Provable countermeasures against Hardware Trojans

Małgorzata Gałązka
no. album: 277288
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw

Dissertation
in the field of **Natural Sciences**
in the discipline of **Computer and Information Sciences**

Dissertation carried out under the supervision of
prof. dr hab. Stefan Dziembowski
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw

Warsaw, April 2024

To my beloved Husband.

Funding

NCN (National Science Centre) **OPUS** Grant *Blockchain wallets – cryptographic theory and applications*
NCBiR **POWER** Grant *Kartezjusz*
FNP **TEAM** Grant *Cryptographic Defence Against Malicious Hardware Manufacturers*

Oświadczenie kierującego pracą

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia jej w postępowaniu o nadanie stopnia doktora w dziedzinie nauk ścisłych i przyrodniczych w dyscyplinie informatyka.

Data Podpis kierującego pracą

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza rozprawa doktorska została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem stopnia doktora w innej jednostce.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora pracy

Contents

[1] **streszczenie** [strɛʃt͡ʃɛɲɛ] - abstract

Abstract

The thesis addresses the problem of countermeasures against Hardware Trojan Horses (HTHs), understood as adversarial and permanent modifications of computational devices. We focus on solutions that can be formally proven to be secure. We model a computational device as a logical circuit and incorporate the possibility of malicious modification. The scope of such modification varies – sometimes, the adversary can replace the circuit with another one, and sometimes, the adversary needs to preserve the original topology. In any case, the HTH is *digital*, by which we mean that it can use only official input/output channels to communicate. We start with a short survey of prior solutions based on cryptographic primitives such as Verifiable Computation, Multi-Party Computation, and Interactive Coding. Then, we proceed to the main technical part of this thesis, i.e., we propose and investigate in-depth novel countermeasures: Very Simple Circuit Compilers and Efficiently Testable Circuits.

All of the protection schemes against HTHs (both already existing in the literature and the ones presented in this thesis) consist of at least one of the following two parts: (i) circuit compiler and (ii) testing procedure. *Circuit compiler* is a procedure which, for a given circuit F which realizes functionality $\mathcal{F}$, outputs another circuit $\widehat{F}$, which also realizes $\mathcal{F}$. $\widehat{F}$ consists of parts the adversary can modify – we say that they are HTH-infected – and, sometimes, a simple trustfully manufactured part called a *master module*. Most of the *testing procedures* work in the following way: the lifecycle of the circuit $\widehat{F}$ is divided into two phases – *lab phase* and *wild phase*. Circuit $\widehat{F}$ during the lab phase is given inputs (a) from an arbitrary set or (b) randomly chosen with a particular distribution; its outputs are compared to reference values. If they agree (i.e., the device passes the lab), it is released into the wild, where $\widehat{F}$ is given inputs (a) chosen by the adversary or (b) from a particular distribution. The lengths of these phases are measured in the number of invocations.

As the first result, we propose a protection scheme against HTHs which can permanently tamper with $\widehat{F}$. This scheme transforms every F into *Efficiently Testable Circuit* $\widehat{F}$ and its test set $\mathbb{T}$. The construction achieves convincing parameters: security is granted with overwhelming or equal 1 probability for the unbounded length of the wild phase; the overhead in the size of the compiled circuit is linear with a factor less than 20; the size of the test set is constant (or linear in the security parameter for more advanced setting). We use a new notion of *information loss* in the security proof. It inherits a basic idea from Shannon's entropy but without relation to the probability measure. The results are presented in papers [Bai+23a] *Efficiently Testable Circuits.* 2023, [Bai+23b] *Efficiently Testable Circuits without Conductivity.* 2023.

The second original result is a notion of *Very Simple Schemes*. Here, the compiled circuit $\widehat{F}$ consists of a master module and some untrusted instanti-

ations of F. This approach has two main objectives: simplifying the device's master module and minimal overhead in the size of the compiled circuit. We show a construction that meets such schemes' achievable security parameters. This countermeasure thwarts HTHs, which can completely modify the circuit; we do not allow the adversary to influence the input the circuit receives in the wild. This result was presented in [Cha+21] *Trojan-resilience without cryptography.* 2021.

Keywords: hardware trojans, tampering model, physical attacks on digital hardware

Dowodliwe metody obrony przed sprzętowymi koniami[1] trojańskimi

Streszczenie

Poniższa praca omawia problem środków zaradczych wobec sprzętowych koni trojańskich (Hardware Trojan Horse, w skrócie HTH) rozumianych jako złośliwe i permanentne modyfikacje urządzeń obliczeniowych. Skupiamy się na rozwiązaniach, których bezpieczeństwo można formalnie wykazać. Urządzenia obliczeniowe modelujemy jako obwody logiczne i uwzględniamy możliwość modyfikacji przez przeciwnika. Zakres tej modyfikacji różni się w zależności od modelu – czasem przeciwnik może zastąpić układ innym, a czasem musi zachować jego pierwotną topologię. W każdym wypadku HTH jest *cyfrowy*, co oznacza, że może używać tylko oficjalnych kanałów wejścia i wyjścia w komunikacji ze światem zewnętrznym. Pracę rozpoczynamy od przeglądu istniejących w literaturze środków zaradczych wykorzystujących takie narzędzia kryptograficzne, jak Verifiable Computation, Multi-Party Computation oraz Interactive Coding. W pozostałej części pracy prezentujemy nowe metod obrony przez HTH: Very Simple Compilers oraz Efficiently Testable Circuits.

Wszystkie metody obrony przed HTH zaprezentowane w pracy (zarówno istniejące, jak i nowe) składają się z co najmniej jednego z następujących dwóch elementów: (i) kompilatora obwodów, (ii) procedury testującej. *Kompilator obwodów* to procedura, która dla danego obwodu F realizującego funkcjonalność $\mathcal{F}$, generuje obwód $\widehat{F}$, który również realizuje $\mathcal{F}$. $\widehat{F}$ składa się z części produkowanych w sposób niezaufany – mówimy, że są zainfekowane HTH – oraz, czasami, prostego modułu produkowanego w sposób zaufany zwanego *modułem głównym*. *Procedura testowania* działa zwykle w następujący sposób: życie obwodu zostaje podzielone na dwie fazy – laboratoryjną oraz na wolności. Obwód $\widehat{F}$ w fazie laboratoryjnej otrzymuje wejścia (a) z dowolnego zbioru lub (b) losowo wybrane z określonego rozkładu; jego wyjścia są porównywane z wartościami referencyjnymi. Jeśli się zgadzają (tj. urządzenie przechodzi test laboratoryjny), rozpoczyna życie na wolności, gdzie otrzymuje wejścia (a) wybrane przez przeciwnika lub (b) z określonego rozkładu. Długości tych faz mierzone są w liczbie wywołań.

Jako pierwszy środek zaradczy prezentujemy schemat ochrony przed HTH, który przekształca dowolny obwód F w obwód efektywnie testowalny (Efficiently Testable Circuit) $\widehat{F}$ i jego zbiór testowy $\mathbb{T}$. Zaproponowana przez nas konstrukcja ma przekonujące parametry – bezpieczeństwo może zostać złamane z zaniedbywalnym prawdopodobieństwem, nawet gdy długość fazy na wolności

[1]Zgodnie z opinią językoznawcy i leksykografa dra hab. Mirosława Bańko, prof. UW, *koniami* jest poprawną formą nadrzędnika liczby mnogiej słowa *koń* w przypadku związku frazeologicznego *koń trojański* odnoszącego się do złośliwego oprogramowania [Bań03].

jest nieograniczona; kompilacja układu zwiększa jego rozmiar w sposób liniowy z czynnikiem mniejszym niż 20; wielkość zbioru testowego jest stała. W dowodzie bezpieczeństwa wykorzystujemy nowe pojęcie *utraty informacji* (*information loss*). Wykorzystuje ono podstawową ideę entropii Shannona, ale bez odniesienia do miary prawdopodobieństwa. Wyniki są przedstawione w pracach [Bai+23a] *Efficiently Testable Circuits.* 2023, [Bai+23b] *Efficiently Testable Circuits without Conductivity.* 2023.

Jako drugi środek zaradczy prezentujemy *Very Simple Schemes*. W tym wypadku skompilowany układ $\widehat{F}$ składa się z modułu głównego oraz niezaufanych instancji F. Głównym celem w tym wypadku jest maksymalne uproszczenie modułu głównego urządzenia oraz minimalne zwiększenie wielkości skompilowanego układu. Prezentujemy konstrukcję, która spełnia osiągalne parametry bezpieczeństwa takich schematów. Ten środek przeciwdziała HTH, które mogą całkowicie zmodyfikować układ; urządzenie na wolności otrzymuje losowe wejścia. Ten wynik został przedstawiony w pracy [Cha+21] *Trojan-resilience without cryptography.* 2021.

Słowa kluczowe: sprzętowe konie trojańskie, tamperowanie obwodów logicznych, ataki fizyczne

1 Introduction

In the present world, we do have conventional wars. Sadly. However, our real-life security depends more and more on the results of battles that happen in cyberspace. Let the numbers do the talking: the global budget spent on digital security is estimated at 154 bln USD in 2022 and is expected to grow up to USD 425 bln in 2030, which gives 13.8% of annual growth[1][23]. This thesis is devoted to one of the aspects of digital security – the countermeasures against *Hardware Trojan Horses*. To introduce this notion, we recall the story of the (antique) Trojan Horse. A wooden structure of a horse given by the Greeks to the city of Troy was filled with soldiers and allowed the Greek army to enter the attacked city. Who could accept such a suspicious gift, one could ask? In the present world, it does not fundamentally differ from assigning the production of computational devices to an untrusted party. Similarly, a device received in such a process does not *look* harmful; nevertheless, the user is unaware of its actual behavior and should remain cautious. Generally speaking, by Hardware Trojan Horse (HTH), we understand an adversarially chosen inconsistency between a computing device's actual and promised functionality [BT18].

The possibility of dishonest production of computing devices is perfectly justified in the modern world. In times of globalization and exorbitant optimization of most economic decisions and processes, some economic schemes, such as outsourcing, are gaining tremendous popularity. Many entities worldwide, e.g., electronic device manufacturers and the army, are interested in the best existing solutions, particularly in the field of circuits. At the same time, building a factory of the most advanced (the fastest, smallest, least energy consuming) circuits is costly. Therefore, there are only a dozen of them in existence [TSZ13]. They produce and sell the devices but should not be considered trusted. Hence, a model that includes the existence of HTHs is well-motivated.

The threats of HTH-infection had already become a reality. The press constantly reports successful attacks based on HTHs. As examples, we can mention spying on the citizens by the US government [Sch15], spying on key US entities by HTH-infected Chinese motherboards [RR18], or spying on the crucial US infrastructure by cranes made in China [Sha23]. We can speculate on how many such attacks were not revealed to the public; remain undetected; and happen at this very moment. To give a flavor of such an adversarial behavior and its consequences, we provide some made-up exemplary threats both on a micro-scale – when an isolated device is used – and on a macro-scale – when the untrusted devices form a network.

[1]Compared to the global military expenditure of USD 2240 bln in 2022 with 3.4% growth from 2021 [Luc22].

Micro-scale. Recall that cryptocurrencies are digital currencies protected by cryptographic protocols against stealth or double-spend. In this context, we can think of *electronic wallets* as exemplary devices vulnerable to micro-scale damage made by HTHs. Having and spending, e.g., bitcoins, ether, or zerocoins, requires keeping some private key trustworthy [Nak+08; Woo+14; Mie+13]. It is a fundamental aspect of cryptocurrencies – in contrast to checkbook money, they have no bank guarantees, so they cannot be recovered when stolen *by a cyberattack*. Because of this, a typical participant in the cryptocurrency market does not want to store her private keys on a laptop or smartphone – these devices are often connected to an untrusted network or peripherals, which makes them vulnerable to malicious software. Trading on the dedicated exchange markets is also considered potentially dangerous because of the risk of going bankrupt[1]. To protect crypto assets, electronic wallets were proposed [Bam+14]. These external, easily disconnectable physical devices store private keys and allow the owners to perform some operations, such as financial transfers, when plugged in. However, as long as an untrusted party performs their production process, they should not be considered honest. An HTH-infected electronic wallet would endanger its owner's money by performing unauthorized operations, leaking the key, or simply deleting it.

Macro-scale. To understand the threat made by HTHs on a macro-scale, we can think of military weapons that receive physical signals from the environment by radio. The adversary can trigger HTH-infected weapons, e.g., by a wave of predefined frequency, and make them (a) deny service, (b) leak some information to her, or (c) start fighting against their army. It is an obvious threat to the *real-world* security on a battlefield, especially on a macro-scale, when such events happen massively in a coordinated way. Many other systems, such as traffic control/electric networks or smart homes/buildings, are also perfect targets for such an attack.

This thesis aims to propose and follow new research paths on the topic of provable countermeasures against Hardware Trojan Horses and to put them into the context of existing solutions to this problem. The provability aspect is essential; the industry heuristic solutions to this problem are insufficient from a cryptographic perspective. However, we do not want to overlook the engineering aspect of this topic. We strive to make our research both formal and practical: the model's development and analysis are at least as crucial as novel, intelligent methods of preventing HTHs in circuits.

[1]Some examples from 2022: FTX, Blockfi, Celsius, and Voyager [Mad23].

1.1 Countermeasures against Hardware Trojan Horses as a part of cryptography

How does the problem of countermeasures against Hardware Trojan Horses relate to codes, passwords, ciphers, or other notions cryptography is known from? We must always be aware that these are only important and easily understandable *tools* to achieve its *goal*, which was perfectly stated almost twenty years ago in *Foundations of Cryptography* [Gol+05]:

> Modern cryptography is concerned with the construction of information systems that are robust against malicious attempts to make these systems deviate from their prescribed functionality.

Since the words above were written, the protected functionalities have continually evolved and improved. Nowadays, humankind uses information systems that are immensely complex in design, production, maintenance, reparation, and usage. Hence, cryptography is challenged by problems that sometimes cannot even be formulated in a simple language of *secure communication* or *encryption scheme*. Protection against Hardware Trojan Horses is one of them.

Cryptographic schemes have two main areas vulnerable to attack: (i) functionality and (ii) implementation; because of the nature of this field of science – it solves the problems for (i) theoretical models of (ii) reality. Attacks on *functionality* happen in the idealized world where honest users always follow the cryptographic protocol and computing devices behave according to their specifications. In this world, we can state and prove theorems. Therefore, attacks of such type happen mostly for heuristic solutions. Attacks on *implementation* are possible since the reality strongly differs from the model. For instance, some people choose easily breakable passwords, or the army orders electronic components from manufacturers of a hostile country that behave far from the specifications. The adversary is released from the prison of the cryptographers' presumptions about reality; she can break the security of a *theoretically* secure system.

To preserve formal cryptographic guarantees, we need to incorporate possible implementation attack scenarios into the model of reality and try to achieve some *provable* security parameters. This is how, e.g., password-based cryptography was born [BM92] – human habits are in contradiction with assumptions about uniformly chosen random keys – these were replaced by *passwords* from good enough distributions.[1] In the real world, successful attacks usually focus on the implementation

[1]Here, the model and the reality met in the middle. Password-based cryptography gives up uniformly distributed *keys* and relies on *passwords* drawn from a good enough distribution. Gen-

because if the security is provable (up to the factor of some complexity assumption), then attacking the functionality works for negligible probability (or would require an algorithm that is strongly believed not to exist).

Electronic devices are inherent in implementing the vast majority of modern information systems. As such, they are potential targets for the adversary, which is far from any black-box model of computation. For example, the adversary can passively measure some observable quantities (such as sound waves, power consumption, and time delay) and reason about the secret key or actively attack the device by inducing an electromagnetic field nearby. The adversarial behavior that uses such methods is called *physical attack*. There exists a broad literature on this topic, headed by papers reporting successful attacks bypassing the underlying theoretical security [Koc96; KJJ99; QS01; GMO01; Pag02], and followed by papers proposing theoretical [ISW03; Ish+06; DDF14; ADF16] and practical countermeasures against them. In this context, HTH is an extreme example of a physical attack.

1.2 What are Hardware Trojan Horses

This section introduces how formal modeling Hardware Trojan Horses can be done. As said before, by HTHs we understand adversarial modifications of the electronic device unknown to the user. The model is parametrized by the scope of modification done by HTH (see Section 1.2.2). In the context of HTHs, the device is modeled by an algebraic/boolean circuit [Wah+16; Ish+06; DFS16] with some alterations and parameters, such as the set of allowed gates (e.g., memory and random gates), method of initialization, volatility, conductivity (see Section 3), etc.

