Efficiently Testable Circuits

Mirza Ahad Baig[1], Suvradip Chakraborty[2] Stefan Dziembowski[3], Małgorzata Gałązka[4], Tomasz Lizurej[3], and Krzysztof Pietrzak[1]

[1] ISTA
[2] ETH Zurich
[3] University of Warsaw and IDEAS NCBR
[4] University of Warsaw

Abstract. In this work, we put forward the notion of "efficiently testable circuits" and provide circuit compilers that transform any circuit into an efficiently testable one. Informally, a circuit is testable if one can detect tampering with the circuit by evaluating it on a small number of inputs from some test set.

Our technical contribution is a compiler that transforms any circuit C into a testable circuit $(\widehat{C}, \widehat{\mathbb{T}})$ for which we can detect arbitrary tampering with *all* wires in $\widehat{C}$. The notion of a testable circuit is weaker or incomparable to existing notions of tamper-resilience, which aim to detect or even correct for errors introduced by tampering during every query, but our new notion is interesting in several settings, and we achieve security against much more general tampering classes – like tampering with all wires – with very modest overhead.

Concretely, starting from a circuit C of size n and depth d, for any L (think of L as a small constant, say $L = 4$), we get a testable $(\widehat{C}, \widehat{\mathbb{T}})$ where $\widehat{C}$ is of size $\approx 12n$ and depth $d + \log(n) + L \cdot n^{1/L}$. The test set $\widehat{\mathbb{T}}$ is of size $4 \cdot 2^L$. The number of extra input and output wires (i.e., pins) we need to add for the testing is $3 + L$ and 2^L, respectively.

1 Introduction

A circuit compiler is a mapping that takes as input (the description of) a circuit C and outputs (the description of) a compiled circuit C'. Many such compilers have been proposed, where the compiler's task is to output a circuit that is *functionally equivalent* to the input circuit but whose physical implementation will provide some useful *additional security properties*. Here "functionally equivalent" means C' computes the same function as C but some compilers require additional input and/or output wires or require the inputs and outputs to be encoded in a special way. An influential line of research in this domain are *private circuits* [13, 12, 7], which aim to hide secrets like cryptographic keys in the

[§]Suvradip Chakraborty was supported in part by ERC grant 724307; Stefan Dziembowski was supported by ERC Grant 885666 and NCN Grant 2019/35/B/ST6/04138; Tomasz Lizurej and Małgorzata Gałązka were supported by NCN Grant 2019/35/B/ST6/04138

compiled circuit in the presence of physical attacks like leakage, tampering, or hardware trojans. We will discuss these in more detail in Section 1.3.

In this work, we consider circuits' integrity rather than secrecy. While private circuits address the setting where an adversary gets access to a device (like a smart card) doing cryptographic computations and aims at extracting the key through leakage, tampering, or by outright replacing the circuit (i.e., hardware trojans). We consider a setting where an adversary can attack the circuit before it is delivered to the honest parties and want the honest parties to be able to efficiently detect whether such tampering took place.

Integrity vs. Secrecy. Detecting tampering is arguably an easier task than protecting secrets. In fact, the approach to safeguard secrets against tampering in "private circuits II" [12] and against trojans in "private circuits III" [7] is to detect integrity violations that could lead to information leakage and self-destruct the circuit once such a violation is detected. These integrity checks must happen continuously whenever the circuit is evaluated, as a single query to a tampered circuit can leak the secret.

In this work, we consider a more benign setting where an adversary can initially tamper with a circuit, which is then delivered to the user. The user can then test the circuit to certify its integrity. This is conceptually similar to traditional testing in circuit manufacturing (see, e.g., [3]). However, while there the goal is the detection of random errors that creep in during manufacturing, we consider extremely powerful adversarial tampering attacks where the circuit (before compilation) can be adversarially designed. The tampering attack can affect the entire (compiled) circuit as long as its topology remains unchanged.

Our notion is motivated by security aspects of cryptographic circuits (which hide a secret) as well as non-cryptographic circuits (where we just care about integrity). As a cryptographic example consider an adversary who tries to tamper with a smart-card used for signing to make it output the secret key if queried on a random message only known to the adversary. Such tampering is not detectable through normal testing, but once the cards are deployed the adversary can extract secret keys through a simple signing query.

As an example of a non-cryptographic circuit consider a chip used in a router which is tampered so it can be triggered with an innocent looking package to route all "interesting" data to some particular server controlled by the adversary. Our compiler could also be interesting in a context where we only consider benign errors, but want to be able to detect those efficiently with 100% certainty. A great example are chips used in space exploration, here integrity is often crucial, while the high radiation takes its toll on circuit components.

1.1 Efficiently Testable Circuits Compiler

We will construct a compiler that maps any circuit $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ into an efficiently testable one, which is a tuple $(\widehat{C}, \widehat{\mathbb{T}})$ where $\widehat{C} : \mathbb{Z}_2^{s+s'} \to \mathbb{Z}_2^{t+t'}$ is a circuit that is functionally equivalent to C when setting the s' extra inputs to 0 and ignoring

the t' extra outputs (the $|t$ subscript indicates that only the first t bits of the output are considered)

$$\forall X \in \mathbb{Z}_2^s \ : \ \widehat{C}_{|t}(X\|0^{s'}) = C(X)$$

and $\mathbb{T} \subseteq \mathbb{Z}_2^{s+s'}$ is a small test set. $\widehat{C}$ is efficiently testable in the sense that when applying a tampering τ to $\widehat{C}$ such that this tampered circuit $\widehat{C}^\tau$ errs on at least one input, there is already an input in the small test set which will certify this

$$\left(\exists X \in \mathbb{Z}_2^s \ : \widehat{C}^\tau_{|t}(X\|0^{s'}) \neq \underbrace{\widehat{C}_{|t}(X\|0^{s'})}_{=C(X)} \right) \ \Rightarrow \left(\exists X \in \mathbb{T} \ : \widehat{C}^\tau(X) \neq \widehat{C}(X). \right)$$

The Tampering Model. The class of tamperings we consider allows for tampering *every* wire in the circuit in an *arbitrary* way. Concretely, the tampering τ specifies for every wire one of four possible actions: do nothing, flip the value or set it to constant 0 or 1. There is one restriction, the tampering is "conductive": if a gate output wire has fan-out > 1, i.e., it is used as input to more than one gate (where a final output is considered a gate), then the adversary can only choose one tampering function which will affect all those wires identically. We will refer to this as the *conductivity assumption*. This assumption has been used in previous works on tamper-resilient circuit compilers including Private Circuits II [12].

While one can turn any circuit into a functionally equivalent one where all gates have fan-out 1 by using COPY gates (cf. Figure 6), this transformation will, unfortunately, not preserve the testability of the circuit, so we have to consider the requried fan-out as a parameter. We refer to a circuit where no wire has fan-out $\geq k$ as k-conductive, and the testable circuit created by our compiler will be 3-conductive.

1.2 Our Technical Contribution

In this section, we will sketch the main ideas behind our compiler using the toy circuit $C(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$ as illustrated in Figure 1.(A) as running example.

Preprocessing - Circuits with Covering Sets. Before we can apply our compiler, we perform some minor *preprocessing* of the circuit. For one thing, we need the preprocessed circuit to be non-conductive (i.e., each gate has fan-out 1) Moreover our compiler will require that the input circuit $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ comes with a *covering set* of inputs $\mathbb{T} \subseteq \mathbb{Z}_2^s$, which is a set of inputs such that for every wire w in C and every $b \in \{0, 1\}$, there is an input $X \in \mathbb{T}$ such that in the evaluation $C(X)$ the wire w carries value b. For our toy circuit $\mathbb{T} = \{0000, 1111\}$ is a covering set (all wires are 0 on input 0000 and all wires are 1 on input 1111). In general, a covering set might not exist or be hard to find (it is easily seen to be NP-complete). We show how to derive from any circuit a functionally equivalent

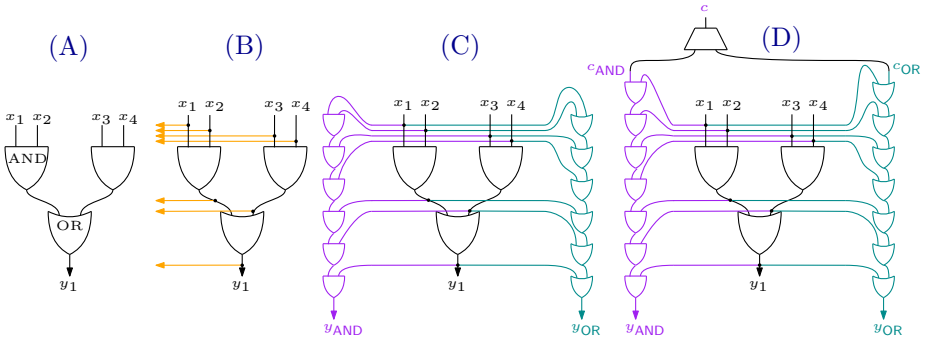

Fig. 1. Illustration of our toy circuit C (A) and its compiled version $\widehat{C}$ (D). (B) is a trivial solution with large output while (C) has only two extra output bits but is only secure assuming no tampering with the chained OR/AND gadgets as explained in the text.

