Efficiently Testable Circuits Without Conductivity

Mirza Ahad Baig[1][0000−0003−3650−7893], Suvradip Chakraborty[2][0000−0002−5352−4946], Stefan Dziembowski[3,4][0000−0002−6914−6425], Małgorzata Gałązka[3], Tomasz Lizurej[3,4][0000−0001−8563−4325], and Krzysztof Pietrzak[1]

[1] ISTA
[2] Visa Research
[3] University of Warsaw**
[4] IDEAS NCBR

Abstract. The notion of "efficiently testable circuits" (ETC) was recently put forward by Baig et al. (ITCS'23). Informally, an ETC compiler takes as input any Boolean circuit C and outputs a circuit/inputs tuple $(C', \mathbb{T})$ where (completeness) C' is functionally equivalent to C and (security) if C' is tampered in some restricted way, then this can be detected as C' will err on at least one input in the small test set $\mathbb{T}$. The compiler of Baig et al. detects tampering even if the adversary can tamper with *all* wires in the compiled circuit. Unfortunately, the model requires a strong "conductivity" restriction: the compiled circuit has gates with fan-out up to 3, but wires can only be tampered in one way even if they have fan-out greater than one. In this paper, we solve the main open question from their work and construct an ETC compiler without this conductivity restriction. While Baig et al. use gadgets computing the AND and OR of particular subsets of the wires, our compiler computes inner products with random vectors. We slightly relax their security notion and only require that tampering is detected with high probability over the choice of the randomness. Our compiler increases the size of the circuit by only a small constant factor. For a parameter λ (think $\lambda \leq 5$), the number of additional input and output wires is $|C|^{1/\lambda}$, while the number of test queries to detect an error with constant probability is around $2^{2\lambda}$.

1 Introduction

Circuit Testing. Detecting errors in circuits is of interest in various areas of engineering and computer science. In circuit manufacturing, the focus is on efficiently detecting errors that randomly occur during production [10]. Querying circuits on a few carefully chosen inputs and checking the output for correctness will typically detect a large fraction of the faulty ones.

** This work was supported by the National Science Centre, Poland, under research project No. 46339.

Private Circuits (PC). The cryptographic community has long focused on errors that are intentionally introduced by an adversary, as such "tampering" or "fault attacks" can be used to extract cryptographic secrets [7, 8]. Compared to testing in manufacturing, protecting circuits against fault attacks is more difficult for at least two reasons (1) the errors are not just random but can be targeted on specific wires or gates in the circuit (2) the errors introduced by tampering must not just be detected, but the circuit must be prevented to leak any information.

For this challenging setting of *private circuits* (PC), Ishai, Prabhakaran, Sahai, and Wagner [24] construct a *circuit compiler* that given (the description of) any circuit C and some parameter k outputs (the description of) a functionally equivalent circuit C_k (i.e., $C(X) = C_k(X)$ for all X) which is secure against fault attacks that can tamper with up to k wires with each query (the faults can be persistent, so ultimately the entire circuit can be tampered with), while blowing up the circuit size by a factor of k^2. The efficacy of the compiler can be somewhat improved by allowing some small information leakage [22].

Efficiently Testable Circuits (ETC). Efficiently testable circuits (ETC), recently introduced in [3], considers a setting that "lies in between" testing for benign errors and private circuits. An ETC compiler takes any Boolean circuit $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ and maps it to a tuple $(C_{\mathsf{test}} : \mathbb{Z}_2^{s+s'} \to \mathbb{Z}_2^{t+t'}, \mathbb{T}_{\mathsf{test}} \subset \mathbb{Z}_2^{s+s'})$ where C_{test} is functionally equivalent to C and $\mathbb{T}_{\mathsf{test}}$ is a test set that will catch any (non-trivial) tampering on C_{test}. A bit more formally, by saying C_{test} is functionally equivalent to C we mean $\forall X \in \mathbb{Z}_2^s : C_{\mathsf{test}}(X \| 0^{s'})_{|t} = C(X)$ ($S_{|t}$ denotes the t bit prefix of S, $\|$ is concatenation and 0^s is the string of s zeros).

The security property states that if for a wire tampering τ on C_{test} the tampered circuit C^{τ}_{test} errs on at least one of the (exponentially many) inputs $X \| 0^{s'}$ (i.e., the t bit prefix of the output is not $C(X)$), then C^{τ}_{test} will err on at least one input in the (small) test set

$$\forall \tau : \exists X \in \mathbb{Z}_2^s \; s.t. \; C^{\tau}_{\mathsf{test}}(X\|0^s)_{|t} \neq \overbrace{C_{\mathsf{test}}(X\|0^{s'})_{|t}}^{=C(X)} \Rightarrow \exists T \in \mathbb{T}_{\mathsf{test}} \; s.t. \; C^{\tau}_{\mathsf{test}}(T) \neq C_{\mathsf{test}}(T) \tag{1}$$

ETC aims at detecting *adversarial* errors like PC, but unlike PC, this detection only happens during a dedicated testing phase, not implicitly with every query. Thus ETC cannot be used to replace PCs which aim to protect secrets on a device that is under adversarial control and can be tampered with. Instead, they ensure that a circuit correctly evaluates on all inputs, even if it was under adversarial control in the past.

Using ETC can also be useful to detect benign errors, particularly in settings where one doesn't want to accept a non-trivial probability of missing a fault, which is the case for the heuristic techniques currently deployed in circuit manufacturing. One such setting is in space exploration where faults can be catastrophic, and to make matters worse, the high radiation in outer space is likely to cause additional faults. Here the ability to run a cheap test repeatedly in a black-box way is useful.

While ETCs provide a weaker security guarantee than PC in terms of how tampering is detected, the construction of the ETC from [3] achieves security under a much stronger tampering model than what is known for PC. Furthermore, ETCs are much more efficient and rely on weaker assumptions: the ETC compiler from [3] blows the circuit up by a small constant factor while allowing for tampering *with all wires*. On the other hand, in *Private Circuits II* [24], to detect tampering with k wires already requires a blow up of k^2.

Conductivity. A major restriction of both, the PC compiler [24] and the ETC compiler from [3], is the fact that wire tamperings are assumed to be *conductive*: while a wire can be tampered (set to constant 0 or 1, or toggling) arbitrarily, if this wire has fan-out greater than 1, i.e., leads to more than one destination which can be an input to another gate or an output wire, all must carry the same value and cannot be tampered individually.[5] This is an arguably unrealistic assumption and not does not capture real tampering attacks: Why should, say, cutting the wire at the input of one gate affect the value at another gate to which this wire is connected? While any circuit can easily be turned into a functionally equivalent one where all wires have fan-out 1 by using copy gates $\mathsf{COPY}(b) = (b, b)$, applying this to the circuit produced by the compiler from [3] will completely break its security as we will sketch below.

Our contribution. In this work we solve the main open problem left in [3] and construct an ETC compiler that maps a circuit C to an ETC $(C_{\mathsf{test}}, \mathbb{T}_{\mathsf{test}})$ where $|\mathbb{T}_{\mathsf{test}}| \leq 6$ and C_{test} has fan-out 1, which means it doesn't rely on the conductivity assumption as there's nothing to conduct.[6]

To get a practical construction with few extra output wires, we need to generalize the notion of ETCs and make it probabilistic. Whether efficient *deterministic* ETCs without the conductivity assumption exist is an interesting open question (our construction can be "derandomized", but this would lead to an impractically large test set of size $|C|^2$). Concretely, the inputs in our test set $\mathbb{T}_{\mathsf{test}}$ are shorter than C_{test}'s input, and during testing the remaining inputs must be chosen at random. The soundness guarantee $\exists T \in \mathbb{T}_{\mathsf{test}}\ s.t.\ C^{\tau}_{\mathsf{test}}(T) \neq C_{\mathsf{test}}(T)$ from eq.(1) is adapted to a probabilistic guarantee

$$\exists T \in \mathbb{T}_{\mathsf{test}}\ s.t.\ \Pr_{R}[C^{\tau}_{\mathsf{test}}(T\|R) \neq C_{\mathsf{test}}(T\|R)] \geq 1/2^{2\lambda} \tag{2}$$

where $\lambda \in \mathbb{N}_0$ is a parameter specifying the number of layers in the testing subcircuit. A larger λ will decrease the extra input/output wires but will increase the required number of test queries, a reasonable range for λ is 1 to 4.

[5] The conductivity assumption for the PC compiler from [24] is slightly stronger than ours, as they additionally assume that "faults on the output side of a NOT gate propagate to the input side".

[6] Ensuring non-conductivity by making sure the fan-out is 1 is done for clarity of exposition. To get a fan-out 1 circuit our complied circuit requires numerous COPY gates. In an actual physical circuit any of those COPY gates can be simply removed by increasing the fan-out of the input wire to that gate by one.

Size, Query and Randomness Efficiency. The number of extra input/output wires is roughly (cf. Table 1 for the exact numbers) $\lambda \cdot |C|^{1/(\lambda+1)}$, e.g. for a circuit with 2^{32} ($\approx$ four billion) gates and $\lambda = 3$ we need roughly $3 \cdot 2^8 = 768$ extra input and output wires. By repeating the testing κ times with fresh randomness, the probability that we fail to detect a non-trivial tampering is at most $(1-1/2^{2\lambda})^{\kappa}$, which for our example is < 0.5 for $\lambda = 3, \kappa = 45$. The number of test queries required for this testing is $|\mathbb{T}_{\mathsf{test}}| \cdot \kappa = 6 \cdot 45 = 270$ (as we don't know which of the $T \in \mathbb{T}_{\mathsf{test}}$ satisfies eq.(2) we have to query with all of them). The number of random bits required for this testing is $\kappa \cdot \lambda \cdot |C|^{1/(\lambda+1)} = 45 \cdot 768 = 34560$ (each test query $T \in \mathbb{T}_{\mathsf{test}}$ must be concatenated with $\lambda \cdot |C|^{1/(\lambda+1)}$ random bits, we can use the same randomness for each $T \in \mathbb{T}_{\mathsf{test}}$, but assume fresh randomness for each of the κ runs of the test). We can get the probability of missing a fault down to any $2^{-\alpha}$ by repeating the above test α times. This is already quite practical despite the fact that in this work we focused on a clean exposition rather than improving concrete parameters.

2 ETC Compilers and their Security

2.1 The Construction from [3] Using Conductivity

Before we describe our construction, let us first give a short summary of the ETC compiler from [3]. The basic construction using a toy circuit $C(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2) \vee (x_3 \vee x_4)$ as input is illustrated in Figure 1.

Wire Covering. In a first step, they compile the basic circuit C into a tuple $(C_{\mathsf{wire}}, \mathbb{T}_{\mathsf{wire}})$ where C_{wire} is functionally equivalent to C and $\mathbb{T}_{\mathsf{wire}}$ is a wire covering for C_{wire}, which means for every wire w in C and $b \in \{0,1\}$ there is some $X \in \mathbb{T}_{\mathsf{wire}}$ such that w carries the value b if C is evaluated on X. For the toy circuit C we can use $(C_{\mathsf{wire}} = C, \mathbb{T}_{\mathsf{wire}} = \{0000, 1111\})$ (here $C_{\mathsf{wire}} = C$, but in general we need up to 3 extra input wires and some extra XOR gates to compile C to C_{wire}).

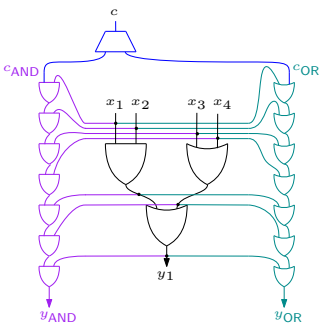

Fig. 1. The compiler from [3] illustrated on a toy circuit.

A naive Construction with Conductivity. From $(C_{\mathsf{wire}}, \mathbb{T}_{\mathsf{wire}})$ [3] then further construct their ETC $(C_{\mathsf{test}}, \mathbb{T}_{\mathsf{test}})$. A naive construction is to let the test set be the wire covering set, i.e., $\mathbb{T}_{\mathsf{test}} = \mathbb{T}_{\mathsf{wire}}$, and derive C_{test} from C_{wire} by increasing the fan-out of every internal wire by one, and use the extra wire as an output. This way any tampering of a wire will be observable on one of the outputs.

Of course, having $|C_{\mathsf{wire}}|$ many output wires is completely impractical so they must be compressed, and we'll sketch how this is done below, but let us

first emphasize here that *conductivity* is absolutely crucial even for this naive construction, a concrete example is given in [3]. Looking ahead, a key observation we make in this work is that by using a more general "gate covering" set, this naive construction will detect tampering even without conductivity.