1.2.1 What Hardware Trojan Horses are not

There are other notions similar to HTHs. Let us now explain the differences.

Unintentional errors. Errors are a normal part of the design/production process, as well as their detection. Unlike HTHs, errors are introduced unintentionally, which

erally, a random variable X has a good enough distribution if its min-entropy is big enough, i.e., $\mathrm{H}_\infty(X) > \lambda$. Typically, we assume $\lambda > 512$, but in the real world, every tenth real-world password is among the 25 most popular passwords, according to [Mas16]. Hence, new, more advanced password requirements emerged – they must be long enough and consist of small and big letters, digits, special signs, etc. These conditions make the real-world min-entropy of passwords higher. This is how a bridge was created between the reality of weak passwords and the theory of uniformly distributed keys.

has significant consequences. Generally, errors occur randomly or in areas especially vulnerable to them – HTHs are introduced maliciously. Therefore, the methods of error and HTHs detection must differ. In particular, the adversary aware of the error detection methods used can find a way to compromise the security. Moreover, the damage caused by the error is unpredictable to the user and the adversary. It contrasts the devastation caused by HTHs, which can be precisely designed.

Software Trojan Horses (STHs). Modes of operation of STHs and HTHs are similar. As an HTH pretends to be a different circuit, an STH imitates a different software, such as some form or an advertisement. However, STHs are much easier to remove from the hardware, while HTHs are the inherent part of the device, which one cannot eliminate.

Physical Trojans. By *physical* Trojans, we understand the device modification, which may consist of some unauthorized communication channels realized by, e.g., an antenna, a clock, a sensor, or a transmitter. We strongly insist that HTH are *digital* [Bai+23a], so as the circuit modification, they can influence only the input-output *behavior* of the infected device, not the input/output by itself. In particular, HTH cannot communicate side-channel with the outer world. In other words, the input and the output of the HTH-infected circuit are according to specification, even if the *computation* done by it is not[1].

1.2.2 Hardware Trojan Horses – taxonomy

One approach to modeling the HTHs allows the adversary to replace the computing device with whatever she wants, up to retaining the specified input/output channels. This model is very challenging because of the role reversal. Usually, the device is a black box to the adversary; here, it is a black box to the *user*. However, such an approach is the most general and allows us to model every digital HTH that we can imagine. This model (Total Hardware Trojan Horses – THTHs) is investigated in the literature [DFS16; Wah+16] with successes in constructions and impossibility results.

This thesis also considers a notion of constrained HTHs, inspired by [Ish+06]. The authors of [Ish+06] describe a method of preventing a physical attack where the adversary can *tamper* with the circuit. By tampering, they mean that values taken by some adversarially chosen wires of a Boolean circuit are (i) toggled (negated), (ii) set to 0, or (iii) set to 1. It leads to questions about the countermeasures against

[1]Practitioners sometimes consider Physical Trojans to be Hardware Trojan Horses [BT18].

HTHs, which must preserve the *topology* of the circuit. Such a model is reasonable from a practical point of view – the circuit's topology is a fundamental property that the adversary cannot arbitrarily change. This line of work is presented in [Bai+23a; Bai+23b] and will be described in detail in Section 3.

Research on the HTHs, whose size strongly depends on the size of the original circuit would be a well-motivated research topic since the size (and weight) is an easily measurable characteristic of an electronic device. For instance, we could ask, what damage can be done by a HTH which is only one, two, or three gates bigger than the original circuit? Unfortunately, questions of this type for arbitrary circuits take us to the land of NP-hard problems, as we informally argue below. Assume that a circuit F is given. In the first place, (if P $\neq$ NP) there exists no efficient *compressing* procedure to find a F' that computes the same function as F and is minimal in the number of gates[1]. Therefore, the HTH-overhead in size cannot be easily bounded – the protocol cannot efficiently compute the minimal size of a circuit for arbitrary function, but at the same time, there can exist an adversary, who *knows* the minimal circuit computing f.[2] Even the existence of an efficient algorithm that finds canonical circuit form for arbitrary function is an open problem [Gar+16]. To our knowledge, there are neither positive nor negative results on countermeasures against HTHs of this type. Still, it would be an exciting research topic at the intersection of cryptography, complexity theory, and approximation algorithmics.

1.2.3 Hardware Trojan Horses – modus operandi

How could the Trojan Horse get to the protected city? As mentioned in the beginning, producing circuits is now widely outsourced, and many problems with HTHs start in this place. The way between an abstract functionality and a physical device that realizes it consists of a few steps, most vulnerable to adversarial behavior [BT18]. For example, the design phase is often supported by optimizing programs (algorithms), which are not trusted and can act maliciously, even if the designer himself is honest. Moreover, during the fabrication process, some errors can unintentionally occur, such as stuck-at-0 and stuck-at-1 errors, meaning that some wires in the circuit always carry 0,1 values, respectively. Malicious manufacturers can intentionally implement such errors and pretend they are a normal part of the production process.

[1] We can recall Circuit Satisfiability Problem – given a circuit F decide if there exists an input, such that its output is equal 1. If there were a compressing procedure, we could check if compressed F' is a trivial circuit that always outputs 0

[2] The existence of such an adversary is not straightforward – recall *The Five Worlds* [Imp95].

An HTH-infected device *may* work incorrectly. We do not expect that HTH deviate from the beginning for every input since some testing is usually a part of the production process. Activation of HTH (by which we mean that it starts malfunctioning) can be done randomly, by a *cheat code*, or by a *timebomb*. A cheat code is a special input recognized by the HTH. This method is possible since we cannot exclude the possibility that the adversary can influence or even control the inputs given to the circuit (however, we sometimes assume that the inputs are random). A time bomb activates the HTH after some predefined (by the adversary) number of invocations of the circuit. Any combination of all these three methods is possible.

Once HTH is active, it starts misbehaving. By this, we can understand both: *Denial of Service* attack and outputting wrong values. The exemplary consequences of the latter are making the device (or the system it is a part of) deviate from the original functionality or leaking secret from a stateful circuit (as its output).

1.3 Countermeasures against HTHs and their limitations

In this section, we present existing countermeasures against HTHs. We focus primarily on provable solutions, but at the very end, we will also briefly present heuristics and solutions coming from practitioners.

1.3.1 Realistic HTH-resilience expectations

The main purpose of research on countermeasures against HTHs is to find methods that make the circuits *resilient* against this type of physical attack. HTH-resilience combines security-based notions such as secrecy, robustness, and correctness in the presence of HTH. Nevertheless, as said before, models with (Total) HTHs are very difficult to work with. Therefore, it is unsurprising that standard cryptographic notions, such as *negligible probability* of deviation, are sometimes relaxed. Such a relaxation must be done in the case of *Very Simple Compilers* – a notion presented in Section 4.

There is no hope for THTH-resilience if the circuit has no THTH-free parts. The impossibility result from [DFS16] shows that no resilience can be granted if the device is produced exclusively by an untrusted party who can infect it with THTH. They showed even more that the circuit must perform some non-trivial computation to have any hope for protection against THTH, even if it consists of some number of subdevices produced by the adversary.

Involving the adversary in the device's manufacturing process has significant implications. We can expect this – one of the exciting results in cryptography is the

	honest device	deviating device
accept	working	insecure
reject	dummy	disappointing

Figure 1: Possible game outcomes between the adversary and the user.

impossibility of general obfuscation [Bar+01a] (in contrast to the indistinguishability obfuscation that is believed to exist). The authors show that the adversary with access to the description of a circuit computing some $\mathcal{F}$ is much more potent than an adversary with only oracle access to $\mathcal{F}$. This has two-level consequences for the problem of protection against HTHs. Firstly, we cannot hope for a generic compiler that would hide the protected functionality from the adversary (who is a part of the production process). Secondly, for similar reasons as in [Bar+01a], the adversary can build an HTH which modifies $\mathcal{F}$ arbitrarily by using the circuit $\mathcal{F}$ as a building block of an HTH. This allows the adversary, e.g., to use a cheat code based on the *output* of $\mathcal{F}$.

When relying on an untrusted party, sometimes nothing can be improved. In the context of HTHs, we can achieve HTH-resilience, but for some adversaries, we can never end up with a working device! We can elaborate on this a bit more closely. There are two axes of possible scenarios. The first one depends on the adversary – she can produce (i) honest or (ii) deviating devices. The second axis depends on the result of the security scheme. The devices can be (a) accepted and used or (b) rejected as potentially malicious. A dummy protocol that always chooses (b) is perfectly secure; nevertheless, security is just a *tool*, or, in other words, just a foundation for some actions that the user wants to perform. The protocol should not reject honest devices. On the other hand, (b) is the only reasonable answer if the adversary chooses (ii). In this case, the scheme cannot achieve correctness because no (accepted) device exists. In other words, the reasonable HTH-resilient scheme cannot accept (ii a) and should avoid (i b) – see Figure 1.

1.3.2 Circuit compiler

In the case of HTH, we focus on the physical modifications of the circuit, which makes our considerations, perhaps surprisingly, simultaneously very abstract and low-level. They are abstract because we model electronic devices as logical/arithmetic circuits.

They are low-level because we focus on the values carried by single wires (see Section 3). Modeling physical attacks allows cryptographic researchers to focus on their impact on the (abstract) circuit, not physical effects or industrial issues. On the other hand, such considerations give great importance to low-level issues, such as the set of allowed gates. Nevertheless, the most common way to achieve HTH-resilience is to *compile* the functionality $\mathcal{F}$. This line of research is widely used, i.e., in the context of indistinguishability obfuscation [Bar+01b] or leakage-resilience [ISW03][1]. It is naturally continued also for other types of physical attacks, such as HTHs [Wah+16; DFS16; Ish+06; Efr+22].

A circuit compiler is a procedure that, given an arbitrary functionality (identified with a circuit), outputs a circuit that realizes it and, at the same time, is resilient against some (prescribed) attack, in particular – HTH. The issue of efficiency is a point to discuss – by this, we mean the overhead in running time, circuit size, output, input, etc. The main benefit of this method is its generality – the designer of HTH-resilient circuits is not interested in what the circuit does by itself; resilience is granted for every input circuit whenever it computes encryption or a majority function.

Many of the compilers invented to counteract possible HTH-infection use advanced cryptographic tools, such as Verifiable Computation [Wah+16], Multi-Party Computation [DFS16], Secret Sharing Schemes [Ish+06], Interactive Coding [Efr+22], etc. Another method of crypto-provenance used to achieve HTH-resilience is the security amplification [DFS16].

1.3.3 Testing

There is an interesting paradox – on the one hand, infection by HTH makes the device entirely under the adversary's control. On the other hand, HTH is very limited – it must behave in the same way every time, not adaptively[2]. This observation brings us to the idea of testing the devices – on a high level, it is possible since the deviation is set *once forever*.

Electronic devices are tested by the circuit industry from the time they are produced. For instance, one can compare the outputs of two or more devices – this method can detect the errors that occur independently and with low probability. Unfortunately, it is not the case in the context of HTHs – the errors can be induced in a coordinated way; for instance, all devices can be modified similarly.

[1] We do not even mention here the most natural "compilers", which, e.g., make the circuit layered or bound fan-in or fan-out of its gates.

[2] In stateful circuits, we treat their inner state as a part of the input.

Much more general attempts to predict the future, which cannot be foreseen, are given by statistics. Among all of its tools, it is worth mentioning the Kaplan-Meier survival estimator [KM58], used to estimate the length of life of some entities, such as people, marriages, animals, etc. Notably, the Kaplan-Meier survival estimator works even if some of the elements of the observable set disappear as long as it happens independently and randomly. We can see an analogous of it in testing the device a random number of times before releasing it.

Testing in the presence of adversarial behavior was investigated in [DFS16], where the lifetime of the device is divided into 2 phases: (i) lab phase (just after leaving the conveyor belt), where the behavior of the circuit is compared to the expectation by executing some number of tests on it and (ii) wild phase where the device is used. If the device passes the tests, some guarantees about its behavior in the wild can be given. This line of research was continued in [Cha+21; Bai+23a; Bai+23b] and resulted in the notion of *Testable Circuits*, which topic will be presented in Sections 3, 4.

1.3.4 Industrial and heuristic solutions

As mentioned before, testing is a part of the standard circuit production process. The tester looks most often for *errors* – which are unintentional. Industrial research on preventing and detecting HTHs [BT18] uses techniques such as split manufacturing, obfuscation- and power-based countermeasures, design-for-trust, etc. Side-channel attacks also inspire a line of research – in this case, the side-channel effects are used to *improve* the security of a system [HBM18]. Recently, a optical detection method of HTHwas presented [Pus+22]. Nevertheless, these methods are heuristics or do not work in our model, so we will not discuss them in detail.

1.4 The thesis organization and personal contribution

I will explain how this thesis is organized. Because of the formal requirements in Poland to present a Ph.D. thesis, I will also describe in detail which parts of the original results in joint papers are mine.

In Section 2, I present a template commonly used to protect a single device from possible damage made by some HTH. I aim to give an easily understandable description of a framework present in literature, and I go as formal as it is needed, no further. The nature of this framework is *synthethical*, by which I mean it is not established prior to the research; instead, I identified some parts of it in different papers and gave a description that can contain the existing solutions.

Section 3 is based on papers [Bai+23a] *Efficiently Testable Circuits*, [Bai+23b] *Efficiently Testable Circuits without Conductivity*. In the paper [Bai+23a] presented at the 14th Innovations in Theoretical Computer Science conference, I proposed the main ideas of construction, such as wire-covering sets, STRmOR, and STRmOR gadgets definitions and their realiations. I also did most of the main proofs, including the idea of information loss. In the paper [Bai+23b] presented at the Theory of Cryptography Conference 2023, I developed the notions of information loss and gate-covering sets. I gave the main contribution in the proof of Theorems 10. The sections and statements mostly developed by the other authors are given for completeness and explicitly marked with footnotes.

Section 4 is based on paper [Cha+21] *Trojan-resilience without cryptography*, which was presented at the Theory of Cryptography Conference 2021. The authors developed the general idea, motivation, and proofs jointly. I wrote down the Section 4.3 and resolved some mediocre and minor issues that arrived during that process. Section 4.2 is cited for completeness.

2 Modeling the Hardware Trojan Horses and the countermeasures against them

This section describes the *template* of the security scheme against HTH along with the security definition. Every paper on countermeasures against HTHs gives precise terminology and modeling. We do not insist on extreme formality here – it is unnecessary and would make things less clear. We want to synthesize existing proofs to make the rest of this thesis more understandable.

Security games. Security games are widely used formal tools to show the security of cryptographic schemes – we can think of them as thought experiments. More precisely, a security game is a procedure that can call the adversary, the algorithms that are part of the security protocol, do some computation, draw random numbers, etc. Ultimately, such a procedure outputs something, usually a bit, that we can interpret as adversary loss/win or maintained/compromised security. When the protocol's security is defined by a game, the security proof is a chain of security games – the first corresponds to the original scheme and modeling. Every other game in the chain is a *hybrid* game, which differs slightly from the previous in terms of how often the adversary wins. The last one is a game, for which we can easily show or see that the adversary cannot win with high probability.

Every template has some structure and parameters. In our case, some parameters are not simple objects such as numbers or graphs but procedures.

2.1 Elements of the model

We can elaborate on the elements of the model we work with.

The set of allowed circuits $\mathcal{C}$. First of all, the model consists of $\mathcal{C}$ – the set of allowed circuits (corresponding to the abstract functionalities) that must be protectable against HTHs. Normally, we expect the scheme to operate on *any* circuit, but still, the model must be precise if we allow memory/random gates; if wires of a circuit are conductive (see Section 3); what is possible fan-in/fan-out of the gates; etc.

Scope of modification done by $\mathcal{A}$. The adversary $\mathcal{A}$ is described by the scope of modification she can make in the part of the circuit under her control. The

compiled circuit $\widehat{F}$ consists of (i) parts assumed to be produced honestly and (ii) parts the adversary $\mathcal{A}$ can modify. In this thesis, we work with two main models of modification. The first one allows the adversary to replace her parts with whatever she wants (see Section 4). These we call Total Hardware Trojan Horses. The second one requires the adversary to preserve the *topology* of a circuit (see Section 3).