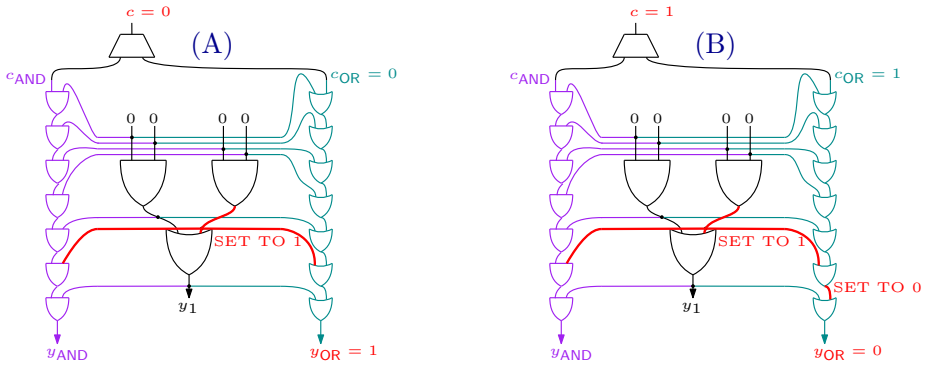

Fig. 2. If in the core circuit some wire is tampered changing 0 to 1 (either flipping the wire or setting to constant 1) this will be detected by the chained OR (1 to 0 by the chained AND) gadget. Trying to tamper with the gadget (on the right one wire is set to 0 trying to "fix" the y_{OR} output) itself will not prevent detection as now the control output y_{OR} will not change when we flip the control input c_{OR}, so the y_{OR} output on either 00000 or 00001 will be wrong.

one together with a covering set of size 4. This requires adding 3 extra input wires and some XOR gates. In the worst case the size of the circuit doubles, but typically it will increase by much less.

A Trivial (but Impractical) Solution. Given a non-conductive circuit C' with covering set $\mathbb{T}$ (as produced by the preprocessing) we can get a testable circuit $(\widehat{C}, \widehat{\mathbb{T}})$ by using the covering set as the test set, i.e., $\widehat{\mathbb{T}} = \mathbb{T}$, and letting $\widehat{C}$ be C' with an extra output wire for every wire in C' as shown in Figure 1.(B). To see why $(\widehat{C}, \widehat{\mathbb{T}})$ is testable consider any wire w which was tampered. On some input $X \in \widehat{\mathbb{T}}$ this wire will take the wrong value, and this value can be directly observed on the outputs.

There is a subtle flaw in the above argument. The tampering on the particular wire w on input X could be "undone" by some tampering higher up in the circuit. To fix the argument it is sufficient to let w be a *topologically first* tampering, meaning no wire on a path from an input to w is tampered. Let us also stress that $\widehat{C}$ is 2-conductive as each wire is additionally used as output. We can't simply use COPY gates to make the circuit non-conductive as then the observed outputs and values on the internal wires could be tampered with individually, and the above argument breaks down. Thus already this impractical construction highlights the importance of the conductivity assumption.

Assuming Non-Tamperable Test Gates. The reason the solution just outlined is not practical is the huge number of output wires. This makes checking whether the test queries are correct expensive. More importantly, while circuits can have billions of gates, the number of pins (input/output wires) is much more limited for technical reasons. Assume we could create a compiler by adding gates and wires to the circuit which the adversary is not allowed to tamper with. Then there is a simple solution using the same test set $\widehat{\mathbb{T}} = \mathbb{T}$ as the trivial solution just outlined, but adding just $2|\mathbb{T}|$ output wires: For every $X \in \mathbb{T}$, consider all wires which should be 1 on this input, and output the AND of all these bits. Similarly, output the OR of all bits set to 0.

Recall that for our toy circuit $\mathbb{C} = \{0000, 1111\}$. On input 0000 all wires are 0 and thus we just need to compute one big OR whose output is denoted y_{OR} in Figure 1.(C). Similarly, on the other input 1111 all wires are 1 and thus we just need one big AND gadget. For reasons that will become clear later, our gadget applies the AND/OR gates sequentially, and not in a tree fashion, which would result in a much lower depth circuit.

For our toy circuit, this just increased the number of outputs by 2, in general, it will increase the number of outputs by at most 8 ($\leq$ 2 outputs for each of the $\leq$ 4 test queries). The compiled toy circuit is 3-conductive. In general, this compiler will output a $(|\mathbb{T}| + 1)$-conductive circuit, but we observe that each wire only needs to go to one OR and one AND gadget even if it is 0 (or 1) for more than one test query. With this observation, we get a 3-conductive circuit no matter how large $\mathbb{T}$ is. Using this observation we also see that the size of the circuit triples (one extra OR and AND gate for each wire).

The Actual Compiler. The previous solution is not secure once the adversary can also tamper with the OR and AND gadgets. There is a surprisingly simple and elegant fix to this. As illustrated in Figure 1.(D) we add one extra input, a control bit $x_{s+1} = c$. This control bit is given as the first input to all sequential AND and OR gadgets. It is safe to use COPY gates to create enough copies of c (i.e., we allow each of the copies to be tampered with independently), so the circuit remains 3-conductive. For our toy circuit we just need two copies $(c_{\mathsf{AND}}, c_{\mathsf{OR}}) \leftarrow \mathsf{COPY}(c)$.

The test set $\widehat{\mathbb{T}}$ for this compiled circuit $\widehat{C}$ is twice as big as for the previous construction where it was just the covering set $\mathbb{T}$: now for each $X \in \mathbb{T}$ we add $X\|0$ and $X\|1$ to $\widehat{\mathbb{T}}$. For our toy circuits, we get

$$\widehat{\mathbb{T}} = \{00000, 00001, 11110, 11111\}.$$

Security. We will prove that a circuit $(\widehat{C}, \widehat{\mathbb{T}})$ constructed as described from an input circuit C is testable. Let us here just give some intuition as to why this is the case for our toy circuit. Assume there is a tampering in the C subcircuit of $\widehat{C}$ that sets some wire to 1 as illustrated in Figure 2.(A). If there is no tampering on the OR gadget then we will detect tampering as we get a wrong output $y_{\mathsf{OR}} = 1$ on input $00000 \in \widehat{\mathbb{T}}$. The adversary can tamper with the OR gadget, say set a wire to 0 as shown in Figure 2.(B), but now $y_{\mathsf{OR}} = 1$ on input $00001 \in \widehat{\mathbb{T}}$ (i.e., same input but the control bit is $c = 1$), which is the wrong value and thus we will detect tampering.

This is not just a particular example, we observe that whenever there is at least one wire (other than the control bit) coming into such a sequential OR gadget that is set to 1, then no matter how the gadget is tampered, the output of this gadget will not change if we just flip the value of the control. An analogous statement holds for the AND gadget assuming some input is 0.

For our toy circuit, this means that once the adversary decided to apply a tampering τ that sets a wire to 1 in the C subcircuit of $\widehat{C}$, and using the convention that $C^{\tau}_{y}(X)$ denotes the value of the wire y when C is tampered according to τ and evaluated on X, we will have

$$\widehat{C}\tau_{y_{\mathsf{OR}}}(00000) = \widehat{C}^{\tau}_{y_{\mathsf{OR}}}(00001)$$

no matter how the gadgets are tampered with, while without tampering

$$\widehat{C}_{y_{\mathsf{OR}}}(00000) = 0 \neq \widehat{C}_{y_{\mathsf{OR}}}(00001) = 1$$

and thus we will detect a wrong output on some test query. Similarly, using the AND gadget we will detect whenever some internal wire is set to 0.

A major drawback of our construction is its depth. For a circuit C with n wires, the OR and AND gadgets in the complied $\widehat{C}$ will have depth up to n (and at least $n/4$), which for most circuits is not acceptable. Unfortunately, we cannot simply replace the sequential OR and AND gadgets with say a tree (that would only have logarithmic depth) as then the gadgets would no longer have the property required for the security proof we just sketched, we illustrate this problem in Figure 3.

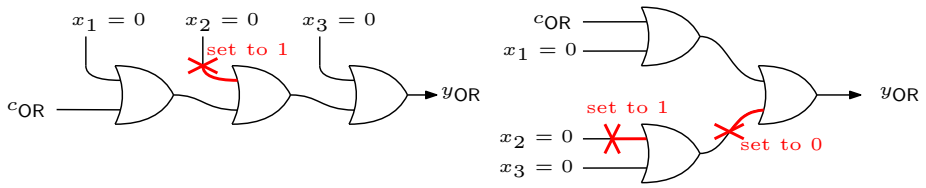

Fig. 3. left: Our OR gadget with four inputs: a control bit c_{OR} and three inputs x_1, x_2, x_3. If $x_1 = x_2 = x_3 = 0$ then the output is the control $y_{\mathsf{OR}} = x_{\mathsf{OR}}$. The key property of this gadget used in the proof is the following: If (at least) one of the inputs is tampered to 1 (in the figure x_2 is set to 1), then, no matter which other wires are tampered with, the output y_{OR} will be the same if we set $c_{\mathsf{OR}} = 0$ and $c_{\mathsf{OR}} = 1$ as the output of the 2nd OR gate will be 1 in both cases, thus "forgetting" the value of c_{OR}. **right:** Arranging the gates in a (low depth) tree structure does not have this property. The figure shows a tampering of x_2 to 1 and one more tampering such that $y_{\mathsf{OR}} = c_{\mathsf{OR}}$ holds if $x_1 = x_2 = x_3 = 0$.

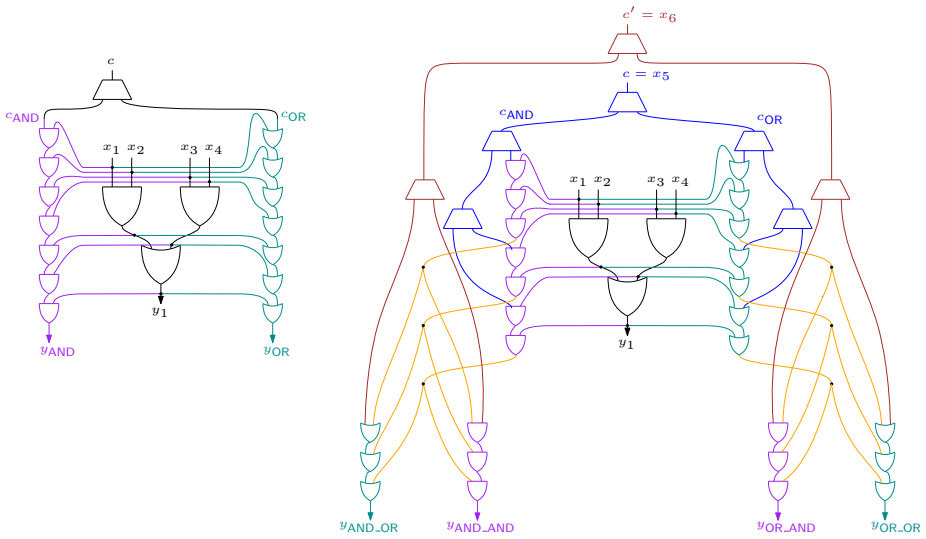

Fig. 4. left: Compilation of our toy circuit C at the basic level $L = 1$. The depth of the sequential AND and OR gadgets is as high as the number of wires in C. **right:** A compilation at level $L = 2$, the sequential gadgets of length m are chopped up into $\approx \sqrt{m}$ chunks, each of length $\approx \sqrt{m}$ (here $m = 7$ and the chunks are of length 3, 2 and 2). The depth of the compression gadgets drops from m to $2\sqrt{m}$ (and $\approx L \cdot m^{1/L}$ for general L).