Compressing the Output. The work of [3] reduce the number of additional output wires by connecting every wire w in C_{wire} to one "OR gadget" and one "AND gadget" in a careful way (so they get fan-out 3, in the figure those gadgets are the purple and cyan subcircuits). These gadgets have just a one bit output (for our toy example we just need one OR and one AND gadget). There's also an extra input bit c (for *c*ontrol) and for every $X \in \mathbb{T}_{\mathsf{wire}}$ in the wire covering, the test set $\mathbb{T}_{\mathsf{test}}$ contains two inputs, $X\|0$ and $X\|1$, i.e., one where the control is 0 and one where it's 1. The wires are connected to the gadgets such that whenever there's some tampering on the internal circuit, some gadget will compute the wrong value on some $X \in \mathbb{T}_{\mathsf{test}}$. The extra control bit is necessary so this holds even if the adversary can also tamper with the gadgets themselves. Understanding the details of their construction and proof are not necessary for the current paper, so we refer to their paper for more details.

2.2 Our Construction without Conductivity

Overcoming Conductivity. Without conductivity, the design principle outlined above, i.e., routing internal wires to some gadgets that try to catch errors, is not sufficient as errors on the internal wires can potentially be "tampered back" to the correct value on the external wires. Our construction makes this approach work even when we cannot rely on conductivity. Instead of trying to catch any tampering error, our gadgets (which compute inner products) are only guaranteed to catch tamperings on a test set if some wire "loses information", which means the wire carries different values on two inputs from the test set, but after tampering the values are identical. A key observation is that one can't undo information loss by tampering a wire. Fortunately, this already will be enough; we prove a dichotomy showing that every tampering either loses information on a "gate covering" set of inputs, or the tampering is additive. The latter case can easily be detected by checking the correctness of the regular (as opposed to the gadget) output on an arbitrary input. We will now illustrate our compiler using the toy circuit C shown in Figure 2.(A).

Gate Covering. Like in [3], in a first step we compile our circuit C into a wire covering. That is, a tuple $(C_{\mathsf{wire}}, \mathbb{T}_{\mathsf{wire}})$ where C_{wire} is functionally equivalent to C and for every wire in C_{wire} and every $b \in \{0,1\}$ there's an $X \in C_{\mathsf{wire}}$ s.t. w takes value b on evaluation $C_{\mathsf{wire}}(X)$. For our toy example, we use $(C_{\mathsf{wire}} = C, \mathbb{T}_{\mathsf{wire}} = \{0000, 1010, 1101\})$ as shown in Fig.2.(A).

We then compile $(C_{\mathsf{wire}}, \mathbb{T}_{\mathsf{wire}})$ into a gate covering $(C_{\mathsf{gate}}, \mathbb{T}_{\mathsf{gate}})$. By this we mean a tuple where C_{gate} is functionally equivalent to C_{wire} when padding the (at most two) new inputs to 0, i.e., $C_{\mathsf{wire}}(X) = C_{\mathsf{gate}}(X\|0^2)$. A gate covering is a wire covering, but additionally, we require that for every gate g, and for

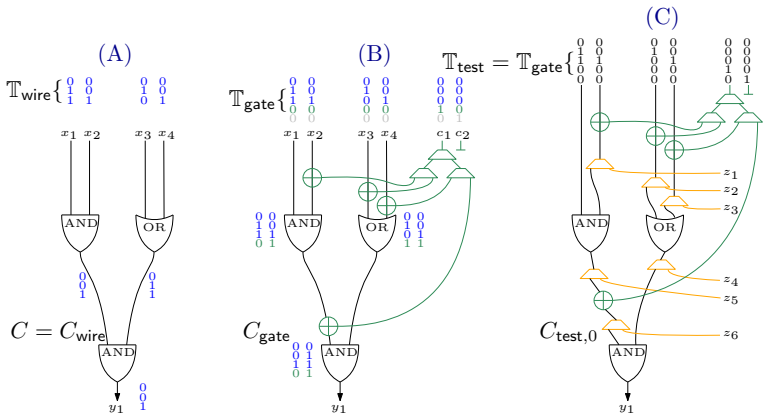

Fig. 2. Illustration of our toy circuit C (A) with a wire covering set $\mathbb{T}_{\mathsf{wire}} = \{0000, 1010, 1101\}$ (B) after adding two extra control inputs c_1, c_2 and XORing them into the circuit to get a gate covering set $\mathbb{T}_{\mathsf{gate}} = \{X\|00 : X \in \mathbb{T}_{\mathsf{wire}}\} \cup 000010$ (in our toy example the 2nd control c_2 and the 2nd input 000001 is not required). (C) We get our 0th layer ETC $(C_{\mathsf{test},0}, \mathbb{T}_{\mathsf{test}})$ setting $\mathbb{T}_{\mathsf{test}} = \mathbb{T}_{\mathsf{gate}}$ and adding copy gates to route the output of every AND,OR and XOR to a new output z_i.

every possible input to that gate, there's a $X \in \mathbb{T}_{\mathsf{gate}}$ such that g is queried on those inputs. There's one relaxation, for XOR gates we just require that three out of the four inputs $\{00, 01, 10, 11\}$ are covered. In Fig.2.(B) we illustrate how to compile the wire covering into a gate covering. This requires adding two extra control bits c_1, c_2 as inputs, some copy gates to create enough copies of those controls, and some XOR gates which add those controls to some carefully chosen wires. The gate cover set $\mathbb{T}_{\mathsf{gate}}$ contains $X\|00$ for every $X \in \mathbb{T}_{\mathsf{wire}}$, and additional two inputs which are all 0 except on c_1 and c_2, respectively. For our toy example we actually just need one control c_1.

Our "0^{th} layer" ETC $(C_{\mathsf{test},0}, \mathbb{T}_{\mathsf{test}})$, as illustrated in Fig.2.(C), is derived from $(C_{\mathsf{gate}}, \mathbb{T}_{\mathsf{gate}})$ by setting $\mathbb{T}_{\mathsf{test}} = \mathbb{T}_{\mathsf{gate}}$, and $C_{\mathsf{test},0}$ is derived from C_{gate} by adding a copy gate to the output of every AND,OR and XOR gate (except if that wire is an output already) to create fresh outputs $z_1, z_2, \ldots, z_6$. Note that by adding copy gates $\mathbb{T}_{\mathsf{test}}$ remains a gate covering for $C_{\mathsf{test},0}$. We will need this fact below.

Of course, the ETC is not practical as there are way too many output wires. Before describing how to compress those outputs we discuss why $(C_{\mathsf{test},0}, \mathbb{T}_{\mathsf{test}})$ is an ETC, i.e., why any non-trivial tampering on the circuit will already cause an error on the outputs for some input in $\mathbb{T}_{\mathsf{test}}$.

Information Loss. As we want a non-conductive circuit, we must use copy gates to route the internal wire values to the outputs and can't just use gates with higher fan-out as in [3]. But now the adversary can tamper with the wires leading to the z_i's individually and thus potentially undo any error in the circuit.

We show that for any circuit with a gate covering – we'll use $(C_{\mathsf{test},0}, \mathbb{T}_{\mathsf{test}})$ from Figure 2 as running example – every tampering τ is either additive in

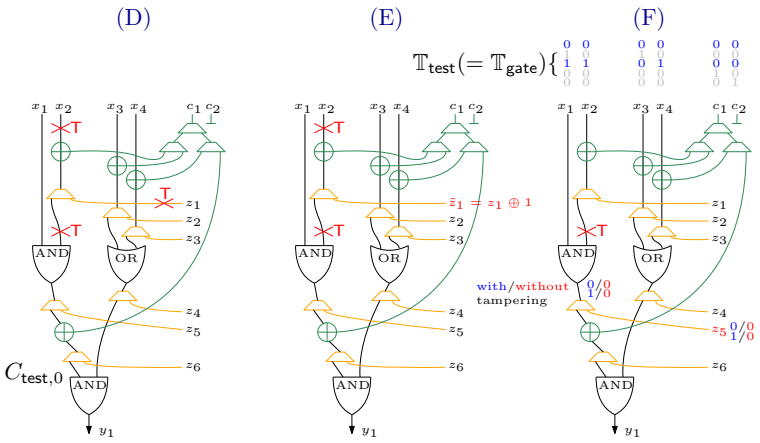

Fig. 3. Illustration of the types on tamperings τ on circuits with gate coverings using the toy example $(C, \mathbb{T}_{\mathsf{gate}})$ from Figure 2. Toggling three wires as indicated in (D) has no effect, i.e., $C^\tau(X) = C(X)$ for all inputs X. Toggling two wires as in (E) creates an additive tampering where $C^\tau(X) = C(X) \oplus B$ (here $B = 100000$), and thus is easily detected with just one query on any input. Tamperings that are not additive create "information loss" at the output of some internal gate, i.e., for two inputs $T_0, T_1 \in \mathbb{T}_{\mathsf{test}}$, some wire will have different values without tampering, but the same value with tampering as illustrated in (F). As we copy all output wires and use them as outputs (and tampering a wire cannot "undo" information loss), we'll observe information loss also on one of the z_i values (illustrated is the loss on z_5 for inputs 000000 and 110100).

the sense that for some fixed B we have $\forall X : C_0^\tau(X) = C_0(X) \oplus B$ or there's *information loss* on some wire w that is the output of a $\mathsf{AND}, \mathsf{OR}$ or XOR gate, which means there are two inputs $X_0, X_1 \in \mathbb{T}_{\mathsf{test}}$ such that the wire w carries different values in the evaluations $C_0(X_0)$ and $C_0(X_1)$, but the same value in the evaluations $C_0^\tau(X_0)$ and $C_0^\tau(X_1)$ of the tampered circuit.

By construction, in our C_0 circuit every such wire w is copied $(w', w'') \leftarrow \mathsf{COPY}(w)$ and w'' is then routed to the output. Here we crucially rely on the fact that we don't merely have arbitrary errors like in [3] but information loss, which cannot be undone even by tampering w'' independently from w, w'.

Let us shortly sketch why $(C_0, \mathbb{T}_{\mathsf{test}})$ is an ETC. Consider any tampering τ. As argued above, the tampering is either (1) additive or (2) we have information loss on the outputs. In case (1) for some B we have $C_0^\tau(X) = C_0(X) \oplus B$. Recall that C_0's outputs contain the actual outputs y_i and the z_i's used for the testing. If B doesn't flip any of the y_i's, then this tampering does not affect correctness and we don't have to bother. If B flips (i.e., XORs a 1 to 0) at least one y_i, we'll observe the mistake by querying C_0 on an arbitrary input. In case (2) there is at least one output *out* (could be a y_i or z_i value) and two inputs $T_0, T_1 \in \mathbb{T}_{\mathsf{test}}$ s.t. *out* has the same value in evaluations $C_0^\tau(T_0), C_0^\tau(T_1)$, but it should be different, thus it will be wrong in one of the two evaluations.

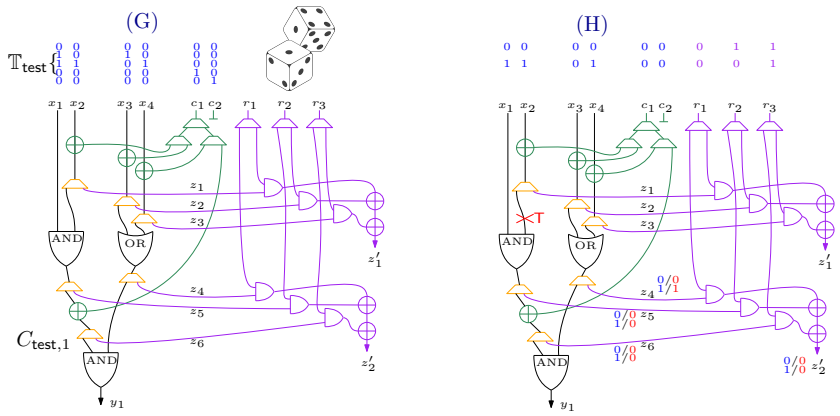

Fig. 4. (G) Illustration of the "information loss preserving" compression circuit (in purple) attached to 0th layer ETC $C_{\mathsf{test},0}$ from Figure 2.(C) to get a 1st layer ETC C_1. (H) For the two inputs $T_0 = 000000, T_1 = 110100$ to $C_{\mathsf{test},0}$ we have information loss on the z_5 and also z_6 output. For randomness $R = 011, Q = 001$ we then also observe information loss at the z'_2 output of $C_{\mathsf{test},1}$ on inputs $T_0\|R, T_1\|Q$.

Compressing the Output. The 0th layer ETC $(C_0, \mathbb{T}_{\mathsf{test}})$ is not practical as the number of output wires required for testing is linear in the size of the circuit.

