On the Leakage of Corrupted Garbled Circuits

Aurélien Dupin[1,2], David Pointcheval[3,4⋆], and Christophe Bidan[2]

[1] Thales Communications & Security, Gennevilliers, France
[2] CentraleSupélec, Rennes, France
[3] DIENS, École normale supérieure, CNRS, PSL University, Paris, France
[4] INRIA, Paris, France

Abstract. Secure two-party computation provides a way for two parties to compute a function, that depends on the two parties' inputs, while keeping them private. Known since the 1980s, Yao's garbled circuits appear to be a general solution to this problem, in the semi-honest model. Decades of optimizations have made this tool a very practical solution. However, it is well known that a malicious adversary could modify a garbled circuit before submitting it. Many protocols, mostly based on cut-&-choose, have been proposed to secure Yao's garbled circuits in the presence of malicious adversaries. Nevertheless, how much an adversary can modify a circuit and make it still executable has not been studied yet. The main contribution of this paper is to prove that any modification made by an adversary is equivalent to adding/removing NOT gates arbitrarily in the original circuit, otherwise the adversary can get caught. Thereafter, we study some evaluation functions for which, even without using cut-&-choose, no adversary can gain more information about the inputs by modifying the circuit. We also give an improvement over most recent cut-&-choose solutions by requiring that different circuits of the same function are used instead of just one.

Keywords: Garbled circuits, Malicious adversaries, Corruption of garbled circuits, Cut-and-choose

1 Introduction

The pioneering work of Yao [21], known as garbled circuits, is a general solution to the secure two-party computation problem, with a generator that builds the garbled circuit to be evaluated, and the evaluator that executes it on its inputs. It was originally designed in the semi-honest model and it was clear that a malicious generator could modify the logic gates of the garbled circuit before sending it to the evaluator for execution. Applying *cut-&-choose* to garbled circuits soon appeared to fix this issue, but requires to generate and evaluate a large number of garbled circuits.

⋆ This work was supported in part by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud)

Since then, a lot of work has been made to optimize the garbled circuits on the one hand [2, 15, 8, 22], and the cut-&-choose on the other hand [13, 10, 18, 14, 9, 19, 1, 20]. The best of these approaches requires s garbled circuits for statistical security 2^{-s}. To mitigate this overhead, other works have studied the amortized setting, where many evaluations of the same function occur [5, 11, 12]. These solutions only require $\mathcal{O}(s/\log\tau)$ circuits for τ evaluations. Other interesting solutions based on a gate-level cut-&-choose have emerged [16, 3, 17]. While these protocols have good asymptotic performances, their implementations still have a higher running time in practice than the best circuit-level cut-&-choose protocols. However, all these techniques aim at avoiding any kind of modification on the circuit. Nevertheless, it has never been studied which modifications a malicious generator can make to a single garbled circuit, still leading to an accepted execution, and then why the cut-&-choose is necessary.

Before the most recent general optimization of semi-honest garbling schemes of Zahur, Rosulek and Evans [22], such a study would have been meaningless. Indeed, it was obvious that an adversary could apply any modification of his choice as long as the topology of the circuit remains the same. In other terms, any binary gate could be turned into any other binary gate and the resulting corrupted garbled circuit would be still executable for any input. However, the recent improvement [22] manages to reduce the size of a garbled gate to only two ciphers (instead of three since the work of Naor et al. [15], or even four before that). Whereas this improvement can be seen as just a nice improvement for an honest party, it is clearly an extra constraint for a malicious party, given that he can now change only two variables instead of three or four. Since then, it is not clear which modifications can actually be made, and we prove in this paper that it is much more limited than suggested in the previous state-of-the-art.

More specifically, our first contribution is to show that an adversary is only able to add NOT gates to a circuit or to allow abortion of the protocol. The latter case is already known in the state-of-the-art as *selective failure attacks*. This result leads to our second contribution: we show some evaluation functions for which no such addition of NOT gates can help an adversary to learn more information about the inputs than the honest circuit. For such functions, this shows that a single circuit, without any cut-&-choose, is the best solution even against malicious adversaries (under the assumption that learning more information is not worth being caught). These results hold for the generic garbling scheme presented in [22].

When one of the parties does not have any inputs, privacy-free garbling schemes should be used instead. Privacy-free garbling schemes were used by Jawurek et al. [6] to build very efficient zero-knowledge proof of knowledge protocols. In this context, a prover has a secret, that satisfies some given statement, and wants to convince a verifier about his knowledge without revealing it. The works of [4] and later [22] showed that in this setting, the size of garbled circuits can be drastically reduced. Our results also hold for the optimal privacy-free scheme of [22], and thus limit a malicious verifier to add NOT gate or to make selective failure attacks.

Table 1. Garbled truth table

$H(k_A^0)$	$H(k_B^0)$	$E_{k_A^0}(E_{k_B^0}(k_C^{g(0,0)}))$
$H(k_A^0)$	$H(k_B^1)$	$E_{k_A^0}(E_{k_B^1}(k_C^{g(0,1)}))$
$H(k_A^1)$	$H(k_B^0)$	$E_{k_A^1}(E_{k_B^0}(k_C^{g(1,0)}))$
$H(k_A^1)$	$H(k_B^1)$	$E_{k_A^1}(E_{k_B^1}(k_C^{g(1,1)}))$

Table 2. Commitments of outputs

$H(k_D^0)$
$H(k_D^1)$

The next section is a reminder of how to garble a circuit with the most recent optimizations. Then, we show in Section 3 that an adversary can add NOT gates to a circuit or make selective failure attack. We also describe how to make these corruptions. The main contribution of our paper comes in Section 4: we prove that no other modification can be made for a large class of circuits that we define. Section 5 gives a construction of garbled circuits that reduces the possible deviations. Finally, we study in Section 6 the impact of our contribution on some real-case circuits and show that some of them do not require cut-&-choose based solutions to ensure privacy, and that for the others, cut-&-choose can be improved for free by recommending that different circuits of the same function are used instead of just one.

2 A Reminder of the Garbled Circuit Optimizations

Let us remind how garbled circuits are designed. The point-and-permute technique [2], the 25-%-row reduction [15] and the free-XOR [8] are briefly explained in the beginning of this section. We refer the reader to these papers for details.

2.1 The Basic Construction

We assume both parties agree on the function to evaluate and the circuit representation of it. One party, called the generator (noted $\mathcal{G}$), randomly chooses two garbled keys k_i^0 and k_i^1 for each wire w_i of the circuit, representing respectively 0 and 1. Then, for the Boolean gate g taking as input the wires w_A and w_B and returning the output in w_C, $\mathcal{G}$ computes the garbled truth table as shown in Table 1, using a hash function H and a symmetric encryption function E.

The rows of this table are randomly shuffled before they are sent to the other party, the evaluator (noted $\mathcal{E}$). Then, with the keys k_A^a and k_B^b, $\mathcal{E}$ is able to compute $k_C^{g(a,b)}$. That way, $\mathcal{E}$ evaluates the circuit and obtains output garbled keys. $\mathcal{G}$ also provides commitments of the garbled keys of the output of the circuit (shown in Table 2) that allow $\mathcal{E}$ to get an exploitable result, and check the correct evaluation. If $\mathcal{E}$ is not supposed to learn the result, these commitments can be randomly shuffled and sent to $\mathcal{E}$. If the result does not match any commitment, then an error or a misbehavior in the generation of the garbled circuit may have occurred.

2.2 The Point-and-Permute Trick

The point-and-permute trick of Beaver, Micali and Rogaway [2] allows to get rid of the two input columns of Table 1. For every wire w_i, $\mathcal{G}$ picks a random bit p_i, called *permute bit*. The least significant bit of a garbled key (later called *select bit*) is now the clear value masked with the permute bit. For example, the select bit of k_A^a is $a \oplus p_A$. Remark that the select bit of every garbled key is arranged so that the two garbled keys of a wire have opposite select bits. Then, the garbled truth table can be arranged by these select bits, as shown in Table 3. Then, $\mathcal{E}$ uses the select bit of the garbled keys to determine which row he should decrypt.

In the rest of the paper, we call $s()$ the function that takes a garbled key as input and outputs the select bit of that key. It tells $\mathcal{E}$ which line of Table 3 he should use while executing: $s(k_A^a) = a \oplus p_A$, $s(k_A^{p_A}) = 0$, and $s(k_A^0) = p_A$.

2.3 The 25%-Row Reduction

The 25%-row reduction of Naor, Pinkas and Sumner [15] allows to reduce the number of ciphertexts per garbled gate. The main idea is to choose one of the output keys so that the first ciphertext is nullified. Then, one less ciphertext has to be transmitted. For example, in Table 4, $k_C^{g(p_A,p_B)}$ is the decryption of zero.

2.4 The Free-XOR Trick

The free-XOR trick of Kolesnikov and Schneider [8] allows to garble XOR gates for free. The idea is to choose a *global offset* Δ that will be used to differentiate the two garbled keys of a same wire. In other words, for any wire, the bitwise XOR of the two garbled keys is Δ. That way, when the evaluator has to evaluate a XOR gate, he just bitwise XOR the two input garbled keys to obtain the output garbled key. Note that, in order to make it compatible with the point-and-permute technique, Δ has to be odd.

2.5 The Two Half-Gates Technique

We now describe how to garble an AND gate using only two ciphertexts. As noticed in the original paper of Zahur, Rosulek and Evans [22]:

$$\forall \gamma \in \mathbb{F}_2, a \wedge b = \underbrace{(a \wedge \gamma)}_{\text{First half-gate}} \oplus \underbrace{(a \wedge (b \oplus \gamma))}_{\text{Second half-gate}}$$

Table 3. Garbled truth table with permute bit

$E_{k_A^{p_A}}(E_{k_B^{p_B}}(k_C^{g(p_A,p_B)}))$
$E_{k_A^{p_A}}(E_{k_B^{\overline{p_B}}}(k_C^{g(p_A,\overline{p_B})}))$
$E_{k_A^{\overline{p_A}}}(E_{k_B^{p_B}}(k_C^{g(\overline{p_A},p_B)}))$
$E_{k_A^{\overline{p_A}}}(E_{k_B^{\overline{p_B}}}(k_C^{g(\overline{p_A},\overline{p_B})}))$

Table 4. 25% reduced garbled truth table

~~$E_{k_A^{p_A}}(E_{k_B^{p_B}}(k_C^{g(p_A,p_B)})) = 0$~~
$E_{k_A^{p_A}}(E_{k_B^{\overline{p_B}}}(k_C^{g(p_A,\overline{p_B})}))$
$E_{k_A^{\overline{p_A}}}(E_{k_B^{p_B}}(k_C^{g(\overline{p_A},p_B)}))$
$E_{k_A^{\overline{p_A}}}(E_{k_B^{\overline{p_B}}}(k_C^{g(\overline{p_A},\overline{p_B})}))$

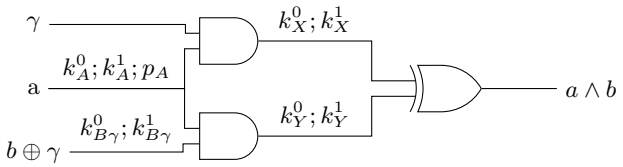

Fig. 1. The two half-gates of an AND gate

Then, an AND gate is replaced by the sub-circuit shown in Fig. 1 with the bit γ randomly chosen by $\mathcal{G}$. A *half-gate* is defined as a gate for which one of the inputs is known by one of the parties. In the first half-gate, γ is known by $\mathcal{G}$, whereas in the second, $b \oplus \gamma$ can be revealed to $\mathcal{E}$ without leaking b (by revealing the permute bit of the corresponding wire). Using this knowledge, each half-gate can be reduced to one ciphertext (called G and E). $\mathcal{G}$ computes these ciphertexts as described in Table 5 and send them to $\mathcal{E}$. We note i and j two distinct and public indexes used as salts for hash function, as detailed in [22].