Decreasing Depth. We will now outline how the depth of the "testing circuit" (i.e., the part of $\widehat{C}$ which is not the initial C circuit) can be brought down from n to $L \cdot n^{1/L}$ for a parameter $L \in \mathbb{N}$, with $L = 1$ giving the basic construction outlined above. While the depth decreases with larger L, the test set and the number of additional output wires will increase exponentially in L. That is not really an issue as already small values of L will be sufficient to make the circuit depth of the testing part very small. As a numerical example, for $n = 2^{32}$ (4 billion gates), with $L = 4$ we get a depth of $\approx 4 \cdot 2^{32/4} = 1024$ while for $L = 5$ that drops to 85.

We will outline our approach for our toy circuit and $L = 2$ as shown on the right side of Figure 4. For $L = 2$, the basic idea is to split the OR and AND gadgets into $\leq \sqrt{n}$ smaller ones of length $\sqrt{n}$. Each of them needs the control bit as the first input, which we create using COPY gates (we can arrange them in a balanced tree, so the depth for this is just $O(\log(n))$). The problem is that while before we had one output from every OR/AND gadget, now we have up to $\sqrt{n}$ outputs (one from each shorter gadget). We can use our compression idea recursively to compress these to 2 values.

For this, we need another control bit c' (the first and second level control bits are shown in blue and brown in the figure). The output of each smaller gadget (shown in orange in the figure) is then forwarded to an OR and an AND gadget which gets the control bit c'. Let us mention that the fan-out of the smaller gadgets is 2, and we can't use a COPY gate to get it down to 1 as the security proof will rely on both output wires carrying the same value.

For general L we split the gadget recursively into chunks of depth $n^{1/L}$. We need L extra input wires for the control bits and the number of additional output wires doubles if we increase L by 1, so it is at most $8 \cdot 2^L$. The test set for this more general construction also doubles if we increase L by 1: for every $x \in \mathbb{T}$ it contains the inputs $xc_1c_2 \ldots c_L$ for all 2^L choices of the L control bits. For our toy example where $\mathbb{T} = \{0000, 1111\}$ and $L = 2$ as illustrated in the figure this means

$$\widehat{\mathbb{T}} = \{000000, 000001, 000010, 000011, 111100, 111101, 111110, 111111\}$$

Security for General L. We show that for any L, the above complied circuit $(\widehat{C}, \widehat{\mathbb{T}})$ is testable.

This can be proven by induction on L and we will give some intuition for our toy circuit for $L = 2$ relying on the intuition we already provided for $L = 1$. Consider again that a (topologically first) tampering is setting a wire to 1 as indicated in Figure 5. This wire is going into an OR gadget, and its output, denoted α in the figure will not switch if we flip the control input c, no matter how the compression gadget is tampered (using the same argument as to why the y_{OR} input didn't flip in the $L = 1$ construction), so α will take the same value on inputs of the form 00000$*$ and 00001$*$ where $*$ can be either 0 or 1.

Let us assume it takes value $\alpha = 0$. In this case the output (labeled $y_{\mathsf{OR\_AND}}$ in the figure) of the AND gadget on the 2nd level will be identical on inputs 000010 and 000011 as the $\alpha = 0$ input to the AND gate in this gadget will make

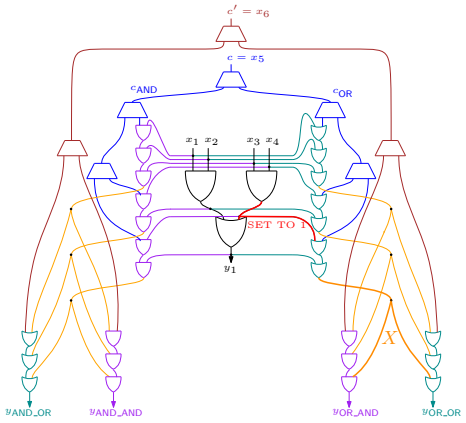

Fig. 5. Intuition for the security proof for our toy circuity using $L = 2$.

the output of this gate independent of the c' control bit. But $y_{\mathsf{OR\_AND}}$ should be different on those two inputs, so we will observe an inconsistent output on one of those two inputs. For the case where $\alpha = 1$ the argument uses the 2nd level OR gadget for inputs 000000 and 000001.

The Sections 4,5,6 of this work give a formal description of a circuit compiler compiling into testable circuits. The result is summarized with TestableCircuitCompiler procedure (see Algorithm 5) that takes two parameters - a circuit C and a number $L \in \mathbb{N}^+$ - and outputs a testable circuit - $(\widehat{C}, \widehat{\mathbb{T}})$. The properties of the construction are summarized in the Theorem 1.

Theorem 1. *Given any circuit $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ (of size n, depth d), and a number $L \in \mathbb{N}^+$, the procedure* TestableCircuitCompiler(C, L) *outputs a pair $(\widehat{C}, \widehat{\mathbb{T}})$ such that:*

- *$\widehat{C}$ is a circuit with additional $3 + L$ input bits and additional 2^L output bits, i.e. $\widehat{C} : \mathbb{Z}_2^{s+3+L} \to \mathbb{Z}_2^{t+2^L}$. The size of $\widehat{C}$ is bounded by $12n$, and its depth is bounded by $d + \log(n) + L \cdot (3n)^{1/L}$. The size of the test set $\widehat{\mathbb{T}}$ is $4 \cdot 2^L$.*
- *The pair $(\widehat{C}, \widehat{\mathbb{T}})$ is a testable circuit, i.e. for any tampering τ*

$$\Big(\exists X \in \mathbb{Z}_2^s : \widehat{C}^{\tau}_{|t}(X||0^{3+L}) \neq \widehat{C}_{|t}(X||0^{3+L}) \Big) \Rightarrow \Big(\exists T \in \widehat{\mathbb{T}} : \widehat{C}^{\tau}(T) \neq \widehat{C}(T) \Big)$$

and $\widehat{C}$ is functionally equivalent to C, i.e. $\forall_{X \in \mathbb{Z}_2^s} : \widehat{C}_{|t}(X||0^{3+L}) = C(X)$.

Section 6 gives a detailed description of the procedure TestableCircuitCompiler and proof of the Theorem 1.

1.3 Related Work

Generally, this work can be viewed as a part of research on securing cryptographic implementations against the so-called *physical attacks*, i.e., attacks in which the

adversary exploits the physical properties of the devices, instead of attacking them on the algorithmic level. The theoretical study of this area was initiated by Micali and Reyzin in [17]. In particular, as already highlighted above, our results are related to the papers studying the so-called *tampering* attacks, which is a long line of work initiated by [11].

The general compilers for protecting against the physical attacks were first considered in the papers on leakage-resilience [13], and then on tamper-resilience [12]. Our model of tampering ("resetting to 0/1", "toggling", and the "conductivity" assumption) is taken from the latter paper. Other works on tamper-resilient compilers include [5, 6, 10, 9, 15] and many more. In the context of trojan-resilience, the general compilers were considered in [7]. The model in [7] is orthogonal to ours: on one hand, it is more general, since it permits the adversary to change the topology of the circuit. On the other hand, it uses a stronger assumption than we do, namely: it relies on the existence of an untamperable "master circuit". The security guarantees of [7] are also weaker than ours, since they only provide trojan-resilience for a limited number of executions of the device.

A general compiler for trojan-resilience has also been recently constructed in [4], where the authors achieve security against adversaries that are more general than the ones considered by us. They also have the advantage of not modifying the circuit specification. On the other hand, as in [7] they require a trusted circuit. Moreover, they make an assumption that the inputs to the device come from an iid source, and they provide only limited security guarantees, namely, they only achieve correctness for a certain fraction of the executions of the device.

Another, orthogonal to ours, model in this area is the one where the adversary is allowed to replace the value of the output of the gate by one of their inputs (the so-called *adversarial short circuit errors*). This model has been introduced in [16], and further studied, e.g., in [14, 8]. Some papers (e.g. [1, 18]) also propose to use the techniques from verifiable computations. They typically provide strong security guarantees at a cost of significant computational overhead. Moreover, they also rely on the "master circuit" assumption.

For an overview of practical methods of defending against the trojan attacks see, e.g., [2]. The problem of the testing errors on wires is also extensively studied in the practical community, see, e.g., [3]. However, all these practical approaches are heuristic and do not come with a well-defined security model and guarantees.

2 Preliminaries

Let $[k] := \{1, 2, ..., k\}$. We denote the set of wires W of a circuit C as $W(C)$. We use a standard computation model of Boolean circuits as e.g. used in the private circuits literature [12]. For the most part, we will only discuss simple stateless Boolean circuits. Extensions to more general models like allowing memory cells or randomness gates are straightforward. A circuit C operates on a set $\mathfrak{G}$ of allowed gates and given some input X produces an output Y. We consider an

adversary who can arbitrarily tamper with every wire of the circuit, that is, choose one of the four possible actions: do nothing, flip the value or set it to constant 0 or 1. It will be convenient to also model the input, output wires of the circuit (aka. pins) as gates with indegree, outdegree 0, respectively.