To compress the output we construct a gadget circuit $G : \{0,1\}^{n_{in}+r} \to \{0,1\}^{n_{in}/r}$ that takes an n_{in} bit string and r random bits as input and outputs an $n_{out} = n_{in}/r$ bits. This compressing circuit on input $X\|R$ chops X into n_{out} strings $X_1, \ldots, X_{n_{out}}$, it then computes and outputs the inner products $\langle X_i, R\rangle$ of each X_i with R. For $n_{in} = 6, r = 3$ this gadget is illustrated by the purple subcircuit in Figure 4.*G*.

We will prove that even if the compressing circuit is tampered with, it will "preserve information loss" with good probability over the randomness. More formally, we consider any four inputs X_0, X_1, X'_0, X'_1 where at least in one position i the X_0, X_1 values are distinct, but the X'_0, X'_1 values are not, i.e., $\exists i \ : \ X_0[i] \neq X_1[i]$ and $X'_0[i] = X'_1[i]$. Now for any tampering τ and random R, Q consider the values

$$(Y_0, Y_1, Y_2, Y_3) = (G(X_0\|R), G(X_1\|R), G(X_0\|Q), G(X_1\|Q))$$
$$(Y'_0, Y'_1, Y'_2, Y'_3) = (G^\tau(X'_0\|R), G^\tau(X'_1\|R), G^\tau(X'_0\|Q), G^\tau(X'_1\|Q))$$

Then $\Pr_{R,Q}\left[\exists i, k < j \ : \ Y_k[i] \neq Y_j[i] \text{ and } Y'_k[i] = Y'_j[i]\right] \geq 1/2$.

Setting $r \approx \sqrt{n_{in}}$ we get $n_{out} = n_{in}/r \approx \sqrt{n_{in}}$ and thus can replace n_{in} output wires with $\sqrt{n_{in}}$ input and output wires (in our toy example we had $n_{in} = 6$ and $r = \lceil\sqrt{6}\rceil = 3, n_{out} = \lfloor\sqrt{6}\rfloor{=}2$).

If we apply the compression gadget just once, it is sufficient to prove that with good probability some of the outputs are wrong, i.e., that for some $j : Y_j \neq Y'_j$. We prove a more general property of information loss at the output of the gadget so we can cascade them. To balance the number of additional input and output

wires, for λ layers we compress the number of wires at each layer by a factor $\sqrt[\lambda+1]{n}$ which results in just $\sqrt[\lambda+1]{n}$ additional output and $\lambda \cdot \sqrt[\lambda+1]{n}$ input wires.

The main disadvantage of choosing a larger λ is the fact that the probability of detecting a tampering decreases, and one thus must make more test queries. Concretely, we must make $|\mathbb{T}_{\mathsf{test}}|2^{\lambda} \leq 6 \cdot 2^{\lambda}$ queries to be guaranteed to catch a tampering with probability $2^{-\lambda}$. By repeating this κ times we can amplify the success probability to $1-(1-2^{-\lambda})^{\kappa}$ at the cost of making $\kappa \cdot 6 \cdot 2^{\lambda}$ queries.

In practice, a very small λ will be sufficient, for example with $\lambda = 4$ we get a total of around 64 output and 256 input wires for a circuit with 2^{32} (around 4 billion) gates, and setting $\kappa = 11$ requires $11 \cdot 6 \cdot 2^4 = 1056$ queries to get a > 0.5 detection probability (that's what's guaranteed in the worst case by our security proof, as we did not optimize for constant, the practical security is certainly much better). Table 1 summarizes the efficiency of our ETC compiler.

	C	C_{wire}	C_{gate}	$C_{\mathsf{test},0}$	$C_{\mathsf{test},\lambda}, \lambda \in \mathbb{N}^+$
Number of gates	n	$\leq (3+o(1))\|C\|$ $\leq 3n + o(n)$	$\leq (7+o(1))\|C\|$ $\leq 7n + o(n)$	$\leq \|C_{\mathsf{gate}}\|+2 \cdot 4n$ $\leq 15n + o(n)$	$\leq \|C_{\mathsf{test},0}\| + 2 \cdot 4n$ $\leq 23n + o(n)$
Input size	s	$\leq s + 3$	$\leq s + 5$	$\leq s + 5$	$\leq s+5+\lambda \cdot \sqrt[\lambda+1]{4n}$
Output size	t	t	t	$\leq t + 4n$	$\leq t + \sqrt[\lambda+1]{4n}$
Cover/Test size		$\|\mathbb{T}_{\mathsf{wire}}\| \leq 4$	$\|\mathbb{T}_{\mathsf{gate}}\| \leq 6$	$\|\mathbb{T}_{\mathsf{test}}\| \leq 6$	$\|\mathbb{T}_{\mathsf{test}}\| \leq 6$
Success Probability		N/A	N/A	1	$2^{-2\lambda}$
Randomness (bits)		0	0	0	$\lambda \cdot \sqrt[\lambda+1]{4n}$

Table 1. Summary of the efficiency of our ETC compiler which first compiles the input circuit C into a (functionally equivalent) C_{wire} with a wire covering set $\mathbb{T}_{\mathsf{wire}}$. This is then compiled into a circuit C_{gate} with a gate covering set $\mathbb{T}_{\mathsf{gate}}$, which is then compiled into the 0th later ETC $C_{\mathsf{test},0}$ with the test set $\mathbb{T}_{\mathsf{test}}$. We then compress the additional output wires from $4n$ (i.e., linear in $|C| = n$) to the $\lambda + 1$th root of that by applying $\lambda \geq 1$ layers of our compression gadget. The testing is now probabilistic but can be repeated with fresh randomness until one gets the desired .

2.3 More Related Work

Testing circuits is a major topic in hardware manufacturing, the books [6, 10] discuss heuristics for testing and practical issues of the problem. Circuit-compilers (as used in this work) which harden a circuit against some "physical attacks" in a provably secure way were first introduced for leakage attacks (concretely, leaking values of a small number of wires) by Ishai et al. in [25]. Based on this compiler they later also gave a compiler to protect against tampering [24]. This line of research was continued in a sequence of papers on tampering wires [13, 14, 23, 21] or gates [26, 19, 28]. As discussed in the introduction, these compilers

aim at protecting secrets in the circuit, while efficiently testable circuits [3] only aim at detecting tampering in a special test phase.

Apart from compilers, a line of research was pioneered by Micali and Reyzin in [30] on reductions or composition of cryptographic building blocks to prevent "general" leakage. The first cryptographic primitive achieving security against a general notion of leakage (bounded leakage) from standard cryptographic building blocks is the leakage-resilient cipher from [17], by now we have leakage-resilient variants of most basic cryptographic primitives including signatures [20, 9] or MACS [5], an excellent overview on the area is [27]. Unfortunately for tampering no construction secure against general tampering – or even a notion of what "general tampering" means – exists. Although Non-malleable codes [18] can protect data in memory (rather than during computation) from very general classes of tampering attacks [29, 12, 1, 4, 15].

The most powerful physical attack model is Trojans, where an attacker can not just tamper the circuit, but completely replace it. Some limited provable-security results against this class of attacks are [16, 11]. There are few attempts to use general verifiable computation to certify the output of circuits [2, 31].

3 Preliminaries

We use notations and a tampering model similar to [3].

3.1 Notation for Circuits

Circuits can be modeled as directed acyclic graphs (DAGs), and we will extensively use standard graph theory notation. Concretely, a circuit is modeled as a DAG $C_\gamma = (V, E)$ where vertices refer to gates and the directed edges refer to wires. The circuit definition C_γ comes with a labeling function $\gamma : V \to \mathfrak{G}$ which assigns specific gates to the vertices, where $\mathfrak{G}$ is the set of gates allowed. We will often omit the parameter γ since it is chosen when specifying the circuit and cannot be changed. Each wire carries a bit from $\mathbb{Z}_2$, and each gate is taken from the set of allowed gates $\mathfrak{G}$ (including $\{\mathsf{AND}, \mathsf{OR}, \mathsf{XOR}, \mathsf{COPY}, \mathsf{NOT}\}$ and two special $\{\mathbf{in}, \mathbf{out}\}$ gates).

For $v \in V$, let $E^-(v) = \{(u, v) \in E\}$ and $E^+(v) = \{(v, u) \in E\}$ be the sets of v's incoming and outgoing edges, respectively. For $e = (u, v) \in E$ we define $V^-(e) = u$ and $V^+(e) = v$. We split the vertices into three sets $V = \mathcal{I} \cup \mathcal{G} \cup \mathcal{O}$, where $\mathcal{I} = \{I_1, I_2, ..., I_s\}$ are vertices which are assigned to **in**, and $\mathcal{O} = \{O_1, O_2, ..., O_t\}$ are these assigned to **out**. Given $C_\gamma = (V, E)$ and an input $X = (x_1, \ldots, x_s) \in \mathbb{Z}_2^s$ we define a valuation function

$$\mathrm{val}_{C_\gamma, X} : V \cup E \to \mathbb{Z}_2 \tag{3}$$

which assigns each gate the value it outputs and each wire the value it holds when the circuit is evaluated on X. More formally the valuation function for vertices $v \in V$ and edges $e \in E$ is defined as

$$\mathrm{val}_{C_\gamma,X=(x_1,x_2,\ldots,x_s)}(v) = \begin{cases} x_i, & \text{if } v = I_i. \\ \gamma(v)(\mathrm{val}_{C_\gamma,X}(E^-(v))), & \text{otherwise.} \end{cases}$$
$$\mathrm{val}_{C_\gamma,X}(e) = \mathrm{val}_{C_\gamma,X}(V^-(e)).$$

We will sometimes just write val_X if the circuit considered is clear from the context. The behaviour of the circuit C can be associated with the function that it evaluates, i.e. $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$. We can define this function as follows: $C(X) = (\mathrm{val}_{C,X}(O_1), \mathrm{val}_{C,X}(O_2), ..., \mathrm{val}_{C,X}(O_t))$.

3.2 Tampering Model

We consider an adversary who can arbitrarily tamper with every wire of the circuit, i.e. flip its value, set it to 0, set it to 1, or leave it untampered. Unlike [3] or [24], we *do not* take advantage of the *conductivity* assumption. This means, we operate on circuits with conductivity 1, where all nodes $n \in V$ of a circuit C have fan-in and fan-out equal to an inherent fan in, fan-out of $\gamma(n)$, and all output wires of all nodes can be tampered independently. We assume that the input circuit is not conductive. In [3], the authors assumed k-conductivity, i.e. a value of some wire in the circuit could be copied to at most k distinct destinations with a restriction that all of them must be tampered equally. Without loss of generality, every k-conductive circuit can be transformed into a 1-conductive circuit, as by using COPY gates, every k-conductive circuit can be turned into a non-conductive one while at most doubling the circuit size and increasing the depth by a factor $\lceil \log(k) \rceil$.

The tampering of a wire is described by a function $\mathbb{Z}_2 \to \mathbb{Z}_2$ from the set of possible bit tamper functions $\mathcal{T} = \{\mathrm{id}, \mathrm{neg}, \mathrm{one}, \mathrm{zero}\}$. The tampering of an entire circuit $C = (V, E)$ is defined by a function $\tau : E \to \mathcal{T}$ mapping each wire to a tampering function. For convenience, we sometimes write τ_e to denote $\tau(e)$.

Now we can extend our notion of the valuation to also take tampering into account in order to define the valuation of a tampered circuit

$$\mathrm{val}_X^\tau : V \cup E \to \mathbb{Z}_2.$$

The only difference to the (non-tampered) valuation function from Eq(3) is that we apply the tampering to each value of an edge after it is being computed, formally:

$$\mathrm{val}^\tau_{C_\gamma,X=(x_1,2,\ldots,x_s)}(v) = \begin{cases} x_i, & \text{if } v = I_i. \\ \gamma(v)(\mathrm{val}^\tau_{C_\gamma,X}(E^-(v))), & \text{otherwise.} \end{cases}$$
$$\mathrm{val}^\tau_{C_\gamma,X}(e) = \tau_e(\mathrm{val}^\tau_{C_\gamma,X}(V^-(e))).$$

By C^τ we can again understand a function that describes the input-output behavior of the *tampered* circuit: $C^\tau(X) = (\mathrm{val}^\tau_{C,X}(O_1), \mathrm{val}^\tau_{C,X}(O_2), ..., \mathrm{val}^\tau_{C,X}(O_t))$.

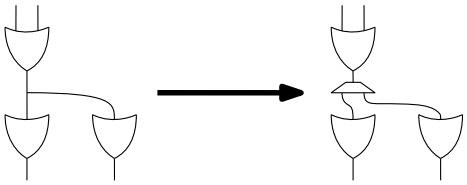

Fig. 5. Examples of $2-$*conductive* (left circuit) and $1-$*conductive* (right circuit) circuits. A single wire on the left circuit is copied to two destinations. The adversary can apply only a single tampering to this wire. On the right circuit, this wire is divided into three parts with a COPY gate. The adversary can apply separate tampering to each of these parts.