Table 5. Garbling the half-gates

First half-gate		Second half-gate	
Garbled table if $\gamma = 0$	Garbled table if $\gamma = 1$	$b \oplus \gamma$	Garbled table
$k_X^0 \oplus H(k_A^{p_A}\|i) = 0$	$k_X^{p_A} \oplus H(k_A^{p_A}\|i) = 0$	0	$k_Y^0 \oplus H(k_{B\gamma}^0\|j) = 0$
$k_X^0 \oplus H(k_A^{\overline{p_A}}\|i) = G$	$k_X^{\overline{p_A}} \oplus H(k_A^{\overline{p_A}}\|i) = G$	1	$k_Y^0 \oplus k_A^0 \oplus H(k_{B\gamma}^1\|j) = E$

To simplify notations, we will omit the salts i and j unless they are necessary to our proofs. Then, $\mathcal{E}$ executes the garbled gate using the garbled inputs. Table 6 shows the four different algorithms of evaluation, depending on the garbled inputs the evaluator has. Of course, three of them output the same garbled key (for an output 0 to the AND gate). Knowing the clear value of $b \oplus \gamma$ and the select bit of k_A^a, the evaluator is able to choose the correct algorithm.

Table 6. Evaluating the half-gates

Inputs		First half-gate	Second half-gate	Garbled output key
$k_A^{p_A}$	$k_{B\gamma}^0$	$H(k_A^{p_A})$	$H(k_{B\gamma}^0)$	$K_1 = H(k_A^{p_A}) \oplus H(k_{B\gamma}^0)$
$k_A^{p_A}$	$k_{B\gamma}^1$	$H(k_A^{p_A})$	$E \oplus H(k_{B\gamma}^1) \oplus k_A^{p_A}$	$K_2 = E \oplus H(k_A^{p_A}) \oplus H(k_{B\gamma}^1) \oplus k_A^{p_A}$
$k_A^{\overline{p_A}}$	$k_{B\gamma}^0$	$G \oplus H(k_A^{\overline{p_A}})$	$H(k_{B\gamma}^0)$	$K_3 = G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^0)$
$k_A^{\overline{p_A}}$	$k_{B\gamma}^1$	$G \oplus H(k_A^{\overline{p_A}})$	$E \oplus H(k_{B\gamma}^1) \oplus k_A^{\overline{p_A}}$	$K_4 = E \oplus G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^1) \oplus k_A^{\overline{p_A}}$

Table 7. Evaluating the privacy-free garbled gate

Inputs		Garbled output key
k_A^0	k_B^0	$K_1 = H(k_A^0)$
k_A^0	k_B^1	$K_2 = H(k_A^0)$
k_A^1	k_B^0	$K_3 = E \oplus H(k_A^1) \oplus k_B^0$
k_A^1	k_B^1	$K_4 = E \oplus H(k_A^1) \oplus k_B^1$

2.6 Linear and Non-Linear Gates

The two half-gates technique works for "any gate whose truth table contains an odd number of ones (e.g. AND, NAND, OR, NOR, etc.)" [22]. Let's call them *non-linear gates* in $\mathbb{F}_2$. For these gates, the garbling scheme of Table 5 is slightly different, but the evaluation scheme of Table 6 is identical. By opposition, we call *linear gates* in $\mathbb{F}_2$ the eight others (e.g., XOR, XNOR, True, False, etc.). Only non-linear gates are garbled, since linear gates are free.

2.7 The Case of Privacy-free Garbled Circuits

Jawurek et al. [6] demonstrated that garbled circuits can be used as a practical solution to zero-knowledge proof protocols. The evaluator (i.e. the prover) can prove any statement "$\exists x : f(x) = 1$" without revealing x, using a single garbled circuit for f.

Frederiksen et al. [4] showed that in this context, the size of the garbled circuits can be significantly reduced. Since the evaluator knows the entire input, he also knows the value of each intermediate wire, and thus there is no need to hide these values to the evaluator. The work of [22] provides an optimal garbling scheme in this context. Since the evaluator knows every value, the non-linear gates can be viewed as half-gates, and thus require a single ciphertext. For an AND gate, this ciphertext is $E = H(K_A^0) \oplus H(K_A^1) \oplus K_B^0$ and the evaluation algorithm is modified as presented in Table 7.

Although only the general case is presented in this paper, our results also hold for the privacy-free garbling scheme of [22].

3 Corruption of a Garbled Circuit

Now that we have seen how to garble a circuit, let us see how the generator $\mathcal{G}$ can cheat. We consider two kinds of corruptions: those that can not be detected, since the evaluation always succeeds, and those that may lead the adversary to get caught, because of an invalid output (inconsistent with the commitments).

3.1 Selective Failure Attacks

We first consider the latter category, that leads to the so-called *selective failure attacks*. These are corruptions of the garbled circuit that make it executable

only if a condition on internal values is met. If not, the protocol aborts: $\mathcal{E}$ does not obtain a correct output and thus can not send back a result to $\mathcal{G}$. Then $\mathcal{G}$ learns whether the condition is met, but, if not, $\mathcal{E}$ detects the corruption and $\mathcal{G}$ gets caught. More specifically, the malicious $\mathcal{G}$ could use inconsistent keys to construct a garbled gate or to exchange inputs during the OT phasis.

Let us see two examples, first, with the modification of an internal gate, and then with a corrupted OT during the initialization phasis.

Alteration of an Internal Garbled Gate. We consider an internal gate garbled as in Table 5. Suppose a key $k^{1*}_{B\gamma}$ has been used for the garbling instead of $k^1_{B\gamma}$. During the evaluation, if $\mathcal{E}$ gets $k^1_{B\gamma}$, then after this corrupted gate, he will get an inconsistent key, that will be used to evaluate the rest of the circuit. It will not be detected until the last gate of the circuit, the output of which will not match any commitment. An example of this corrupted garbled gate is shown in Annex A.

Because he can not return a valid output, $\mathcal{E}$ is forced to abort the protocol. If the protocol aborts, $\mathcal{G}$ learns that $k^1_{B\gamma}$ should have been used and $\mathcal{E}$ detects the attack. But if the protocol runs correctly, $\mathcal{G}$ learns the normal output, plus an internal bit $k^0_{B\gamma}$, and the $\mathcal{E}$ does not detect it. In the previous works, as any other corruption of the circuit, this attack is prevented by cut-&-choose solutions.

Corruption during the OT. We now consider $\mathcal{E}$ has some input bit b and $\mathcal{G}$ generates honestly the circuit using k^0_B and k^1_B. However, during the OT phasis, $\mathcal{G}$ uses k^0_B and k^{1*}_B. Then, if $b = 1$, $\mathcal{E}$ gets an inconsistent key and the leakage of information is just as before. Note that the circuit itself is not modified, meaning that cut-&-choose does not solve this issue. More specific and efficient solutions have been designed, such as *s-probe-resistant matrices* [10, 19].

Information vs. Detection. In both above cases, the malicious generator can get detected since the failure is part of the way to learn information. Hence, the adversary must make the protocol fail with non-negligible probability to learn something. In the rest of the paper, we restrict the study to context where the potential gain of information is not worth the risk of getting caught by the honest party. Moreover, if the garbled circuit and the inputs were signed by the generator, the evaluator could easily prove to some authority that the garbled circuit is indeed non-executable. This seems reasonable in many real-life cases. We thus limit alterations to the protocol that do never lead to a failure.

3.2 Undetectable Corruptions

In order to be undetectable, the corrupted circuit must keep the same topology and the outputs must match the commitments. We later prove that this limits modifications to turning any non-linear gate into any other non-linear gate.

But before showing this is the only possible alteration, let us show how such an alteration can work: if $\mathcal{G}$ garbles the half-gates by switching some garbled keys, as shown in Table 8, it is easy to prove that the resulting gate computes

$\bar{a} \wedge b$, and that the execution algorithm of $\mathcal{E}$ remains unchanged. Moreover, this modified garbled truth table is actually the correct way of garbling $\bar{a} \wedge b$.

Table 8. Turning $a \wedge b$ into $\bar{a} \wedge b$

First half-gate		Second half-gate	
Garbled table if $\gamma = 0$	Garbled table if $\gamma = 1$	$b \oplus \gamma$	Garbled table
$k_X^0 \oplus H(k_A^{p_A}) = 0$	$k_X^{\overline{p_A}} \oplus H(k_A^{p_A}) = 0$	0	$k_Y^0 \oplus H(k_{B\gamma}^0) = 0$
$k_X^0 \oplus H(k_A^{\overline{p_A}}) = G$	$k_X^{p_A} \oplus H(k_A^{\overline{p_A}}) = G$	1	$k_Y^0 \oplus k_A^1 \oplus H(k_{B\gamma}^1) = E$

Similarly, we show in Annex A how to obtain a correct garbling of $a \wedge \bar{b}$ and $\overline{a \wedge b}$ from a corrupted AND gate. Combining these three modifications, one can turn a AND gate into any of the eight non-linear gates. The example of the OR gate is also given in Annex A. Note that other ways exist to obtain the same results, but we chose these ones because they represent the honest ways of garbling $\bar{a} \wedge b$, $a \wedge \bar{b}$ and $\overline{a \wedge b}$.