The compiler $\mathfrak{C}$ and the security guarantess $\mathcal{S}$ it gives. The compiler $\mathfrak{C}$ compiles any circuit F from the set $\mathcal{C}$ into $\widehat{F}$ so that: (i) $\widehat{F}$ is functionally equivalent to F, i.e., $\widehat{F}$ can easily emulate the input-output behavior of F; (ii) $\widehat{F}$ is secure against the adversary $\mathcal{A}$ in the sense of the security guarantees $\mathcal{S}$. The security guarantees include, e.g., the upper bound on the probability of deviation of the circuit $\widehat{F}$ in the wild phase.

2.2 Security scheme

Now, we are ready to explain the steps of the security scheme.

1. The adversary $\mathcal{A}$ chooses a circuit F from the set of allowed circuits $\mathcal{C}$.
2. The compiler $\mathfrak{C}$ compiles some circuit F, and outputs a tuple $(\widehat{F}, \mathbb{T})$, where $\widehat{F}$ is expected to be functionally equivalent to F.
3. The adversary chooses a modification τ of $\widehat{F}$
4. The infected $\widehat{F}^{\tau}$ is tested by the lab phase $\mathbb{T}$.
5. If $\widehat{F}^{\tau}$ *passes* the lab phase $\mathbb{T}$, some proven security guarantees $\mathcal{S}$ are given to it in the wild phase.

To clarify the game, we discuss these steps in detail and show how it works in Figure 2.3.

The choice of functionality F. The adversary chooses F, because the compiler $\mathfrak{C}$ must work for *every* allowed circuit. We identify the circuits with their functionalities here.

The output $(\widehat{F}, \mathbb{T})$ of the compiler. The compiler provides a tuple $(\widehat{F}, \mathbb{T})$ for every circuit from $\mathcal{C}$, in particular for the circuit chosen by the adversary. $\widehat{F}$ must be *functionally equivalent* to F; by this, we mean that the compiled circuit can easily emulate the original circuit F behavior. For example, for $F : \mathbb{F}^n \to \mathbb{F}^m$, $\widehat{F} : \mathbb{F}^{n+n'} \to \mathbb{F}^{m+m'}$, we can say, that $\widehat{F}$ is functionally equivalent, if $\widehat{F}(X|0^{n'}) = F(X)|0^{m'}$ for every input X (of length n), where "|" is a concatenation operator. It is indeed: the circuit $\widehat{F}$ emulates the F behavior easily – to call F on input X, one can call $\widehat{F}$ on input $X|0^{n'}$ and output the first m bits of the result.

Description of $\widehat{F}$ also provides the information of which parts are produced honestly and which (possibly) not – in other words, which parts can be modified by the adversary.

Modification τ of the circuit. The adversary can choose the modification of the parts of the circuits according to her limitations.

Description of and performing the lab phase $\mathbb{T}$. The lab phase $\mathbb{T}$ describes the testing procedure, such as the (possibly random) number of tests or the set of tests to be performed. We usually assume access to a *testing oracle* – we can think of it as a trusted software emulation of the circuit $\widehat{F}$ run on some trusted device (computer) or simply the honest circuit $\widehat{F}$. Recall that the motivation for investigating the countermeasures against HTHs lies in optimizing the production process of circuits. Therefore, the assumption that *some* (possibly inefficient) version of the circuit can be produced honestly is not unrealistic. We usually say that $\widehat{F}^{\tau}$ passes the lab phase if all of its outputs in this phase are correct, but there are possible other definitions of passing the lab phase.

Security guarantees. Security granted by a specific protocol may differ in many ways. We can ask, e.g., if the protocol is resilient against Denial of Service attacks, if it compromises the secret key, what the length of the wild phase is, etc.

2.3 Definitions

Let us give a template definition of a circuit compiler being a countermeasure against HTHs.

Definition 1 *We say that $\mathfrak{C}$ is a circuit compiler for a set of circuits $\mathcal{C}$ if for any $F \in \mathcal{C}$ (corresponding to its abstract functionality $\mathcal{F}$) it outputs a tuple $(\widehat{F}, \mathbb{T})$. Here,*

$\widehat{F}$ is an augmented description of the compiled circuit, and $\mathbb{T}$ is the lab (testing) phase description.

Some issues, such as the lab phase, are most often understood directly from the construction; for instance, we usually identify $\mathbb{T}$ with a set of tests given to the compiled circuit.

Given all these, we can define the security granted by $\mathfrak{C}$.

Definition 2 *We say, that $\mathfrak{C}$ for a set of circuits $\mathcal{C}$ is $\mathcal{S}$-resilient against $\mathcal{A}$ if the tuple $(\widehat{F}, \mathbb{T}) = \mathfrak{C}(F)$ (where $F \in \mathcal{C}$) meets the following conditions: whenever $\widetilde{\widehat{F}} = \mathcal{A}(\widehat{F})$ passes $\mathbb{T}$, it achieves security guarantees $\mathcal{S}$ in the wild.*

As mentioned, the proofs of HTH-resilience often use some security game as a building block. In this case, the security guarantee is equivalent to a bounded probability of winning some security game, parametrized by the setup, lab, and wild phase. The winning condition is often simply an inconsistency between the outputs of $\widehat{F}$ and $\widehat{F}^{\tau}$ on some inputs given to the circuit in the wild (see Figure 2.3).

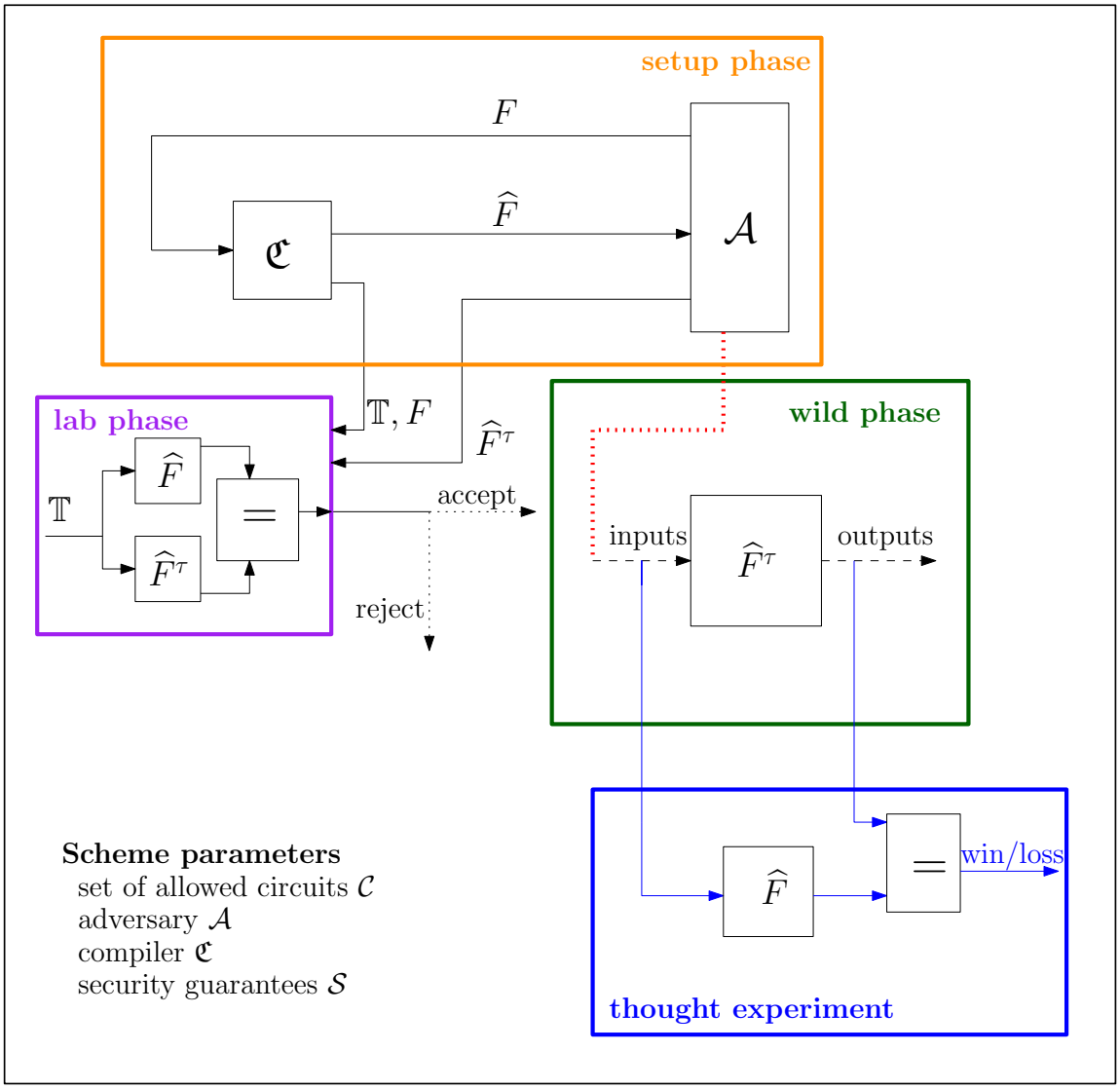


Figure 2: This figure shows the synthesis of protection schemes against HTH. Firstly, there is a setup phase – choice of F and compilation. The adversary knows $\mathfrak{C}$, but it can be randomized. Therefore, she is not assumed to get $\mathbb{T}$. Secondly, there is the lab phase. Testing compares the input's actual and expected values defined by $\mathbb{T}$. If these values differ, the device is rejected; otherwise, it is accepted, and the wild phase begins. In this phase, the untrusted device is getting some inputs that can be controlled by $\mathcal{A}$. The last part of the analysis is the thought experiment – in the proofs, the actual and expected outputs of the device are compared. If the adversary manages to make them different, she wins the game – the security is broken.

3 Efficient testability against restricted Hardware Trojan Horses[1]

In this section, we investigate a restricted model of HTHs – the one where the *topology* of the Boolean circuit is preserved. More precisely, we allow the adversary to modify a circuit by *tampering* with its wires. Tampering is a notion extensively described in [Ish+06]. In the model which we work in this section with, the adversary can apply to *every* wire of the attacked circuit one of the following modifications: **neg**, **one**, **zero**. By **neg** tampering, we understand that the value obtained from the preceding gate is negated (another name for this operation is *toggle*). By **zero** and **one** tamperings, we understand that the wire transmits 0 and 1, respectively, no matter what it gets from the preceding gate. Some of the wires of the adversarial choice can remain untampered.

Dummy tampering. Obviously, the adversary's goal is to make a circuit malfunction, not just to tamper. We can imagine a tampering of a circuit that does not change its input-output behavior (see Figure 3). We do not insist on preventing such *dummy tampering* attacks in this section.

3.1 Introduction

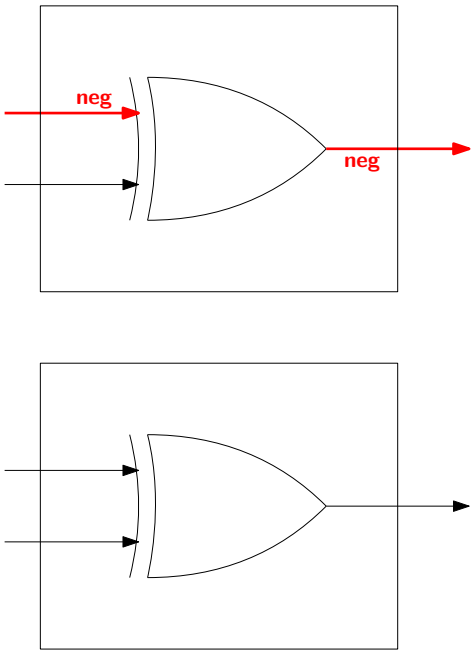
The protection schemes in this section fit perfectly into the template presented in Section 2. Recall its structure and set the parameters.

The set of allowed circuits $\mathcal{C}$. In this case, the set $\mathcal{C}$ consists of stateless circuits with gates from the set $\{\mathsf{AND}, \mathsf{OR}, \mathsf{XOR}, \mathsf{COPY}, \mathsf{NOT}\}$ with bounded fan-in and fan-out.

Scope of modif existication done by $\mathcal{A}$. The adversary $\mathcal{A}$ can tamper with the circuit $\widehat{F}$, by which we mean assigning the wire tampering function τ to some of its three possible wire tampering functions: **neg**, **one**, **zero**.

The compiler $\mathfrak{C}$ and security guarantees $\mathcal{S}$ it gives. The design of the compilers will be given in the next paragraphs and sections. For now, we can state

[1] Section based on papers [Bai+23a] *Efficiently Testable Circuits*, [Bai+23b] *Efficiently Testable Circuits without Conductivity.*



input	output
00	0
01	1
10	1
11	0

Figure 3: Dummy tampering on a circuit consisting of a single XOR gate. The truth table for both circuits is the same. More about incorporating dummy tamperings into a circuit, see Theorem 9.

the informal security guarantee they give: once the device passes the lab phase, it deviates in the wild with negligible/zero probability.

Now, we describe the steps of the protection schemes.

1. Some functionality along with its circuit F is chosen (the theorems work for every circuit and every adversary; therefore, we do not insist that the adversary chooses F).

2. The compiler $\mathfrak{C}$ compiles F into a tuple $(\widehat{F}, \mathbb{T})$, where $\widehat{F}$ is functionally equivalent to F, and the test set $\mathbb{T}$ simply consists of some number of tests.

3. The adversary $\mathcal{A}$ is given a circuit $\widehat{F}$. She tampers with it by assigning to some unbounded number of its wires the tampering functions from the set $\{\mathsf{neg}, \mathsf{one}, \mathsf{zero}\}$. Importantly, we do not assume any tamper-free parts.

4. The tampered circuit $\widehat{F}^{\tau}$ is tested on the test set $\mathbb{T}$.

5. If $\widehat{F}^{\tau}$ passes the lab phase, it can be used in the wild with security guarantees $\mathcal{S}$.

One factor that differs between the two solutions presented in [Bai+23a; Bai+23b] is the issue of *conductivity*. We assume that every outcoming wire of some gate (except the COPY gates) is tampered with in the same way. The COPY gates have fan-out equal 2, and their outcoming wires can be tampered with different functions.

Conductivity. The n-conductivity of a circuit F means that every gate of F (except the COPY gates) must have the same tampering on every outcoming wire, and its fan-out is bounded by n. We say that 1-conductive circuits are non-conductive. The assumption of n-conductivity is a model constraint – the higher the n, the more limited the model is. The protection scheme presented in [Bai+23a] works for 3-conductive circuits, and the protection scheme from [Bai+23b] works for non-conductive circuits (so the latter is more general *due to factor of conductivity*; the advantages of the former will be presented later).

3.1.1 Tampering model – discussion

The tampering model we work with may look slightly artificial, but it meets the real world. The process of circuit production consists of many steps; circuits are vulnerable to stuck-at-0 and stuck-at-1 (unintentional) faults [Cha+98], which are functionally equivalent to **zero** and **one** (intentional) tamperings. The testing community explores heuristic algorithms that guarantee high fault coverage [MT00], meaning

that a high percentage of the wires in the circuit are checked against such faults. Since it is barely possible to perform all production processes in a trusted manner, the possibility of attacks that mimic unintentional faults cannot be excluded. In other words, the adversary can easily introduce **zero** and **one** tamperings during fabrication, e.g., for wires examined by the heuristic algorithms against faults only with a small probability. In contrast, the **neg** tampering does not mimic any production error. Negation of values transmitted by wires sometimes happens temporarily in the wild.[1] Our model does not contain the possibility of *temporary* tampering, but, as we see in the following paragraphs, the permanent **neg** wire modification is also worth investigating.

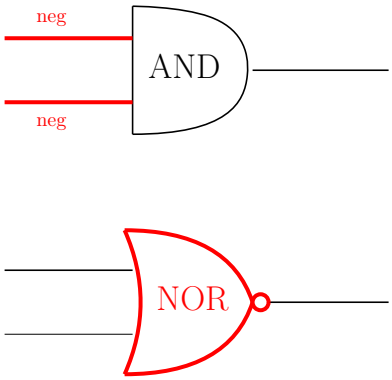
A natural extension of a wire-tampering attack is a gate-tampering attack. By this, we understand that an adversary can replace some gates in a circuit with any gates with the same fan-in and fan-out, even if the model of computation does not allow them to be used as a part of the untampered circuit. For instance, for NAND circuits, the adversary can replace gates with any of 16 possible gates with fan-in equal to 2 and fan-out equal to 1. The motivation for such a model is twofold. Firstly, HTH, which does not change the circuit's topology, seems easier to install than more general HTHs. The adversary needs only to substitute gates, not redesign the circuit, its topology, orientation in the space, etc. Secondly, such an attack is more difficult to detect on the physical level because the topology of a circuit can be verified with, e.g., optical methods.