We require that tampering is *conductive*. This means that if an output wire of a gate is used as input to more than one gate, we think of this as a single wire for which only one tampering action can be chosen. We leave it as a main open problem to construct a compiler not relying on this assumption. This assumption is conciliated by the fact that in the compiled circuit any wire is used as input to at most three gates, which we refer to as 3-conductive. We assume that the input circuit is not conductive (which means 1-conductive, i.e., each wire is used as input to exactly one gate), this is without loss of generality as by using copy gates $(x, x) = \mathsf{COPY}(x)$ 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 $\lceil \log(k) \rceil$.

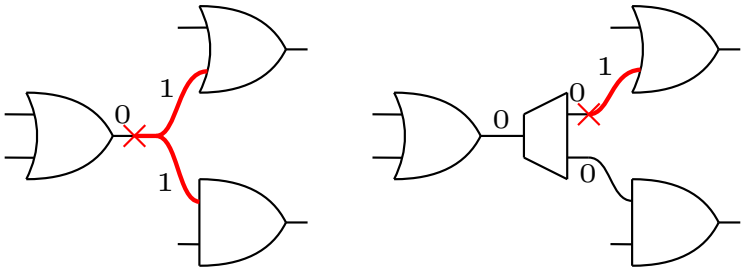

Fig. 6. left: Part of a 2-conductive circuit, any tampering (a toggle is illustrated) will affect the two inputs in the same way. **right:** Using a COPY gate the circuit can be made 1-conductive, allowing more general tampering attacks where each wire can now be tampered individually.

2.1 Notation for Circuits

In this section, we define the notation for the Boolean circuits which we will use throughout the paper. 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. We will often omit the parameter γ since it is chosen when specifying the circuit and cannot be tampered with. 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 $V^+(e) = v$.

Fan-Out and Conductivity. Each gate type in $\mathfrak{G}$ has some given number of input and output wires as indicated in Table 2.1. Each output wire is used as an input wire to at least one other gate (if not the gate is redundant and can be removed). An output wire can lead to more than one input wire. The *fan-out* of an output wire is the number of input wires it leads to. A circuit is k-*conductive* if no wire has a fan-out greater than k; a 1-conductive circuit is also called non-conductive. Every k-conductive circuit can be turned into non-conductive one by using copy gates $\mathsf{COPY}(x) = (x, x)$ as illustrated in Figure 6. As said before, this transformation does not preserve security against wire tampering as considered in this work, where tampering with a wire of fan-out > 1 affects all the input wires in the same way.

Gates	Inputs	Outputs
OR, XOR, AND	2	1
out	1	0
in	0	1
NOT	1	1
COPY	1	2

Table 1. Number of inputs and outputs for the gates from $\mathfrak{G}$.

It will be convenient to define two non-standard gates: the input and output gates denoted **in**, **out**.

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 $X = (x_1, ..., x_s) \in \mathbb{Z}_2^s$ we define a valuation function

$$\mathrm{val}_{C_\gamma, X} : V \cup E \to \mathbb{Z}_2 \tag{1}$$

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

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

$$\mathrm{val}_{C_\gamma, X}(e) = \mathrm{val}_{C_\gamma, X}(V^-(e)).$$

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:

$$C(X) = (\mathrm{val}_{C,X}(O_1), \mathrm{val}_{C,X}(O_2), ..., \mathrm{val}_{C,X}(O_t)).$$

2.2 Tampering Model

We consider an adversary who can tamper with *every* wire in the circuit. The tampering of a wire is described by a function $\mathbb{Z}_2 \to \mathbb{Z}_2$ from the class of the four 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 $\tau(e)$.

Conductivity and Tampering. The only restriction we put on the tampering is: for the output wires from a vertex with fan-out > 1 the same tampering is applied. In our graph notation, this means that for every $v \in V$ and $e, e' \in E^+(v)$ we have $\tau(e) = \tau(e')$. The only exception is the copy gate as it has two output wires. Here the tampering must fall into one of two choices (one for each output wire), i.e., for $v \in V$ where $\gamma(v) = \mathsf{COPY}$ we have $|\{\tau(e) \ : \ e \in E^+(v)\}| \leq 2$.

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

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

The only difference to the (non-tampered) valuation function from eq.(1) 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 behaviour of the *tampered* circuit:

$$C^\tau(X) = (\mathrm{val}^{\tau}_{C,X}(O_1), \mathrm{val}^{\tau}_{C,X}(O_2), ..., \mathrm{val}^{\tau}_{C,X}(O_t)).$$

3 A compiler transforming circuits into testable circuits

In this work, we develop a notion of (efficiently) *testable* circuits. Consider some circuit $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$, then its testable version is a circuit $(\widehat{C} : \mathbb{Z}_2^{s+s'} \to \mathbb{Z}_2^{t+t'}, \widehat{\mathbb{T}} \subset \mathbb{Z}_2^{t+t'})$ which is functionally equivalent (i.e. $\widehat{C}$ computes the same function by setting the s' extra input bits to 0 and ignoring the t' extra output bits), but additionally is able to detect any non-trivial tampering τ (i.e. any tampering that makes $\widehat{C}$ output something else than output of C, ignoring the last t' test bits) by observing its input-output behaviour on some small set $\widehat{\mathbb{T}}$ of test inputs.

Definition 1 (A testable circuit). *For any circuit $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$, a pair $(\widehat{C}, \widehat{\mathbb{T}})$ - a circuit $\widehat{C} : \mathbb{Z}_2^{s+s'} \to \mathbb{Z}_2^{t+t'}$ along with a testing set $\widehat{\mathbb{T}} \subset \mathbb{Z}_2^{s+s'}$ - is a testable circuit if and only if:*

– *$\widehat{C}$ is functionally equivalent to C, i.e.*

$$\forall X \in \mathbb{Z}_2^s \ : \ \widehat{C}_{|t}(X \| 0^{s'}) = C(X)$$

– *$(\widehat{C}, \widehat{\mathbb{T}})$ is testable, i.e. for any tampering τ*

$$\exists X \in \mathbb{Z}_2^s : \widehat{C}^{\tau}_{|t}(X || 0^{s'}) \neq \widehat{C}_{|t}(X) \ \Rightarrow \ \exists T \in \widehat{\mathbb{T}} : \widehat{C}^{\tau}(T) \neq \widehat{C}(T)$$

In the subsequent sections we construct a compiler that takes any circuit and maps it to a testable circuit.

Definition 2 (Compiler transforming into testable circuits). *A circuit compiler $\mathfrak{C}$ is an algorithm transforming a circuit $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ into a* testable *circuit $(\widehat{C} : \mathbb{Z}_2^{s+s'} \to \mathbb{Z}_2^{t+t'}, \widehat{\mathbb{T}} \subset \mathbb{Z}_2^{s+s'})$.*

3.1 Parameters of our Construction

A useful circuit compiler $\mathfrak{C}$ should meet the following (informal) conditions:

– $\widehat{C}$ is not much more complex than the input circuit C (since we want the compiled circuit to be practical in production and use),
– $\widehat{\mathbb{T}}$ is not too large (to allow an efficient verification).

As mentioned in the introduction, our compiler first precompiles the circuit C into an intermediary $(C', \mathbb{T})$ and finally outputs the compiled $(\widehat{C}, \widehat{\mathbb{T}})$. Let $n, n', \widehat{n}$ denote the number of vertices, and $d, d', \widehat{d}$ the depth of C, precompiled C' and compiled $\widehat{C}$, respectively. Our compiler takes as additional input a *recursion parameter* L, which then gives the following parameters:

	Input C	**Preprocessed C'**	**Compiled $\widehat{C}$**
Input size	s	$s+3$	$s+3+L$
Output size	t	t	$t+2^L$
Number of gates	n	$3n$	$12n$
Circuit depth	d	$d+\log(n)$	$d+\log(n)+L\cdot[3n]^{1/L}$
Test set size			$4\cdot 2^L$

The circuit $\widehat{C}$ is 3-conductive, recall that this means that a gate can have an output wire with fan-out of up to 3, and a tampering affects all these wires in the same way. We assume the input circuit C is non-conductive (i.e., 1-conductive), which is without loss of generality as any circuit can be transformed into a non-conductive one at a modest cost using COPY gates. We cannot make $\widehat{C}$ non-conductive this way as this would not preserve its testability.

4 Compilers

In this section, we will show an efficient circuit compiler, which works for arbitrary circuit. As a warm-up, we will present two impractical (but educative) solutions: they make I/O size or the depth of the compiled circuit linear in the size of the original circuit. We will then build on these basic ideas to make a practical construction.

Unlike some other circuit compilers, our compiled circuit C will be a subcircuit of the compiled $\widehat{C}$. Let us also stress that we do not rely on any tamper-free components, randomness gates or alike.

4.1 The basic ideas

4.1.1 Covering sets Recall, that the tampering on a wire e is a function $\tau_e : \mathbb{Z}_2 \to \mathbb{Z}_2$ assigned to that wire. If any of the functions associated with wires on the circuit is not identity, the circuit is tampered. So the general idea is to check the behaviour of every wire when it should transfer the value 0 and when it should transfer the value 1 in a smart way. To work with this intuition, we define a *covering set*. It is a set of inputs for which every wire would take both values, 0 and 1. Formally speaking,

Definition 3. *We say, that a set of inputs $\mathbb{T} = \{t_1, ..., t_k\}$ is a covering set (for some circuit C) if*

$$\forall\ e \in E(C),\ b \in \{0,1\}\ \exists X \in \mathbb{T}\ :\ \mathrm{val}_X(e) = b\ .$$

In general, a circuit may not have a covering set at all, as some wires could be 0 or 1 on every input to the circuit. Fortunately, every circuit can be efficiently modified into a functionally equivalent circuit with a covering set of constant size. We will demonstrate the construction in Section 5.