4 Gate Covering Sets

The authors of [3] developed a notion of *wire covering set* $\mathbb{T}_{\mathsf{wire}}$. It is a set of inputs to a specific circuit, such that every wire is evaluated to both 0 and 1, given some inputs from the test set $\mathbb{T}_{\mathsf{wire}}$ (see Definition 1 below). Moreover, the paper states that every circuit can be efficiently compiled into its wire-covered version (Theorem 10 from [3]). We denote the compilation procedure (Algorithm 10 from [3]) as Algorithm A. It compiles a circuit C into a functionally equivalent C_{wire}, along with its wire covering set $\mathbb{T}_{\mathsf{wire}}$.

Definition 1 (Definition 4 from [3]). *The set* $\mathbb{T}_{\mathsf{wire}}$ *is a wire covering set for a circuit* C *if* $\forall\ e \in E(C),\ b \in \{0,1\}\ \exists\ X \in \mathbb{T}_{\mathsf{wire}}\ :\ \mathrm{val}_{C,X}(e) = b$.

In this paper, we develop a stronger notion called *gate covering set*. Here, we not only require covering all wires of the circuit, but also pairs of wires that form an input to multi-input gates (in our model the **and**, **or**, **xor** gates).

Before we proceed, we expand our notation by the sequences of the input values to the gate *i.e.*, by $(\mathrm{val}_{C,X}(e))_{e\in E^-(v)}$ we understand the sequence of the values given to v, when the circuit is evaluated on the input X. E.g. for a gate v in a circuit C that is evaluated on 0 and 1 given input X to C, we write $(\mathrm{val}_{C,X}(e))_{e\in E^-(v)} = 01$. Now, we give the definition of the gate covering set.

Definition 2. $\mathbb{T}_{\mathsf{gate}}$ *is a gate covering set for a circuit* C *with gate assignment* γ *if it satisfies:*

- $\forall_{v\in V(C)\ :\ \gamma(v)\in\{\mathsf{COPY},\mathsf{NOT},\mathsf{OUTPUT}\}} : |\{(\mathrm{val}_{C,X}(e))_{e\in E^-(v)} : X \in \mathbb{T}_{\mathsf{gate}}\}| \geq 2$,
- $\forall_{v\in V(C)\ :\ \gamma(v)\in\{\mathsf{AND},\mathsf{OR}\}} : |\{(\mathrm{val}_{C,X}(e))_{e\in E^-(v)} : X \in \mathbb{T}_{\mathsf{gate}}\}| = 4$,
- $\forall_{v\in V(C)\ :\ \gamma(v)\in\{\mathsf{XOR}\}} : |\{(\mathrm{val}_{C,X}(e))_{e\in E^-(v)} : X \in \mathbb{T}_{\mathsf{gate}}\}| \geq 3$.

We call a circuit with a gate-covering set a gate-covered circuit. Any node in a circuit that has enough evaluation sequences as in the Definition 2, given any test set, we call gate-covered.

To construct a gate covering of a circuit we first use the recalled Algorithm A to obtain a wire covering of the circuit and then we go through the multi-input gates of the intermediary circuit topologically to ensure that they are evaluated on a sufficient number of their inputs combinations. In the algorithm, it is sufficient to add XOR gates to the input wires of the topologically traversed gates of the circuit to gate-cover them. The Algorithm 1 describes a procedure that takes as input a circuit C and outputs a functionally-equivalent circuit C_{gate} along with a gate-covering set $\mathbb{T}_{\mathsf{gate}}$ for C_{gate}.

Algorithm 1: Algorithm for constructing a gate covering set for C

Input: $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$
Output: $C_{\mathsf{gate}}, \mathbb{T}_{\mathsf{gate}}$
$(C_{\mathsf{wire}} : \mathbb{Z}_2^{s+s_w} \to \mathbb{Z}_2^t, \mathbb{T}_{\mathsf{wire}}) = \mathit{Algorithm}\ A(C)$ `/* Wire-covered intermediary circuit */`
Initialize $C_{\mathsf{gate}} = C_{\mathsf{wire}}$
Initialize $\mathbb{T}_{\mathsf{gate}} = \mathbb{T}_{\mathsf{wire}}$
Append 00 to every $X \in \mathbb{T}_{\mathsf{gate}}$
$X_0 = 0^{s+s_w}10$
$X_1 = 0^{s+s_w}01$
for $v \in V(C_{\mathsf{gate}}) : \gamma(v) \in \{\mathsf{OR}, \mathsf{AND}, \mathsf{XOR}\}$ (processed in a *topological* order) **do**
  $S = \{(\mathrm{val}_{C,X}(e))_{e \in E^-(v)} : X \in \mathbb{T}_{\mathsf{gate}}\}$ `/* Assert` $|S| \geq 2$ `*/`
  for $e_i \in \{00, 01, 10, 11\} \setminus S$ **do**
    $p_i = (\mathrm{val}_{C,X_i}(e))_{e \in E^-(v)}$ `/* Get current valuation of the` v`'s input wires on the input` X_i `*/`
    for $j \in \{0, 1\}$ **do**
      if $e_i[j] \neq p_i[j]$ **then**
        `/* Given the input` X_i`, the` $s + s_w + i$`'th input wire of the circuit has value 1 */`
        Update C_{gate} by adding a XOR gate that has one of the inputs connected $s + s_w + i$'th input wire of the circuit and its second input is the j'th input wire of v. The output of the XOR gate will be a new j'th input of v `/* When the control bits` X_0, X_1 `are used at least twice, one needs to use linear number of COPY gates in the construction to assure that` C_{gate} `remains non-conductive */`
return $C_{\mathsf{gate}}, \mathbb{T}_{\mathsf{gate}} \cup \{X_0, X_1\}$

Proposition 1 *The Algorithm 1 transforms a circuit C into a functionally equivalent circuit C_{gate} along with gate-covering set $\mathbb{T}_{\mathsf{gate}}$.*

Proof. It is easy to see that the circuit C_{gate} is functionally equivalent to C, since it does not add any new output bits to the circuit, and all the new gates are XOR gates connected via a sequence of COPY gates to the new control bits. Whenever

these bits are set to 0, the new XOR gates do not affect the behaviour of the circuit. Note that after adding the new XOR gates, all of the wires connected directly to the old gates of the circuit remain wire-covered by the old test set adjusted by adding 00 to its every input. What is more, the new control bits wire-cover every new wire added to the circuit. This implies that every gate from the set $\{\mathsf{COPY}, \mathsf{NOT}, \mathsf{OUTPUT}\}$ in the updated circuit is trivially gate-covered (evaluated to both $0, 1$ given some inputs from the test set).

When we topologically go through the gates from the set $\{\mathsf{OR}, \mathsf{AND}, \mathsf{XOR}\}$ of the intermediary circuit, we can see that since the input wires to the circuit are wire-covered by the adjusted old test set, then these gates are partially covered before and after adding new XOR gates to their input wires (i.e. $\{(\mathrm{val}_{C,X}(e))_{e\in E^-(v)} : X \in \mathbb{T}_{\mathsf{gate}}\} \geq 2$). In the step 9 of the Algorithm, we add XOR gates connected to the new control bits to cover at most two missing evaluation sequences. The XOR gates added during the topological procedure are evaluated on two distinct input sequences, given inputs from the adjusted old wire-covering set. The third distinct input comes from setting their respective control bit to 1 □

Proposition 2 *For any circuit C with max fan-in 2, number of gates n, the Algorithm 1 creates a circuit with additional 5 input bits, test set of size 6, additional $6n$ gates.*

Proof. The Algorithm A from [3] compiles into a circuit with additional 3 input bits, test set of size 4 and adds at most n XOR gates and n COPY gates. Now, adding 2 input bits and 2 test inputs to the test set in the second part of Algorithm 1, and adding at most $2n$ XOR gates and $2n$ COPY gates during the iteration concludes the result □

5 Information Loss in Gate-Covered Circuits

In this section, we define information loss and show that it is easily trackable in any C_{gate} that has a gate-covering set $\mathbb{T}_{\mathsf{gate}}$. For any such circuit, we show the following property: for any tampering applied to the wires of the C_{gate}, either we observe an information loss on one of the output wires of the multi-input gates $\mathsf{AND}, \mathsf{OR}, \mathsf{XOR}$ (given only the inputs from the gate-covering set $\mathbb{T}_{\mathsf{gate}}$), or the output wires of the circuit are always set to a constant value or always toggled or always correctly evaluated.

Theorem 1. *For any circuit $C_{\mathsf{gate}} : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ with gate-covering set $\mathbb{T}_{\mathsf{gate}}$, for any tampering function τ applied to the circuit then at least one of the following holds:*

- ***Information loss on multi-input gates***

$$\exists_{X_0, X_1 \in \mathbb{T}_{\mathsf{gate}},\ n \in V(C_{\mathsf{gate}})\ :\ \gamma(n) \in \{\mathsf{AND},\mathsf{OR},\mathsf{XOR}\}}\ :$$

$$\left(\mathrm{val}_{X_0, C_{\mathsf{gate}}}(n) = 0 \wedge \mathrm{val}_{X_1, C_{\mathsf{gate}}}(n) = 1\right) \wedge \left(\mathrm{val}^{\tau}_{X_0, C_{\mathsf{gate}}}(n) = \mathrm{val}^{\tau}_{X_1, C_{\mathsf{gate}}}(n)\right)$$

- ***Constant output***

$$\exists_{i\in[t],c\in\{0,1\}}\forall X\in\mathbb{Z}_2^s \ : \ C^{\tau}_{\mathsf{gate}}(X)[i]=c$$

- ***At most toggled output***

$$\exists_{T\in\{0,1\}^t}\forall X\in\mathbb{Z}_2^s : \ C^{\tau}_{\mathsf{gate}}(X)=C_{\mathsf{gate}}(X)+T$$

Proof. The proof follows a modular argument. For this, we need a definition of Topological Layers of Computation on any circuit C_γ. In the definition below, we say that a wire e is connected to a gate g in a circuit C_γ described with a DAG (denoted by predicate $connected_{C_\gamma}(g,e)$ holds) if and only if there exists a direct connection or connection going through a path of COPY or NOT gates between g and the predecessor of e in the circuit.

Definition 3 (Topological Layers of Computation). *For any circuit C, we recursively define its Topological Layers of Computation:*

- 0^{th}*-layer of Computation* $\mathcal{L}_0=\mathcal{I}(C)$
- i^{th}*-layer of Computation* $\mathcal{L}_i=\{g\in V(C_\gamma) : \forall_{e\in E^-(g)} : connected_{C_\gamma}(g',e)$ *for some* $g'\in\mathcal{L}_0\cup\ldots\cup\mathcal{L}_{i-1}$ *and* $\gamma(g)\in\{\mathsf{XOR},\mathsf{AND},\mathsf{OR}\}\}$.

By $G_i(C)$ we denote a subgraph induced by the layers $\mathcal{L}_0,\ldots,\mathcal{L}_i$ of the circuit.

Below we consider $C=C_{\mathsf{gate}}$. We run an experiment that evaluates layer by layer the tampered C (assuming C has $L+1$ layers). In the i'th layer either there is an information loss and we stop the experiment or the output of the layer is at most toggled [see event $\mathcal{E}_2$ below] and the experiment proceeds to the next layer. We define the following predicates for a gate g in layer i:

- $\mathcal{E}_1(g,i)$ holds if $g\in\mathcal{L}_i \ \wedge \ \exists_{X_0,X_1\in\mathbb{T}'}\mathrm{val}_{X_0,C}(g)=0 \ \wedge \ \mathrm{val}_{X_1,C}(g)=1 \ \wedge \ \mathrm{val}^{\tau}_{X_0,C}(g)=\mathrm{val}^{\tau}_{X_1,C}(g)$,
- $\mathcal{E}_2(g,i)$ holds if $g\in\mathcal{L}_i \ \wedge \ \forall_{X\in\mathbb{Z}_2^s} : \mathrm{val}^{\tau}_{X,C}(g)=\mathrm{val}_{X,C}(g)+f\Big[\tau(e) : e\in E(G_i(C))\Big]$.

