlying encryption function (which could be even unknown at the compilation stage). Additionally, the implementation of the GC generation and evaluation engine should ensure that the protocol is secure even if a gate has duplicate input wires, e.g., by using the unaffected encryption functions from Tab. 1, or by applying the first or second countermeasure for the vulnerable schemes.

6 Related Work

Our work fits into a series of works that describe subtle issues in the area of garbling schemes, in particular their usage in higher-level protocols, proving their security, and their implementation.

Kiraz and Schoenmakers [KS06] describe an attack on Garbled Circuit (GC)-based protocols with stronger security using the Cut-and-Choose technique as it was performed in the initial work of Pinkas [Pin03] and the implementation in Fairplay [MNP$^+$04]. In these protocols, multiple GCs are generated of which some are opened and checked for correctness. The problem discovered by Kiraz and Schoenmakers stems from the way Oblivious Transfers (OTs) are used with which the evaluator obliviously obtains the wire labels corresponding to his inputs. They show that classical OT is not enough and propose to use committing OT instead. The issues pointed out were resolved in subsequent Cut-and-Choose-based protocols such as [LP07; LP12; LPS08; PSS$^+$09].

Choi et al. [CKK$^+$12] show that not only a variant of correlation robustness (as claimed in [KS08]), but also a form of circular security is required to prove security of the Free-XOR technique [KS08] using a weaker assumption than a random oracle.

Levi and Hazay [LH23] give side-channel attacks on implementations of GCs where all wire labels are correlated due to the Free-XOR technique [KS08].

7 Conclusion

In this work, we presented an attack against garbling schemes that use deterministic encryption functions to derive per-wire encryption pads for the left and right input wire that are XORed together. The attack stems from the implicit, but oftentimes not explicitly mentioned assumption that the input wires to an AND gates must be different. We showed that our attack affects several prominent garbling schemes and can occur in practical systems. We also showed how to easily and efficiently fix the vulnerable constructions to prevent our attack.

In the future, we hope that our findings help to decouple or at least make explicit the required properties about the circuit (or more general the function to be computed) which is the interface between compilation and the underlying secure computation protocols. This is important not only in papers with cryptographic constructions and proofs, but especially in implementation frameworks and in languages and compilers that aim at formally verified secure computation, e.g., [RHH14; ABB$^+$17; RSH19].

Acknowledgments

This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the Deutsche Forschungsgemeinschaft (DFG) within SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230.

References

[ABB$^+$17] J. B. Almeida, M. Barbosa, G. Barthe, F. Dupressoir, B. Grégoire, V. Laporte, and V. Pereira. A fast and verified software stack for secure function evaluation. In *CCS*, 2017 (cited on pages 4, 5, 7).

[BDK$^+$18] N. Büscher, D. Demmler, S. Katzenbeisser, D. Kretzmer, and T. Schneider. HyCC: Compilation of hybrid protocols for practical secure computation. In *CCS*, 2018 (cited on page 1).

[BHK$^+$13] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *S&P*, 2013 (cited on pages 1, 5, 6).

[BHR12] M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In *CCS*, 2012. Full version: https://ia.cr/2012/265 (cited on pages 4, 5).

[BLO16] A. Ben-Efraim, Y. Lindell, and E. Omri. Optimizing semi-honest secure multiparty computation for the Internet. In *CCS*, 2016 (cited on page 5).

[BLO17] A. Ben-Efraim, Y. Lindell, and E. Omri. Efficient scalable constant-round MPC via garbled circuits. In *ASIACRYPT*, 2017 (cited on page 5).

[BMR90] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *STOC*, 1990 (cited on pages 1, 4, 5).

[BNP08] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: A system for secure multi-party computation. In *CCS*, 2008 (cited on page 1).

[CKK+12] S. G. Choi, J. Katz, R. Kumaresan, and H.-S. Zhou. On the security of the "Free-XOR" technique. In *TCC*, 2012 (cited on pages 5, 7).

[DSZ15] D. Demmler, T. Schneider, and M. Zohner. ABY — A framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015 (cited on page 6).

[FHK+14] M. Franz, A. Holzer, S. Katzenbeisser, C. Schallhart, and H. Veith. CBMC-GC: An ANSI C compiler for secure two-party computations. In *Compiler Construction (CC)*, 2014 (cited on page 1).