These modifications can be made arbitrarily by the generator and it will not be detected by the evaluator, unless some cut-&-choose solution is used. In the rest of the paper, we are proving that no other modification can be made by a probabilistic polynomial-time adversary, or the protocol may abort, but the adversary does not want to take the risk of getting caught.

4 Delimitation of the Corruption

Let us now prove that the above modifications and their combinations are the only ones that can be made by an adversarial generator $\mathcal{G}$, if it does not want to get detected. We call f the function to evaluate and C_f a Boolean circuit representation of it.

We assume in this section that the (possibly corrupted) garbled circuit is executable for all inputs, since the adversary does not want to get detected.

Let us start with the obvious limitations. First, as already noticed, the topology of the Boolean circuit to evaluate is public, which ensures that $\mathcal{G}$ can not cheat on the number of gates or the way they are connected. Second, because of the free-XOR trick [8], XOR gates have no garbled truth tables to transmit, then they can not be corrupted either.

But $\mathcal{G}$ can still garble "correctly" another circuit $\mathsf{C}_{f'}$ (computing some other function f' instead of f). By correct garbling, we mean that $\mathcal{G}$ garbles $\mathsf{C}_{f'}$ in accordance with the garbling algorithm (and its optimizations), and keeps the number of gates and the way they are connected to each other unchanged, as if f' was the correct function to evaluate. XOR gates of C_f must also be present in $\mathsf{C}_{f'}$. More specifically, we have the following restrictions :

1. Only two ciphers are sent for each non-linear gates.
2. XOR gates are not transmitted.

3. There is a global offset that differentiates the two garbled keys of each wire of $\mathsf{C}_{f'}$ (in accordance with the free-XOR trick [8]) and this offset is odd (as required by the point-and-permute technique [2]).
4. $\mathsf{C}_{f'}$ is Boolean: for every wire of the circuit, there are two garbled keys.

It is obvious that the first two requirements are met. Otherwise, $\mathcal{E}$ will refuse to evaluate the circuit. In this section, we show that if the input wires of C_f are correctly garbled (i.e. have a common odd offset), then the rest of the circuit is also correctly garbled, or the protocol may abort. Thereafter, we provide a construction to ensure that input wires are correct. This will help to prove that the adversary is only able to turn a non-linear gate into another non-linear gate.

For the sake of simplicity, we consider that the original circuit is only composed of XOR and AND gates and we show later that the same result applies for the other gates.

4.1 Impossibility of Reducing the Number of Garbled Keys to One

The first thing to prove is that, for any garbled gate, there are at least two output garbled keys. Consider the case where an adversary wants to alter an AND gate (w.l.o.g.) so that it always outputs True (or always False), whatever the inputs are. Then, he must choose E and G in Table 6, so that the four garbled output keys are equal. Then, we have the following system of equations:

$$\begin{cases} K_2 = K_1 \\ K_3 = K_1 \\ K_4 = K_1 \end{cases} \iff \begin{cases} E = H(k^0_{B\gamma}) \oplus H(k^1_{B\gamma}) \oplus k^{p_A}_A \\ G = H(k^{p_A}_A) \oplus H(k^{\overline{p_A}}_A) \\ k^{p_A}_A = k^{\overline{p_A}}_A \end{cases}$$

Lemma 1. *For any garbled gate, if the first operand has two garbled keys with an odd offset, then the output wire has at least two possible garbled keys.*

Proof. If we indeed have $k^{p_A}_A \oplus k^{\overline{p_A}}_A = \Delta$ that is odd, then the four keys can not be equal. □

4.2 Impossibility of Three-Key Wires - Part 1

In the last part, we showed that if the input wires are correct, there are at least two garbled keys per wire. In this section, we aim to prove there is no wire having more than two possible garbled keys, while the circuit remains evaluable. As described in Section 2, the garbled circuit is considered to have two commitments on the garbled keys of its output wires. This ensures that output wires have at most two possible keys, or the protocol aborts when a third key is obtained. Then, if some wire of the circuit has three possible keys or more, then there must be a gate that reduces it to only two. We show that such a gate is impossible.

As defined in Section 2, $s()$ refers to the function that takes a garbled key as input and outputs the select bit of that key. This function tells the evaluator what line of Table 6 he should use while evaluating: $s(k^{p_A}_A) = 0$ and $s(k^0_A) = p_A$.

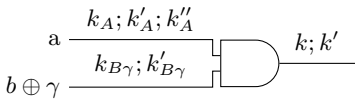

Fig. 2. Reducing the number of keys of the first operand: Impossible

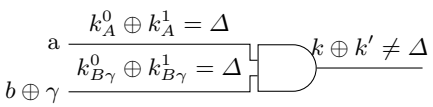

Fig. 3. Modification of the offset: Impossible

Since the previous notations are irrelevant, if there are more than two keys or if the point-and-permute trick is not followed, we now call k_X, k'_X the two distinct garbled key of a wire, and k''_X a third garbled key when needed.

We remind that $H()$ is a hash function that is assumed to behave like a random function from $\mathbb{F}_{2^N}$ to $\mathbb{F}_{2^N}$ and we expect the following problems to be computationally unfeasible by any polynomially bounded adversary :

1. Finding distinct $k_1, k'_1 \in \mathbb{F}_{2^N}$, so that $H(k_1) = H(k'_1)$ requires $2^{N/2}$ evaluations of $H()$ on average (Birthday paradox).
2. Finding distinct $k_1, k'_1 \in \mathbb{F}_{2^N}$, so that $H(k_1) \oplus k_1 = H(k'_1) \oplus k'_1$ requires $2^{N/2}$ evaluations of $H()$ on average (Equivalent to the birthday paradox).
3. For given i and j, finding $k_1, k_1^{'}, k_2, k_2^{'} \in \mathbb{F}_{2^N}$, so that $k_1 \neq k_1^{'}$, $k_2 \neq k_2^{'}$ and $H(k_1|i) \oplus H(k_1^{'}|i) \oplus H(k_2|j) \oplus H(k_2^{'}|j) = 0$ requires $2^{N/4}$ evaluation of $H()$ on average.
4. For given i and j, finding $k_1, k_1^{'}, k_2, k_2^{'} \in \mathbb{F}_{2^N}$, so that $k_1 \neq k_1^{'}$, $k_2 \neq k_2^{'}$ and $H(k_1|i) \oplus k_1 \oplus H(k_1^{'}|i) \oplus k_1^{'} \oplus H(k_2|j) \oplus H(k_2^{'}|j) = 0$ requires $2^{N/4}$ evaluations of $H()$ on average.

All these properties can be proven if H is modeled as a random oracle, using the birthday paradox bound. Note that in the definition of these problems, the adversary can freely choose the garbled keys k_1, $k_1^{'}$, k_2 and $k_2^{'}$, whereas for garbled gates, they are constrained by the garbling of the previous gates. Intuitively, solving these problems requires a lot more evaluations than listed above. These properties lead to the following lemma, which proof can be found in Annex B, illustrated in Figure 2:

Lemma 2. *For any garbled gate, if the first operand has at least three possible garbed keys, and the second has at least two, then the output wire has at least three garbled keys.*

4.3 Impossibility of Three-Key Wires - Part 2

In this part, we study the opposite problem, where the second operand has at least three garbled keys and the first has at least two. The proof being more tricky, we need to demonstrate Lemma 3 as a preliminary step.

Lemma 3. *For any gate, if the operands have two garbled keys and have the same odd offset, then the output wire has the same offset or at least three keys.*

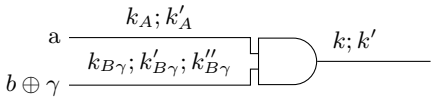

Fig. 4. Reducing the number of keys of the second operand: Impossible

Proof. This situation is illustrated in Figure 3. Consider the case where the adversary wants to corrupt a garbled AND gate (w.l.o.g.) so that the offset is altered in the process. Then, he must choose such E and G in Table 6. We prove here that it can not be done. As stated in Section 2, there are four evaluation algorithms, the output of which, noted K_1 to K_4 collide so that there are only two distinct results (the case $K_1 = K_2 = K_3 = K_4$ is already proven to be impossible). We develop here the case $K_1 = K_2 = K_3$ and the others in Annex C.

$$\begin{cases} K_1 = H(k_A^{p_A}) \oplus H(k_{B\gamma}^0) \\ K_2 = E \oplus H(k_A^{p_A}) \oplus H(k_{B\gamma}^1) \oplus k_A^{p_A} = K_1 \\ K_3 = G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^0) = K_1 \\ K_4 = E \oplus G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^1) \oplus k_A^{\overline{p_A}} \end{cases} \Rightarrow \begin{cases} K_1 = H(k_A^{p_A}) \oplus H(k_{B\gamma}^0) \\ E = H(k_{B\gamma}^0) \oplus H(k_{B\gamma}^1) \oplus k_A^{p_A} \\ G = H(k_A^{p_A}) \oplus H(k_A^{\overline{p_A}}) \\ K_4 = K_1 \oplus \Delta \end{cases}$$

The other cases give the same result or are easily proven to be impossible. For XOR gates, since garbled keys are simply XORed together, the output wire has the same offset as the operands, which ends the proof of Lemma 3. □

We now aim at concluding the last case with the following lemma:

Lemma 4. *If the input wires of the circuit have garbled keys with an odd global offset, then the garbled circuit cannot have a gate such that the second operand has at least three possible garbed keys, and the first has at least two, while the output wire has only two garbled keys.*

Proof. From Lemma 2, we know this is true if the two operands have at least three keys. We thus focus to the case where the first operand has two keys and the second operand has three keys, as illustrated in Figure 4. For this proof, we consider that the input wires are correctly garbled : these wires have two garbled keys and they have an odd global offset Δ. We study the case of the first gate, called $\mathcal{F}$, of the circuit (in topological order) that has two garbled inputs for the first operand and three (or more) for the second.

Since $\mathcal{F}$ is the first of its kind in the circuit and because of Lemma 2, the sub-circuit that links the inputs of the circuit to the first operand wire of $\mathcal{F}$ have only wires with exactly two garbled keys. Moreover, since all input wires of this sub-circuit have the global offset Δ and because of Lemma 3, all wires of the sub-circuit, including the first operand of $\mathcal{F}$, have this same odd offset Δ.

Remark that an input wire of the circuit can not have three keys. Then the three keys (or more) of the second operand of $\mathcal{F}$ come from a corrupted gate $\mathcal{F}'$ that outputs three distinct keys (or more). However, the two operand wires of

$\mathcal{F}'$ have two possible garbled keys, and, with a similar approach, we can show that they have the same offset Δ as the first operand of $\mathcal{F}$.