In the 0^{th}-layer of the circuit, by definition of the tampering function, for any node $g\in\mathcal{L}_0(C)$: $X\in\mathbb{Z}_2^s : \mathrm{val}^{\tau}_{X,C}(g)=\mathrm{val}_{X,C}(g)$. This implies event $\mathcal{E}_2(g,0)$ on any gate from this layer. We prove the following for the tampered circuit C:

$$\forall_{\tau(C),i\in\{1,\ldots,L\}} : \forall_{j\in\{0,\ldots,i-1\},g'\in\mathcal{L}_j} : \mathcal{E}_2(g',j) \implies \forall_{g\in\mathcal{L}_i(C)} \ : \ \mathcal{E}_1(g,i)\vee\mathcal{E}_2(g,i)$$

We first study the gates of the first layer:

- The AND gate in the 1^{st}-layer is connected to the input gates only via a sequence of COPY and NOT gates. The computation on this gate can be described as $P_g(a,b)=a\cdot b$. The tampered output of the gate is $\tilde{P}_g(a,b)=\tilde{a}\cdot\tilde{b}$, where $\tilde{a}\in\{a,a+1,0,1\}$, $\tilde{b}\in\{b,b+1,0,1\}$. The tampering of a wire a is set to 1 or 0 whenever there is a constant tampering on its path from the

0^{th}layer, $a+1$ or $b+1$ whenever on the path there is an odd number of toggle tamperings, and a or b whenever there is an even number of toggle tamperings on the path. Whenever $\tilde{a} = 0 \vee \tilde{b} = 0$, then $\tilde{P}_g(a,1) = 0$ and $P(a,1) = a$, we get an information loss. Now, since by the construction of the Algorithm 2, the wire P is connected via a COPY to the output, the event $\mathcal{E}_1(g,1)$ occurs.
In other cases:
  - if $\tilde{a} = 1$ (or $\tilde{b} = 1$), $P(a,1) = a$ and $\tilde{P}(a,1) = const.$ (resp. $P(b,1) = b$ and $\tilde{P}(b,1) = const.$) [$\mathcal{E}_1(g,1)$ occurs],
  - when $\tilde{a} = a+1$ (or $\tilde{b} = b+1$), then $P(1,b) = b$ and $\tilde{P}(1,b) = 0$ (resp. $P(1,a) = 1$ and $\tilde{P}(a,1) = 0$) [$\mathcal{E}_1(g,1)$ occurs],
  - otherwise $\tilde{P}(a,b) = ab$ [$\mathcal{E}_2(g,1)$ occurs].
- Similar argument as above applies for the OR gate,
- The input wires of the XOR gate are also connected only via a sequence of COPY and NOT gates to the input. We observe that $P_g(a,b) = a+b$, and the tampered output $\tilde{P}_g(a,b) = \tilde{a}+\tilde{b}$, where $\tilde{a} \in \{a, a+1, 0, 1\}$, $\tilde{b} \in \{b, b+1, 0, 1\}$.
  - if $\tilde{a} = const.$ (or $\tilde{b} = const.$), $P(a,0) = a$ and $\tilde{P}(a,0) = const.$ (resp. $P(0,b) = b$ and $\tilde{P}(0,b) = const.$) [$\mathcal{E}_1(g,1)$ occurs],
  - when $\tilde{a} = a+c_a, \tilde{b} = b+c_b$, then $P(a,b) = a+b$, $\tilde{P}(a,b) = a+b+c_a+c_b$ [$\mathcal{E}_2(g,1)$ occurs].

In the i'th layer, the inputs to all of the gates are, again, connected to the gates of the previous layers only via a sequence of COPY, NOT gates. Now, once the induction assumption holds in the layers $\{1,\ldots,i-1\}$, the event $\mathcal{E}_2$ on all gates assures that the case analysis from the first layer may be repeated, but the tampered wires $\tilde{a}, \tilde{b}$ will now get a constant tampering 0 or 1, or a toggle bit depending on the tamperings chosen on the edges of the graph induced by layers from the set $\{0,\ldots,i\}$.

This implies that on multi-input gates of the circuit, we either get event $\mathcal{E}_1$ or $\mathcal{E}_2$. Whenever the event $\mathcal{E}_1$ occurs, the information loss on one of the multi-input gates of the circuit occurs. Otherwise only the event $\mathcal{E}_2$ on these gates may occur. The OUTPUT gates of the circuit are connected via a sequence of COPY and NOT gates to the gates of the topological layers of computation of the circuit. If on their paths one finds a constant tampering, then some output bit is set constant; if only toggles are found there, the output bits are at most toggled □

5.1 Routing the Information Loss in Gate-Covered Circuits

In this section, we show that any gate-covered circuit can be converted to another gate-covered circuit for which any information loss that appears on its multi-input gates is routed to the output of the circuit. We present Algorithm 2 that adds a COPY gate to the output wires of the multi-input gates in the gate-covered circuit. The added COPY gates forward one copy of the original wires to their previous destinations and another copy directly to the output (Fig 6).

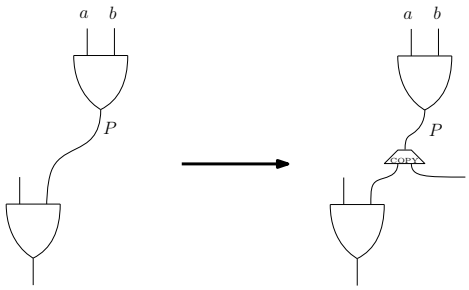

Fig. 6. Adding a COPY gate to the wire P in the Algorithm 2. This creates two wires; the left one is connected to the previous successor of the wire P, and the right one is sent to the output of the circuit. Algorithm 2 takes into account only the wires P which originate at AND, OR, XOR gates in the original circuit.

Algorithm 2: Algorithm for Routing the Information Loss in a Gate-Covered C

Input: $C_{\mathsf{gate}} : \mathbb{Z}_2^s \to \mathbb{Z}_2^t, \mathbb{T}_{\mathsf{gate}}$
Output: $C_{\mathsf{test},0}, \mathbb{T}_0$
Initialize $C_{\mathsf{test},0} = C_{\mathsf{gate}}$, $\mathbb{T}_0 = \mathbb{T}_{\mathsf{gate}}$
for $g \in V(C_{\mathsf{gate}})$ **do**
  if $g \in \{\mathsf{AND}, \mathsf{OR}, \mathsf{XOR}\} \wedge E^+(g)$ *is not an output wire of* $C_{\mathsf{test},0}$ **then**
    Insert to $C_{\mathsf{test},0}$ a COPY gate between g and $V^+(w)$.
    One of the output wires of the new gate should go to $V^+(w)$, the other one should be left as an additional output wire of the modified circuit.
return $C_{\mathsf{test},0}, \mathbb{T}_0$

Proposition 3 *The Algorithm 2 transforms a gate-covered circuit* $C_{\mathsf{gate}} : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ *with gate-covering set* $\mathbb{T}_{\mathsf{gate}}$ *into another gate-covered circuit* $C_{\mathsf{test},0} : \mathbb{Z}_2^s \to \mathbb{Z}_2^{t+t_0}$ *with additional output bits and the same gate-covering set,* $\mathbb{T}_0 = \mathbb{T}_{\mathsf{gate}}$*, where one observes for any tampering* τ *of the circuit* $C_{\mathsf{test},0}$ *at least one of the following holds:*

- ***Information loss on output:*** $\exists b \in \{0,1\}, X_0, X_1 \in \mathbb{T}_0, i \in \{1, \ldots, t+t_0\}$ *such that*

$$\mathrm{val}_{X_0}(C_{\mathsf{test},0})[i] = 0, \mathrm{val}_{X_1}(C_{\mathsf{test},0})[i] = 1, \mathrm{val}^{\tau}_{X_0}(C_{\mathsf{test},0})[i] = \mathrm{val}^{\tau}_{X_1}(C_{\mathsf{test},0})[i] = b$$

- ***At most toggled output***

$$\exists_{B \in \{0,1\}^t} \forall X \in \mathbb{Z}_2^s \ \exists Y \in \mathbb{Z}_2^{t_0} : \ C^{\tau}_{\mathsf{test},0}(X) = C_{\mathsf{gate}}(X) \| Y + B \| 0^{t_0}$$

Proof. It is easy to see that the same test set $\mathbb{T}_0 = \mathbb{T}_{\mathsf{gate}}$ is a gate covering set for the transformed $C_{\mathsf{test},0}$. Now, according to Theorem 1 on the transformed circuit the following cases may follow:

1. Information-loss on one of the multi-input gates of $C_{\mathsf{test},0}$: in this case, one of the output wires of $C_{\mathsf{test},0}$ is connected via a COPY gate to the output of the multi-input gate and the information loss is propagated to this wire.

2. One of the output wires of $C_{\mathsf{test},0}$ always evaluates to a constant value: in this case, we observe an information loss on this wire because it is wire-covered according to the definition of the gate-covering set $\mathbb{T}_0$.
3. At most, toggled output on the circuit.

6 Minimizing the Number of External Wires

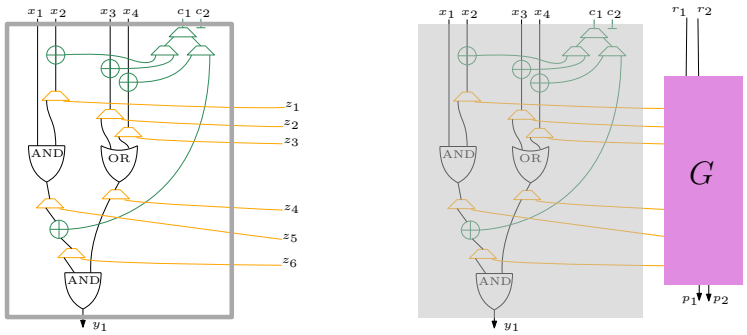

Fig. 7. To reduce the number of external wires, we add a compressing gadget $G_{n,\lambda,d}$.

In the previous Section, we introduced the notion of information loss and showed that without limitations on the number of *additional* input/output wires, the tamper-resilience can be achieved for 1-conductive circuits. For practical reasons, the number of the external wires, n_{external}, is more limited compared to the number of internal wires, n_{internal}. Typically, the external wires are on the border of a square which contains internal wires [6, 10], thus a convincing relation between these numbers can be given by $n^2_{\text{external}} \leq c \cdot n_{\text{internal}}$, where c is some constant. Unfortunately, in the construction given in the previous Section, the number of external wires in the compiled circuit is *linear* in the number of the internal wires in the original ones, which is not practical to implement in a real-life circuit. Thus, in this section, we focus on minimizing the number of external wires in our precompiled circuit.

We will define a *compressing gadget* $G_{n,\lambda,d}$. The gadget will compress an input of length n. It will be composed of λ layers of smaller sub-gadgets each of which will need d additional wires with uniformly random bits as input. The gadget $G_{n,\lambda,d}$ will take $\lambda \cdot d$ additional input wires and will have a limited number of output wires (much lower than the number of its input wires - see Figure 7). We will show that even if the adversary tampers with the compressing gadget $G_{n,\lambda,d}$, the information loss on any of its input wires will survive through it and can be detected on the output of the gadget with sufficiently high probability. In practice, we will be able to keep λ at most 5.

6.1 Construction of One Layer Compression

The $G_{n,\lambda,d}$ gadget consists of λ layers that we define as subgadgets $S_{m,d}$ (with varying parameter m). The single-layer compression gadget $S_{m,d}$ compresses m bit input into $\lceil \frac{m}{d} \rceil$ bit output (using d additional input wires which will be uniformly random bits during the testing procedure). For ease of analysis, we can consider m to be a multiple of d (otherwise we can add $m - \lfloor \frac{m}{d} \rfloor$ spare input wires set to 0 to the $S_{m,d}$ single-layer gadget). Sometimes we refer to $S_{m,d}$ as simply S_d when m is clear from the context.

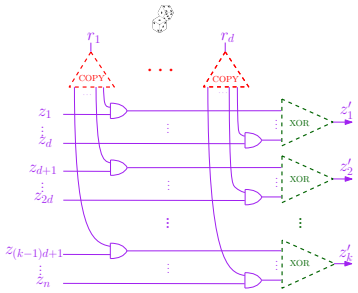

Fig. 8. Construction of - $S_{m,d}$. The dotted red triangle represents the copying tree, $\triangle_k$. The dotted green triangle represents the xoring tree, $\triangleright_d$.

$S_{m,d}$ takes $m + d$ wires as input, where d are additional inputs composed of uniformly random bits, and outputs $\frac{m}{d}$ wires. First, $S_{m,d}$ divides the input sequence that it receives (say $z_1, z_2, \ldots, z_m$) into $\frac{m}{d}$ blocks of length d

$$\left((z_1, \ldots, z_d), (z_{d+1}, \ldots, z_{2d}), \ldots, (z_{(\frac{m}{d}-1)d+1}, \ldots z_{(\frac{m}{d}-1)d+d})\right).$$

Then using the additional sequence of d input bits $r_1, ..., r_d$ it outputs the value of the inner product of each length d block of $z's$ and the additional sequence. More formally, for $S_{m,d}$ given input wires $(z_1, ..., z_m)$ and additional input wires $(r_1, ..., r_d)$, $S_{m,d}$ outputs $\frac{m}{d}$ bits:

$$S_{m,d}\left((z_i)_{i=1,\ldots,m}, (r_i)_{i=1,\ldots,d}\right) = \left(\sum_{j=1,\ldots,d} z_{id+j} \cdot r_j\right)_{i=0,\ldots,\frac{m}{d}-1}.$$

The construction of $S_{m,d}$ is shown in Figure 8. An instantiation with $m = 8, d = 4$ is shown in Figure 10. Construction of $S_{m,d}$ needs as building blocks 2 types of gadgets: copying tree (with fan-in 1, but high fan-out) consisting of COPY gates and xoring tree (with high fan-in, but fan-out 1) consisting of XOR gates. They are realized by tree-like gadgets that we denote with the following symbols - $\triangle_{m'}, \triangleright_d$ (see Figure 9).