Intuitively, a covering set seems to be the smallest testing set for the testable circuit. If the circuit would be tested on a set of inputs that do not form a covering set, then there would exist a wire which would take only one value for each test, say 0. Then we would not detect zero tampering on this wire.

The rest of this section assumes that the circuit C under consideration has been transformed into a circuit C' with a small covering set $\mathbb{T}$.

4.1.2 The simplest solution A trivial solution is to simply output every wire value. For this we increase the fan-out of every wire by one, and connect each wire to new output gates $O_{t+1}, O_{t+2}, ..., O_{t+n}$. Now we can simply check the values of these output gates on our covering set. Since the circuit is a DAG, there always exists a topologically first tampered wire. The output value related to this wire must show an inconsistency for some element of the covering set.

Let us stress that already this solution increases the conductivity by one, i.e. if we start with a non-conductive circuit, we get a 2-conductive circuit. Simply using a COPY gate for each wire and outputting the extra wire would not increase

the conductivity, but it would also not guarantee that we detect every tampering even after evaluating the compiled circuit on a covering set.

Nevertheless, the above construction is totally impractical. Although the circuit size only doubles, it massively increases the number of output wires. This not only makes testing the correctness of the outputs impractical but is also unimplementable. In practice, circuit pins (input/output wires) are expensive and they should only constitute a tiny fraction of the circuit size as they are large and must be placed at the border.

In the rest of this section, we will show how to compress these extra outputs in a way that still allows us to detect inconsistency. The challenge is to achieve this while also allowing for the compressing part of the circuit to be tampered with.

4.1.3 Sufficiently tamper-resilient gadgets In our construction we will use gadgets (circuits) that compute multi-input OR (mOR) and multi-input AND (mAND) functions, and are tamper-resilient *to some extent*. We will show their construction a bit later. For now, we only need to define their properties. In general, the gadget has normal and additional inputs. It should compute OR/AND functions on its input and prevent some class of tampering on some subset of the input wires.

Definition 4 (Sufficiently Tamper Resilient mOR). *A multi-input* OR *with inputs* $x_1, x_2, ..., x_m, c$ *and output* y*, is a sufficiently tamper-resilient* mOR (STRmOR) *gadget if:*

1. *It computes* OR *on its inputs, i.e.* $\mathrm{val}_{x_1 \dots x_m x_m c}(y) = 0$ *iff* $x_1 = x_2 = ... = x_m = c = 0$.
2. *If the tampering* τ *changes some* 0 *to* 1 *among* $x_1, ..., x_m$*, then*

$$\mathrm{val}^{\tau}_{0^{m+1}}(y) = \mathrm{val}^{\tau}_{0^m 1}(y).$$

Similarly,

Definition 5 (Sufficiently Tamper Resilient mAND). *A multi-input* AND *with input* $x_1, x_2, ..., x_m, c$ *and output* y*, is a sufficiently tamper-resilient* mAND (STRmAND) *gadget if:*

1. *It computes* AND *on its inputs, i.e.* $\mathrm{val}_{x_1 \dots x_m x_m c}(y) = 1$ *iff* $x_1 = x_2 = ... = x_m = c = 1$.
2. *If the tampering* τ *changes some* 1 *to* 0 *among* $x_1, ..., x_m$*, then*

$$\mathrm{val}^{\tau}_{1^{m+1}}(y) = \mathrm{val}^{\tau}_{1^m 0}(y).$$

At the first sight, the definition says nothing about the *inconsistency* between the tampered and untampered version of the gadget. However, since STRmOR (STRmAND) computes the general OR (AND) function, the following holds for untampered versions of the gadgets:

$$\mathrm{val}_{\mathsf{STRmOR},0^{m+1}}(y) \neq \mathrm{val}_{\mathsf{STRmOR},0^m 1}(y),$$

$$\mathrm{val}_{\mathsf{STRmAND},1^{m+1}}(y) \neq \mathrm{val}_{\mathsf{STRmAND},1^m0}(y).$$

Thus there is an inconsistency between the outputs of the untampered circuit and its tampered version, which can be easily verified on the input sets $\{0^{m+1}, 0^m1\}, \{1^{m+1}, 1^m0\}$, respectively.

The STRmOR and STRmAND can be realized as the ChainMultiOR, presented in the Figure 7. To simulate mOR function with m inputs we would need only m simple OR gates.

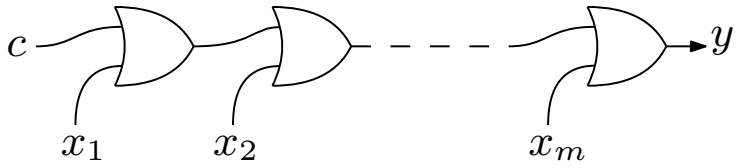

Fig. 7. Construction of the ChainMultiOR. The actual inputs are on the bottom.

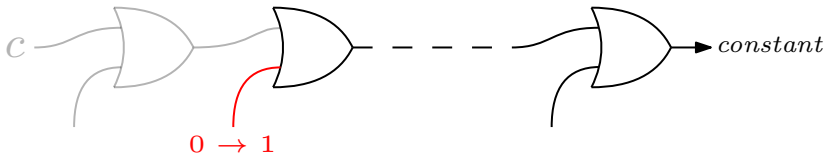

Fig. 8. For any tampering on the input wire which would change the value from 0 to 1, the output becomes blind to everything that happens before. In particular, the output does not depend on c.

Lemma 1. ChainMultiOR *is a* STRmOR.

Proof. The first condition holds obviously. Let τ be a tampering such that x_i is tampered to 1. Consider the function $f(c) := \mathrm{val}^{\tau}_{00\ldots0c}(y)$. Then the OR gate, which takes x_i as input, is unaffected by the second input (see Figure 8), since the value 1 on on of its inputs determines the output of OR. Therefore any change in the value of c cannot change the value of the output of this gadget, so $f(c)$ is a constant. □

An analogous proof can be made for ChainMultiAND.

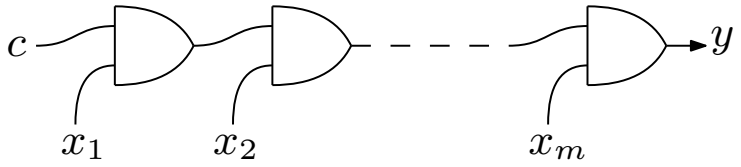

Fig. 9. Construction of the ChainMultiAND. The actual inputs are on the bottom.

Corollary 1. ChainMultiAND *is a* STRmAND.

4.1.4 Costly compression Now we can present a bit more practical solution (as we will see, it has 1 main drawback - it increases the depth of the circuit). For this we assume not 2-, but 3-conductivity. Every internal wire of the original circuit is branched into 3 wires *along with its tampering*. One continues into the circuit as before, and two others are used for the testing.

Let $\mathbb{T} = \{t_1, ..., t_k\}$ be a covering set for C'. Then for each $e \in E(C')$ we can choose two test inputs: one for the value 0, and the second one for the value 1. In other words, for every $e \in E(C')$ there exist $i, j \in [k]$ such that

$$\mathrm{val}_{t_i}(e) = 0, \quad \mathrm{val}_{t_j}(e) = 1.$$

For all $j \in [k]$ and $b \in \{0, 1\}$ define

$$W_j^b := \{e \in E(C') : \mathrm{val}_{t_j}(e) = b\}$$

Intuitively, W_j^b consists of all the wires which will be checked for tampering on those wires which changes the value from b to $1 - b$ using the input t_j.

To make C' testable, we extend it by adding an input gate, I_{s+1}, and k similar gadgets. For every $j \in [k]$ we add to the original circuit the following elements:

- STRmOR_j gadget with inputs from the set $W_j^0 \cup \{I_{s+1}\}$, which outputs O_j^{OR}
- $\mathsf{STRmAND}_j$ gadget with inputs from the set $W_j^1 \cup \{I_{s+1}\}$, which outputs O_j^{AND}

We call this extension of C' as $\widehat{C}$. Observe, that if $C' : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$, then $\widehat{C} : \mathbb{Z}_2^{s+1} \to \mathbb{Z}_2^{t+2k}$. Outputs are given in the order $O_1, O_2, ..., O_t, O_1^{\mathrm{OR}}, O_1^{\mathrm{AND}}, ..., O_k^{\mathrm{OR}}, O_k^{\mathrm{AND}}$.

For every $t_j \in \mathbb{T}$ define the following 2 inputs to the extended circuit $\widehat{C}$: $T_j^0 := t_j||0, T_j^1 := t_j||1$. Let $\widehat{\mathbb{T}} := \{T_j^b\}_{b=0,1;j=1,...,k}$, which is the testing set for our extended circuit, then we would get a testable circuit.

Lemma 2. *Let* $(C', \mathbb{T})$ *be a circuit with covering set and* $\textsc{Compress}_0$ *be the Algorithm 1. Then* $(\widehat{C}, \widehat{\mathbb{T}}) = \textsc{Compress}_0(C', \mathbb{T})$ *is a testable circuit.*

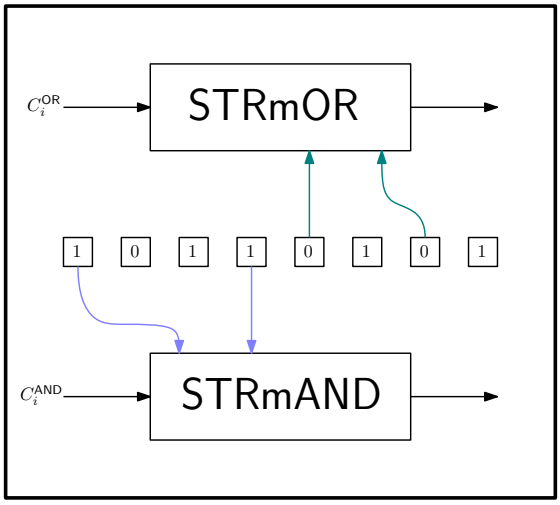

Fig. 10. For every element $t_i \in \mathbb{T}$ we extend the circuit by one mOR, one mAND gadget, single input gate, and 2 output gates.