The gate-tampering model is more general than the model of wire-tampering attacks in the following sense: for every circuit F, every functionality $\mathcal{F}^\tau$ which can be obtained by an adversary who tampers with the wires of F can be obtained by an adversary who tampers with the gates of F. For example, an adversary can replace its preceding gate with its negation function to simulate the toggle operation on some internal wire. Figure 4 gives other examples of such simulations. In this context, the **neg** tampering can be seen as an overture to the model, including the possibility of gate-tampering.

3.1.2 Results

The protection schemes against tampering HTHs we refer to in this section use the compilers that output a compiled circuit and its test set. We want the test set to be exhaustive: if the device passes the lab phase, it is impossible or improbable that it deviates on any input in the wild. This takes us to the notion of *testability*.

[1] This insight was given to us by Prof. Ingrid Verbauwhede (COSIC, KU Leuven).



input	output
00	1
01	0
10	0
11	0

Figure 4: Another example of wire-tampering simulated by gate-tampering. To simulate the **neg** tampering on both input wires to the AND gate, the adversary can substitute the AND gate with a NOR gate. The truth table of both functions is identical.

Testable circuits. The center of our research on the tampering HTHs is the aspect of testability. We say that a tuple of a circuit F, some functionally equivalent circuit $\widehat{F}$ and some set of inputs $\mathbb{T}$ is a testable circuit if we are (almost) sure that if the *tampered* $\widehat{F}^\tau$ perfectly mimics input-output behavior of $\widehat{F}$ for every input from $\mathbb{T}$, then $\widehat{F}^\tau$ is functionally equivalent to F (see Definition 5). We intentionally do not describe the behavior of $\widehat{F}^\tau$ for every possible input it can get as a circuit - recall that $\widehat{F}^\tau$ in the wild receives only the input of the form $X|0^m$, where X is a possible input to F.

We present two schemes that protect any circuit from $\mathcal{C}$ against damage done by tampering HTH. They both achieve convincing parameters in security and size/time overhead. The models in which they are proven secure differ slightly, which will be discussed in detail in Section 3.6. The first protection scheme is denoted with $\mathfrak{L}$ (see [Bai+23a]), which comes from its construction based on $\mathfrak{L}$ogical operations. We state the achieved security informally here (see Section 3.4 and Theorem 8 for formality).

Theorem 1 (informal) *Given any circuit F of input size s, output size t, size n and depth d, and some small number L (think of 3, 4 or 5), the scheme $\mathfrak{L}$ outputs a tuple $(\widehat{F}, \mathbb{T})$, such that*

- *$\widehat{F}$ is a small extension of F. More precisely, $\widehat{F}$ is a circuit with a few additional input bits and several dozen additional output bits. The size of $\widehat{F}$ is bounded*

by $12n$*, and its depth is bounded by* $d + O(n^{1/L})$*.*

- *The size of the test set* $\mathbb{T}$ *is less than 150.*
- *The tuple* $(F, \widehat{F}, \mathbb{T})$ *is a testable circuit.*

The paper [Bai+23b] presents the scheme denoted with $\mathfrak{R}$, which comes from $\mathfrak{R}$andomness used in constructing the test set. We can informally state achieved security guarantees (for formality, see Section 3.5 and Theorem 11).

Theorem 2 (informal) *Given any circuit* F*, the scheme* $\mathfrak{R}$ *outputs a tuple* $(\widehat{F}, \mathbb{T})$*, such that*

- $\widehat{F}$ *is a small extension of* F*.*
- *The tuple* $(F, \widehat{F}, \mathbb{T})$ *is a testable circuit with overwhelming (in* $|\mathbb{T}|$*) probability.*

We can see that the scheme $\mathfrak{R}$ has a significant advantage over the scheme $\mathfrak{L}$ – it does not rely on the conductivity assumption. However, the scheme $\mathfrak{L}$ also has its quality – as we will see in Section 3.6, its testing procedure makes it more likely to extend to stateful circuits.

3.1.3 Methods

The proofs of resilience against HTHs require using some dedicated tools. Let us introduce them informally below. Their formal definitions will be presented later.

Wire- and gate-covering sets. Recall that the tampering on a wire is a function of type $\mathbb{F} \to \mathbb{F}$ assigned to that wire. If any of the functions associated with wires on the circuit is not the identity function, the circuit has been tampered with. So, intuitively, every wire's lab phase behavior must be verified for two cases: when it should transfer the value 0 and when it should transfer the value 1. We define a *wire-covering set* to work with this intuition. It is a set of inputs for the circuit for which every wire would take values 0 and 1. Intuitively, the wire-covering sets are the only reasonable testing sets. If the circuit were tested on a set of inputs that do not form a wire-covering set, there would be a wire that would take only one value for each test, say 0. Then, during the lab phase, the **zero** tampering on this wire could not be detected. For technical reasons, we also define a *gate-covering set*, an analog of a wire covering set. It is a set of inputs for which incoming wires to every gate take all possible valuations. Section 3.3 presents more details on covering circuits.

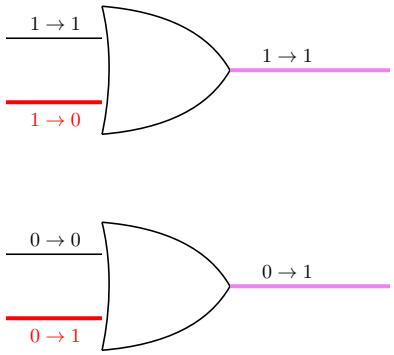


Figure 5: The pictures show the information loss on the circuit consisting of a single OR gate. The second input to this gate is tampered by **neg** tampering (red wire). The first circuit shows honest and tampered evaluation on input 11. The second one shows the evaluation on input 00. Note that the information loss is observable on the *output wire* (violet) and not on the tampered wire.

Information loss. Another interesting tool we developed is the concept of *information loss*. It inherits a basic idea of Shannon's entropy [Sha48], but its definition does not need any probability measure. Recall that for any discrete random variable X and any function f, it holds $H(X) \geq H(f(X))$. In particular, the above inequality is strict if and only if the function f restricted to the image of X is not injective. Information loss can be defined for honest specification of some circuit F, its tampered version F^τ and some specific wire e. It also involves 2 circuit inputs. We say that such a tuple forms an *information-losing* tuple if, for the original circuit, the wire e gets two different evaluations, and in the tampered circuit, both evaluations are the same. The denotation *information loss* comes from the idea that the wire e could be used to distinguish two inputs in the original circuit, which is no longer true for the tampered circuit. See Figure 5 for more explanations.

3.1.4 Related works

The idea of an adversary who modifies computation done by a circuit by **zero**, **one**, and **neg** operations performed on the wires was investigated in [Ish+06]. The main difference between our work and theirs is that the authors of [Ish+06] work with a different model and achieve different security guarantees. First of all, in [Ish+06], the adversary can tamper with a (bounded) number of wires for each call – the wires tampered with in previous steps remain tampered; secondly, the circuit detects by itself that the adversary is present, and the construction allows to keep the inner

state secret by self-destroying all inner values.

Other works addressing a similar problem are [KLR12; KLM97]. Here, the adversary can modify the circuit differently – she can introduce *short circuit* errors, i.e., replace some of the gates by gates outputting one of its incoming values. According to our terminology, they work with a restricted model of gate-tampering adversary. The authors of [KLR12] propose a compiler that transforms arbitrary circuit F into a circuit $\widehat{F}$ resilient to short-circuit errors. The proposed construction works as long as the fraction of replaced gates does not exceed 1/51 in every path from input to output. If the size of the original circuit is equal to n, then the size of the compiled circuit $\widehat{F}$ is quasi-polynomial: $n^{O(\log n)}$.

3.2 Notation

In this section, we define some useful notations for the Boolean circuits. We model circuits as graphs and extensively use standard graph theory notation.

3.2.1 General notation for circuits

A circuit is modeled as a DAG (directed acyclic graph) $F_\gamma = (V, E)$ where vertices refer to gates and the directed edges refer to wires. Each wire transmits a bit from $\mathbb{F}$ according to the wire's direction. The circuit definition F_γ is parameterized by a labeling function $\gamma : V \to \mathfrak{G}$ which assigns specific functions to the vertices, where $\mathfrak{G} = \{\mathsf{AND}, \mathsf{OR}, \mathsf{XOR}, \mathsf{COPY}, \mathsf{NOT}\} \cup \{\mathbf{in}, \mathbf{out}\}$. The functions $\mathbf{in}, \mathbf{out}$ are special guard functions introduced for notation issues. We sometimes omit the parameter γ since it is chosen when specifying the circuit and remains unchanged. For each circuit gate $v \in V$, we denote the sets of its incoming and outcoming edges by $E^-(v) = \{(u, v) \in E\}$ and $E^+(v) = \{(v, u) \in E\}$, respectively. For each circuit wire $e = (u, v) \in E$ we denote its tail gate and head gate by $V^-(e) = u$ and $V^+(e) = v$, respectively.

Fan-Out and Conductivity. Each gate type in $\mathfrak{G}$ has a specific number of input/output wires, as indicated in Table 3.2.1. Each output wire is used as an input 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 a non-conductive one by using copy gates $\mathsf{COPY}(x) = (x, x)$. However, 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 output wires 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 is 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 $F_\gamma = (V, E)$ and an input $X = (x_1, ..., x_s) \in \mathbb{Z}_2^s$ we define a valuation function

$$\mathrm{val}_{F_\gamma, X} : V \cup E \to \mathbb{F} \tag{1}$$

which assigns each input and inner gate the value it outputs and each wire the value it carries 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

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

We will sometimes write val_X if the circuit considered is clear from the context. The behavior of the circuit F can be associated with the function that it evaluates, i.e., $F : \mathbb{F}^s \to \mathbb{F}^t$. We define this function as follows:

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

3.2.2 Notation for tampering

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

possible bit tamper functions $\mathcal{T} = \{\mathsf{id}, \mathsf{neg}, \mathsf{one}, \mathsf{zero}\}$. The tampering of an entire circuit $F = (V, E)$ is defined by a function

$$\tau : E \ \rightarrow \mathcal{T},$$

mapping each wire to a tampering function. We sometimes write τ_e to denote $\tau(e)$ for convenience. Now, we can extend our notion of the valuation to also take tampering into account to define the valuation

$$\mathrm{val}^{\tau}_{X} : V \cup E \rightarrow \mathbb{F}$$

of a tampered circuit. 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:

$$\begin{aligned} \mathrm{val}^{\tau}_{F_\gamma, X=(x_1,2,\ldots,x_s)}(v) &= \begin{cases} x_i, & \text{if } v = I_i, \\ \mathrm{val}^{\tau}_{F_\gamma,X}(E^-(v)) & \text{if } v \in \mathcal{O}, \\ \gamma(v)(\mathrm{val}^{\tau}_{F_\gamma,X}(E^-(v))) & \text{otherwise.} \end{cases} \\ \mathrm{val}^{\tau}_{F_\gamma,X}(e) &= \tau_e(\mathrm{val}^{\tau}_{F_\gamma,X}(V^-(e))). \end{aligned}$$

By F^τ we can again understand a function that describes the input-output behavior of the *tampered* circuit:

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

Now, we are ready to state the definition of *dummy* tampering formally.

Definition 3 *We say that the tampering τ of a circuit F is* dummy *iff τ does not change the input-output behavior of F, i.e.,*

$$\forall_X F(X) = F^\tau(X).$$

The tampering that is not dummy is called nontrivial.

We also define the *honest* tampering.

Definition 4 *We say that the tampering τ of a circuit F is* honest *if it assigns the* **id** *tampering to every wire. We say that τ is* dishonest *if it is not honest.*

Note that every honest tampering is also dummy tampering. We can also formally define the testability property.

Definition 5 *The tuple $(F, \widehat{F}, \mathbb{T})$ is a* testable circuit *iff*

$$\forall_\tau \left(\left(\ \exists_X : \widehat{F}^\tau(X|0^m) \neq F(X) \ \right) \ \Rightarrow \ \left(\exists_{T \in \mathbb{T}} : \widehat{F}^\tau(T) \neq \widehat{F}(T) \right) \right.$$

3.3 Covering circuits

As mentioned, essential building blocks of the protection schemes against HTHs are wire- and gate-covering sets.

Wire-covering sets. Wire-covering set of a circuit F is a set of inputs, such that if the circuit is called on all of these inputs, every wire transmits at least once all the possible values (in our case, 0 and 1). We define it formally.

Definition 6 *We say that a set of inputs* $\mathbb{T}_{\mathtt{w}} = \{T_1, ..., T_k\}$ *is* a wire-covering set *for a circuit* F *if*

$$\forall_{\ e \in E(F),\ \ b \in \{0,1\}}\ \exists_{\ i}\ :\ \mathrm{val}_{T_i}(e) = b\ .$$

For many practical circuits, we expect that some number of random inputs form, with a high probability, a wire-covering set for the majority of its wires –e.g., the majority of wires in circuits computing pseudorandom values take random values given random inputs. However, the general problem of finding or even deciding if a circuit has a wire-covering set for an arbitrary circuit F is NP-hard. Let us argue informally why this is the case. Consider the NP-complete Circuit Satisfiability problem (CIRCUIT-SAT). It is a decision problem for a circuit F – we ask if there exists an input X, such that $F(X) = 1$. Assume that we can solve the problem of deciding if there exists a wire-covering set in polynomial time. If there exists a wire-covering set for a circuit F, then F is satisfiable. The opposite implication does not hold since the wire that cannot be covered is not necessarily the output wire; however, the wires of F can be ordered in topological order, where the heads of the wires are compared. Once the order is done, we can proceed with the circuit from the first wire and add the next wire in every step. For such a subcircuit F', we check if it has the wire tampering. If yes, we proceed further. If not, the wire not covered is the last wire in F', say v. We check which value b is carried by v by evaluating the circuit F' on any input. Then, we simplify the rest of F by replacing v with b, deleting gates, etc. Then, we proceed further. In the last step, $F = F'$, so we check if the output wire of the original circuit F can be covered. Fortunately, there exists an efficient compiler that transforms every circuit into a functionally equivalent circuit with a covering set of constant size. We demonstrate it in Section 3.3.1.

Gate-covering sets. We also consider *gate-covering* sets, which are an inherent part of the proof of Theorem 11. Gate-covering is an analog of wire-covering and can be defined as a set of inputs for which the set of incoming wires to every gate take all possible valuations. For instance, for a gate with a fan-in equal to 2, there are four

valuations on its incoming wires: 00, 01, 10, and 11. Therefore, any gate-covering set of a circuit with gates of fan-in 2 has at least 4 elements. Again, we can state the definition formally (for a linear gate XOR, the condition of covering all possible input valuations is relaxed – we demand only 3 out of 4 possible input pairs).

Definition 7 $\mathbb{T}_{\mathbf{g}}$ *is a* gate-covering set *for a circuit* F *iff it meets all the following conditions*

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

As we can see, the gate-covering conditions are relaxed for XOR gates for two reasons. Firstly, we introduce gate-covering to prove Theorem 11, and it turns out that partial gate-covering for XOR gates is sufficient. Secondly, we do not know the compiler that would transform every circuit into a functionally equivalent, every-gate-full-covered circuit with a constant-size test set. The construction that achieves the gate-covering described in Definition 7 will also be presented in Section 3.3.1.

Gate-covering vs. wire-covering. We can easily see that for non-degenerated circuits[1] a gate-covering set is also wire-covering. Indeed, every wire that is an input wire to some gate is covered – for every gate except the XOR gate, it is trivial. Wires from XOR gates are covered since there are no three different valuations on its inputs, such that they are all the same for one of its elements. Since the circuit is not degenerate, every output wire is output from some gate, and we assume that gates with inputs are not constant. Therefore, the output wires are also covered.