The *copying tree*, $\triangle_{m'}$, takes a single wire as input and outputs m' wires which, as the name suggests, are copies of the input in untampered computation. This is achieved by a complete binary tree with m' leaves where the root is the input, the leaves are the output and all internal nodes are COPY gates. The direction of computation is from the root to the leaves.

The *xoring tree*, $\triangleright_d$ takes d wires as input and outputs 1 wire which in the untampered circuit is the **xor** of all the inputs. It achieves this by a complete binary tree with d leaves, where leaves are the input, the root is the output and all nodes are XOR gates. The direction of computation is from the leaves to the root. From the construction above, we obtain the following properties of $S_{m,d}$.

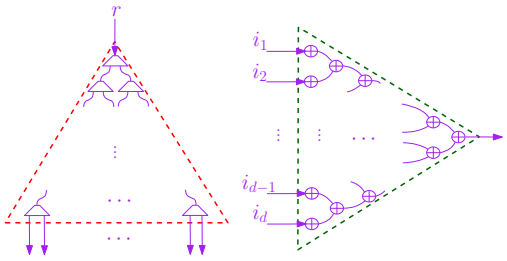

Fig. 9. The gadgets $\triangle, \triangleright$ are realized by complete binary trees with COPY, XOR in nodes, respectively.

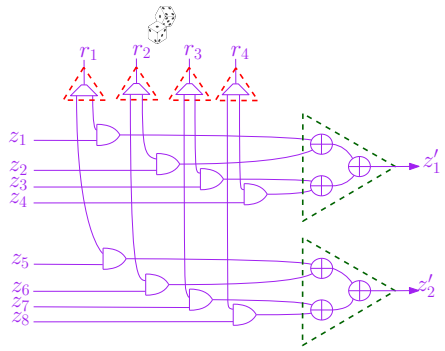

Fig. 10. A single layer of compression - $S_{m,d}$ (in this case $m = 8, d = 4$). Trapeziums represent copy gates. The dotted red triangle represents the copying tree, $\triangle_2$. The dotted green triangle represents the xoring tree, $\triangleright_4$.

Lemma 1. *For the $S_{m,d}$ gadget with m input wires and additional d input wires for randomness (Figure 8), the following holds: (1) The number of output wires is $\frac{m}{d}$, (2) The depth is less than $\log \frac{m}{d} + \log d + 1 = \log m + 1$, (3) The total number of gates is less than $d \cdot \frac{m}{d} + n + \frac{m}{d} \cdot d = 3m$ (number of gates in the copying trees, plus the number of multiplication gates plus number of gates in the xoring trees).*

6.2 Composing The Layers

Now we are ready to present the full construction of $G_{n,\lambda,d}$. This is achieved by simply adding λ layers of $S_{m,d}$ gadgets, with varying parameter m depending on the layer (see Figure 11). The first layer of S takes as input the input wires to the gadget G; the next layer takes as input the output of the previous layer, and so on. We change the parameter m of inputs to $S_{m,d}$ in each layer accordingly. The output of the last layer is the output of the G. Every layer reduces the number of output wires by a factor of d. Every layer is given d extra input wires which would be uniformly random bits.

Intuition: In Proposition 3, it was shown, that any non-trivial tampering implies an error on the standard output wires or an information loss on the auxiliary output wires (which are input wires to the compressing gadget G). Here we focus on the second case. We can conclude, that if there is any error on the input wire corresponding to the value z_i (what is implied by the information loss), we may hope it to survive through λ of the S_d layers - sometimes the value on this particular wire will be changing the value of the respective inner product, and sometimes not, independent of everything else except the value of some r_j.

From the construction above, we obtain the following.

Lemma 2. *Let $G_{n,\lambda,d}$ receive a sequence of length $n = m \cdot d^{\lambda}$ as an input to be compressed. Then the following statements are true: (1) $G_{n,\lambda,d}$ outputs m bits. (2) It needs $\lambda \cdot d$ auxiliary random bits. (3) The depth of $G_{n,\lambda,d}$ is bounded by $\lambda \cdot (\log n + 1)$. (4) The total number of its gates is not greater than $\sum_{i=0}^{\lambda-1} \left(3\frac{n}{d^i}\right) = 3n\frac{d-d^{1-\lambda}}{d-1}$.*

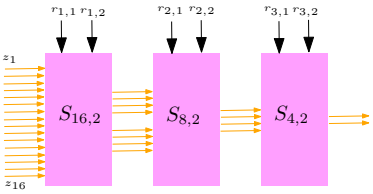

Fig. 11. The compressing gadget $G_{n,\lambda,d}$ consists of λ layers of the compression subgadgets $S_{m,d}$, where the number of input wires m decreases layer by layer. Example parameters are $n = 16, \lambda = 3, d = 2$.

6.3 Information Losing Tuples

Recall that in Proposition 3, we show that any meaningful error of computation will result in an information loss on one of the output wires of the precompiled circuit. In the following Sections, we will describe that the $G_{n,\lambda,d}$ gadget propagates the information loss on some of its input wires to one of its output wires with good probability. The reason that we focus on the propagation of the information loss, not a single error on computation is that the values of the input the $G_{n,\lambda,d}$ gadget and the tamperings may be adversarially chosen in a way that the error vanishes. E.g. imagine a wire in $G_{n,\lambda,d}$ that is (almost) always evaluated to 0 on the test inputs in an untampered evaluation, and the adversarial tampering flips the value of this wire to 1, given some specific inputs. Then the adversary may undo the wrong evaluation on this wire with another constant tampering. In general, it is easy for an adversary to undo the (almost) always correct or (almost) always incorrect evaluations. We will thus make use of the information loss - a pair of evaluations on a single wire that ensures that this wire evaluates to both 0 and 1, and an error occurs on one of these evaluations.

We introduce the notion of information-losing tuples which separates the idea of information loss from the process of evaluation of the whole circuit. In the definition below the n-ary vectors over $\mathbb{Z}_2$ denoted with X_i denote honest evaluations of n wires and the vectors denoted with Y_i denote tampered evaluations of the same wires of some circuit C.

Definition 4. *We say that $(X_1, ..., X_m; Y_1, ..., Y_m)$ - a tuple of n-ary vectors over $\mathbb{Z}_2$ - is an information-losing tuple if $\exists_{i,j,k} \left((X_i[k] \neq X_j[k]) \wedge (Y_i[k] = Y_j[k])\right)$. The triple (i, j, k) is called an information-losing witness for $(X_1, ..., X_m; Y_1, ..., Y_m)$*

Recall Proposition 3. Let $(X_i), (Y_i)$ denote the values on the output wires of $C_{\mathsf{gate}}, C^{\tau}_{\mathsf{gate}}$ for the gate covering set $\mathbb{T}_{\mathsf{gate}} = (T_i)$. Then *information loss on the output* means, that $((X_i), (Y_i))$ forms an information-losing tuple if the information loss occurs on the output of the circuit.

6.4 Algebraic Values on the Wires

Now we analyze what are the (parameterized by the input) possible values on wires in *tampered* realization of $G_{n,\lambda,d}$. We will use an algebraic notation for the computation on the circuit. The wires of the circuit carry not only the elements of $\mathbb{Z}_2$, but elements of a ring of multivariate polynomials over $\mathbb{Z}_2$. The indeterminates of this ring for a single circuit will be associated with its input wires and will be denoted with lowercase letters; sometimes we will be using auxiliary indeterminates. To compute the results of the val function, we simply extrapolate the functions from $\mathfrak{G}$ to the ring. From now on, whenever we refer to *value* on the wire, we allow the value to be an element of the ring.

In this setting, how does the tampering of the wire affect its value? It works the same way as before - toggling is simply adding 1 to the polynomial, and setting 0/1 is setting the polynomial to be equal to 0/1, without indeterminates. Therefore we can make some observations on the gadgets $\triangleright, \triangle$ (from Figure 9), and the output of the multiplication gates in S_d.

Proposition 4 (Output of the copying trees) *Let $\triangle^\tau$ be given r as input, and r' be any of its output. Then $r' \in \{0, 1, r, r+1\}$.*

Every output of a copying tree is either constant, toggled or the original value of its single input wire, depending on the number of toggling or constant tamperings on the path from the root of the copying tree to the output wire.

Proposition 5 (Output of the xoring trees) *Let $a_1, ..., a_d$ be the input values to $\triangleright^\tau$ and p be its output. Then $p = \beta + \sum_{i=1,...,d} \alpha_i a_i$, where $\alpha_i, \beta \in \{0, 1\}$.*

The single output of the xoring tree is a linear combination of its input. If there is a constant tampering on a path from some input wire to the output wire, the coefficient α_i of the input value a_i is set to 0, the coefficient β depends on the number of toggling tamperings and values of the constant tamperings.

Proposition 6 (Output of the multiplication gates) *Let (z_i, r_i) be a pair of input wires to some multiplication gate in S^τ and let $mult_i$ denote the output value of this multiplication gate. Then $mult_i = \alpha_i(z_i)r_i + \beta_i(z_i)$, where α_i, β_i are linear functions over $\mathbb{Z}_2$ for all i's.*

Given that for any fixed τ on S^τ, $tamp(z_i) \in \{0, 1, z_i, z_i + 1\}$, $tamp(r_i) \in \{0, 1, r_i, r_i + 1\}$, we can set $mult_i = tamp(z_i) \cdot tamp(r_i)$. The above Proposition states that for the fixed tampering τ, the output value of the multiplication gate m_i can be described as a linear function of r_i.

Proposition 7 (Output of the one layer compression gagdet) *Let p_m be the output value of the gadget $\triangleright^\tau$ from the construction of S_d^τ which takes as input values $z_{md+1}, z_{md+2}, ..., z_{md+d}, r_1, r_2, ..., r_d$. Then $p_m = \beta(z_{md+1}, ..., z_{md+d}) + \sum_{i=1,...,d} \alpha_i(z_{md+i})r_i$, where α_i and β_i are linear (multilinear) functions over $\mathbb{Z}_2$.*

Given the Propositions 4, 5, 6, in Proposition 7 we can conclude on the output values of $S_{m'd,d}$ given values $z_1, ..., z_{m'd}; r_1, ..., r_d$ as input in the above statement.

6.5 Information Loss Survival for S_d

Now, we will prove that information loss survives a single-layer computation S_d with probability at least $\frac{1}{2}$. Since the full compression gadget $G_{n,\lambda,d}$ is built using λ layers of S_d gadgets, this result will lead us to the final conclusion, that $G_{n,\lambda,d}$ compresses the size of the output and propagates the information loss to the output with a probability at least $1/2^{\lambda}$.

We are given an information-losing tuple $(X_1, ..., X_z; X_1^\tau, ..., X_z^\tau)$ which represents z different (untampered and tampered) evaluation vectors of input wires to the single layer compressing gadget S_d. Given z uniformly random pairs of randomness vectors R_i, Q_i each of the evaluation vectors X_i will be used twice as the input to the gadget S_d. This will suffice to propagate the information loss to the output of the gadget with good probability.

Theorem 2 (Information loss through one layer). *Let $(X_1, ..., X_z; X_1^\tau, ..., X_z^\tau)$ be an information-losing tuple. Let R_i, Q_i for $i = 1, ..., z$ be vectors in $\mathbb{Z}_2^d$ chosen independently and uniformly at random. Let*

$$Y_i = S_d(X_i|R_i),\ Y_{i+z} = S_d(X_i|Q_i),\ Y_i^\tau = S_d^\tau(X_i^\tau|R_i),\ Y_{i+z}^\tau = S_d^\tau(X_i^\tau|Q_i),$$

for $i = 1, ..., z$. Then $(Y_1, ..., Y_{2z}; Y_1^\tau, ..., Y_{2z}^\tau)$ is an information-losing tuple with probability at least $\frac{1}{2}$.