Using the same convention as before, we call k_A and k'_A the keys of the first operand and $k_{B\gamma}$, $k'_{B\gamma}$ and $k''_{B\gamma}$ the keys of the second operand. As stated above, the computation of $k_{B\gamma}$, $k'_{B\gamma}$ and $k''_{B\gamma}$ engages the choice of Δ, and consequently $k_A \oplus k'_A$. We develop here the trickiest case $s(k_A) = s(k_{B\gamma}) = s(k'_{B\gamma}) = 0$ and $s(k'_A) = s(k''_{B\gamma}) = 1$, which gives the following set of keys :

$$\begin{array}{ll} K_1 = H(k_A) \oplus H(k_{B\gamma}) & K_4 = G \oplus H(k'_A) \oplus H(k_{B\gamma}) \\ K_2 = H(k_A) \oplus H(k'_{B\gamma}) & K_5 = G \oplus H(k'_A) \oplus H(k'_{B\gamma}) \\ K_3 = E \oplus H(k_A) \oplus H(k''_{B\gamma}) \oplus k_A & K_6 = E \oplus G \oplus H(k'_A) \oplus H(k''_{B\gamma}) \oplus k'_A \end{array}$$

Thanks to the property of the hash function, K_1 and K_2 are different. We show in Annex D that there is no E and G such that K_3 to K_6 collide with K_1 or K_2. All other cases are also proven to be impossible in Annex D. This ends the proof of Lemma 4. □

4.4 Impossibility of Turning a Non-Linear Gate into a Linear Gate

In Section 3, we showed how to turn a non-linear gate into any other non-linear gate. We will now prove that, since an adversarial generator is limited to Boolean circuits and can not deviate from the global offset, he can not turn a non-linear gate into a linear gate. We focus on the case of an AND gate.

Lemma 5. *For any non-linear gate, if the two operands have two garbled keys and have the same odd offset, then it can not be turned into a linear gate.*

Proof. We already demonstrated that an AND gate can not be corrupted into a gate that always outputs True (or False). All other cases are proven to be impossible in Annex E. □

Two particular cases of this lemma clearly reduce the possibilities of a malicious generator. First, an adversary can not force the output of a non-linear gate, and thus can not trivially force the output of the entire garbled circuit. Moreover, the adversary can not alter a gate so that it always outputs the first input a ($K_1 = K_2$ and $K_3 = K_4$). This last example is interesting: it actually means that the malicious generator cannot modify the circuit so that the evaluator's inputs go directly to the output through the circuit.

4.5 About Other Non-Linear Gates

We showed in Section 3 how to turn $a \wedge b$ into $\bar{a} \wedge b$, $a \wedge \bar{b}$ and $\overline{a \wedge b}$. It appears that these deviations and their combinations are identical to the honest ways of garbling these respective gates, described in [22].

Then, an honest garbling of $\overline{a \wedge b}$ (or any other non-linear gate) can be obtained from a corruption of $a \wedge b$. Thus, there is no modification that can be made on $\overline{a \wedge b}$ and that cannot be made on $a \wedge b$. Therefore, any non-linear gate can only be turned into another non-linear gate.

4.6 Fitting Everything Together

Assembling the lemmata previously proved, we obtain Theorem 6, which is the main contribution of this paper.

Theorem 6. *If all the operands of the first garbled gates can take the two values according to the evaluator's inputs (while the generator's inputs are fixed), and if there are output commitments, then the adversarial generator is limited to turn any non-linear gates into other non-linear gates.*

This theorem means that if we can guarantee that the first garbled gates (the non-linear gates that are the closest to the input wires) can take the two possible inputs, independently on each wire, according to the evaluator's choice, then all the garbled gates can only be altered into any non-linear gates.

Proof. Using Lemma 1, if the input wires of the first garbled gates all have two possible garbled keys, then there is no wire in the rest of the circuit that has only one possible key. Combining Lemmata 2 and 4, if the input wires of the first garbled gates of the circuit all have the same odd global offset and if the circuit has output commitments, then no wire of the rest of the circuit has more than two possible garbled keys. Moreover, with the same conditions, Lemma 3 shows that all wires share the same odd global offset. Then, Lemma 5 comes last and shows that non-linear gates can only be turned into other non-linear gates, and that this is the only possible corruption. □

It remains to study the conditions so that the starting point of this theorem is satisfied: all the inputs of the first garbled gates have two possible garbled keys. How to guarantee some wires to have two possible garbled keys, with the same global odd offset? We will show below that it is possible to make sure that all the evaluator's inputs are converted into garbled keys with a common global odd offset. But there is no way to do the same for the generator's inputs. Indeed, he can not be forced to choose his inputs after generating the garbled circuit. On the other hand, XOR gates can not be corrupted, and so a XOR gate with an evaluator's input will necessarily have two distinct outputs. Hence, here are some interesting cases that will meet our above requirements:

- one wants to evaluate $f(y)$, for a public function f, so that the evaluator chooses y, but the generator will get the result;
- one evaluates $f(x, y)$, and any input wires of the first non-linear gates is either an y_j chosen by the evaluator, or $x_i \oplus y_j$, where x_i is chosen by the generator. Indeed, in both cases, y_j or $x_i \oplus y_j$, when x_i is fixed, the inputs of the first gates can take the two possible values according to y_j.

The latter case applies to a large class of circuits, including the addition, the greater-than (as defined in [7]), the equality test, combination of those, or even more complex circuits, such as AES.

The former case is known as *privacy-free* garbled circuits [4] and was shown to be efficient zero-knowledge proof protocols [6]. As mentioned in Section 2, there are more efficient garbling schemes in this context. The work of [22] also provides an optimal garbling scheme for this purpose. Our results also hold with this garbling scheme, but only the general solution is presented here.

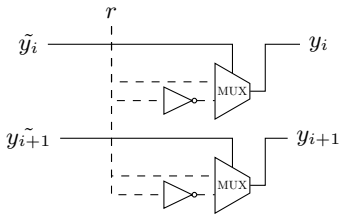

Fig. 5. Overview of the sub-circuit

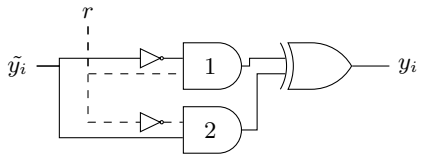

Fig. 6. Implementation of the MUX

5 Ensuring the Correct Garbling of Input Wires

In this section, we describe a construction to guarantee that input wires of the evaluator $\mathcal{E}$ are correctly garbled by the generator $\mathcal{G}$. Rather than modifying the garbling scheme, we propose to modify the circuit representation of the function to evaluate, by adding a sub-circuit in front of the original circuit. This sub-circuit is illustrated in Figure 5. Figure 6 gives details of the multiplexer, but is not required for the correctness. We call x the input of $\mathcal{G}$, and y the input of $\mathcal{E}$.

5.1 Construction

The main idea is that rather than transmitting the input garbled keys of $\mathcal{E}$ through an oblivious transfer, the inputs are now connected to the outputs of this sub-circuit. The sub-circuit has the same number of inputs of $\mathcal{E}$ as the original circuit plus one: a bit r that is randomly chosen by $\mathcal{E}$. For each input y_i of the original circuit, the sub-circuit has an input $\tilde{y}_i = y_i \oplus r$ and an output y_i. The new inputs are transmitted as usual through an oblivious transfer.

We also give restrictions on some permute bits: the permute bit of w_R (the wire carrying r) and $w_{\tilde{Y}_i}$ (carrying $\tilde{y}_i$) must be zero. Also the permute bit of w_{Y_i} (the wire carrying y_i) must be public. This is to ensure that $\mathcal{G}$ does not force the inputs of $\mathcal{E}$ during the oblivious transfer phase.

Because of r and of those permute bits, the protocol has to be slightly modified, as suggested by the following sketch:

1. $\mathcal{G}$ garbles the concatenation of the two circuits using the usual garbling scheme and sends it to $\mathcal{E}$, along with his garbled input keys for x and the permute bit for w_{Y_i}, for all i;
2. $\mathcal{E}$ randomly picks a bit r;
3. $\mathcal{E}$ and $\mathcal{G}$ perform oblivious transfers in order $\mathcal{E}$ to obtain the garbled keys of $\tilde{y}_i$ and r, and $\mathcal{E}$ checks that the select bits of these keys match the clear values or aborts. This ensures two possible keys for the evaluator's inputs;
4. $\mathcal{E}$ evaluates the sub-circuit and checks if the select bits of the keys for the input y match the clear value, or aborts;
5. $\mathcal{E}$ evaluates the rest of the circuit and returns the result.

Since the functionality of the circuit is not changed by the sub-circuit (as long as the new input $\tilde{y}$ is chosen according to r), the correctness is preserved.

5.2 Analysis

Our security goal is to ensure that all output wires of the sub-circuit (i.e. inputs of the rest of the circuit) share the same odd global offset, or the protocol aborts for some specific inputs. To prove it, we need two more lemmata.

Lemma 7. *For any garbled gate, if the two operands have distinct but odd offsets, then the offset of the first operand is propagated to the output wire.*

Proof. The proof of this lemma is identical to the proof of Lemma 3. Indeed, in the proof of Lemma 3, the offset of the second operand ($k^0_{B\gamma} \oplus k^1_{B\gamma}$) never appears. □

Lemma 8. *For any XOR gate, if the offsets of the operands are different or if one of the operands has more than two garbled keys, there are at least four distinct garbled keys at the output.*

Proof. The proof of this lemma is trivial since the output keys of a XOR gate are the input keys XORed together. □

Let us analyze the propagation of offsets in one of the multiplexer of the sub-circuit. Remark that there can not be only one possible garbled key for w_{Y_i}. Indeed, since the permute bit of this wire is known by the evaluator, then there must be at least two possible keys with opposite select bits. We consider the multiplexer illustrated in Fig. 6. We stress that the order of the operands matters. Let w_1 and w_2 refer to the output wires of the AND gates noted respectively 1 and 2. We also note Δ the offset of wire w_r carrying r and $\Delta_{\tilde{Y}_i}$ the offset of the wire carrying $\tilde{y}_i$. We can enumerate the different corruption cases:

1. The offsets Δ and $\Delta_{\tilde{Y}_i}$ are different but odd.
2. Δ is even and $\Delta_{\tilde{Y}_i}$ is odd.
3. Δ is odd and $\Delta_{\tilde{Y}_i}$ is even.
4. Both offsets are even (distinct or not).