3.3.1 Compiling circuit into wire-covered circuit

Our first main goal is to show the compiler that transforms any circuit $F : \mathbb{F}^s \to \mathbb{F}^t$ into a circuit $F_{\mathbf{w}} : \mathbb{F}^{s+s'} \to \mathbb{F}^t$ with a wire-covering set $\mathbb{T}_{\mathbf{w}}$ of a *small* size. The new circuit $F_{\mathbf{w}}$ will be functionally equivalent to F (i.e., $\forall_{X \in \mathbb{F}^s} F_{\mathbf{w}}(X||0^{s'}) = F(X)$). In the case of standard circuits with a maximum fan-in of the gates equal to 2, we obtain a 4-element covering set.

[1]By *degenerated circuits* we understand here circuits which contain wires disconnected from any gate – these wires are only transmitting a value from input to output – or circuits which contain dummy gates, i.e., gates which output is independent of at least one of its inputs.

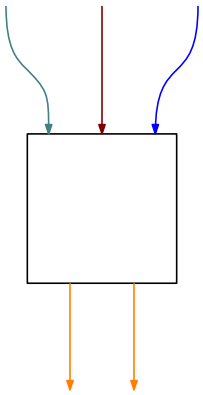


Figure 6: The picture visualises the properties of k-divisions introduced on a set of wires of a circuit F. All wires output by the same gate must belong to the same W_i. The output wires of a gate must belong to W_j, which is different from W_k of the input wires to the gate.

Now, we present a brief description of the compiler. It starts the creation of the wire-covering set by evaluating the circuit on input: $T_0 = 1^s 0^{s'}$. Next, we observe that on this input each wire $e \in E$ is evaluated to either 0 or 1 (i.e., it is evaluated to some $b = \mathrm{val}_{F,T_0}(e)$). Finally, the algorithm divides the wires in E from F into a few subsets of wires W_i. Every subset proceeds in one step, in which the wires from W_i are *fixed* (i.e., each wire $e \in W_i$ now is evaluated to $1 - b$) by adding only 1 additional *control* bit at each step. All $e \in W_i$ are fixed by XORing the new *control* bit with the wires of input bits to $V^-(w)$ (an example is given in the Figure 7). It is impossible to fix all the wires $e \in W_i$ *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. One needs to see that the modifications introduced in the circuit, as gates are processed in topological order, may already be sufficient to fix the outputs of the topologically subsequent gates.

k-divisible circuits. One of the steps of the algorithm above is dividing the circuit wires into subsets. This is essential, since to fix a wire e the algorithm adds an XOR gate on some wire $e' \in E^-(V^-(e))$. It influences a value transmitted by e' and can possibly make it uncovered. Therefore, e and e' must be processed in different steps. We state it formally below.

Definition 8 *For a circuit F we say that a function $f : E \to [k]$ is a* k-dividing *function if all of the following conditions hold:*

1. $\forall_{v \in V} |f(E^+(v))| \leq 1$,

2. $\forall_{v \in V} f(E^+(v)) \cap \mathsf{F}(E^-(v)) = \emptyset$.

The sets $W_i = \{e \in E : f(e) = i\}$ *for* $i \in [k]$ *is a* k-division *of* E. *A circuit for which exists a* k*-division is* k-divisible.

The first condition means that for each gate, all of its outcoming wires belong to the same subset. The second condition justifies the claim already mentioned – for each gate, the outcoming wires must be processed in different steps than the incoming wires. The properties of k-divisions are presented visually in Figure 6.

The algorithm k-DIVISION constructs a k-division set for circuits with fan-in bounded by k. We leave this statement without any proof - the algorithm is self-explanatory.

Algorithm 1: (k)-DIVISION

Data: $F : \mathbb{F}^s \to \mathbb{F}^t$ with fan-in $\leq k-1$
Result: k-division set $W_1, \ldots, W_k$
/*`Construction of` (k)-`dividing function` $f : E \to [k]$ */
for $v \in \mathcal{I}$ **do**
  $f(E^+(v)) = 1$
end
for $v \in V \backslash \mathcal{I}$ *processed in topological order* **do**
  Let $i \in [k] \backslash f(E^-(v))$ /*`The existence of such an index` i `is granted by the bounded fan-in.` */
  for $e \in E^+(v)$ **do**
    $f(e) = i$
  end
end
return $W_i = \{e \in E : f(e) = i\}$ for $i \in [k]$

Corollary 1 *Every circuit with fan-in* $\leq k-1$ *is* k*-divisible.*

3.3.2 Constructing small wire-covering sets for k-divisible circuits

Let $F : \mathbb{F}^s \to \mathbb{F}^t$ be a circuit and $(W_1, ..., W_k)$ be its k-division. The Algorithm 2 WIRE-COVERING constructs a circuit $F_{\mathbf{w}} : \mathbb{F}^{s+s'} \to \mathbb{F}^t$ with $s' = k$ that is functionally equivalent to F and has a covering set $\mathbb{T}_{\mathbf{w}}$ of size $k+1$. The algorithm works in steps corresponding 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 $e \in W_i$ is evaluated to

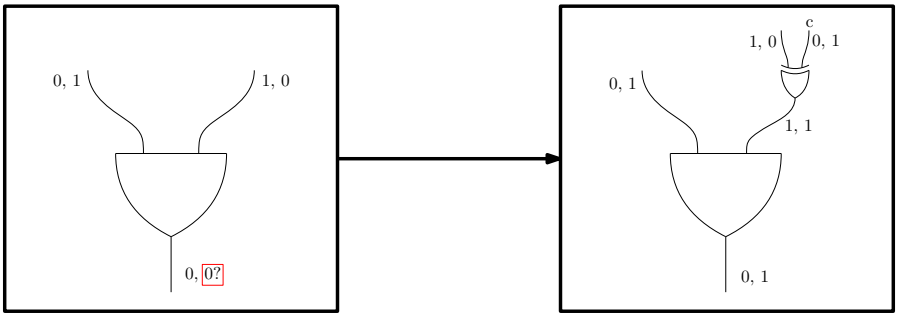


Figure 7: An example showing how the output wire of an AND gate can be fixed with a single *control* bit. Before fixing, it is always evaluated to 0. Therefore, in this step, we add an XOR gate to one of the incoming wires to make the actual output equal to 1. The righthand side input wire to the AND gate is covered, as we see in the proof of Theorem 3.

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.

$$F(X) = F_{\mathbf{w}}(X||0^k).$$

Theorem 3 *Let* $(F_{\mathbf{w}} : \mathbb{F}^{s+k} \to \mathbb{F}^t, \mathbb{T}_{\mathbf{w}}) = \textsc{Wire-Covering}(F : \mathbb{F}^s \to \mathbb{F}^t)$. *Then:*

1. $\mathbb{T}_{\mathbf{w}}$ *is a covering set for* $F_{\mathbf{w}}$,
2. $F_{\mathbf{w}}$ *is functionally equivalent to* F.

Proof. To see that $\forall_{X \in \mathbb{F}^s} F(X) = F_{\mathbf{w}}(X||0^k)$, note that when *control* bits are equal to 0, then the XOR gates that were added to $F_{\mathbf{w}}$ by the Wire-Covering do not affect the behavior of its subcircuit F.

Now we show that $\mathbb{T}_{\mathbf{w}}$ is a wire-covering set for $F_{\mathbf{w}}$, i.e., every $e \in E(F_{\mathbf{w}})$ takes both values when $F_{\mathbf{w}}$ is called on the test inputs from the $\mathbb{T}_{\mathbf{w}}$. Note that the Algorithm 2 adds only a single XOR gate for every wire in the original circuit F because all output wires from every gate are evaluated to the same value (see the details in Section 3.2.1) and belong to the same subset of k-division (according to the definition of k-division). 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). Consider the following cases, based on the gate $v = V^-(e)$ preceding the wire e:

Algorithm 2: WIRE-COVERING

Data: circuit $F_\gamma = (V, E) : \mathbb{F}^s \to \mathbb{F}^t$
k-division $W_1, ..., W_k$ of E
Result: $F_{\mathbf{w}} : \mathbb{F}^{s+k} \to \mathbb{F}^t, \mathbb{T}_{\mathbf{w}}$

for $e \in E$ **do**
  $C(e) = \emptyset$; /*the table C is indexed with wires; the field $C(e)$ holds already covered values on the wire e */
end
Initialize $T_0 = 1^s 0^k$; /*the first testing input */
for $e \in E$ **do**
  $C[e] = \{\mathrm{val}_{F,T_0}(e)\}$
end
Initialize $F_{\mathbf{w}} = F$;
Initialize $\mathbb{T}_{\mathbf{w}} = \{T_0\}$;
for $i \in [k]$ **do**
  $T_i = 0^{s+i-1}10^{k-i}$;
  $\mathbb{T}_{\mathbf{w}} = \mathbb{T}_{\mathbf{w}} \cup \{T_i\}$;
  Add i-th new *control* input wire C_i to $F_{\mathbf{w}}$;
  for $v \in V(F)$ (processed in a *topological* order) **do**
    if $E^+(v) \subseteq W_i \wedge b \notin C[E^+(v)]$ /*check if the output wire(s) of v is to be fixed now (i.e., belongs to W_i) and if it needs to be fixed now (i.e., is not covered yet) */
    then
      Set $b : b \notin C[E^+(v)]$;
      Update $F_{\mathbf{w}}$ by adding a XOR gate between the input wire C_i and some of the $E^-(v)$ that sets $\mathsf{Eval}_{F_{\mathbf{w}},T_i}(e) = b$; /*see Figure 7 */
      Append b to $C[E^+(v)]$;
    end
  end
end
return $F_{\mathbf{w}}, \mathbb{T}_{\mathbf{w}}$;

- v is one of the input gate for $F_{\mathbf{w}}$. Therefore, e either carries one of the first s input bits or one of the new k control bits. In the first case, e takes the value 1 for the input $T_0 = 1^s0^k$, and the value 0 for every other input from $\mathbb{T}_{\mathbf{w}}$ of the form $T_i = 0^{s+i-1}10^{k-1}$. In the second case, we have $v = C_i$ for some $i \in [k]$. Then e carries the value 0 for the input $T_0 = 1^s0^k$ and the value 1 for the input $T_i = 0^{s+i-1}10^{k-1}$.

- v is one of the inner gates of the original circuit F. In this case, e is evaluated to b for T_0. To see that it is also evaluated to $1-b$, note that there exists some i, such that $e \in W_i$. Then, during the fixing i-th step of the Algorithm 2, it is assured to be evaluated to the bit $1-b$ on the test input with the i-th control bit set to 1.

- v is one of the XOR gates added during the processing of the Algorithm 2. Let e', e'' be its input wires – let the wire e' transmit the i-th control bit. In this case, e'' transmits the output of some inner gate w of the original circuit F. Therefore e'' is wire-covered by $\mathbb{T}_{\mathbf{w}}$. Therefore, e is wire-covered by $\mathbb{T}_{\mathbf{w}}$ if it is wire-covered by these elements of $\mathbb{T}_{\mathbf{w}}$ that evaluate on e' to 0. These are exactly the elements of the set $\mathbb{T}_{\mathbf{w}}^{\setminus i} = \mathbb{T}_{\mathbf{w}} \setminus \{T_i\}$. Note that v is added to fix the output wire f from the gate $V^+(e)$. The k-divisibility construction shows that e and f belong to different division sets. Hence, e is wire-covered by $\mathbb{T}_{\mathbf{w}}^{\setminus i}$.

There are no other wires in the circuit; therefore, $\mathbb{T}_{\mathbf{w}}$ is a wire-covering set for $F_{\mathbf{w}}$. ■

Corollary 2 *A circuit F with fan-in at most 2 has a k-division of size $2+1=3$. Thus, the Algorithm 2* Wire-Covering *on such an input produces a circuit $F_{\mathbf{w}}$ with 3 additional input bits and a wire-covering set $\mathbb{T}_{\mathbf{w}}$ of size $1+3=4$.*

3.3.3 Compiling circuits into gate-covered circuits

The Algorithm 3 Gate-Covering describes a detailed procedure that compiles any circuit F into a new functionally-equivalent circuit $F_{\mathbf{g}}$ along with its gate-covering set $\mathbb{T}_{\mathbf{g}}$. We can now explain how it works. Firstly, it calls the Algorithm 2 Wire-Covering to prepare a wire-covering tuple $(F_{\mathbf{w}}, \mathbb{T}_{\mathbf{w}})$. Secondly, the algorithm performs a step where multi-input gates of the intermediary circuit $F_{\mathbf{w}}$ are checked to see if they are covered and, if not – fixed. Recall that every gate in the gate-covered circuit must be evaluated for a sufficient number of input combinations (i.e., for XOR gates – 3 input combinations, for the other gates – 4 combinations, see Definition 7). Since the circuit $F_{\mathbf{w}}$ is wire-covered, we know that all multi-input gates are evaluated

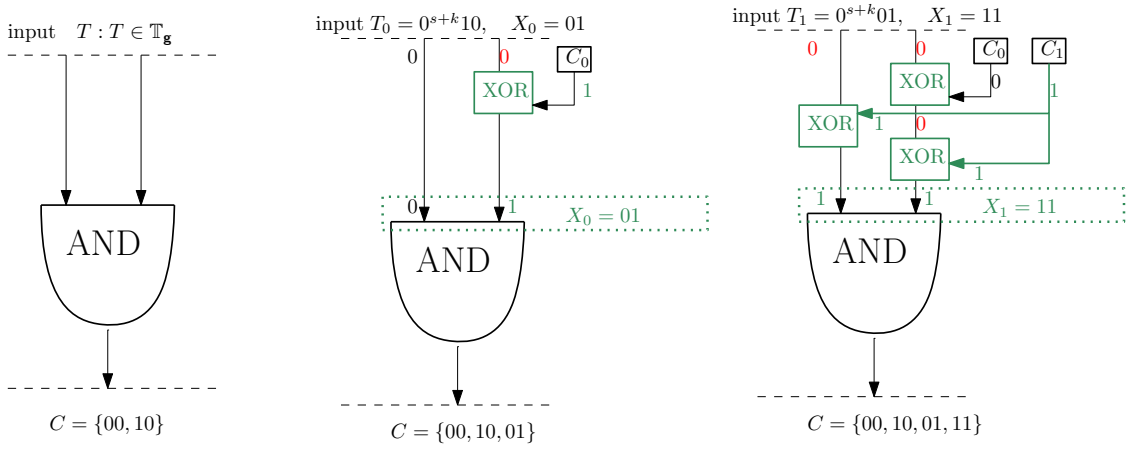


Figure 8: Fixing of a single gate AND. For all elements of $\mathbb{T}_{\mathbf{g}}$, the input wires take only two evaluations: $00, 10$ (left-hand side). There are two evaluations left that we must force to appear. The first one is $X_0 = 01$, which is made with the control bit C_0 on the input T_0 (in the middle). The second one is $X_1 = 11$, which is made with the second control bit C_1. In this case, the XOR gates are added on both incoming wires. They do not affect the previous evaluations on incoming wires because $C_1 = 0$ for all inputs in $\mathbb{T}_{\mathbf{g}}$ but T_1.

for at least two different input combinations. Therefore, each multi-input gate needs at least two more input combinations. The Algorithm 3 adds two additional control input bits C_0, C_1 corresponding to two additional test inputs. Then, it proceeds gate-by-gate in topological order. For each v and each input combination X_i that is not taken by v, it fixes it by xoring some of the input wires to v with control bit C_i. Figure 8 presents an exemplary fixing step.

Theorem 4 *The Algorithm 3* GATE-COVERING *transforms a circuit F into a functionally equivalent circuit $F_{\mathbf{g}}$ along with its gate-covering set $\mathbb{T}_{\mathbf{g}}$.*

Proof. It is easy to see that the circuit $F_{\mathbf{g}}$ is functionally equivalent to F since it does not add any new output bits to the circuit, and all the new gates are XOR gates connected to the new control bits. Whenever these bits are set to 0, the new XOR gates do not affect the behavior of the subcircuit F.