Algorithm 1: COMPRESS$_0$

Input: $(C', \mathbb{T})$
/* a circuit along with its covering set */
Output: $(\widehat{C}, \widehat{\mathbb{T}})$
/* a testable circuit functionally equivalent to C' along with its testing set */

$\widehat{C} := C'$
$k := |\mathbb{T}|$
Construct the sets $W_j^b = \{e \in E(C') : \mathrm{val}_{t_j}(e) = b\}$ for $j = 1, ..., k$
Add I_{s+1} to the set of input gates of $\widehat{C}$
for $j = 1, ..., k$ **do**
    Extend $\widehat{C}$ by STRmOR_j gadget with inputs from $W_j^0 \cup \{I_{s+1}\}$, output O_j^{OR}
    Extend $\widehat{C}$ by $\mathsf{STRmAND}_j$ gadget with inputs from $W_j^1 \cup \{I_{s+1}\}$, output O_j^{AND}
end
$\widehat{\mathbb{T}} = \emptyset$
for $j = 1, ..., k$ **do**
    $T_j^0 := t_j|0$
    $T_j^1 := t_j|1$ $\widehat{\mathbb{T}} = \widehat{\mathbb{T}} \cup \{T_j^0, T_j^1\}$
end
return $(\widehat{C}, \widehat{\mathbb{T}})$

Proof. Assume that $\widehat{C}^{\tau}$ is tampered non-trivially. Thus at least one wire in the original subcircuit C' is tampered. Take the *topologically first* tampered wire $e \in E(C')$. Let $i, j \in [k]$ be indices such that $\mathrm{val}_{C',t_i}(e) = 0,\ \mathrm{val}_{C',t_j}(e) = 1$.

Assume WLOG, that the tampering on e changes 0 to 1 (i.e tampering is one or *toggle* function). We will show, that the gadget STRmOR_i enables us to detect the tampering on $\widehat{C}^{\tau}$. Consider the tests T_i^0, T_i^1. Since the STRmOR_i gadget is sufficiently tamper-resilient, and at least one of the incoming wires is tampered from 0 to 1, we will detect the inconsistency. □

The solution presented above has a major drawback - it makes the depth of the compiled circuit linear in the size of the original circuit, which is unacceptable for many practical applications. Fortunately, this can be resolved.

4.2 General construction

We are now ready to present a solution with low depth and small I/O size. The starting point is the naive solution from the section 4.1.2. The full output will be then compressed layer by layer - each layer will be reducing the size of the additional output by the factor $\frac{n^{\frac{1}{L}}}{2}$, where L is the number of layers. For every layer the additional outputs will be divided into groups of $n^{\frac{1}{L}}$ and every group will be compressed using the chain gadget from section 4.1.3 to 2 output bits. We call the factor $n^{\frac{1}{L}}$ the *width* of the layer.

A fragment of the construction of a single layer is presented in the Figure 11.

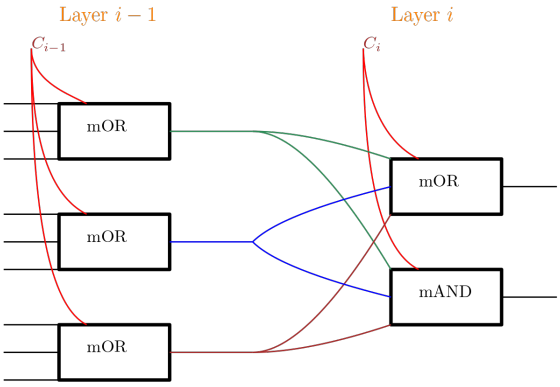

Fig. 11. A fragment of the subcircuit produced by Algorithm 2. We assume the chain length parameter $w^{1/L} = 3$. In the picture we can see a fragment of the construction starting with the inputs of the Layer $i-1 \geq 1$ organized into groups of 3. They are then processed by mOR gates connected to a single control bit C_{i-1}. The outputs from these gates (indicated with green, blue, and brown lines) are then connected to new gates mOR and mAND which are again connected to new mOR and mAND gates connected to a single control bit C_i (lines 13 and 14 of the Algorithm 2). The mOR and mAND gates are implemented with STRmOR and STRmAND gadgets.

Algorithm 2: COMPRESS

Input: $(C' : \mathbb{Z}_2^s \to \mathbb{Z}_2^t, \mathbb{T}), L$
/* a circuit with its covering set, parameter */
Output: $(\widehat{C}, \widehat{\mathbb{T}})$
/* efficiently testable circuit */

$\widehat{C} := C'$;
for $e \in E(C')$ **do**
  Append O_e as the output gate to $\widehat{C}$
end
for $b = 0, 1; j = 1, ..., |\mathbb{T}|$ **do**
  make W_j^b set out from $(O_e)_{e \in E(C')}$ gates;
end
for *l=1,...,L* **do**
  Append I_{s+l} gate to $\widehat{C}$;
  for $j = 1, ..., |\mathbb{T}|$ **do**
    Divide the set W_j^b into the subsets $W_{j,i}^b$ of the size $n^{\frac{1}{L}}$;
    for *i=1,...* **do**
      Extend $\widehat{C}$ by ChainMultiOR made from $W_{j,i}^0 \cup I_{s+l}$ with V_i^0 as the output;
      Extend $\widehat{C}$ by ChainMultiAND made from $W_{j,i}^1 \cup I_{s+l}$ with V_i^1 as the output;
    end
    $W_j^b = \bigcup_i \{V_i^b\}$
  end
end
$\widehat{\mathbb{T}} := \bigcup_{T \in \mathbb{T}} T | \mathbb{Z}_2^L$;
return $(\widehat{C}, \widehat{\mathbb{T}})$;

Lemma 3. *Let $(C', \mathbb{T})$ be a testable circuit of conductivity 1, size n, depth d, input size $s + k$ and output size $t + r$, where r output bits are used for testing (consistency check). Then $(\widehat{C}, \widehat{\mathbb{T}}) = \textsc{Compress}(C, \mathbb{T}, L)$ is a testable circuit of conductivity 3, depth $< d + l \cdot n^{\frac{1}{L}} + \log(n)$, input size $s + l$, output size $t + 2^L$ and $|\mathbb{T}'| = 2^L|\mathbb{T}|$.*

Proof. First, we prove, that $(\widehat{C}, \widehat{\mathbb{T}})$ is a testable circuit. Let τ be nontrivial tampering over C'. Consider the topologically first tampered e along with associated with O_e. Then for some $b \in \mathbb{Z}_2$ we have

$$\tau(e)(b) = 1 - b.$$

Let $i_0, i_1 \in [k]$ be indices such that $\mathrm{val}_{C,t_{i_b}}(e) = b$ for $b \in \{0, 1\}$. The algorithm collects O_e by 2 chain gadgets. For $b = 0, 1$ the output of ChainMultiOR, ChainMultiAND, respectively, is constant (along with test T_{i_b}). For the next step of the algorithm (i.e. $l = 2$) this constant value is collected by another pair of ChainMultiOR/ChainMultiAND gadgets, so it works just like the tampering on the inputs to these gadgets. For one of the gadgets this tampering meets the requirement for STRmOR/STRmAND (see Definition 4). By induction, we obtain, that for some $q \in [2^L], X \in \mathbb{Z}_2^{L-1}$

$$\mathrm{val}^{\tau}_{\widehat{C},T_{i_b}X0}(O_{t+q}) = \mathrm{val}^{\tau}_{\widehat{C},T_{i_b}X1}(O_{t+q}),$$

where for the not tampered $\widehat{C}$ these 2 values should be different. Therefore the inconsistency will be detected when $\widehat{C}$ is tested on all the elements of $\widehat{\mathbb{T}}$. So the pair $(\widehat{C}, \widehat{\mathbb{T}})$ is a testable circuit.

Next, we need to prove, that the algorithm Compress actually achieves the desired parameters. It is not difficult; we can observe, that the algorithm adds L layers. Each of them:

- adds a single input gate;
- multiplies the number of additional output wires by the factor $\frac{2}{n^{\frac{1}{L}}}$
- makes the depth of the circuit higher by $n^{\frac{1}{L}}$.

Moreover, the conductivity of the internal wires need to be 3, and of the rest of the wires - at most 2. Every test from the original test set is duplicated 2^L times (by adding a sequence of $0, 1$ of length L). □

5 Small covering set for every circuit

5.1 Introduction

Recall that the construction $\widehat{C}$ from the previous section assumes that each wire from the input circuit C' takes values 0 and 1 given some inputs from the testing set - it has a *covering set* (see Definition 3).

In fact, for many practical circuits, some number of random inputs should form, with a high likelihood, a *covering* set for the majority of its wires (e.g.

one may expect that the majority of wires in circuits computing pseudorandom values take random values given random inputs), but the general problem of the existence of covering set for the set of all wires $W(C)$ of an arbitrary circuit C is NP-hard (consider SAT instances). In this section we will design an algorithm that transforms any circuit $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ into a circuit $C' : \mathbb{Z}_2^{s+s'} \to \mathbb{Z}_2^t$ with a *covering* set for $W(C')$ of a *small* size. The new circuit C' will be functionally equivalent to C (i.e. $\forall_{X \in \mathbb{Z}_2^s} C'_{|t}(X||0^{s'}) = C(X)$. In the case of standard circuits with maximum fan-in of the gates equal to 2, we will obtain a 4-element covering set.