Consider the first case. According to Lemma 7, the different offsets propagate so that w_1 has offset $\Delta_{\tilde{Y}_i}$ and w_2 has offset Δ, or one of them two wires have more than two keys. In either case, using Lemma 8, the output of the XOR gate gives at least three different keys. Given that these three (or more) keys engages the value of Δ, we can show that it can not be reduced back to two in the rest of the circuit, using the same method as for Lemma 4.

Consider the second case, if Δ is even, then the garbled keys of w_R have equal select bits. In other words, the select bit of one of the garbled keys does not match the clear value of r. Since r is known to evaluator and since the permute bit must be set to zero, this situation is detected and leads the evaluator to abort. The exact same reasonning works for the third and fourth cases.

We can now conclude that the output wires of the sub-circuit have exactly two possible garbled keys with the same odd global offset, or the protocol aborts for some inputs of the evaluator or some r.

Fig. 7. Impossible corruption

6 Applications to Real Circuits

In the previous sections, we have defined precisely how a malicious generator can corrupt a garbled circuit. Turning non-linear gates into other non-linear gates is equivalent at adding NOT gates to the circuit. Then, we consider in this section that the adversary is able to add a NOT gate to any wire of the circuit. An important consequence is that a circuit can not be modified so that the inputs of the evaluator's inputs go through the gates to the outputs of the circuit. More precisely, the corruption of a gate as shown in Figure 7 can not be generated. Thus, the question "does a corrupted circuit leak more information than the original circuit?" turns out to be trickier than suggested in the previous works.

In this section, we don't provide a general answer, but we see the impact of corruptions on some real circuits. We measure this impact with the Shannon entropy of the evaluator's input. We call x and y the respective inputs of the generator and the evaluator. Let $z = f(x, y)$ be the function to evaluate and C_f a boolean circuit computing it. We note $\mathcal{C}_f$ the set of all circuits that can be obtained by corrupting C_f (i.e. by adding NOT gates to C_f). In other words, there exists a corruption of C_f that leads to $\mathsf{C}_{f'}$, that computes some other function f', if and only if $\mathsf{C}_{f'} \in \mathcal{C}_f$. We formalize the problem as follows :

Problem: *For a circuit C_f does it exist a corrupted circuit $\mathsf{C}_{f'} \in \mathcal{C}_f$, such that the obtained function f' leaks more information on the evaluator's input : $H(Y|X = x, Z = f(x, y)) > H(Y|X = x', Z = f'(x', y))$?*

Remark that in the entropy equation, the generator knows x since this is his input. In our computations, we consider that the adversarial generator chooses his input in order to increase the leakage.

To help us answer that question, we implemented a tool to exhaustively compute all corruptions $\mathsf{C}_{f'}$ of a circuit C_f and check if one of them leaks more information. More details about this tools are given in Annex F.

6.1 The Greater-Than Function

Let us now see a practical example: the greater-than function, that returns a single bit (1 if $x > y$, 0 otherwise). Assuming the adversary takes the middle of the set as input (which leaks the most information), the original function leaks one bit of entropy. Since there is a single output wire, whatever the modification made on the circuit, it does not leak more than one bit of entropy on y. But it is interesting to see that the adversary is limited in the choice of that bit. For example, if we consider the greater-than circuit defined in [7], it can not be

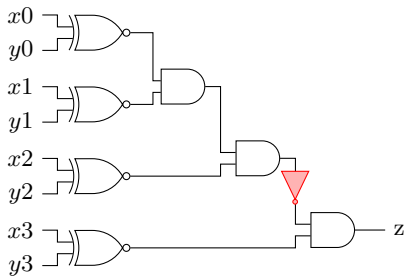

Fig. 8. Circuit for the 4-bit-equality test and its best corrupted circuit in red

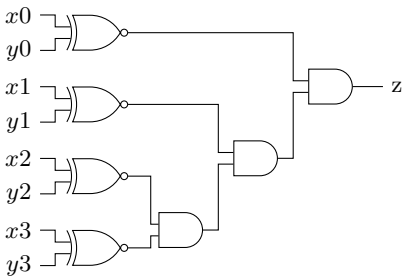

Fig. 9. Another circuit for the 4-bit-equality test

modified to output the parity bit of y. This can be proven exhaustively for the 3-bit greater-than circuit and then recursively.

In the particular case of greater-than circuit, remark that the best strategy of an adversarial $\mathcal{G}$, willing to retrieve the input y, consists in not modifying the circuit. If y is ℓ-bit long, then it would require ℓ evaluations for $\mathcal{G}$ to find y, and it can not be reduced by corrupting it. Thus, in this context, using cut-&-choose based solutions does not enhance privacy (but ensures the correctness).

6.2 The Addition Function

Let us study now the addition function f, the circuit C_f of which is defined and optimized in [7]. Consider that $\mathcal{E}$ has two inputs $y, y' \in \mathbb{F}_2^\ell$ and the generator none. This circuit computes the addition of y and y' in $\mathbb{F}_2^\ell$ (the carry bit is not returned). The original function f does not leak any information on y (or on y'). Up to $\ell = 10$, we exhaustively demonstrated that no modification leaks any information on y: $H(Y|Z = f'(y, y')) = H(Y|Z = f(y, y')) = \ell$ This result can be extended recursively for larger values of ℓ.

6.3 The Equality-Test Function

Unfortunately, it is not the case for all circuit. Consider now the equality-test function, that returns 1 if and only if $x = y$. The Boolean circuit we study for the 4-bit case is shown in Fig. 8. Inputs are 4-bit long and after the evaluation of the original function, it remains 3.66 bits of entropy. This circuit is vulnerable to the addition of NOT gates. Indeed, we demonstrated exhaustively that the best corruption required to add a single NOT gate, as shown in red in Fig. 8. Now, the remaining entropy is $H(Y|X = x', Z = f'(x', y)) = 3.01$ bits. Consequently, almost 1 bit is leaked by this function f'. Actually, f' returns $x_3 \oplus y_3$ if x_{0-2} and y_{0-2} are different and 0 otherwise. Clearly, this same attack would work for larger equality-test circuits.

But note that this attack is entirely based on the topological representation of the function. If we inverted the direction of the cascade of AND gates (ascending

instead of descending in Fig. 8), the leaked bit would be $x_0 \oplus y_0$. Based on this fact, we propose a generic solution to reduce the leakage of a circuit in Annex G. Unfortunately, this fix also requires to increase the size of the circuit.

6.4 Trade-off with Cut-&-Choose

Then, for some classes of circuits, there exist corrupted circuits that leak more information than the original function. In such cases, cut-&-choose remains necessary if we want to avoid this leakage. Based on the fact that this leakage depends on the topology of the circuit, our results still allow to improve for free any cut-&-choose based solutions since [9].

Since several garbled circuits are generated, we recommend to use different circuits of the same function (with different topologies). Then, even if the adversary manages to guess correctly which circuits are opened and which are evaluated, he is limited to corruptions that can be obtained from all unopened circuits and their respective topologies. Indeed, if different corrupted circuits do not compute the same (corrupted) function, then they may output different results, which allows the evaluator to learn the adversarial inputs thanks to [9, 1].

For example, let us consider the two circuits of Fig. 8 and 9 of the same function. Say that cut-&-choose is used with half of the circuits with the first topology and the other half with the second. Assume that at least one circuit of each is unopened. Then, we demonstrated exhaustively that any corrupted function that can be obtained from both topologies does not leak any information on the evaluator's inputs more than the original function already does.

7 Conclusion

The main contribution of this paper is to show that, for a large class of circuits, a malicious generator can corrupt a garbled circuit by only two ways. He can add NOT gates arbitrarily in the circuit, or make selective failure attacks on inputs/outputs of a non-linear gate. This is drastically lower than the previous state-of-the-art suggests. We believe this work can lead to some more optimized secure solutions in the malicious setting, more efficient than the regular cut-&-choose schemes.

The second contribution is the analysis of the impact of NOT gates in real-life circuits. We show that some circuits do not leak more information when NOT gates are added, and thus cut-&-choose solutions are unnecessary to enhance the privacy security property. However, for some other circuits, the addition of NOT gates can lead them to reveal more information, but in that case we give recommendations to improve cut-&-choose solutions for free.

References

1. Afshar, A., Mohassel, P., Pinkas, B., Riva, B.: Non-interactive secure computation based on cut-and-choose. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 387–404. Springer, Heidelberg (May 2014)
2. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: 22nd ACM STOC. pp. 503–513. ACM Press (May 1990)
3. Frederiksen, T.K., Jakobsen, T.P., Nielsen, J.B., Nordholt, P.S., Orlandi, C.: MiniLEGO: Efficient secure two-party computation from general assumptions. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 537–556. Springer, Heidelberg (May 2013)
4. Frederiksen, T.K., Nielsen, J.B., Orlandi, C.: Privacy-free garbled circuits with applications to efficient zero-knowledge. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 191–219. Springer, Heidelberg (Apr 2015)
5. Huang, Y., Katz, J., Kolesnikov, V., Kumaresan, R., Malozemoff, A.J.: Amortizing garbled circuits. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 458–475. Springer, Heidelberg (Aug 2014)
6. Jawurek, M., Kerschbaum, F., Orlandi, C.: Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 13. pp. 955–966. ACM Press (Nov 2013)
7. Kolesnikov, V., Sadeghi, A.R., Schneider, T.: Improved garbled circuit building blocks and applications to auctions and computing minima. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 09. LNCS, vol. 5888, pp. 1–20. Springer, Heidelberg (Dec 2009)
8. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (Jul 2008)
9. Lindell, Y.: Fast cut-and-choose based protocols for malicious and covert adversaries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 1–17. Springer, Heidelberg (Aug 2013)
10. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (May 2007)
11. Lindell, Y., Riva, B.: Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 476–494. Springer, Heidelberg (Aug 2014)
12. Lindell, Y., Riva, B.: Blazing fast 2PC in the offline/online setting with security for malicious adversaries. In: Ray, I., Li, N., Kruegel:, C. (eds.) ACM CCS 15. pp. 579–590. ACM Press (Oct 2015)
13. Mohassel, P., Franklin, M.: Efficiency tradeoffs for malicious two-party computation. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 458–473. Springer, Heidelberg (Apr 2006)
14. Mohassel, P., Riva, B.: Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 36–53. Springer, Heidelberg (Aug 2013)
15. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Proceedings of the 1st ACM conference on Electronic commerce. pp. 129–139. ACM (Nov 1999)

16. Nielsen, J.B., Orlandi, C.: LEGO for two-party secure computation. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 368–386. Springer, Heidelberg (Mar 2009)
17. Nielsen, J.B., Schneider, T., Trifiletti, R.: Constant round maliciously secure 2PC with function-independent preprocessing using LEGO. In: NDSS 2017. The Internet Society (2017)
18. shelat, a., Shen, C.H.: Two-output secure computation with malicious adversaries. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 386–405. Springer, Heidelberg (May 2011)
19. shelat, a., Shen, C.H.: Fast two-party secure computation with minimal assumptions. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 13. pp. 523–534. ACM Press (Nov 2013)
20. Wang, X., Malozemoff, A.J., Katz, J.: Faster secure two-party computation in the single-execution setting. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part III. LNCS, vol. 10212, pp. 399–424. Springer, Heidelberg (May 2017)
21. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS. pp. 162–167. IEEE Computer Society Press (Oct 1986)
22. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole - reducing data transfer in garbled circuits using half gates. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (Apr 2015)

Supplementary Material

Since the evaluation algorithms shown in Table 6 are the keystone of our proofs, we recall here in Table 9 a shorter version of it.