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 every input from $\mathbb{T}_{\mathbf{w}}$ (see line 4 of the Algorithm 3). Moreover, the

Algorithm 3: GATE-COVERING

Data: $F_\gamma : \mathbb{F}^s \to \mathbb{F}^t$ of fan-in $\leq k-1$
Result: $F_{\mathbf{g}}, \mathbb{T}_{\mathbf{g}}$

Initialize $W_1, ..., W_k = \text{DIVISION}(F)$
Initialize $(F_{\mathbf{w}}, \mathbb{T}_{\mathbf{w}}) = \text{WIRE-COVERING}(F, W_1, ..., W_k)$ /*Wire-covered intermediary circuit */
Initialize $F_{\mathbf{g}} = F_{\mathbf{w}}$
Initialize $\mathbb{T}_{\mathbf{g}} = \{T : \exists_{T' \in \mathbb{T}_{\mathbf{w}}} T = T'|00\}$
$T_0 = 0^{s+k}10$
$T_1 = 0^{s+k}01$
$\mathbb{T}_{\mathbf{g}} = \mathbb{T}_{\mathbf{g}} \cup \{T_0, T_1\}$
Add two control input wires C_0, C_1 to $F_{\mathbf{g}}$ /*these refer to the test inputs T_0, T_1 */
for $v \in V(F_{\mathbf{g}}) : \gamma(v) \in \{\mathsf{OR}, \mathsf{AND}, \mathsf{XOR}\}$ (processed in a *topological* order) **do**
    $C = \{(\text{val}_{F_{\mathbf{g}},T}(e))_{e \in E^-(v)} : T \in \mathbb{T}_{\mathbf{g}}\}$ /*At this point $|C| \geq 2$ */
    for $X_i \in \{00, 01, 10, 11\} \setminus C$ /*i can take $0, 1$ or 2 values; it takes values from the set $\{0, 1\}$ */ **do**
        Update $F_{\mathbf{g}}$ by adding the XOR gates on some of the input wires of v to fix the incoming values to X_i (see Figure 8).
    end
end
return $F_{\mathbf{g}}, \mathbb{T}_{\mathbf{g}}$

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 go topologically through the gates from the set $\{\mathsf{OR}, \mathsf{AND}, \mathsf{XOR}\}$ of the intermediary subcircuit $F_{\mathbf{w}}$, 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., $\{(\mathrm{val}_{F_{\mathbf{g}},T}(e))_{e\in E^-(v)} : T \in \mathbb{T}_{\mathbf{g}}\} \geq 2$). In line 13 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. ∎

Corollary 3 *For any circuit F with fan-in ≤ 2, number of gates n, the Algorithm 3* Gate-Covering *creates a circuit with additional 5 input bits, a test set of size 6 and at most additional $6n$ gates.*

Proof. The Algorithm 2 compiles F into a circuit with additional 3 input bits, a test set of size 4 and adds at most n XOR gates and n COPY gates. Algorithm 3 adds 2 input bits and 2 test inputs to the test set, and at most $2n$ XOR gates and $2n$ COPY gates during the iteration, which concludes the result. ∎

3.3.4 Reducing high conductivity of the control wires

Since in any k-division $W_1, \ldots, W_k$ of a set of wires E of a circuit F, 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 $n = |V(F)|$. The i-th control bit in the Algorithm 2 can be thus used even as many as n times. Similarly, every control bit in Algorithm 3 can be used $2n$ times. This means that the circuit specification requires the implementation of highly conductive control wires. The following corollary states that we can reduce this requirement by reapplying COPY gates to the control wires added by the Algorithms 2, 3.

Corollary 4 *$(F'_{\mathbf{w}} : \mathbb{F}^{s+k} \to \mathbb{F}^t, \mathbb{T}_{\mathbf{w}})$ and $(F'_{\mathbf{g}} : \mathbb{F}^{s+l} \to \mathbb{F}^t, \mathbb{T}_{\mathbf{g}})$ created by running the Algorithms 2, 3, respectively, on $F : \mathbb{F}^s \to \mathbb{F}^t$ and replacing every highly conductive control wire of the intermediary wire with a log-depth tree of copy gates is a pair such that:*

1. *$\mathbb{T}_{\mathbf{w}}, \mathbb{T}_{\mathbf{g}}$ is a wire-, gate-covering set for $F'_{\mathbf{w}}, F'_{\mathbf{g}}$, respectively;*

2. $\forall_{X\in\mathbb{F}^s} F(X) = F'_{\mathbf{w}}(X||0^k) = F'_{\mathbf{g}}(X||0^l)$.

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. ∎

3.4 $\mathfrak{L}$-scheme

In this section, we show an efficient scheme $\mathfrak{L}$ against tampering HTHs that protects an arbitrary 3-conductive circuit. The compiled circuit $\widehat{F}$ consists of F as a subcircuit and has a small number of auxiliary *control* input and output bits. The scheme $\mathfrak{L}$ does not rely on tamper-free components, randomness gates, etc. Moreover, the testing procedure is deterministic and consists of approximately 150 tests (for practical reasons, this number can be higher; see Section 3.6). The consistency check in the lab requires only comparing the output on the control bits, not the original output bits.

The only restriction we put on the tampering *for this construction* is that the compiled circuit $\widehat{F}$ is 3-conductive, i.e., all but COPY gates can output up to 3 wires, and the tampering must be done in the same way for all of them. 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 that can be tampered with with different functions (note that without this restriction, the *practical* conductivity of the circuit would be unbounded).

As a warm-up, we present two impractical (but educative) solutions: they make the *I/O size* or the *depth* of the compiled circuit linear in the size of the original circuit. The final solution is built on these basic ideas.

3.4.1 The trivial solution for 2-conductive circuits

Let $F_{\mathbf{w}}$ be a circuit wire-covered by $\mathbb{T}_{\mathbf{w}}$. A trivial circuit compiler $\mathfrak{C}$, when given $F_{\mathbf{w}}$, outputs a 2-conductive $\widehat{F}$ that outputs all internal values on wires of $F_{\mathbf{w}}$. For this, the compiler $\mathfrak{C}$ increases the fan-out of every inner gate of $F_{\mathbf{w}}$ by one and connects each wire to new output gates $O_{t+1}, O_{t+2}, ..., O_{t+n}$. In the lab phase, the scheme works as follows: the consistency on $O_{t+1}, O_{t+2}, ..., O_{t+n}$ is checked when $\widehat{F}^\tau$ is given inputs from $\mathbb{T}_{\mathbf{w}}$. Assume that the tampering τ is dishonest. 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 wire-covering set.

We must stress that 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 should only constitute a tiny fraction of the circuit size as they are large and must be placed at the border. More precisely, the number of the external wires n_{ext} *is* limited by some function of the number n_{int} of the internal ones. It can be easily seen if we realize that the external wires are typically on the border of a square which contains internal wires [BT18; BA04]. Therefore, quite a convincing relation between these numbers can be given by

$$\frac{n_{\text{ext}}^2}{n_{\text{int}}} \approx c,$$

where c is some constant.

In the rest of Section 3.4, we show how to *compress* these extra outputs so that detection of dishonest tampering is still possible. By compression, we mean the number of additional output/input wires must remain relatively small to the size of the circuit. The main challenge is to achieve this goal *without* tamper-free parts of the circuit. In particular, the adversary is allowed to tamper with the *compressing gadget*, i.e., the part of the circuit responsible for compression.

3.4.2 Sufficiently tamper-resilient gadgets.

The scheme $\mathfrak{L}$ uses 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 allow to detect of some class of tampering on some subset of the input wires.

Definition 9 (Sufficiently Tamper Resilient mOR) *A circuit F_{OR} with input wires $I_1, ..., I_m, I'$ and output wire O, is a* sufficiently tamper-resilient multi-input OR gadget (STRmOR) *if:*

1. *It computes* OR *on its inputs, i.e., $F_{\mathsf{OR}}(X) = 0$ iff $X = 0^{m+1}$.*
2. *If the tampering τ changes some 0 to 1 among on one of the input wires $I_1, ..., I_m$, i.e., $\tau(I_i) \in \{\mathbf{neg}, \mathbf{one}\}$ for some $i \in [m]$, then*

$$F_{\mathsf{OR}}^{\tau}(0^{m+1}) = F_{\mathsf{OR}}^{\tau}(0^m 1).$$

We define Sufficiently Tamper Resilient mAND similarly.

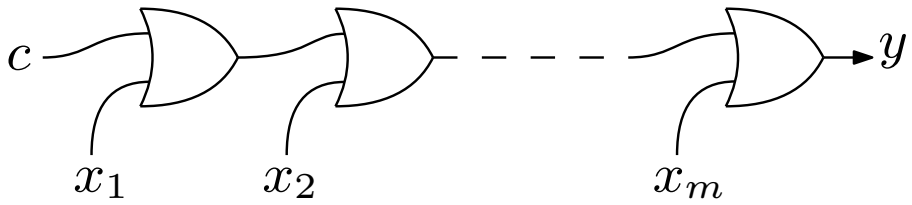


Figure 9: Construction of the ChainMultiOR. The actual inputs are on the bottom.

Definition 10 (Sufficiently Tamper Resilient mAND**)** *A circuit F_{AND} with input wires $I_1, ..., I_m, I'$ and output wire O, is a* sufficiently tamper-resilient multi-input AND gadget (STRmAND) *if:*

1. *It computes* AND *on its inputs, i.e., $F_{\mathsf{AND}}(X) = 1$ iff $X = 1^{m+1}$.*
2. *If the tampering τ changes some 1 to 0 among on one of the input wires $I_1, ..., I_m$, i.e., $\tau(I_i) \in \{\mathbf{neg}, \mathbf{zero}\}$ for some $i \in [m]$, then*

$$F^{\tau}_{\mathsf{AND}}(0^{m+1}) = F^{\tau}_{\mathsf{AND}}(0^m 1).$$

At 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:

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

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

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 $\{0^{m+1}, 0^m 1\}$, $\{1^{m+1}, 1^m 0\}$. We can understand it in terms of *information loss* introduced before. Informally, whenever $x_1, ..., x_m = 1$, the output wire from STRmAND should keep the information about the value of the control bit c. However, any tampering that changes any of the other inputs from 1 to 0 makes it blind. In other words, let the tampering τ assign any of the tamperings $\{\mathbf{neg}, \mathbf{zero}\}$ to any of the input wires to the STRmAND gadget – then $1^{m+1}, 1^m 0$ and $\mathsf{STRmAND}, \mathsf{STRmAND}^{\tau}$ form an information loosing tuple.

The STRmOR can be realized by the ChainMultiOR gadget, presented in Figure 9. To simulate the mOR function with m inputs, we would need only m simple OR gates.

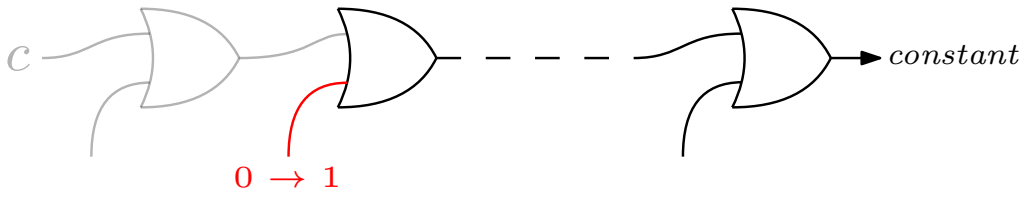


Figure 10: For any tampering on the input wire, which would change the value from 0 to 1, the output becomes blind to everything that happened before. In particular, the output does not depend on c.

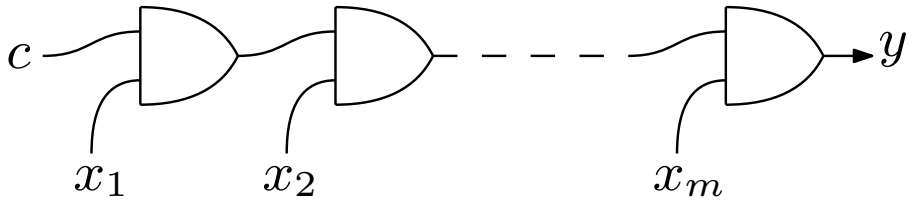


Figure 11: Construction of the ChainMultiAND. The actual inputs are on the bottom.

Theorem 5 ChainMultiOR *is a* STRmOR.

Proof. The first condition holds obviously (from the associative property of the OR operator). Let τ be tampering that applies to some wire I_i a tampering function changing 0 to 1. Consider the function $f(c) := \mathsf{ChainMultiOR}^{\tau}(00...0c)$. Then the OR gate, which takes I_i as input, is unaffected by the second input (see Figure 10) since the value 1 on the wire I_i completely 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. ∎

Corollary 5 ChainMultiAND *(see Figure 11) is a sufficiently tamper resilient* mAND.

3.4.3 Intermidiate solution for 3-conductive circuits

Now we can present a bit more practical solution. For this, we assume not 2-, but 3-conductivity. Every internal wire of the original circuit is branched into three wires *along with its tampering*. One continues into the circuit as before, and two others are used for the testing. Note that for wire-covered circuit $F_{\mathbf{w}}$, we have that for each $e \in E(F_{\mathbf{w}})$, we can choose two test inputs from $\mathbb{T}_{\mathbf{w}}$: one for the value 0, and the second one for the value 1. Recall that it is essential to make any reasonable testing.

Definition 11 *Let $F_{\mathbf{w}}$ be a circuit wire-covered by a set $\mathbb{T}_{\mathbf{w}} = \{T_1, ..., T_k\}$. We say, that a function $\mathbb{i}_b : E(F_{\mathbf{w}}) \to [k]$ is an* index$_b$ function *for $(F_{\mathbf{w}}, \mathbb{T}_{\mathbf{w}})$ iff*

$$\forall_{e \in E(F_{\mathbf{w}})} \mathrm{val}_{T_{\mathbb{i}_b(e)}}(e) = b.$$

Definition 12 *Let $F_{\mathbf{w}}$ be a circuit and let $\mathbb{T}_{\mathbf{w}} = \{T_1, ..., T_k\}$ be its wire-covering set. Let $\mathbb{i}_b$ be some index$_b$ function for $(F_{\mathbf{w}}, \mathbb{T}_{\mathbf{w}})$. We say that a set W_i^b is an* b-testing *for T_i and $\mathbb{i}_b$ iff*

$$W_i^b := \{e \in E(F_{\mathbf{w}}) : \mathbb{i}_b = i\}.$$

Intuitively, W_i^b consists of all the wires that will be checked for tampering that changes the value from b to $1 - b$ on them using the input T_i. To make $F_{\mathbf{w}}$ testable, we extend it by adding an input gate, I_{s+1}, and $2k$ similar gadgets. More precisely, for every $i \in [k]$ we add to the original circuit $F_{\mathbf{w}}$ the gadgets $\mathsf{STRmOR}_i, \mathsf{STRmAND}_i$ with inputs from the sets $W_i^0 \cup \{I_{s+1}\}, W_i^1 \cup \{I_{s+1}\}$, respectively. Intuitively, e.g., the gadget STRmOR_i is used to check if there exists any tampering on wires from W_i^0 that change value from 0 to 1. Such a construction makes a circuit testable. We now state it more formally.

Theorem 6 *Let $F_{\mathbf{w}}$ be wire-covered by $\mathbb{T}_{\mathbf{w}} = \{T_1, ..., T_k\}$. Let $\mathbb{i}_0, \mathbb{i}_1$ be some index$_0$, index$_1$ functions for $(F_{\mathbf{w}}, \mathbb{T}_{\mathbf{w}})$. For $b = 0, 1, j \in [k]$ let W_j^b be a b-testing for $T_j, \mathbb{i}_b$. Let $\widehat{F}$ be $F_{\mathbf{w}}$ extended by:*

- *a single input gate I_{s+1};*
- STRmOR_j *gadget with inputs from the set $W_j^0 \cup \{I_{s+1}\}$ that outputs O_j^{mOR} for $j \in [k]$;*
- $\mathsf{STRmAND}_j$ *gadget with inputs from the set $W_j^1 \cup \{I_{s+1}\}$ that outputs O_j^{mAND} for $j \in [k]$.*

For $j \in [k]$ let $T_j^0 := T_j||0, T_j^1 := T_j||1$. Let $\mathbb{T} := \{T_j^b\}_{b=0,1;j=1,...,k}$. The tuple $(\widehat{F}, \mathbb{T})$ is a testable circuit.