Now we will give a brief description of the algorithm. It starts the creation of the *covering set* by evaluating the circuit on all *zeroes* input and all *ones* input. Next, we observe that at all *zeroes* input each wire $w_j \in W(C) : V^-(w) \neq \mathbf{in}$ is evaluated to either 0 or 1 (i.e. it is evaluated to some $v_j = \mathrm{val}_{C,0^s}(w_j)$). Finally, the algorithm divides the wires in W from C into a small number of subsets of wires W_i. Every subset is proceeded in one step, in which the wires from W_i are *fixed* (i.e. each wire $w_j \in W_i$ now is evaluated to $v'_j \neq v_j$) by adding only 1 additional *control* bit at each step. The details of the algorithm are given in the next subsections.

5.2 k-divisible circuits

In the final step of the algorithm, we fix the values on all $w \in W_i$, by manipulating the inputs of $V^-(w)$ using a single *control* bit for each W_i. To this end, we need to divide W to k disjoint subsets $W_1, W_2, \ldots, W_k$, such that:

1. outputs of some gate should belong to a different W_i than inputs to this gate, i.e.
$$\forall i \in [k], w_{in}, w_{out} \in W_i : V^+(w_{in}) \neq V^-(w_{out})$$
2. for each gate, all of its output wires belong to the same subset, i.e.
$$\nexists i \neq j, w_1 \in W_i, w_2 \in W_j : V^-(w_1) = V^-(w_2)$$

We say that a set of wires W in a circuit C is *k-divisible* if it can be partitioned into k subsets $W_1, \ldots, W_k$ meeting the conditions above. The properties of k-divisions are presented visually in the Figure 12. Any family of subsets $W_1, W_2, ..., W_k$ which proves that C is k-divisible is called a k-division for C.

5.3 Every circuit with maximum fan-in d is $(d+1)$-divisible

Let $C = (V, E)$ be a circuit, where every $n \in V$ has maximum fan-in not greater than d. We show a greedy algorithm DIVIDE that constructs its $(d+1)$-division efficiently.

It starts with the family of empty sets $W_1, \ldots, W_{d+1}$. Firstly we assign all the input wires of the circuit C (i.e. all $w \in E : \gamma(V^-(w)) = \mathbf{in}$) to W_1. Then we process the gates of the circuit C in the topological order: for every gate g

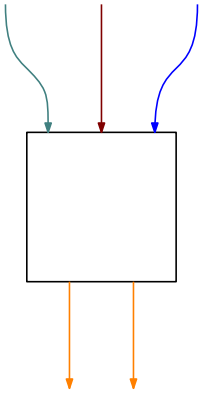

Fig. 12. The picture visualises the properties of k-divisions introduced on a set of wires of a circuit C. All wires output by the same gate must belong to the same W_i. The output wires of a gate must belong to W_j which is different from W_k's of the input wires to the gate.

we assign its all output wires - $w \in E^+(g)$ - to W_i where the index i is the smallest one not assigned to any of the input wires of g. It is easy to see that after finishing the algorithm, all of the wires of the circuit $w \in W$ belong to some W_i, and one needs a family of size $d+1$ to finish the algorithm on a circuit C with maximum fan-in d.

Algorithm 3: DIVIDE

Input: $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t, d$
/* a circuit, its fan-in */
Output: $W_1, ..., W_{d+1}$
/* $(d+1)$ - division of C */
1 **for** $i = 1, ..., d+1$ **do**
2 | $W_i := \emptyset$
3 **end**
4 $W_1 := W_1 \cup \{I_1, ..., I_s\}$
5 **for** $v \in V(C)$ **do**
| /* v proceeded in topological order */
6 | Choose any i such that $E^-(v) \cap W_i = \emptyset$
7 | $W_i := W_i \cup E^+(v)$
8 **end**
9 **return** $W_1, ..., W_{d+1}$

5.4 Constructing small covering sets for k-divisible circuits

Let $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ be a circuit $(W_1, ..., W_k)$ and its k-division. We construct a circuit $C' : \mathbb{Z}_2^{s+s'} \to \mathbb{Z}_2^t$ with $s' = k$ which can emulate C and has a covering set T of size $k + 1$.

The algorithm works in steps that correspond to parts W_i of the division. In every step, we try to fix the values taken by the wires from the corresponding part, after which every $w \in W_i$ is evaluated to both 0 and 1 on some inputs from the test set. As mentioned before, the fixing is achieved by extending the input with a new input bit - called a *control* bit - and adding to the test set a new input. All $w \in W_i$ are fixed by XORing the new *control* bit with the wires of input bits to $V^-(w)$ (an example is given in the Figure 13). It is not possible to fix all $w \in W_i$ *at once* by inserting the required XOR gates, since the valuations of the fixed wires may depend on each other. For this reason, we process all wires from W_i in topological order.

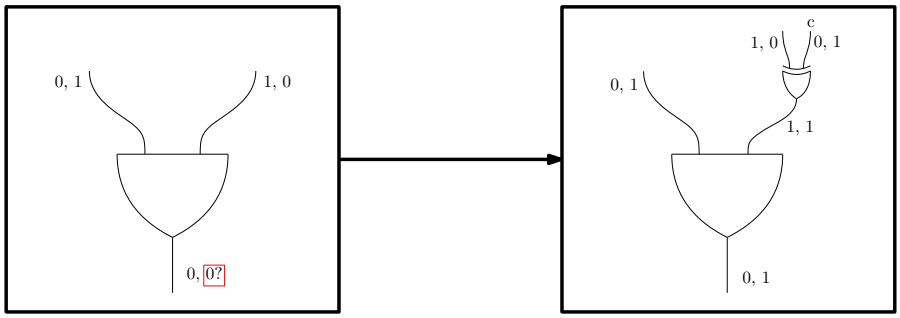

Fig. 13. An example showing how the output wire of an AND gate can be fixed with a single *control* bit. Before fixing it is always evaluated to 0, therefore we add a XOR gate to one of the incoming wires to make the actual output equal to 1 in this step. The process can be repeated on the subsequent gates.

The Algorithm 4 below takes as input a definition of C and produces a new C', functionally equivalent to C, along with a covering set $\mathbb{T}$ for the C'. It constructs the new C' and the new $\mathbb{T}$ in steps, keeping track of valuations of w in the structure V, i.e. in every moment of the execution of the algorithm $b \in V[w] \implies \exists_{T \in \mathbb{T}} : \text{val}_{C',T}(w) = b$. One needs to see that the modifications introduced in the circuit, as gates are processed in a topological order, may already be sufficient to fix outputs of the topologically subsequent gates, therefore the processing in the line 15 of the algorithm may not always be necessary.

To compute $C(X)$ we simply complement X with 0s on the control input wires:

$$C(x) = C'(x||0^k).$$

Theorem 2. *Given a circuit $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$, the procedure* Covering *(C) returns a pair $(C', \mathbb{T})$ such that:*

Algorithm 4: COVERING

Input: $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$
Output: $(C', \mathbb{T})$

$W_1, \ldots, W_{k+1} = k$-division of C `/* Computed using the algorithm from Section 5.3 */`
Initialize $C' = C$
Add k new *control* input wires to C'
Initialize $V = \{\}$, $\mathbb{T} = \{1^s 0^k\}$
for $w \in W, b \in \{0, 1\}$ **do**
    Append b to $V[w]$ whenever $\mathrm{val}_{C', 1^s 0^k}(w) = b$.
end
for $i \in [k]$ **do**
    $T_i = 0^{s+i-1} 10^{k-i}$;
    $\mathbb{T} = \mathbb{T} \cup \{T_i\}$
    for $n \in V(G)$ (processed in a *topological* order) **do**
        if $E^+(g) \subseteq W_i$ *and* $\exists_{b \in \{0,1\}} : \forall_{w \in E^+(g)} b \notin V[w]$ **then**
            Set $L = E^+(g)$
            if $\forall_{w \in L} \mathrm{val}_{C', T_i}(w) \neq b$ **then**
                Update C' by adding a XOR gate between the $s + i$'th input wire of C' and and input of n that sets $w \in L$ to b on the input T_i
            end
            Append b to $V[w]$ for all $w \in L$
        end
    end
end
return $G', \mathbb{T}$

1. *C' is a circuit with additional k bits, i.e. $C' : \mathbb{Z}_2^{s+k} \to \mathbb{Z}_2^t$,*
2. *C' is functionally equivalent to C, i.e. $\forall_{X \in \mathbb{Z}_2^s} C(x) = C'(x||0^k)$,*
3. *$\mathbb{T}$ is a covering set for C'.*

Proof. To see that $\forall_{X \in \mathbb{Z}_2^s} C(x) = C'(x||0^k)$, please note that when *control* bits are equal to 0, then the XOR gates added to the C will not change its standard behaviour. Note that we need only a single XOR gate for every wire in the original circuit, because all output wires from every gate are evaluated to the same value (see the details in Section 2.1) and belong to the same subset of k-division (according to its definition). Moreover, every new XOR gate refers to some W_i (a XOR gate refers to W_i, when it was made to fix one of the elements of W_i).

Now we will show, that every $w' \in E(C')$ takes both values when C' on some of the test inputs from the $\mathbb{T}$. Consider the following cases:

- $\gamma(V^-(w')) = \mathbf{in}$, therefore w' either belongs to one of the first s input bits, or is one of the new control bits. In the first case, w' takes value 1 on the test input introduced during the initialization in the line 4 of the Algorithm and the value 1 on one of the inputs added during the processing of one of the W_i. In the second case on the test input introduced during the initialization in the line 4 the w' is evaluated to 0 and on one of the tests T_i added during the processing of every W_i in the k-division it takes value 1.
- $\gamma(V^-(w'))$ is one of the gates of the origin C (except from input gates). In this case, w' was evaluated to one bit b during the initialization phase in the line 4 of the Algorithm. What is more assuming that $w' \in W_i$, then during the fixing i'th step of the Algorithm it is assured to be evaluated to the bit $1 - b$ on the test input with the i'th control bit set to 1,
- $\gamma(V^-(w'))$ is one of the XOR gates added during the processing of the Algorithm. In this case, one of the input wires w'' of the XOR gate must be processed in the Algorithm at least once. For this reason when the other - control - input wire to the XOR gate is set to 0, then the wire w' takes its value from w''. As shown in the point above, w'' must be evaluated to both bits at some inputs from the test set.