[GLN+15] S. Gueron, Y. Lindell, A. Nof, and B. Pinkas. Fast garbling of circuits under standard assumptions. In *CCS*, 2015 (cited on pages 4, 5).

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *STOC*, 1987 (cited on page 1).

[HEK+11] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security*, 2011 (cited on pages 1, 6).

[HKS+10] W. Henecka, S. Kögl, A. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for automating secure two-party computations. In *CCS*, 2010 (cited on pages 1, 6).

[JKS+10] K. Järvinen, V. Kolesnikov, A. Sadeghi, and T. Schneider. Embedded SFE: Offloading server and network using hardware tokens. In *FC*, 2010 (cited on page 4).

[KS06] M. Kiraz and B. Schoenmakers. A protocol issue for the malicious case of Yao's garbled circuit construction. In *Information Theory in the Benelux (SITB)*, 2006 (cited on page 7).

[KS08] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP*, 2008 (cited on pages 1–3, 5–7).

[KSM+13] B. Kreuter, A. Shelat, B. Mood, and K. Butler. PCF: A portable circuit format for scalable two-party secure computation. In *USENIX Security*, 2013 (cited on page 1).

[KSS12] B. Kreuter, A. Shelat, and C.-H. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security*, 2012 (cited on pages 1, 5).

[LH23] I. Levi and C. Hazay. Garbled circuits from an SCA perspective: Free XOR can be quite expensive. . . In *CHES*, 2023 (cited on page 7).

[LP07] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, 2007 (cited on page 7).

[LP09] Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptology*, 2009 (cited on page 5).

[LP12] Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *J. Cryptology*, 2012 (cited on page 7).

[LPS08] Y. Lindell, B. Pinkas, and N. P. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *SCN*, 2008 (cited on pages 4–7).

[LWN+15] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi. ObliVM: A programming framework for secure computation. In *S&P*, 2015 (cited on page 6).

[Mal11] L. Malka. VMCrypt: Modular software architecture for scalable secure computation. In *CCS*, 2011 (cited on pages 1, 6).

[MGC+16] B. Mood, D. Gupta, H. Carter, K. Butler, and P. Traynor. Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation. In *EuroS&P*, 2016 (cited on pages 1, 6).

[MLB12] B. Mood, L. Letaw, and K. Butler. Memory-efficient garbled circuit generation for mobile devices. In *FC*, 2012 (cited on page 1).

[MNP+04] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — Secure two-party computation system. In *USENIX Security*, 2004 (cited on pages 1, 4–7, 9).

[NPS99] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Electronic Commerce (EC)*, 1999 (cited on pages 1, 3–5).

[NS23] R. Nieminen and T. Schneider. Breaking and fixing garbled circuits when a gate has duplicate input wires. *J. Cryptology*, 2023. To appear (cited on page 1).

[Pin03] B. Pinkas. Fair secure two-party computation. In *EUROCRYPT*, 2003 (cited on page 7).

[PSS+09] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In *ASIACRYPT*, 2009 (cited on pages 4–7).

[RHH14] A. Rastogi, M. A. Hammer, and M. Hicks. Wysteria: A programming language for generic, mixed-mode multiparty computations. In *S&P*, 2014 (cited on page 7).

[RR21] M. Rosulek and L. Roy. Three halves make a whole? Beating the Half-Gates lower bound for garbled circuits. In *CRYPTO*, 2021 (cited on pages 1, 4, 5).

[RSH19] A. Rastogi, N. Swamy, and M. Hicks. Wys*: A DSL for verified secure multi-party computations. In *Principles of Security and Trust (POST)*, 2019 (cited on page 7).

[SHS+15] E. M. Songhori, S. U. Hussain, A. Sadeghi, T. Schneider, and F. Koushanfar. TinyGarble: Highly compressed and scalable sequential garbled circuits. In *S&P*, 2015 (cited on pages 1, 6).

[SZ13] T. Schneider and M. Zoher. GMW vs. Yao? Efficient secure two-party computation with low depth circuits. In *FC*, 2013 (cited on page 1).

[Yao86] A. C. Yao. How to generate and exchange secrets. In *FOCS*, 1986 (cited on page 1).