Proof. Assume that $\widehat{F}^\tau$ is tampered dishonestly. Thus, at least one wire in the original subcircuit $F_{\mathbf{w}}$ is tampered with a function from $\{\mathbf{one}, \mathbf{zero}, \mathbf{neg}\}$. Take the *topologically first* tampered wire $e \in E(F_{\mathbf{w}})$. Assume WLOG that the τ tampering changes on e the value 0 to the value 1 (i.e, $\tau(e) \in \{\mathbf{one}, \mathbf{neg}\}$). We will show that the output of gadget $\mathsf{STRmOR}_{\mathbb{i}_0(e)}$ enables to detect the tampering on $\widehat{F}^\tau$ in the lab phase. Consider the tests $T^0_{\mathbb{i}_0(e)}, T^1_{\mathbb{i}_0(e)}$. Since the $\mathsf{STRmOR}_{\mathbb{i}_0(e)}$ gadget is sufficiently

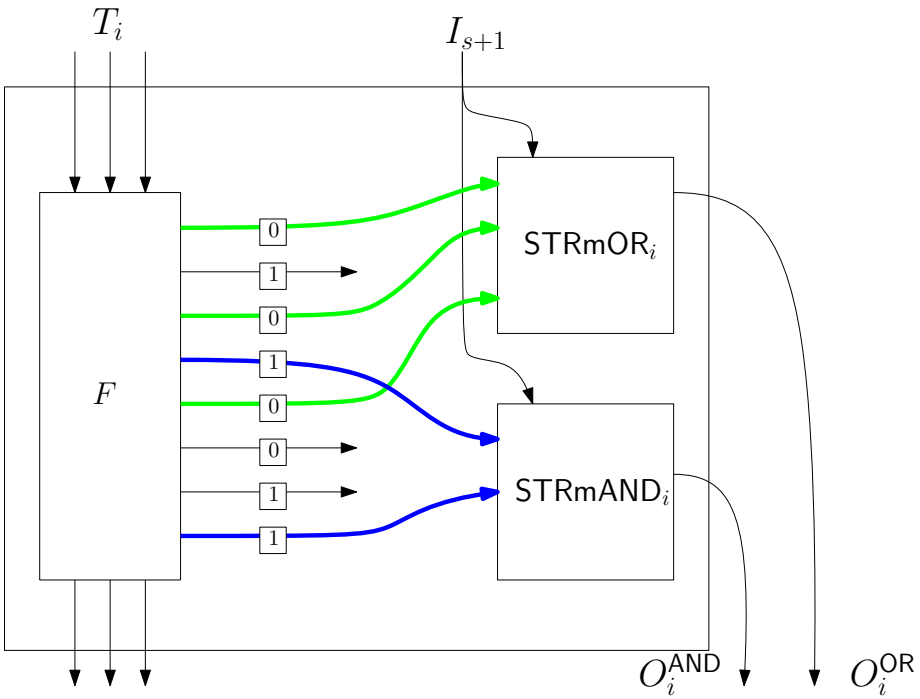


Figure 12: The picture shows the extension of $F_{\mathbf{w}}$ for a single test input T_i. It contains gadgets STRmOR_i, $\mathsf{STRmAND}_i$ that take as input a copy of I_{s+1} and the elements of W_i^0, W_i^1, respectively (in the picture represented in green and blue).

tamper-resilient, and at least one of the incoming wires is tampered with from 0 to 1, the inconsistency will be detected in the lab phase on its output (for tampering that changes 1 to 0 we would consider the output of the $\mathsf{STRmAND}_{\mathbb{i}_1(e)}$ gadget). ■

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, as we will see in the next section.

3.4.4 The $\mathfrak{C}_{\mathfrak{L}}$ compiler construction

We are now ready to present a solution with a small $\mathsf{I/O}$ size *and* a low depth. The starting point is the naive solution from Section 3.4.1. Recall that it simply outputs the values of all internal wires. The compiler $\mathfrak{C}_{\mathfrak{L}}$ adds a gadget that compresses it layer by layer. Each layer reduces the number of additional output bits by the factor $n^{1/L}/2$, where L is the number of layers. For each layer, the additional output bits are divided into groups of $n^{\frac{1}{L}}$ and every group is compressed using the chain gadget from Section 3.4.2 to 2 output bits. A fragment of the construction of a single layer is presented in Figure 13.

Theorem 7 *Let $(F_{\mathbf{w}}, \mathbb{T}_{\mathbf{w}})$ be a wire-covered circuit of conductivity 1, size n, depth d, input size $s+k$ and output size t. Then $(\widehat{F}, \mathbb{T}) = \textsc{Compress}_L(F_{\mathbf{w}}, \mathbb{T}_{\mathbf{w}})$ 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 $|\mathbb{T}| = 2^L |\mathbb{T}_{\mathbf{w}}|$.*

Proof. First, we prove that $(\widehat{F}, \mathbb{T})$ is a testable circuit. Let τ be some nontrivial tampering over F. Consider the topologically first tampered e along with associated gate O_e. Then for some $b \in \mathbb{F}$ we have

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

Let $i_b = \mathbb{i}_b(e)$ for $b \in \{0, 1\}$. The algorithm collects O_e by 2 chain gadgets. For $b = 0, 1$ the output of $\mathsf{ChainMultiOR}$, $\mathsf{ChainMultiAND}$, 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 $\mathsf{ChainMultiOR/ChainMultiAND}$ 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 $\mathsf{STRmOR/STRmAND}$ (see Definition 9). By induction, we obtain, that for some $q \in [2^L], X \in \mathbb{F}^{L-1}$

$$\mathrm{val}^{\tau}_{\widehat{F}, T_{\mathbb{i}_b(e)} X 0}(O_{t+q}) = \mathrm{val}^{\tau}_{\widehat{F}, T_{\mathbb{i}_b(e)} X 1}(O_{t+q}),$$

Algorithm 4: $\textsc{Compress}_L$

Data: wire-covered circuit $F_{\mathbf{w}} : \mathbb{F}^s \to \mathbb{F}^t, \mathbb{T}_{\mathbf{w}} = \{T_1, ..., T_k\}$
Result: $\widehat{F}, \mathbb{T}$

Initialize $\widehat{F} := F_{\mathbf{w}}$;
for $e \in E(\widehat{F})$ **do**
  Append O_e as the output gate to $\widehat{F}$
end
for $b = 0, 1; j = 1, ..., |\mathbb{T}_{\mathbf{w}}|$ **do**
  make b-testing sets W_j^b out from $(O_e)_{e \in E(F_{\mathbf{w}})}$ gates for some index_b functions $\mathbb{i}_b$ and T_j;
end
for $l = 1, ..., L$ **do**
  Append I_{s+l} gate to $\widehat{F}$;
  for $j = 1, ..., |\mathbb{T}_{\mathbf{w}}|$ **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**
      Append to $\widehat{F}$ the ChainMultiOR gadget with inputs from $W_{j,i}^0 \cup I_{s+l}$ with V_i^0 as the output;
      Append to $\widehat{F}$ the ChainMultiAND gadget with inputs 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
$\mathbb{T} := \bigcup_{T \in \mathbb{T}_{\mathbf{w}}, X \in \mathbb{F}^L} \{T||X\}$;
return $\widehat{F}, \mathbb{T}$;

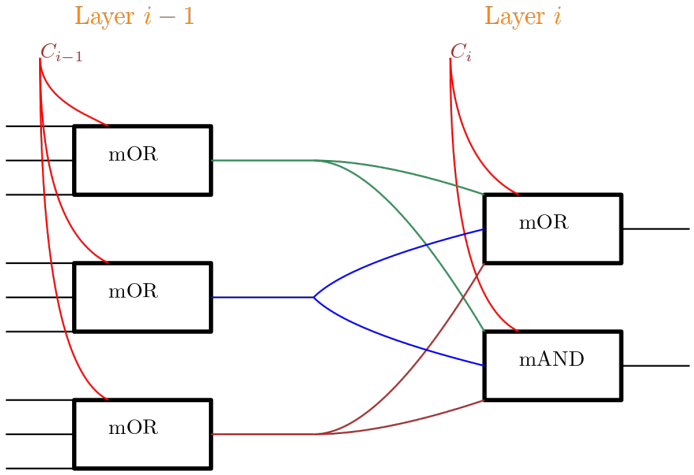


Figure 13: A fragment of the subcircuit produced by Algorithm 4. We assume the chain length parameter $w^{1/L} = 3$. The picture shows a construction fragment starting with the inputs of the Layer $i - 1 \geq 1$ organized into groups of 3. They are then processed by mOR gates connected to a single control 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. The STRmOR and STRmAND gates are implemented with ChainMultiOR and ChainMultiAND gadgets.

where for the untampered $\widehat{F}$, these 2 values should be different. Therefore, the inconsistency will be detected when $\widehat{F}$ is tested on all the elements of $\mathbb{T}$. So, the pair $(\widehat{F}, \mathbb{T})$ is a testable circuit.

Next, we must prove that the algorithm $\textsc{Compress}_L$ 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 must 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). ∎

3.4.5 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 $F : \mathbb{F}^s \to \mathbb{F}^t$ and a number $L \in \mathbb{N}^+$. It first transforms the circuit F into a circuit $F_{\mathtt{w}}$ with a covering set $\mathbb{T}_{\mathtt{w}}$, using the Wire-Covering procedure. Then, it transforms the intermediary circuit into a testable circuit with an extended test set using the procedure $\textsc{Compress}_L$.

Algorithm 5: TestableCircuitCompiler

Data: circuit ($F : \mathbb{F}^s \to \mathbb{F}^t$ of fan-in $\leq k-1$
number of layers L
Result: $(\widehat{F}, \mathbb{T})$
; /*A testable circuit $\widehat{F} : \mathbb{F}^{s+3+L} \to \mathbb{F}^{t+2^L}$ and its test set $\mathbb{T}$ */
1 $W_1, ...W_k \leftarrow k$-Division(F) $(F_{\mathtt{w}}, \mathbb{T}_{\mathtt{w}}) \leftarrow$ Wire-Covering$(F, W_1, ..., W_k)$;
2 $(\widehat{F}, \mathbb{T}) \leftarrow \textsc{Compress}_L(F_{\mathtt{w}}, \mathbb{T}_{\mathtt{w}})$;
3 **return** $(\widehat{F}, \mathbb{T})$;

Theorem 8 *Given any circuit $F : \mathbb{F}^s \to \mathbb{F}^t$ (of size n, depth d), and a number $L \in \mathbb{N}^+$, the procedure* TestableCircuitCompiler(F, L) *outputs a pair $(\widehat{F}, \mathbb{T})$ such that:*

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

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

Proof. The testability and functional equivalence of the circuit $\widehat{F}$ follows directly from the Theorems 3, 5, and 7.

Below, we discuss 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 $F_{\mathbf{w}}$ with depth $d' \leq d + \log(n)$, size $n' \leq n+n+n$, and its wire-covering set of size $k' = 4$. What is more the intermediary circuit has added only 3 control bits to the input, i.e.: $F_{\mathbf{w}} : \mathbb{F}^{s+3} \to \mathbb{F}^t$. By applying the Algorithm $\textsc{Compress}_L$, we can calculate the following parameters of the output circuit:

- its modified input and output size – $\widehat{F} : \mathbb{F}^{s+3+L} \to \mathbb{F}^{t+2^L}$,

- its modified circuit size is the number of the gates n' of the circuit $F_{\mathbf{w}}$ 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 second 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:

$$n'' \leq n' + \sum_{i=1}^{L} 2^{i-1}\frac{w'}{w'^{(i-1)/L}} + \sum_{i=1}^{L} 2^{i-1}\frac{w'}{w'^{i/L}} =$$
$$n' + \frac{w'^{1/L}(w' - 2^L)}{w'^{1/L} - 2} + \frac{w' - 2^L}{w'^{1/L} - 2} \leq$$
$$n' + 2w' + w' \leq 3n + 3 \cdot 3n$$

- its modified circuit depth $d'' \leq d' + \sum_{i \in \{1,\ldots,L\}} w'^{1/L}$,

  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 $k'' = 4 \cdot 2^L$.

■

3.5 $\mathfrak{R}$-scheme

We now present the $\mathfrak{R}$-scheme, a different way of compiling circuits to make them tamper-resilient. It achieves a little bit different properties compared to $\mathfrak{L}$-scheme.

3.5.1 Information Loss in Gate-Covered Circuits[1]

In this section, we define information loss and show that it is easily trackable in any $F_{\mathbf{g}}$ with a gate-covering set $\mathbb{T}_{\mathbf{g}}$. For any such circuit, we show the following property: for any tampering applied to the wires of the $F_{\mathbf{g}}$, 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 $\mathbb{T}_{\mathbf{g}}$), or the output wires of the circuit are always set to a constant value or always toggled or always correctly evaluated.

Theorem 9 *For any circuit $F_{\mathbf{g}} : \mathbb{F}^s \to \mathbb{F}^t$ with gate-covering set $\mathbb{T}_{\mathbf{g}}$, for any tampering function τ applied to the circuit then at least one of the following holds:*

- *Information loss on multi-input gates*

$$\exists_{X_0, X_1 \in \mathbb{T}_{\mathbf{g}},\ v \in V(F_{\mathbf{g}})\ :\ \gamma(v) \in \{\mathsf{AND},\mathsf{OR},\mathsf{XOR}\}}\ :$$

$$\left(\mathrm{val}_{X_0, F_{\mathbf{g}}}(v) = 0 \wedge \mathrm{val}_{X_1, F_{\mathbf{g}}}(v) = 1\right) \wedge \left(\mathrm{val}^{\tau}_{X_0, F_{\mathbf{g}}}(v) = \mathrm{val}^{\tau}_{X_1, F_{\mathbf{g}}}(v)\right)$$

- *Constant output*

$$\exists_{i \in [t], b \in \{0,1\}} \forall X \in \mathbb{F}^s\ :\ F^{\tau}_{\mathbf{g}}(X)[i] = b$$

- *At most toggled output*

$$\exists_{T \in \{0,1\}^t} \forall_{X \in \mathbb{F}^s} F^{\tau}_{\mathbf{g}}(X) = F_{\mathbf{g}}(X) + T$$

Proof. The proof follows a modular argument. For this, we introduce the concept of Topological Layers of Computation on any circuit F_γ. In the definition below, we say that a wire e is connected to a gate v in a circuit F_γ described with a DAG (denoted by predicate $\mathit{connected}_{F_\gamma}(g, e)$ 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 13 (Topological Layers of Computation) *For any circuit F, we recursively define its Topological Layers of Computation:*

[1] cited from [Bai+23b] for completeness

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

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

Below we consider $F = F_{\mathbf{g}}$. We run an experiment that evaluates layer by layer the tampered F (assuming F 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 v in layer $\mathcal{L}_i$:

- $\mathcal{E}_1(v, i)$ holds if $v \in \mathcal{L}_i \ \wedge \ \exists_{X_0, X_1 \in \mathbb{T}_{\mathbf{g}}} \mathrm{val}_{X_0,F}(v) = 0 \ \wedge \ \mathrm{val}_{X_1,F}(v) = 1 \ \wedge \ \mathrm{val}^{\tau}_{X_0,F}(v) = \mathrm{val}^{\tau}_{X_1,F}(v)$,
- $\mathcal{E}_2(v, i)$ holds if $v \in \mathcal{L}_i \ \wedge \ \forall_{X \in \mathbb{F}^s} : \mathrm{val}^{\tau}_{X,F}(v) = \mathrm{val}_{X,F}(v) + f\Big[\tau(e) : e \in E(G_i(F))\Big]$.

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

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

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 $P_v(a, b) = a \cdot b$. The tampered output of the gate is $\tilde{P}_v(a, b) = \tilde{a} \cdot \tilde{b}$, 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 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 $\tilde{P}_v(a, 1) = 0$ and $P(a, 1) = a$, we get an information loss. Now, since by the construction of the Algorithm 6, the wire P is connected via a COPY to the output, the event $\mathcal{E}_1(v, 1)$ 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 $\tilde{P}(b, 1) = const.$) [$\mathcal{E}_1(v, 1)$ 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 $\tilde{P}(a, 1) = 0$) [$\mathcal{E}_1(v, 1)$ occurs],
- otherwise $\tilde{P}(a, b) = ab$ [$\mathcal{E}_2(v, 1)$ occurs].

- A similar argument as above applies for the XOR 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 $P_v(a, b) = a + b$, and the tampered output $\tilde{P}_v(a, b) = \tilde{a} + \tilde{b}$, 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 $\tilde{P}(0, b) = const.$) [$\mathcal{E}_1(v, 1)$ occurs],
  - when $\tilde{a} = a + c_a, \tilde{b} = b + c_b$, then $P(a, b) = a + b$, $\tilde{P}(a, b) = a + b + c_a + c_b$ [$\mathcal{E}_2(v, 1)$ 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 as constant; if only toggles are found there, the output bits are at most toggled. ■

3.5.2 Routing the Information Loss in Gate-Covered Circuits.[1]

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 6 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 (Figure 14).