□

Corollary 2. *A circuit C with max fan-in 2 has a k-division of size $2 + 1 = 3$, thus the Algorithm 4 will produce on such input a circuit C' with 3 additional input bits and a test set $\mathbb{T}$ of size $1 + 3 = 4$.*

5.5 Reducing high conductivity of the control wires

Since in any k-division $W_1, \ldots, W_k$ of a set of wires W of a circuit C all wires going out of a single gate must belong to the same W_i, the size of W_i is bounded by the number of gates of the circuit $n = |V(C)|$. The *$i'th$* control bit in the Algorithm 4 can be thus used even as many as n times. This means that the specification of the circuit requires the implementation of highly conductive control wires. We can reduce this requirement, by reapplying COPY gates to the control wires added by the Algorithm 4.

Corollary 3. *$(\widehat{C} : \mathbb{Z}_2^{s+k} \to \mathbb{Z}_2^t, \mathbb{T})$ created by running the Algorithm 4 on $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ and replacing every highly conductive control wire of the intermediary wire with a log-depth construction of copy gates is a pair such that:*

1. *$\mathbb{T}$ is a covering set for $\widehat{C}$,*
2. *$\forall_{X \in \mathbb{Z}_2^s} C(X) = \widehat{C}(X||0^k)$.*

Proof. Note that replacing every highly conductive control wire with a construction of copy gates creates a set of wires which are all evaluated to bit b whenever the i'th control input is set to b. □

6 The main result

Finally, we can collect partial results from the previous sections and define a complete circuit compiler. The Algorithm 5 TestableCircuitCompiler takes as parameters a circuit $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ and a number $L \in \mathbb{N}^+$. It firstly transforms the circuit C into a circuit C' with a covering set, using the Covering procedure defined in the Section 5. Finally, it transforms the intermediary circuit into a testable circuit with an extended test set using the procedure TestableCircuitCompiler defined in the Section 4. Algorithm 5 TestableCircuitCompiler is a circuit compiler transforming into testable circuits $\mathfrak{C}$.

Algorithm 5: TestableCircuitCompiler

Input: (C, L)
`/* A circuit` $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ `and a number of layers` $L \in \mathbb{N}^+$ `*/`
Output: $(\widehat{C}, \widehat{\mathbb{T}})$
`/* A testable circuit` $\widehat{C} : \mathbb{Z}_2^{s+3+L} \to \mathbb{Z}_2^{t+2^L}$ `and its test set` $\widehat{\mathbb{T}}$ `*/`
1 $(C', \mathbb{T}) \leftarrow$ Covering(C)
2 $(\widehat{C}, \widehat{\mathbb{T}}) \leftarrow$ Compress$(C', \mathbb{T}, L)$
3 **return** $(\widehat{C}, \widehat{\mathbb{T}})$

Theorem 1. *Given any circuit $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ (of size n, depth d), and a number $L \in \mathbb{N}^+$, the procedure* TestableCircuitCompiler(C, L) *outputs a pair $(\widehat{C}, \widehat{\mathbb{T}})$ such that:*

- *$\widehat{C}$ is a circuit with additional $3 + L$ input bits and additional 2^L output bits, i.e. $\widehat{C} : \mathbb{Z}_2^{s+3+L} \to \mathbb{Z}_2^{t+2^L}$. The size of $\widehat{C}$ is bounded by $12n$, and its depth is bounded by $d + \log(n) + L \cdot (3n)^{1/L}$. The size of the test set $\widehat{\mathbb{T}}$ is $4 \cdot 2^L$.*
- *The pair $(\widehat{C}, \widehat{\mathbb{T}})$ is a testable circuit, i.e. for any tampering τ*

$$\left(\exists X \in \mathbb{Z}_2^s : \widehat{C}^\tau_{|t}(X||0^{3+L}) \neq \widehat{C}_{|t}(X||0^{3+L}) \right) \Rightarrow \left(\exists T \in \widehat{\mathbb{T}} : \widehat{C}^\tau(T) \neq \widehat{C}(T) \right)$$

and $\widehat{C}$ is functionally equivalent to C, i.e. $\forall_{X \in \mathbb{Z}_2^s} : \widehat{C}_{|t}(X||0^{3+L}) = C(X)$.

Proof. The testability and functional equivalence of the circuit $\widehat{C}$ follows from the Theorems 3 and 2.

Below we give a discussion on the parameters of our compiler. We assume that after each subprocedure the number of wires w is approximately the same as the number of gates n in the circuit. The first subprocedure of the algorithm produces a circuit C' with depth $d' \leq d + \log(n)$, size $n' \leq n + n + n$, and its *covering* set of size $k' = 4$. What is more the intermediary circuit has added only 3 control bits to the input, i.e.: $C' : \mathbb{Z}_2^{s+3} \to \mathbb{Z}_2^t$. By applying the Algorithm Compress with L layers, we can calculate the following parameters of the output circuit:

- its modified input and output size - $\widehat{C} : \mathbb{Z}_2^{s+3+L} \to \mathbb{Z}_2^{t+2^L}$,
- its modified circuit size is the number of the gates n' of the circuit C' plus the number of gates used for each layer. In the construction with L layers, the algorithm chains of length $w'^{1/L}$ are used. The first layer adds $w'^{1/L}\frac{w'}{w'^{1/L}} = w'$ new gates and gives $\frac{w'}{w'^{1/L}}$ output wires), the 2'nd layer adds $2\frac{w'}{w'^{1/L}}$ gates build upon $\frac{w'}{w'^{1/L}}$ output bits from the first layer, the i'th layer adds $2^{i-1}\frac{w'}{w'^{(i-1)/L}}$ gates. Finally, a linear number of copy gates is added to deliver $i'th$ control bit to chains in each layer. In general:

$$\begin{aligned}\widehat{n} \leq n' + \sum_{i=1}^{L} 2^{i-1}\frac{w'}{w'^{(i-1)/L}} + \sum_{i=1}^{L} 2^{i-1}\frac{w'}{w'^{i/L}} = \\ n' + \frac{w'^{1/L}(w' - 2^L)}{w'^{1/L} - 2} + \frac{w' - 2^L}{w'^{1/L} - 2} \leq \\ n' + 2w' + w' \leq \\ 3n + 3 \cdot 3n = \\ \mathcal{O}(n)\end{aligned}$$

- its modified circuit depth $\widehat{d} \leq d' + \sum_{i\in\{1,\ldots,L\}} w'^{1/L}$,
  i.e. $\widehat{d} \leq d + \log(n) + L \cdot (n')^{1/L} = d + \log(n) + L \cdot [3n]^{1/L}$,
- the new size of the test set $\widehat{k} = 4 \cdot 2^L$.

□

7 Conclusions and Open Problems

In this work, we introduced the notion of efficiently testable circuits and provided a compiler that transforms any circuit into an efficiently testable one that detects tampering with *every* wire of the circuit.

While the tampering model is already quite powerful, there are two natural ways in which one could hope to strengthen it:

Conductivity. Our compiled circuit has gates with a fan-out up to 3, i.e., an output wire can be used as input wire for up to 3 gates. While one can reduce the fan-out to 1 using COPY gates, this will break security as the tampering model we (and also other works) use crucially requires that gate input wires that come from the same output wire are tampered in the same way.

We leave it as an open problem to find a compiler satisfying a stronger notion, where all input wires can be individually tampered, or show that such a compiler does not exist.

Gate Tampering. While our tampering model allows tampering with every wire, it does not allow tampering with the gates.

Recall that our construction starts with a precompiled circuit that comes with a small set of inputs to the circuit (the covering set) such that for every wire there are inputs in that set which set this wire to 0 and 1, respectively. If we use a more demanding set which, for every gate, contains circuit inputs that set the input wires to that gate to all possible values (i.e., $00, 01, 10, 11$ for a gate with two inputs), our compiled circuit does achieve security against gate tampering, but only for the subcircuit which is the input circuit, not the additional gates added by the compiler.

Coming up with a compiler that can tolerate gate tampering on the entire compiled circuit is an interesting open problem.

References

[1] 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. URL: https://doi.org/10.1007/978-3-030-01446-9_5.

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

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

[4] S. Chakraborty, S. Dziembowski, M. Galazka, T. Lizurej, K. Pietrzak, and M. Yeo. *Trojan-Resilience without Cryptography*. Cryptology ePrint Archive, Paper 2021/1224. https://eprint.iacr.org/2021/1224. 2021. URL: https://eprint.iacr.org/2021/1224.

[5] 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. URL: https://doi.org/10.1007/978-3-642-32009-5_31.

[6] 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. URL: https://doi.org/10.1007/978-3-642-54242-8_23.

[7] 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.

[8] 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. URL: https://doi.org/10.1145/3519935.3520007.

[9] 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.

[10] D. Genkin, Y. Ishai, M. M. Prabhakaran, A. Sahai, and E. Tromer. "Circuits Resilient to Additive Attacks with Applications to Secure Computation". 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. URL: https://doi.org/10.1145/2591796.2591861.

[11] R. Gennaro, A. Lysyanskaya, T. Malkin, S. Micali, and T. Rabin. "Algorithmic Tamper-Proof (ATP) Security: Theoretical Foundations for Security against Hardware Tampering". In: *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*. Ed. by M. Naor. Vol. 2951. Lecture Notes in Computer Science. Springer, 2004, pp. 258–277. DOI: 10.1007/978-3-540-24638-1_15. URL: https://doi.org/10.1007/978-3-540-24638-1_15.

[12] 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. URL: https://doi.org/10.1007/11761679_19.

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

[14] 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. URL: https://doi.org/10.1109/FOCS.2012.69.

[15] 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.

[16] D. J. Kleitman, F. T. Leighton, and Y. Ma. "On the design of reliable Boolean circuits that contain partially unreliable gates". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science* (1994), pp. 332–346.

[17] 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.

[18] 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.