Proof. Let (i, j, k) be a information-loss witness for $(X_1, ..., X_z; X_1^\tau, ..., X_z^\tau)$. Then

$$(X_i[k] \neq X_j[k]) \wedge \left(X_i^\tau[k] = X_j^\tau[k]\right). \tag{4}$$

Denote the input to S_d by $U = (x_1, ..., x_d, r_1, ..., r_d)$. Let O_s be the s'th output wire of S which is possibly affected by the value of x_k. Obviously $s = \lceil \frac{k}{d} \rceil$, and $k = sd + k'$ where $k' \in [d]$. Then the value of the selected output wire in the untampered S_d is:

$$\mathrm{val}_U(O_s) = \sum_{t=1,...,d} x_{sd+t} r_t = \sum_{t=1,...,d} \gamma_t(x_{sd+t}) r_t, \tag{5}$$

where γ_t is the identity function. From Proposition 7 we know, that the tampered value of the selected output wire can be described with the following expression:

$$\mathrm{val}_U^\tau(O_s) = \sum_{t=1,...,d} \alpha_t\left(x_{sd+t}\right) r_t + \beta_t\left(x_{sd+t}\right), \tag{6}$$

where α_i and β_i are linear (multilinear) functions over $\mathbb{Z}_2$.

Now we can instantiate $(x_1, ..., x_z)$ four times, with $X_i, X_j, X_i' X_j'$. In every such case $\alpha_t(\cdot), \beta_t(\cdot), \gamma_t(\cdot)$ are evaluated to some elements of $\mathbb{Z}_2$. Let us denote these elements with $\beta_t^{X_i'} = \beta_t(X_i'[sd + t])$. Since (i, j, k) is the witness of information loss we know, that $\gamma_{k'}^{X_i} \neq \gamma_{k'}^{X_j}$, $\alpha_{k'}^{X_i'} = \alpha_{k'}^{X_j'}$. WLOG let $\gamma_{k'}^{X_i} \neq \alpha_{k'}^{X_i'}$.

Moreover, the evaluations 5, 6 are simply linear/affine combinations of $r_1, ...r_d$ over $\mathbb{Z}_2$, respectively. Consider,

$$\mathrm{DIFF_i}(r_1, ..., r_t) := \mathrm{val}_{X_i|r_1,...,r_d}(O_s) - \mathrm{val}^{\tau}_{X'_i|r_1,...,r_d}(O_s) = r_{k'} + \sum_{t=1,...,d;t\neq t'} \delta_t r_t + \epsilon_i, \tag{7}$$

for some $\epsilon_i, \delta_t \in \mathbb{Z}_2$.

Now let instantiate $(r_1, ..., r_d)$ with uniform random variable R over $\mathbb{Z}_2^d$. Firstly, observe that $\Pr[\mathrm{DIFF_i}(R) = 1] = \frac{1}{2}$ which means, that for the pair X_i, X'_i for exactly half of the choices of R there will occur an error on the O_s - i.e. the expected and actual values will differ. Since $\mathrm{DIFF_j}(r_1, ..., r_t) = \mathrm{val}_{X_j|r_1,...,r_d}(O_s) - \mathrm{val}^{\tau}_{X'_j|r_1,...,r_d}(O_s) = \sum_{t=1,...,d} \kappa_t r_t + \epsilon_j$, for some $\epsilon_j, \kappa_t \in \mathbb{Z}_2$, we know that $\Pr[\mathrm{DIFF_j}(R) = 1] \in \left\{0, \frac{1}{2}, 1\right\}$. Thus for the pair X_j, X'_j there is an error never or always or for half of the choices of R. Finally,

$$\left\{\frac{1}{2}\right\} \subseteq \left\{\Pr[\mathrm{val}_{X_i|R}(O_s) = 0], \Pr[\mathrm{val}_{X_j|R}(O_s) = 0]\right\} \subseteq \left\{0, \frac{1}{2}\right\}. \tag{8}$$

The first inclusion is true since $1 \in \{\gamma^{X_i}_{k'}, \gamma^{X_j}_{k'}\}$. The second inclusion is true, since $\mathrm{val}_{X_i|R}(O_s), \mathrm{val}_{X_j|R}(O_s)$ are linear combinations of r_t's.

Let us denote $x_1(R) = S(X_i|R)[k']$, $x_2(R) = S(X_j|R)[k']$, $y_1(R) = S^{\tau}(X^{\tau}_i|R)[k']$, $y_2(R) = S^{\tau}(X^{\tau}_j|R)[k']$. Informally speaking, for independent and uniformly random R_i, R_j, Q_i, Q_j the tuple $\mathcal{V}$ below
$\mathcal{V} = (x_1(R_i), x_1(Q_i), x_2(R_j), x_2(Q_j); y_1(R_i), y_1(Q_i), y_2(R_j), y_2(Q_j))$ contains a tampered evaluation y_1 that has an error with probability $1/2$ with respect to its correct evaluation x_1 and at least one evaluation x_1 or x_2 that is correctly evaluated to 1. Now, we can use Lemma 3 below to prove the above informal statement and say that given uniformly random R_i, R_j, Q_i, Q_j, the tuple $\mathcal{V}$ is an information-losing tuple with a probability of at least $1/2$.

Thus we conclude that $(Y_1, ..., Y_{2z}; Y^{\tau}_1, ..., Y^{\tau}_{2z})$ is an information-losing tuple with at least one of $(i, j, k'), (i, j+z, k'), (i+z, j, k'), (i+z, j+z, k')$ as a witness with probability at least $1/2$. □

Finally, we formulate the Lemma which lets us conclude that the information loss survives a single layer of compression S_d with probability at least $1/2$.

Lemma 3. *Let x_1, x_2, y_1, y_2 be functions from $\mathbb{Z}_2^d$ to $\mathbb{Z}_2$, and let R be a random variable over $\mathbb{Z}_2^d$, such that:*

$$\Pr[x_1(R) = y_1(R)] = \frac{1}{2}, \tag{9}$$

$$\Pr[x_2(R) = y_2(R)] \in \left\{0, \frac{1}{2}, 1\right\}, \tag{10}$$

$$\left\{\frac{1}{2}\right\} \subseteq \{\Pr[x_1(R) = 0], \Pr[x_2(R) = 0]\} \subseteq \left\{\frac{1}{2}, 1\right\}, \tag{11}$$

Then for independent and uniformly random R_1, R_2, Q_1, Q_2 the tuple

$$(x_1(R_1), x_1(Q_1), x_2(R_2), x_2(Q_2); y_1(R_1), y_1(Q_1), y_2(R_2), y_2(Q_2))$$

is information losing with probability $\geq \frac{1}{2}$.

Proof. We want to show that for any constraints with probability $\geq \frac{1}{2}$ the tuple $(x_1(R_1), x_1(Q_1), x_2(R_2), x_2(Q_2); y_1(R_1), y_1(Q_1), y_2(R_2), y_2(Q_2))$ has some two $x's$ different and corresponding $y's$ being equal. Consider any four tuples (a, a'), (b, b'), (c, c') and (d, d') where $a, b, c, d, a', b', c', d' \in \{0, 1\}$. First, we claim that $(a, b, c, d; a', b', c', d')$ forms an information-losing tuple if and only if none of the following is true: (1) $a = b = c = d$, (2) $(a = a') \wedge (b = b') \wedge (c = c') \wedge (d = d')$, (3) $(a = 1 - a') \wedge (b = 1 - b') \wedge (c = 1 - c') \wedge (d = 1 - d')$.

Clearly, if any of the above conditions is true then $(a, b, c, d; a', b', c', d')$ is not information-losing. For the reverse, assume none of the conditions is true. WLOG let $a = 1, b = 0$. WLOG we have two cases $c = 1, d = 1$ or $c = 1, d = 0$. In the first case, if $b' = 0$, then at least one of a', c', d' must be 0, otherwise, condition 2 above would hold. WLOG let $a' = 0$. Thus $a \neq b$ but $a' = b'$ and we get information loss. Similarly, if $b' = 1$ using condition 3, we will find information loss. In the second case of $a = 1, b = 0, c = 1, d = 0$, if $a' \neq c'$, then $b' = a'$ or c' hence information loss. If $a' = c'$, then we get information loss if either b' or d' is equal to a'. The only way for neither b' nor d' to be equal to a' is for one of conditions 2 or 3 to hold true, but this would be a contradiction.

Thus we define three events corresponding to the three conditions above:

E_1: $x_1(R_1) = x_1(Q_1) = x_2(R_2) = x_2(Q_2)$
E_2: $(x_1(R_1) = y_1(R_1)) \wedge (x_1(Q_1) = y_1(Q_1)) \wedge (x_2(R_2) = y_2(R_2)) \wedge (x_2(Q_2) = y_2(Q_2))$
E_3: $(x_1(R_1) = 1 - y_1(R_1)) \wedge (x_1(Q_1) = 1 - y_1(Q_1)) \wedge (x_2(R_2) = 1 - y_2(R_2)) \wedge (x_2(Q_2) = 1 - y_2(Q_2))$

Thus given a tuple of evaluations

$$\mathcal{V} = (x_1(R_1), x_1(Q_1), x_2(R_2), x_2(Q_2); y_1(R_1), y_1(Q_1), y_2(R_2), y_2(Q_2)),$$

we get that $\Pr[\mathcal{V} \text{ is information losing}] = 1 - \Pr[E_1 \vee E_2 \vee E_3]$.

We will bound $\Pr[E_1 \vee E_2 \vee E_3] \leq \frac{1}{2}$, thus proving the desired result. For this we first use union bound to get $\Pr[E_1 \vee E_2 \vee E_3] \leq \Pr[E_1] + \Pr[E_2 \vee E_3]$. Now for $\Pr[E_1]$, we have that at either $\Pr[x_1 = 0] = \frac{1}{2}$ or $\Pr[x_2 = 0] = \frac{1}{2}$ (eq. 11). Thus $\Pr[x_1(R_1) = x_1(Q_1) = x_2(R_2) = x_2(Q_2)] \leq \frac{1}{4}$.

For $\Pr[E_2 \vee E_3] \leq \Pr[E_2] + \Pr[E_3]$. We have three cases by eq. 10:

1. $\Pr[x_2(R) = y_2(R)] = 0$: In this case $\Pr[E_2] = 0$. Additionally using eq. 9 we get that $\Pr[E_3] = \frac{1}{4}$. Hence, $\Pr[E_2 \vee E_3] = \frac{1}{4}$
2. $\Pr[x_2(R) = y_2(R)] = 1$: In this case $\Pr[E_3] = 0$. Additionally using eq. 9 we get that $\Pr[E_2] = \frac{1}{4}$. Hence, $\Pr[E_2 \vee E_3] = \frac{1}{4}$
3. $\Pr[x_2(R) = y_2(R)] = \frac{1}{2}$: Additionally using eq. 9 we get $\Pr[E_2] = \Pr[E_3] = \frac{1}{16}$. Hence, $\Pr[E_2 \vee E_3] \leq \frac{1}{8}$

We get $\Pr[E_1 \vee E_2 \vee E_3] \leq \frac{1}{2}$, and the probability of inf. loss at least $\frac{1}{2}$. ■

7 The Compiler

Finally, building upon results from the previous sections, we define a compiler that compiles any circuit $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ into another functionally equivalent circuit $C_{\mathsf{test},\lambda} : \mathbb{Z}_2^{s+s'} \to \mathbb{Z}_2^{t+t'}$ such that for any non-trivial tampering of the circuit $C_{\mathsf{test},\lambda}$, running the testing procedure on the tampered $C_{\mathsf{test},\lambda}$, one always detects an error with high probability.