[ZE15] S. Zahur and D. Evans. Obliv-C: A language for extensible data-oblivious computation. *Cryptology ePrint Archive 1153*, 2015. URL: http://eprint.iacr.org/2015/1153 (cited on page 6).

[ZRE15] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole — Reducing data transfer in garbled circuits using half gates. In *EUROCRYPT*, 2015 (cited on pages 1, 4–6).

A Our Attack on Fairplay

In this section, we show hot to apply our attack on the Fairplay MPC framework [MNP+04]. We start with the original circuit in Fig. 1(a), where Alice is the attacker with two input wires and Bob has one input wire.

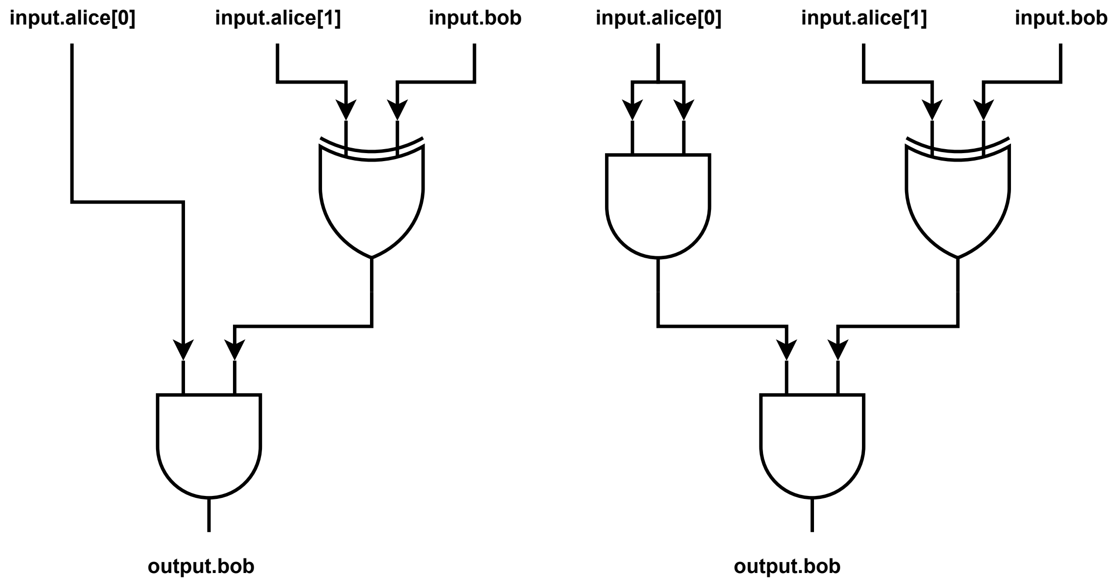
First, Alice turns the original circuit into the modified circuit shown in Fig. 1(b), where the first input of Alice goes into both inputs of the first AND gate. Note that both circuits compute the same function. Since the compiler of Fairplay either removes or replaces the first AND gate with an identity gate, Alice must manually modify the circuit description in Fairplay's Secure Hardware Definition Language (SHDL). The corresponding file `Opt.circuit` is given in List. 1.1.

The idea is that now the evaluator Alice gets both input wire labels for the final AND gate and can perform our *General Attack* as explained in Sect. 2.1 from which she can determine the output value of the XOR gate. This value reveals the input value of Bob, since Alice knows the other input value for the XOR circuit (and the output value from the attack). In this example also the output value is revealed, since Alice now knows both input values for the final AND gate.

A similar attack can be applied to more complex circuits in order to reveal Bob's input values.

Listing 1.1: Modified circuit for our attack (see Fig. 1(b)) in Fairplay's Secure Hardware Definition Language (SHDL).

```
0 input            //output$input.bob$0
1 input            //output$input.alice[1]$0
2 input            //output$input.alice[0]$0
3 gate arity 2 table [ 0 0 0 1 ] inputs [ 2 2 ] // duplicate input wires
4 gate arity 2 table [ 0 1 1 0 ] inputs [ 0 1 ]
5 output gate arity 2 table [ 0 0 0 1 ] inputs [ 4 3 ] //output$output.bob$0
```



(a) Original circuit (b) Modified circuit for our attack

Fig. 1: Example circuit