Table 9. Evaluating the half-gates

Select bits		Inputs		Garbled output key
0	0	$k_A^{p_A}$	$k_{B\gamma}^0$	$K_1 = H(k_A^{p_A}) \oplus H(k_{B\gamma}^0)$
0	1	$k_A^{p_A}$	$k_{B\gamma}^1$	$K_2 = E \oplus H(k_A^{p_A}) \oplus H(k_{B\gamma}^1) \oplus k_A^{p_A}$
1	0	$k_A^{\overline{p_A}}$	$k_{B\gamma}^0$	$K_3 = G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^0)$
1	1	$k_A^{\overline{p_A}}$	$k_{B\gamma}^1$	$K_4 = E \oplus G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^1) \oplus k_A^{\overline{p_A}}$

A Other Corruptions

Table 10 is an example of gate that leads to an abortion if $k_{B\gamma}^1$, since the gate was garbled using $k_{B\gamma}^{1*}$.

Table 10. Generating half-gates that abort if $b \oplus \gamma = 1$

First half-gate		Second half-gate	
Garbled table if $\gamma = 0$	Garbled table if $\gamma = 1$	$b \oplus \gamma$	Garbled table
$k_X^0 \oplus H(k_A^{p_A}) = 0$	$k_X^{p_A} \oplus H(k_A^{p_A}) = 0$	0	$k_Y^0 \oplus H(k_{B\gamma}^0) = 0$
$k_X^0 \oplus H(k_A^{\overline{p_A}}) = G$	$k_X^{\overline{p_A}} \oplus H(k_A^{\overline{p_A}}) = G$	1	$k_Y^0 \oplus k_A^0 \oplus H(k_{B\gamma}^{1*}) = E$

Table 11 shows how to transform an AND gate into a gate that computes $a \wedge \bar{b}$.

Table 11. Turning $a \wedge b$ into $a \wedge \bar{b}$

First half-gate		Second half-gate	
Garbled table if $\gamma = 1$	Garbled table if $\gamma = 0$	$b \oplus \gamma$	Garbled table
$k_X^0 \oplus H(k_A^{p_A}) = 0$	$k_X^{p_A} \oplus H(k_A^{p_A}) = 0$	0	$k_Y^0 \oplus H(k_{B\gamma}^0) = 0$
$k_X^0 \oplus H(k_A^{\overline{p_A}}) = G$	$k_X^{\overline{p_A}} \oplus H(k_A^{\overline{p_A}}) = G$	1	$k_Y^0 \oplus k_A^0 \oplus H(k_{B\gamma}^1) = E$

Table 12 shows how to transform an AND gate into a NAND.

Table 13 shows how to transform an AND gate into a OR. It is a combination of the three previous corruptions.

Table 12. Turning $a \wedge b$ into $\overline{a \wedge b}$

First half-gate		Second half-gate	
Garbled table if $\gamma = 0$	Garbled table if $\gamma = 1$	$b \oplus \gamma$	Garbled table
$k_X^1 \oplus H(k_A^{p_A}) = 0$	$k_X^{\overline{p_A}} \oplus H(k_A^{p_A}) = 0$	0	$k_Y^0 \oplus H(k_{B\gamma}^0) = 0$
$k_X^1 \oplus H(k_A^{\overline{p_A}}) = G$	$k_X^{p_A} \oplus H(k_A^{\overline{p_A}}) = G$	1	$k_Y^0 \oplus k_A^0 \oplus H(k_{B\gamma}^1) = E$

Table 13. Turning $a \wedge b$ into $a \vee b$

First half-gate		Second half-gate	
Garbled table if $\gamma = 1$	Garbled table if $\gamma = 0$	$b \oplus \gamma$	Garbled table
$k_X^1 \oplus H(k_A^{p_A}) = 0$	$k_X^{p_A} \oplus H(k_A^{p_A}) = 0$	0	$k_Y^0 \oplus H(k_{B\gamma}^0) = 0$
$k_X^1 \oplus H(k_A^{\overline{p_A}}) = G$	$k_X^{\overline{p_A}} \oplus H(k_A^{\overline{p_A}}) = G$	1	$k_Y^0 \oplus k_A^1 \oplus H(k_{B\gamma}^1) = E$

B Proof of Lemma 2

We note k_A, k'_A and k''_A the three keys of the first operand and $k_{B\gamma}, k'_{B\gamma}$ the two keys of the second operand. We can show that there can not be only two output keys and that the gate illustrated in Figure 2 is computationally unfeasible.

Proof. Suppose that we have the following select bits $s(k_A) = s(k'_A) = 0$, $s(k_{B\gamma}) = 0$ and $s(k'_{B\gamma}) = 1$. We add no constraint on $s(k''_A)$. Then, we can apply the evaluation algorithms for each combination, as shown in Table 9, and we obtain the following set of garbled output keys :

$$\begin{cases} K_1 = H(k_A) \oplus H(k_{B\gamma}) \\ K_2 = E \oplus H(k_A) \oplus H(k'_{B\gamma}) \oplus k_A \\ K_3 = H(k'_A) \oplus H(k_{B\gamma}) \\ K_4 = E \oplus H(k'_A) \oplus H(k'_{B\gamma}) \oplus k'_A \end{cases}$$

We also have K_5 and K_6 that depends on k''_A. Then, the adversary has to reduce the number of keys to two. Thanks to the properties of the hash function, K_1 and K_3 are different. Then, he has to choose E that maps K_2 and K_4 to K_1 or K_3. If we try $K_2 = K_1$, then we get $K_4 = H(k'_A) \oplus H(k_{B\gamma}) \oplus k_A \oplus k'_A$.

Because of the properties of the hash function, K_4 can not be mapped with either K_1 or K_3. A similar result would have been obtained if we first assumed $K_2 = K_3$. Then, the combination of select bits $s(k_A) = s(k'_A) = 0$, $s(k_{B\gamma}) = 0$ and $s(k'_{B\gamma}) = 1$ and any k''_A can not be reduced to two garbled keys.

Let us see the case $s(k_A) = s(k'_A) = s(k_{B\gamma}) = s(k'_{B\gamma}) = 0$. We add no constraint on $s(k''_A)$.

$$\begin{cases} K_1 = H(k_A) \oplus H(k_{B\gamma}) \\ K_2 = H(k_A) \oplus H(k'_{B\gamma}) \\ K_3 = H(k'_A) \oplus H(k_{B\gamma}) \\ K_4 = H(k'_A) \oplus H(k'_{B\gamma}) \end{cases}$$

We also have K_5 and K_6 that depends on k''_A. Then, the adversary has to reduce the number of keys to two. Thanks to the properties of the hash function, K_1, K_2, K_3 and K_4 are different.

Let us see the case $s(k_A) = s(k'_A) = s(k_{B\gamma}) = s(k'_{B\gamma}) = 1$. We add no constraint on $s(k''_A)$.

$$\begin{cases} K_1 = E \oplus G \oplus H(k_A) \oplus H(k_{B\gamma}) \oplus k_A \\ K_2 = E \oplus G \oplus H(k_A) \oplus H(k'_{B\gamma}) \oplus k_A \\ K_3 = E \oplus G \oplus H(k'_A) \oplus H(k_{B\gamma}) \oplus k'_A \\ K_4 = E \oplus G \oplus H(k'_A) \oplus H(k'_{B\gamma}) \oplus k'_A \end{cases}$$

We also have K_5 and K_6 that depends on k''_A. Then, the adversary has to reduce the number of keys to two. Note that in this particular case, the choice of E and G has no impact on the number of distinct keys. Thanks to the properties of the hash function, K_1, K_2, K_3 and K_4 are different.

Let us see the case $s(k_A) = s(k'_A) = 1$, $s(k_{B\gamma}) = 0$ and $s(k'_{B\gamma}) = 1$. We add no constraint on $s(k''_A)$.

$$\begin{cases} K_1 = G \oplus H(k_A) \oplus H(k_{B\gamma}) \\ K_2 = E \oplus G \oplus H(k_A) \oplus H(k'_{B\gamma}) \oplus k_A \\ K_3 = G \oplus H(k'_A) \oplus H(k_{B\gamma}) \\ K_4 = E \oplus G \oplus H(k'_A) \oplus H(k'_{B\gamma}) \oplus k'_A \end{cases}$$

We also have K_5 and K_6 that depends on k''_A. Then, the adversary has to reduce the number of keys to two. Thanks to the properties of the hash function, K_1 and K_3 are different. Then, he has to choose E that maps K_2 and K_4 to K_1 or K_3. We describe below the case $K_2 = K_1$ and show that K_4 does not map any other key.

$$\begin{aligned} K_2 = K_1 &\iff E = H(k_{B\gamma}) \oplus H(k'_{B\gamma}) \oplus k_A \\ &\iff K_4 = G \oplus H(k'_A) \oplus H(k_{B\gamma}) \oplus k_A \oplus k'_A \end{aligned}$$

Because of the properties of the hash function, K_4 can not be mapped with either K_1 or K_3. A similar result would have been obtained if we first assumed $K_2 = K_3$.