Algorithm 3: The Compiler

Input: $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t, \lambda$
Output: $C_{\mathsf{test},\lambda}$

1 Compile circuit C into a gate-covered $C_{\mathsf{gate}} : \mathbb{Z}_2^{s+s_g} \to \mathbb{Z}_2^t, \mathbb{T}_{\mathsf{gate}}$, by running Algorithm 1 on it
2 Add the COPY gates that route the information loss in the gate-covered circuit to the testing gadget, by running Algorithm 2 on the pair $C_{\mathsf{gate}}, \mathbb{T}_{\mathsf{gate}}$. This procedure gives a circuit with additional t_0 output bits - $C_{\mathsf{test},0} : \mathbb{Z}_2^{s+s_g} \to \mathbb{Z}_2^{t+t_0}$ along with a test set $\mathbb{T}_0$
3 Append the $G_{n,\lambda,d}$ gadget to the t_0 wires added in the previous step, where $d = \lceil t_0^{\frac{1}{\lambda+1}} \rceil$. This step adds s_λ wires to the input of the circuit, but replaces the t_0 output bits created in the previous step with λ new output bits, producing a circuit $C_{\mathsf{test},\lambda} : \mathbb{Z}_2^{s+s_g+s_\lambda} \to \mathbb{Z}_2^{t+t_\lambda}$
4 **return** $C_{\mathsf{test},\lambda}$

Theorem 3 (Testing Probability of Final Circuit). *On input circuit* $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ *along with parameter* λ*, Algorithm 3 outputs a circuit* $C_{\mathsf{test},\lambda} : \mathbb{Z}_2^{s+s_g+s_\lambda} \to \mathbb{Z}_2^{t+t_\lambda}$ *such that for any tampering* τ *of* $C_{\mathsf{test},\lambda}$ *if*

$$\exists X \in \mathbb{Z}_2^s : C^\tau_{\mathsf{test},\lambda}(X||0^{s_g+s_\lambda}) \neq C_{\mathsf{test},\lambda}(X||0^{s_g+s_\lambda})$$

then when observing behaviour of the circuit $C_{\mathsf{test},\lambda}$ *on its test set* $\mathbb{T}_{\mathsf{test}}$*,*

- ***Either the output is wrong:***

$$\exists X \in \mathbb{T}_{\mathsf{test}} : C^\tau_{\mathsf{test},\lambda}(X||0^{s_\lambda}) \neq C_{\mathsf{test},\lambda}(X||0^{s_\lambda}),$$

- ***or the testing gadget detects an inconsistency:***

$$\exists X \in \mathbb{T}_{\mathsf{test}} : \Pr_{R \leftarrow \mathbb{Z}_2^{s_\lambda}} \left[C^\tau_{\mathsf{test},\lambda}(X||R) \neq C_{\mathsf{test},\lambda}(X||R) \right] \geq \frac{1}{2^{2\lambda}}.$$

Proof. By the Proposition 3 we know that either we observe an information loss on the first t bits of the intermediary circuit $C_{\mathsf{test},0}$ or its output is toggled, or we observe information loss on t_0 wires added during Step 2 of the Algorithm 3, or the output is always correct. Any error on the first t bits of the circuit will

be detected on at least of query from ($\mathbb{T}_0 \subseteq \mathbb{T}_{\mathsf{test}}$ is the gate-covering set of the $C_{\mathsf{test},0}$ (Propositions 1 and 3)). Next, when we append the gadget $G_{n,\lambda,d}$ to the remaining t_0 wires of the construction. By Theorem 2 we know that the information loss on these wires survives with probability $1/2$ in each layer when queried with fresh randomness twice. Hence the information loss would survive with probability $1/2^{\lambda}$ if we query with two fresh randomness vectors in each layer. Thus if we query with only one random string the probability of information loss surviving and hence the error showing up on the output is $\frac{1}{2^{\lambda}}/2^{\lambda} = 1/2^{2\lambda}$. ■

Testing Procedure Given any circuit $C^{\tau}_{\mathsf{test},\lambda}$ with any tampering τ on its wires we test it by querying it on all the test inputs in $\mathbb{T}_{\mathsf{test}}$ along with uniform random $R \in \mathbb{Z}_2^{s_\lambda}$. We can repeat the testing procedure κ times with fresh randomness to get the probability of catching an error $1 - (1 - 1/2^{2\lambda})^{\kappa}$

Circuit Parameters For any circuit $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ with n gates, using the Algorithm 3 with parameter λ. The first step of the Algorithm produces a gate-covered circuit C_{gate} with 5 new input bits and a test set of size 6 and creates a circuit of size $\approx 7n$ gates (see Proposition 2). The second step of the algorithm adds XOR and COPY gate to every nonlinear gate of the circuit, adding $\approx 2 \cdot 4n$ gates and roughly $\approx 4n$ output wires (in the previous estimation at least $3n$ out of $7n$ gates are the COPY gates). The third step of the algorithm replaces the $4n$ intermediary wires with $\approx L \cdot \sqrt[\lambda+1]{4n}$ input bits and $\approx \sqrt[\lambda+1]{4n}$ output bits.

8 Conclusions and Open Problems

In this work, we construct an efficiently testable circuit compiler that detects tampering on all wires and does not assume conductivity, solving one of the two open problems put forward in [3] (the other being a construction that can handle tampering of all *gates*). Unlike in [3], our testing procedure is randomized, and it's an interesting open question whether this is inherent. We can "derandomize" our construction by using $\lambda = \log(n)$ layers and then making test queries for all $2^{2\lambda} = n^2$ possible choices of the randomness. The number of test queries will be quadratic in the size of the circuit, which is not practical.

We hope that more applications, besides testing and security against tampering, will be found in the future as was the case for non-malleable codes [18] (like ETC, non-malleable codes originally also aimed at preventing tampering, but only on static memory). E.g., an arithmetic version of our ETC could have applications to multiparty computation (as it would strengthen the additive tampering notion used in [23]), or to the construction of succinct proofs systems, where starting from an ETC rather than a general (arithmetic) circuit could give some security benefits, like avoiding a trusted setup, and instead just checking whether the setup works on all the values in the testset.

References

[1] D. Aggarwal, Y. Dodis, and S. Lovett. "Non-malleable codes from additive combinatorics". In: *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*. Ed. by D. B. Shmoys. ACM, 2014, pp. 774–783. DOI: 10.1145/2591796.2591804.

[2] G. Ateniese, A. Kiayias, B. Magri, Y. Tselekounis, and D. Venturi. "Secure Outsourcing of Cryptographic Circuits Manufacturing". In: *ProvSec*. Ed. by J. Baek, W. Susilo, and J. Kim. Vol. 11192. Lecture Notes in Computer Science. Springer, 2018, pp. 75–93. DOI: 10.1007/978-3-030-01446-9_5.

[3] M. A. Baig, S. Chakraborty, S. Dziembowski, M. Gałązka, T. Lizurej, and K. Pietrzak. "Efficiently Testable Circuits". In: *ITCS - Innovations in Theoretical Computer Science*. 2023.

[4] M. Ball, D. Dachman-Soled, S. Guo, T. Malkin, and L. Tan. "Non-Malleable Codes for Small-Depth Circuits". In: *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*. Ed. by M. Thorup. IEEE Computer Society, 2018, pp. 826–837. DOI: 10.1109/FOCS.2018.00083.

[5] F. Berti, C. Guo, T. Peters, and F. Standaert. "Efficient Leakage-Resilient MACs Without Idealized Assumptions". In: *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part II*. Ed. by M. Tibouchi and H. Wang. Vol. 13091. Lecture Notes in Computer Science. Springer, 2021, pp. 95–123. DOI: 10.1007/978-3-030-92075-3_4.

[6] S. Bhunia and M Tehranipoor. "The Hardware Trojan War". In: *Cham„ Switzerland: Springer* (2018).

[7] E. Biham and A. Shamir. "Differential Fault Analysis of Secret Key Cryptosystems". In: *CRYPTO*. Vol. 1294. Lecture Notes in Computer Science. Springer, 1997, pp. 513–525.

[8] D. Boneh, R. A. DeMillo, and R. J. Lipton. "On the Importance of Eliminating Errors in Cryptographic Computations". In: *J. Cryptol.* 14.2 (2001), pp. 101–119.

[9] E. Boyle, G. Segev, and D. Wichs. "Fully Leakage-Resilient Signatures". In: *J. Cryptol.* 26.3 (2013), pp. 513–558. DOI: 10.1007/s00145-012-9136-3.

[10] M. Bushnell and V. Agrawal. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. Vol. 17. Springer Science & Business Media, 2004.

[11] S. Chakraborty, S. Dziembowski, M. Gałązka, T. Lizurej, K. Pietrzak, and M. Yeo. "Trojan-Resilience Without Cryptography". In: *Theory of Cryptography*. Ed. by K. Nissim and B. Waters. Cham: Springer International Publishing, 2021, pp. 397–428. ISBN: 978-3-030-90453-1.

[12] E. Chattopadhyay and X. Li. "Non-malleable codes and extractors for small-depth circuits, and affine functions". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017,*

Montreal, QC, Canada, June 19-23, 2017. Ed. by H. Hatami, P. McKenzie, and V. King. ACM, 2017, pp. 1171–1184. DOI: 10.1145/3055399.3055483.

[13] D. Dachman-Soled and Y. T. Kalai. "Securing Circuits against Constant-Rate Tampering". In: *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. Ed. by R. Safavi-Naini and R. Canetti. Vol. 7417. Lecture Notes in Computer Science. Springer, 2012, pp. 533–551. DOI: 10.1007/978-3-642-32009-5_31.

[14] D. Dachman-Soled and Y. T. Kalai. "Securing Circuits and Protocols against 1/poly(k) Tampering Rate". In: *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*. Ed. by Y. Lindell. Vol. 8349. Lecture Notes in Computer Science. Springer, 2014, pp. 540–565. DOI: 10.1007/978-3-642-54242-8_23.

[15] D. Dachman-Soled, F. Liu, E. Shi, and H. Zhou. "Locally Decodable and Updatable Non-malleable Codes and Their Applications". In: *J. Cryptol.* 33.1 (2020), pp. 319–355. DOI: 10.1007/s00145-018-9306-z.

[16] S. Dziembowski, S. Faust, and F. Standaert. "Private Circuits III: Hardware Trojan-Resilience via Testing Amplification". In: *ACM CCS*. Ed. by E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi. ACM, 2016, pp. 142–153. DOI: 10.1145/2976749.2978419.

[17] S. Dziembowski and K. Pietrzak. "Leakage-Resilient Cryptography". In: *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*. IEEE Computer Society, 2008, pp. 293–302. DOI: 10.1109/FOCS.2008.56.

[18] S. Dziembowski, K. Pietrzak, and D. Wichs. "Non-Malleable Codes". In: *J. ACM* 65.4 (2018), 20:1–20:32. DOI: 10.1145/3178432.

[19] K. Efremenko et al. "Circuits Resilient to Short-Circuit Errors". In: *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2022. Rome, Italy: Association for Computing Machinery, 2022, 582–594. ISBN: 9781450392648. DOI: 10.1145/3519935.3520007.

[20] S. Faust, E. Kiltz, K. Pietrzak, and G. N. Rothblum. "Leakage-Resilient Signatures". In: *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*. Ed. by D. Micciancio. Vol. 5978. Lecture Notes in Computer Science. Springer, 2010, pp. 343–360. DOI: 10.1007/978-3-642-11799-2_21.

[21] S. Faust, P. Mukherjee, J. B. Nielsen, and D. Venturi. "A Tamper and Leakage Resilient von Neumann Architecture". In: *Public-Key Cryptography – PKC 2015*. Ed. by J. Katz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 579–603. ISBN: 978-3-662-46447-2.

[22] S. Faust, K. Pietrzak, and D. Venturi. "Tamper-Proof Circuits: How to Trade Leakage for Tamper-Resilience". In: 2011, pp. 391–402. DOI: 10.1007/978-3-642-22006-7_33.

[23] D. Genkin, Y. Ishai, M. M. Prabhakaran, A. Sahai, and E. Tromer. "Circuits Resilient to Additive Attacks with Applications to Secure Computa-

tion". In: *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*. STOC '14. New York, New York: Association for Computing Machinery, 2014, 495–504. ISBN: 9781450327107. DOI: 10.1145/2591796.2591861.

[24] Y. Ishai, M. Prabhakaran, A. Sahai, and D. A. Wagner. "Private Circuits II: Keeping Secrets in Tamperable Circuits". In: *EUROCRYPT*. Ed. by S. Vaudenay. Vol. 4004. Lecture Notes in Computer Science. Springer, 2006, pp. 308–327. DOI: 10.1007/11761679_19.

[25] Y. Ishai, A. Sahai, and D. Wagner. "Private circuits: Securing hardware against probing attacks". In: *Annual International Cryptology Conference*. Springer. 2003, pp. 463–481.

[26] Y. T. Kalai, A. B. Lewko, and A. Rao. "Formulas Resilient to Short-Circuit Errors". In: *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*. IEEE Computer Society, 2012, pp. 490–499. DOI: 10.1109/FOCS.2012.69.

[27] Y. T. Kalai and L. Reyzin. "A survey of leakage-resilient cryptography". In: *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. Ed. by O. Goldreich. ACM, 2019, pp. 727–794. DOI: 10.1145/3335741.3335768.

[28] A. Kiayias and Y. Tselekounis. "Tamper Resilient Circuits: The Adversary at the Gates". In: *Advances in Cryptology - ASIACRYPT 2013*. Ed. by K. Sako and P. Sarkar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 161–180. ISBN: 978-3-642-42045-0.

[29] X. Li. "Improved non-malleable extractors, non-malleable codes and independent source extractors". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. Ed. by H. Hatami, P. McKenzie, and V. King. ACM, 2017, pp. 1144–1156. DOI: 10.1145/3055399.3055486.

[30] S. Micali and L. Reyzin. "Physically Observable Cryptography". In: *Theory of Cryptography*. Ed. by M. Naor. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 278–296. ISBN: 978-3-540-24638-1.

[31] R. S. Wahby, M. Howald, S. Garg, A. Shelat, and M. Walfish. "Verifiable ASICs". In: *IEEE SP*. IEEE Computer Society, 2016, pp. 759–778. DOI: 10.1109/SP.2016.51.