All other cases are changes of variables of the already studied cases, which ends the proof of Lemma 2. □

C End of Proof of Lemma 3

Proof. Let us see the case $K_1 = K_2 = K_4$.

$$
\begin{cases}
K_1 = H(k_A^{p_A}) \oplus H(k_{B\gamma}^0) \\
K_2 = E \oplus H(k_A^{p_A}) \oplus H(k_{B\gamma}^1) \oplus k_A^{p_A} = K_1 \\
K_3 = G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^0) \\
K_4 = E \oplus G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^1) \oplus k_A^{\overline{p_A}} = K_1
\end{cases}
$$

$$
\Longrightarrow
\begin{cases}
K_1 = H(k_A^{p_A}) \oplus H(k_{B\gamma}^0) \\
E = H(k_{B\gamma}^0) \oplus H(k_{B\gamma}^1) \oplus k_A^{p_A} \\
G = H(k_A^{p_A}) \oplus H(k_A^{\overline{p_A}}) \oplus \Delta \\
K_3 = H(k_A^{p_A}) \oplus H(k_{B\gamma}^0) \oplus \Delta = K_1 \oplus \Delta
\end{cases}
$$

Let us see the case $K_1 = K_3 = K_4$.

$$
\begin{cases}
K_1 = H(k_A^{p_A}) \oplus H(k_{B\gamma}^0) \\
K_2 = E \oplus H(k_A^{p_A}) \oplus H(k_{B\gamma}^1) \oplus k_A^{p_A} \\
K_3 = G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^0) = K_1 \\
K_4 = E \oplus G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^1) \oplus k_A^{\overline{p_A}} = K_1
\end{cases}
$$

$$
\Longrightarrow
\begin{cases}
K_1 = H(k_A^{p_A}) \oplus H(k_{B\gamma}^0) \\
E = H(k_{B\gamma}^0) \oplus H(k_{B\gamma}^1) \oplus k_A^{\overline{p_A}} \\
G = H(k_A^{p_A}) \oplus H(k_A^{\overline{p_A}}) \\
K_2 = H(k_A^{p_A}) \oplus H(k_{B\gamma}^0) \oplus \Delta = K_1 \oplus \Delta
\end{cases}
$$

Let us see the case $K_2 = K_3 = K_4$.

$$
\begin{cases}
K_1 = H(k_A^{p_A}) \oplus H(k_{B\gamma}^0) \\
K_2 = E \oplus H(k_A^{p_A}) \oplus H(k_{B\gamma}^1) \oplus k_A^{p_A} \\
K_3 = G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^0) = K_2 \\
K_4 = E \oplus G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^1) \oplus k_A^{\overline{p_A}} = K_2
\end{cases}
$$

$$
\Longrightarrow
\begin{cases}
K_1 = H(k_A^{p_A}) \oplus H(k_{B\gamma}^0) \\
K_2 = E \oplus H(k_A^{p_A}) \oplus H(k_{B\gamma}^1) \oplus k_A^{p_A} \\
E = G \oplus H(k_A^{p_A}) \oplus H(k_{B\gamma}^1) \oplus k_A^{p_A} \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^0) \\
K_4 = H(k_A^{p_A}) \oplus H(k_{B\gamma}^0) \oplus \Delta
\end{cases}
$$

$$
\Longrightarrow K_4 = K_1 \oplus \Delta = K_2 = K_3
$$

Let us see the case $K_1 = K_2$ and $K_3 = K_4$.

$$
\begin{cases} K_1 = K_2 \\ K_3 = K_4 \end{cases}
\Leftrightarrow
\begin{cases}
E \oplus H(k_A^{p_A}) \oplus H(k_{B\gamma}^1) \oplus k_A^{p_A} = H(k_A^{p_A}) \oplus H(k_{B\gamma}^0) \\
G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^0) = E \oplus G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^1) \oplus k_A^{\overline{p_A}}
\end{cases}
$$

$$
\Leftrightarrow
\begin{cases}
E = H(k_{B\gamma}^0) \oplus H(k_{B\gamma}^1) \oplus k_A^{p_A} \\
E = H(k_{B\gamma}^0) \oplus H(k_{B\gamma}^1) \oplus k_A^{\overline{p_A}}
\end{cases}
$$

$$
\Leftrightarrow \Delta = 0
$$

Then, the case $K_1 = K_2$ and $K_3 = K_4$ is impossible.

Let us see the case $K_1 = K_4$ and $K_2 = K_3$.

$$\begin{aligned}
\begin{cases} K_1 = K_4 \\ K_2 = K_3 \end{cases} &\Leftrightarrow \begin{cases} E \oplus G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^1) \oplus k_A^{\overline{p_A}} = H(k_A^{p_A}) \oplus H(k_{B\gamma}^0) \\ G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^0) = E \oplus H(k_A^{p_A}) \oplus H(k_{B\gamma}^1) \oplus k_A^{p_A} \end{cases} \\
&\Leftrightarrow \begin{cases} E \oplus G = H(k_A^{p_A}) \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^0) \oplus H(k_{B\gamma}^1) \oplus k_A^{p_A} \\ E \oplus G = H(k_A^{p_A}) \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^0) \oplus H(k_{B\gamma}^1) \oplus k_A^{\overline{p_A}} \end{cases} \\
&\Leftrightarrow \Delta = 0
\end{aligned}$$

Then, the case $K_1 = K_4$ and $K_2 = K_3$ is impossible.

Let us see the case $K_1 = K_3$ and $K_2 = K_4$.

$$\begin{aligned}
&\begin{cases} K_1 = K_3 \\ K_2 = K_4 \end{cases} \\
\Longleftrightarrow &\begin{cases} G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^0) = H(k_A^{p_A}) \oplus H(k_{B\gamma}^0) \\ E \oplus H(k_A^{p_A}) \oplus H(k_{B\gamma}^1) \oplus k_A^{p_A} = E \oplus G \oplus H(k_A^{\overline{p_A}}) \oplus H(k_{B\gamma}^1) \oplus k_A^{\overline{p_A}} \end{cases} \\
\Longleftrightarrow &\begin{cases} G = H(k_A^{p_A}) \oplus H(k_A^{\overline{p_A}}) \\ G = H(k_A^{p_A}) \oplus H(k_A^{\overline{p_A}}) \oplus k_A^{p_A} \oplus k_A^{\overline{p_A}} \end{cases} \\
\Longleftrightarrow &\Delta = 0
\end{aligned}$$

Then, the case $K_1 = K_3$ and $K_2 = K_4$ is impossible.

In the case of a XOR gate, the offset is also propagated, since garbled keys are simply XORed together.

$$\begin{cases} K_1 = k_A^0 \oplus k_B^0 \\ K_2 = k_A^0 \oplus k_B^0 \oplus \Delta = K_1 \oplus \Delta \\ K_3 = k_A^0 \oplus \Delta \oplus k_B^0 = K_1 \oplus \Delta \\ K_4 = k_A^0 \oplus \Delta \oplus k_B^0 \oplus \Delta = K_1 \end{cases}$$

This ends the proof of Lemma 3. □

D End of Proof of Lemma 4

We call k_A and k'_A the keys of the first operand and $k_{B\gamma}$, $k'_{B\gamma}$ and $k''_{B\gamma}$ the keys of the second operand.

Proof. As already stated, the computation of $k_{B\gamma}$, $k'_{B\gamma}$ and $k''_{B\gamma}$ engages the choice of Δ, and consequently $k_A \oplus k'_A$. We develop here the trickiest case $s(k_A) = s(k_{B\gamma}) = s(k'_{B\gamma}) = 0$ and $s(k'_A) = s(k''_{B\gamma}) = 1$. Using the four evaluation algorithms on each of the combinations, we have the following set of keys :

$$\begin{cases} K_1 = H(k_A) \oplus H(k_{B\gamma}) \\ K_2 = H(k_A) \oplus H(k'_{B\gamma}) \\ K_3 = E \oplus H(k_A) \oplus H(k''_{B\gamma}) \oplus k_A \\ K_4 = G \oplus H(k'_A) \oplus H(k_{B\gamma}) \\ K_5 = G \oplus H(k'_A) \oplus H(k'_{B\gamma}) \\ K_6 = E \oplus G \oplus H(k'_A) \oplus H(k''_{B\gamma}) \oplus k'_A \end{cases}$$

Thanks to the property of the hash function, K_1 and K_2 are different.Then, the adversary must choose E and G, so that K_3 to K_6 collide with K_1 or K_2. Let us first consider the case $K_4 = K_1$, then the system of keys becomes :

$$K_4 = K_1 \implies \begin{cases} G = H(k_A) \oplus H(k'_A) \\ K_1 = K_4 = H(k_A) \oplus H(k_{B\gamma}) \\ K_2 = K_5 = H(k_A) \oplus H(k'_{B\gamma}) \\ K_3 = E \oplus H(k_A) \oplus H(k''_{B\gamma}) \oplus k_A \\ K_6 = E \oplus H(k_A) \oplus H(k''_{B\gamma}) \oplus k'_A \end{cases}$$

Then we have two more cases to enumerate : $K_3 = K_1$ and $K_3 = K_2$.

$$\begin{cases} K_4 = K_1 \\ K_3 = K_1 \end{cases} \implies \begin{cases} G = H(k_A) \oplus H(k'_A) \\ E = H(k_{B\gamma}) \oplus H(k''_{B\gamma}) \oplus k_A \\ K_1 = K_3 = K_4 = H(k_A) \oplus H(k_{B\gamma}) \\ K_2 = K_5 = H(k_A) \oplus H(k'_{B\gamma}) \\ K_6 = H(k_A) \oplus H(k_{B\gamma}) \oplus k_A \oplus k'_A \end{cases}$$

Then, K_6 is different from K_1 since Δ is odd and thus non-zero. Matching K_6 and K_2 is computationally unfeasible since it would require that $H(k_{B\gamma}) \oplus H(k'_{B\gamma}) = \Delta$ and we demonstrated above that the values of $k_{B\gamma}$ and $k'_{B\gamma}$ commits the value of Δ. The same result can be obtained if we assumed that $K_4 = K_2$ and/or $K_3 = K_2$

Let us now see the case $s(k_A) = s(k_{B\gamma}) = 0$ and $s(k'_A) = s(k'_{B\gamma}) = s(k''_{B\gamma}) = 1$. Using the four evaluation algorithms on each of the combinations, we have the following set of keys :

$$\begin{cases} K_1 = H(k_A) \oplus H(k_{B\gamma}) \\ K_2 = E \oplus H(k_A) \oplus H(k'_{B\gamma}) \oplus k_A \\ K_3 = E \oplus H(k_A) \oplus H(k''_{B\gamma}) \oplus k_A \\ K_4 = G \oplus H(k'_A) \oplus H(k_{B\gamma}) \\ K_5 = E \oplus G \oplus H(k'_A) \oplus H(k'_{B\gamma}) \oplus k'_A \\ K_6 = E \oplus G \oplus H(k'_A) \oplus H(k''_{B\gamma}) \oplus k'_A \end{cases}$$

Thanks to the property of the hash function, K_2 and K_3 are different. Then, the adversary must choose E and G, so that K_1 and K_4 to K_6 collide with K_2 or K_3. Let us first consider the case $K_1 = K_2$, then the system of keys becomes :

$$K_1 = K_2 \implies \begin{cases} E = H(k_{B\gamma}) \oplus H(k'_{B\gamma}) \oplus k_A \\ K_1 = K_2 = H(k_A) \oplus H(k_{B\gamma}) \\ K_3 = H(k_A) \oplus H(k_{B\gamma}) \oplus H(k'_{B\gamma}) \oplus H(k''_{B\gamma}) \\ K_4 = G \oplus H(k'_A) \oplus H(k_{B\gamma}) \\ K_5 = G \oplus H(k'_A) \oplus H(k_{B\gamma}) \oplus k'_A \oplus k_A \\ K_6 = G \oplus H(k'_A) \oplus H(k_{B\gamma}) \oplus H(k'_{B\gamma}) \oplus H(k''_{B\gamma}) \oplus k'_A \oplus k_A \end{cases}$$

Then we have two more cases to enumerate : $K_4 = K_2$ and $K_4 = K_3$.

$$\begin{cases} K_1 = K_2 \\ K_4 = K_2 \end{cases} \implies \begin{cases} E = H(k_{B\gamma}) \oplus H(k'_{B\gamma}) \oplus k_A \\ G = H(k_A) \oplus H(k'_A) \\ K_1 = K_2 = K_4 = H(k_A) \oplus H(k_{B\gamma}) \\ K_3 = H(k_A) \oplus H(k_{B\gamma}) \oplus H(k'_{B\gamma}) \oplus H(k''_{B\gamma}) \\ K_5 = H(k_A) \oplus H(k_{B\gamma}) \oplus k'_A \oplus k_A \\ K_6 = H(k_A) \oplus H(k_{B\gamma}) \oplus H(k'_{B\gamma}) \oplus H(k''_{B\gamma}) \oplus k'_A \oplus k_A \end{cases}$$

$$\begin{cases} K_1 = K_2 \\ K_4 = K_3 \end{cases} \implies \begin{cases} E = H(k_{B\gamma}) \oplus H(k'_{B\gamma}) \oplus k_A \\ G = H(k_A) \oplus H(k'_A) \oplus H(k'_{B\gamma}) \oplus H(k''_{B\gamma}) \\ K_1 = K_2 = H(k_A) \oplus H(k_{B\gamma}) \\ K_3 = K_4 = H(k_A) \oplus H(k_{B\gamma}) \oplus H(k'_{B\gamma}) \oplus H(k''_{B\gamma}) \\ K_5 = H(k_A) \oplus H(k_{B\gamma}) \oplus H(k'_{B\gamma}) \oplus H(k''_{B\gamma}) \oplus k'_A \oplus k_A \\ K_6 = H(k_A) \oplus H(k_{B\gamma}) \oplus k'_A \oplus k_A \end{cases}$$

Note that in the two cases, K_5 and K_6 are simply switched around. Then, we only focus on the former case. K_5 is different from K_2 since Δ is odd and thus non-zero. Matching K_6 and K_3 is computationally unfeasible since it would require that $H(k_{B\gamma}) \oplus H(k'_{B\gamma}) = \Delta$ and we demonstrated above that the values of $k_{B\gamma}$ and $k'_{B\gamma}$ commits the value of Δ. The same result would be obtained if we assumed first that $K_1 = K_3$. Indeed, it would only be a permutation of the keys $k'_{B\gamma}$ and $k''_{B\gamma}$.

Let us see the case $s(k_A) = s(k_{B\gamma}) = s(k'_{B\gamma}) = s(k''_{B\gamma}) = 0$ and $s(k'_A) = 1$. Using the four evaluation algorithms on each of the combinations, we have the following set of keys :

$$\begin{cases} K_1 = H(k_A) \oplus H(k_{B\gamma}) \\ K_2 = H(k_A) \oplus H(k'_{B\gamma}) \\ K_3 = H(k_A) \oplus H(k''_{B\gamma}) \\ K_4 = G \oplus H(k'_A) \oplus H(k_{B\gamma}) \\ K_5 = G \oplus H(k'_A) \oplus H(k'_{B\gamma}) \\ K_6 = G \oplus H(k'_A) \oplus H(k''_{B\gamma}) \end{cases}$$

This case is easier since the three first keys are different because of the properties of the hash function.

Let us see the case $s(k_A) = 0$ and $s(k'_A) = s(k_{B\gamma}) = s(k'_{B\gamma}) = s(k''_{B\gamma}) = 1$. Using the four evaluation algorithms on each of the combinations, we have the following set of keys :

$$\begin{cases} K_1 = E \oplus H(k_A) \oplus H(k_{B\gamma}) \oplus k_A \\ K_2 = E \oplus H(k_A) \oplus H(k'_{B\gamma}) \oplus k_A \\ K_3 = E \oplus H(k_A) \oplus H(k''_{B\gamma}) \oplus k_A \\ K_4 = E \oplus G \oplus H(k'_A) \oplus H(k_{B\gamma}) \oplus k'_A \\ K_5 = E \oplus G \oplus H(k'_A) \oplus H(k'_{B\gamma}) \oplus k'_A \\ K_6 = E \oplus G \oplus H(k'_A) \oplus H(k''_{B\gamma}) \oplus k'_A \end{cases}$$

The three first keys are different because of the properties of the hash function.

So far, we studied all the cases where $s(k_A) = 0$ and $s(k'_A) = 1$ (and by change of variables $s(k_A) \neq s(k'_A)$). All cases where $s(k_A) = s(k'_A)$ is equivalent to one already seen in Annex B (it follows from the fact that we do not use the third key k''_A in the referred annex), which ends the proof of Lemma 4. □

E Proof of Lemma 5

Proof. All cases are already seen in other proofs:

- The case $K_1 = K_2 = K_3 = K_4$ is already shown to be unsolvable in the proof of Lemma 1.
- The case $K_1 = K_2$ and $K_3 = K_4$ (the gate that always output the first operand a or $\overline{a}$) is proved to be impossible in Annex C.
- The case $K_1 = K_3$ and $K_2 = K_4$ (the gate that always output the second operand b or $\overline{b}$) is proved to be impossible in Annex C.
- The case $K_1 = K_4$ and $K_2 = K_3$ (the gate that computes $a \oplus b$ or $\overline{a \oplus b}$) is proved to be impossible in Annex C.

This ends the proofs of Lemma 5. □

F Finding the Best Corrupted Circuit

To help us answer the problem stated in Section 6, we implemented a tool to exhaustively compute all corruptions $\mathsf{C}_{f'}$ of a circuit C_f. For every corrupted circuit $\mathsf{C}_{f'}$, it computes the entropy $H(Y|X = x', Z = f'(x', y))$ and checks whether it leaks more information than C_f. Since the complexity of this method is exponential in the size of the original circuit, it is obviously limited to small circuits. Algorithm 1 gives the overview of our algorithm.

G Corruption of Circuits

In order to reduce the leakage of a circuit, we take advantage of the fact that this leakage depends on the tyopology of the circuit. In the case of the equality

Input: A circuit C_f of n wires w_1 to w_n (arbitrary order).
Output: The corrupted circuit that leaks the most information.
Set an n-nits integer ω to zero.
Compute the initial entropy
$H = H(Y|X = x, f(x, y))$.
while $\omega < 2^n$ **do**
  $\omega \leftarrow \omega + 1$
  $\mathsf{C}_{f'} \leftarrow \mathsf{C}_f$
  foreach ω_i **do**
    If $\omega_i = 1$, add a NOT gate to the wire w_i of $\mathsf{C}_{f'}$.
  end
  Compute the truth table of $\mathsf{C}_{f'}$.
  Compute the entropy
  $H' = H(Y|X = x', f'(x', y))$.
  if $H' < H$ **then**
    $H \leftarrow H'$
    $\mathsf{C}_{f'}$ becomes the best corrupted circuit so far.
  end
  return The best corrupted circuit found.
end

Algorithm 1: Finding the best corruption of a circuit.

test (w.l.o.g.), we propose to evaluate two parallel sub-circuits with different topologies, and to output only one, chosen by the evaluator. This approach is illustrated in Figure 10. A dashed sub-circuit performing differently the equality test is added to the previous one, and a multiplexer (described in [8]) allows the evaluator to choose which of the two results is returned (let c represent this choice). If the generator is honest, the circuit remains correct : the sub-circuits have the same result and the multiplexer has no influence. Otherwise, the generator does not know which one of the two sub-circuits have returned the result.

However, the attacker can still add NOT gates to this new circuit. Using the same method, we computed that the best corruption requires six NOT gates to be added, as illustrated in Fig. 11. By studying the multiplexer of [8], we can show that there is no way for the adversary to force the choice of c. Then, we do not detail the multiplexer in the following Boolean circuit. The remaining entropy on the evaluator's input after evaluating this corrupted circuit is $H(Y|X = x', Z = f'(x', y)) = 3.35$ bits. Then, we have considerably reduced the leakage of information.

This result was obtained by computing all possible additions of NOT gates. Then, for each corrupted circuit, we have computed the truth table of the corrupted function and measured the Shannon entropy of the evaluator's input.

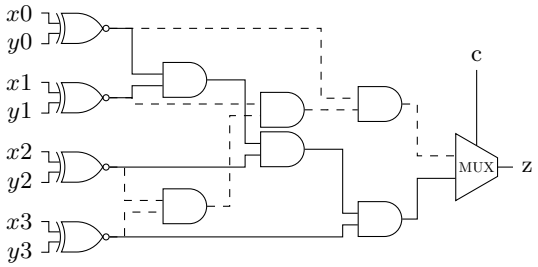

Fig. 10. Improved circuit for the 4-bit-equality test

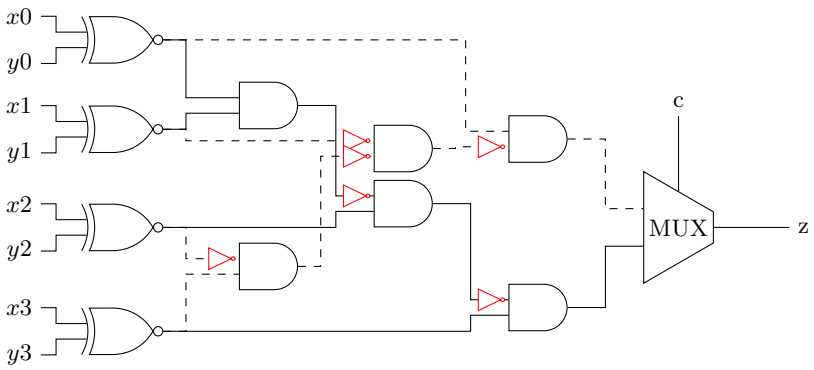

Fig. 11. Best corrupted circuit for the improved 4-bit-equality test