[1] cited from [Bai+23b] for completeness

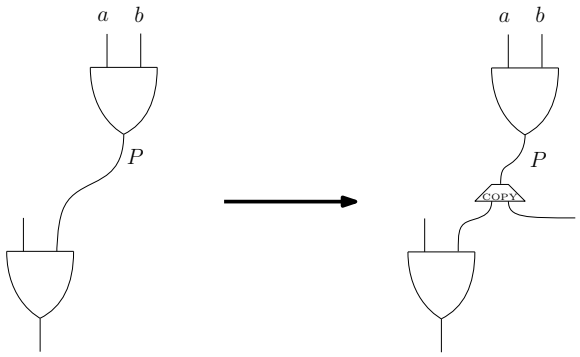


Figure 14: Adding a COPY gate to the wire P in the Algorithm 6. This creates two wires; the left one is connected to the previous successor of the wire P, and the right one is sent to the circuit output. Algorithm 6 considers only the wires P which originate at $\mathsf{AND}, \mathsf{OR}, \mathsf{XOR}$ gates in the original circuit.

Proposition 1 *The Algorithm 6 transforms a gate-covered circuit $F_{\mathbf{g}} : \mathbb{F}^s \to \mathbb{F}^t$ with gate-covering set $\mathbb{T}_{\mathbf{g}}$ into another gate-covered circuit $F_{\mathbf{t},0} : \mathbb{F}^s \to \mathbb{F}^{t+t_0}$ with additional output bits and the same gate-covering set, $\mathbb{T}_0 = \mathbb{T}_{\mathbf{g}}$, where one observes for any tampering τ of the circuit $F_{\mathbf{t},0}$ 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*

$$\mathrm{val}_{X_0}(F_{\mathbf{t},0})[i] = 0, \mathrm{val}_{X_1}(F_{\mathbf{t},0})[i] = 1, \mathrm{val}^{\tau}_{X_0}(F_{\mathbf{t},0})[i] = \mathrm{val}^{\tau}_{X_1}(F_{\mathbf{t},0})[i] = b$$

- *At most toggled output*

$$\exists_{B\in\{0,1\}^t} \forall X \in \mathbb{F}^s \; \exists Y \in \mathbb{F}^{t_0} : \; F^{\tau}_{\mathbf{t},0}(X) = F_{\mathbf{g}}(X) \| Y + B \| 0^{t_0}$$

Proof. It is easy to see that the same test set $\mathbb{T}_0 = \mathbb{T}_{\mathbf{g}}$ is a gate covering set for the transformed $F_{\mathbf{t},0}$. Now, according to Theorem 9 on the transformed circuit, the following cases may follow:

1. Information-loss on one of the multi-input gates of $F_{\mathbf{t},0}$: in this case, one of the output wires of $F_{\mathbf{t},0}$ 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 $F_{\mathbf{t},0}$ constantly evaluates to a constant value: in this case, we observe an information loss on this wire because it is wire-covered according to the gate-covering set $\mathbb{T}_0$ definition.

Algorithm 6: Algorithm for Routing the Information Loss in a Gate-Covered F

Data: $F_{\mathbf{g}} : \mathbb{F}^s \to \mathbb{F}^t, \mathbb{T}_{\mathbf{g}}$
Result: $F_{\mathbf{t},0}, \mathbb{T}_0$
Initialize $F_{\mathbf{t},0} = F_{\mathbf{g}}$, $\mathbb{T}_0 = \mathbb{T}_{\mathbf{g}}$
for $v \in V(F_{\mathbf{g}})$ **do**
  if $\gamma(v) \in \{\mathsf{AND}, \mathsf{OR}, \mathsf{XOR}\} \wedge e = E^+(v)$ *is not an output wire of* $F_{\mathbf{t},0}$ **then**
    Insert to $F_{\mathbf{t},0}$ a COPY gate between v and $V^+(e)$.
    One of the output wires of the new gate should go to $V^+(e)$; the other one should be left as an additional output wire of the modified circuit.
  end
end
return $F_{\mathbf{t},0}, \mathbb{T}_0$

3. At most, toggled output on the circuit.

■

3.5.3 Minimizing the Number of External Wires

In the previous sections, we introduced the notion of information loss, and we showed that without limitations on the number of *additional* input/output wires, the tamper-resilience can be achieved for 1-conductive circuits. As in the case of 3-conductive circuits, such a solution is impractical. The rest of this section is devoted to demonstrating the *compressing gadget* $G_{n,\lambda,d}$ that works for nonconductive circuits even if the adversary tampers with it.

The gadget $G_{n,\lambda,d}$ will compress an input of length n. It will comprise 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 15). 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 can keep λ at most 5.

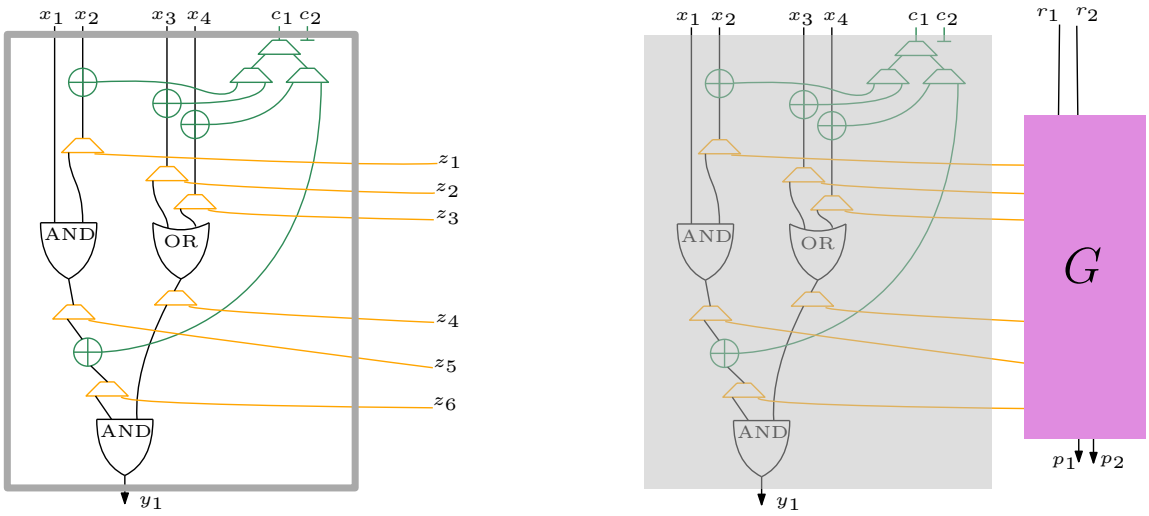


Figure 15: To reduce the number of external wires, we add a compressing gadget $G_{n,\lambda,d}$.

3.5.3.1 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.

The $S_{m,d}$ gadget 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:

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

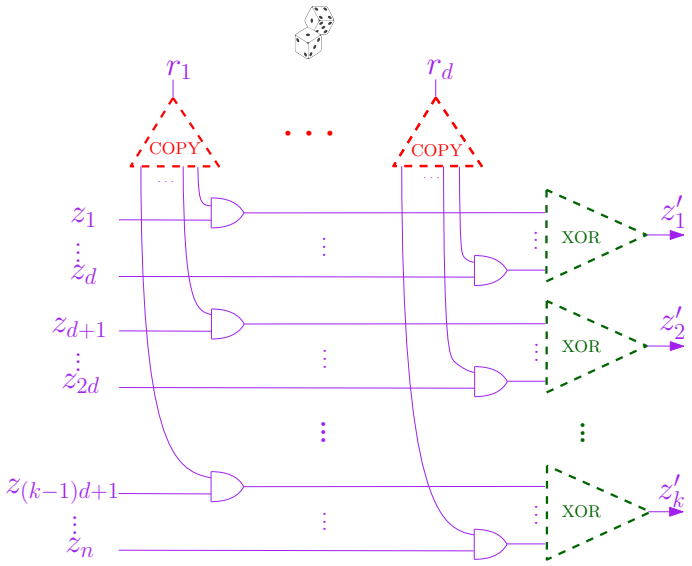


Figure 16: Construction of the $S_{m,d}$ gadget. The dotted red triangle represents the copying tree, $\triangle_k$. The dotted green triangle represents the xoring tree $\triangleright_d$.

The construction of $S_{m,d}$ is shown in Figure 16. An instantiation with $m = 8, d = 4$ is shown in Figure 18. Construction of $S_{m,d}$ needs as building blocks two 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 17).

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 $S_{m,d}$.

Lemma 1 *For the $S_{m,d}$ gadget with m input wires and additional d input wires for randomness (Figure 16), 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*

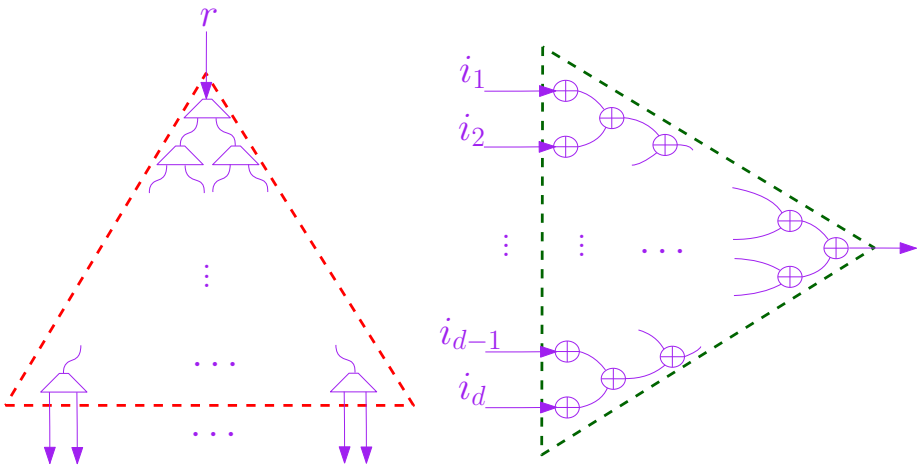


Figure 17: The gadgets $\triangle, \triangleright$ are realized by complete binary trees with COPY, XOR in nodes, respectively.

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).*

3.5.3.2 Composing The Layers

We are ready to present the complete 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 19). 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 d times. Every layer is given d extra input wires, which would be uniformly random bits.

Intuition: In Proposition 1, 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.

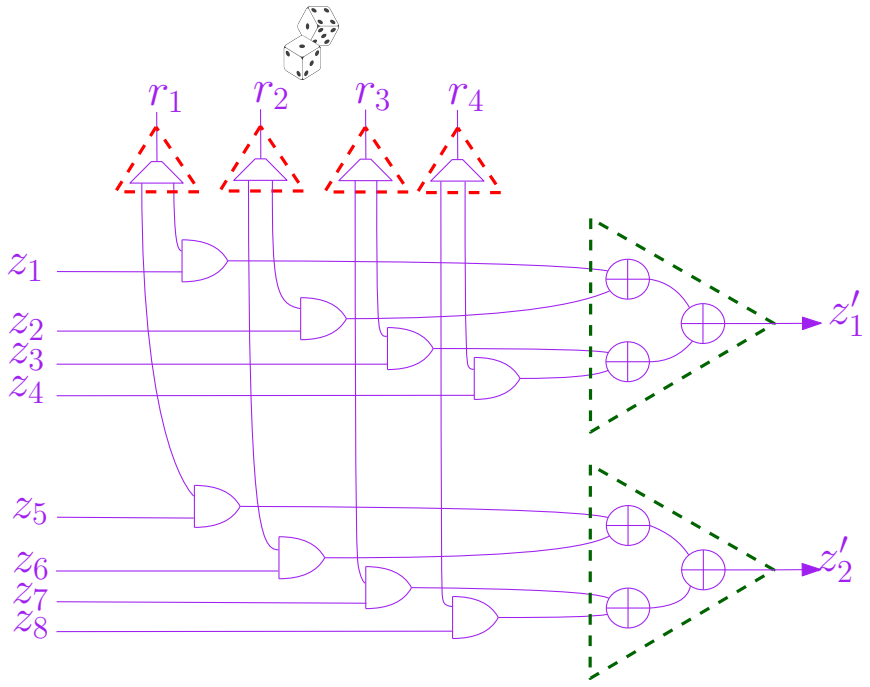


Figure 18: A single layer of compression - $S_{m,d}$ (in this case $m = 8, d = 4$). Trapeziums represent copy gates. The dotted red triangle represents the copying tree, $\triangle_2$. The dotted green triangle represents the xoring tree, $\triangleright_4$.

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 $\sum_{i=0}^{\lambda-1} \left(3\frac{n}{d^i}\right) = 3n\frac{d-d^{1-\lambda}}{d-1}$.*

3.5.3.3 Information Losing Tuples

Recall that in Proposition 1, we show that any meaningful computation error 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 reasonable 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.

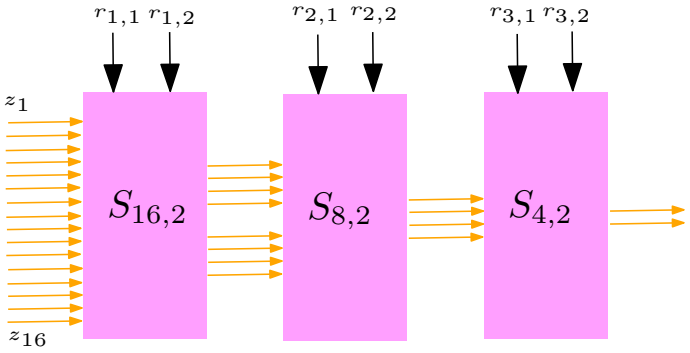


Figure 19: The compressing gadget $G_{n,\lambda,d}$ consists of λ layers of the compression subgadgets $S_{m,d}$, where the number of input wires m decreases layer by layer. Example parameters are $n = 16, \lambda = 3, d = 2$.

For instance, 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. Thus, we will use 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 evaluation process 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 F.

Definition 14 *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 1. Let $(X_i), (Y_i)$ denote the values on the output wires of $F_{\mathbf{g}}, F_{\mathbf{g}}^{\tau}$ for the gate covering set $\mathbb{T}_{\mathbf{g}} = \{T_i\}_i$. 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.

3.5.3.4 Algebraic Values on the Wires

Now we analyze what the (parameterized by the input) possible values on wires in *tampered* realization of $G_{n,\lambda,d}$ are. We will use an algebraic notation for the evaluation of the circuit. We make the following extension – the circuit wires 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 denoted with lowercase letters; sometimes, we will be using auxiliary indeterminates. To compute the results of the val function, we 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 wire tampering 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 equal to $0/1$ without indeterminates. Therefore, we can make some observations on the gadgets $\triangleright, \triangle$ (from Figure 17) and the output of the multiplication gates in S_d.

Proposition 2 (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 3 (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 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 4 (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 $\mathbb{F}$ 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 5 (Output of the one layer compression gagdet) *Let p_m be the output value of the gadget $\rhd^\tau$ 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 $\mathbb{F}$.

Given the Propositions 2, 3, 4, in Proposition 5 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.

3.5.3.5 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 complete compression gadget $G_{n,\lambda,d}$ is built using λ layers of S_d gadgets, this result will lead us to the 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 evaluation vector X_i will be used twice as the input to the gadget S_d. This will suffice to propagate the information loss to the gadget's output with good probability.

Theorem 10 (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

$$(X_i[k] \neq X_j[k]) \wedge \left(X_i^\tau[k] = X_j^\tau[k]\right). \tag{2}$$

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:

$$\mathrm{val}_U(O_s) = \sum_{t=1,\ldots,d} x_{sd+t} r_t = \sum_{t=1,\ldots,d} \gamma_t(x_{sd+t}) r_t, \tag{3}$$

where γ_t is the identity function. From Proposition 5, we know that the tampered value of the selected output wire can be described with the following expression:

$$\mathrm{val}^{\tau}_U(O_s) = \sum_{t=1,\ldots,d} \alpha_t\left(x_{sd+t}\right) r_t + \beta_t\left(x_{sd+t}\right), \tag{4}$$

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

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 3, 4 are simply linear/affine combinations of $r_1, ...r_d$ over $\mathbb{Z}_2$, respectively. Consider,

$$\mathrm{DIFF_i}(r_1, ..., r_t) := \mathrm{val}_{X_i|r_1,\ldots,r_d}(O_s) - \mathrm{val}^{\tau}_{X'_i|r_1,\ldots,r_d}(O_s) = r_{k'} + \sum_{t=1,\ldots,d; t\neq t'} \delta_t r_t + \epsilon_i, \tag{5}$$

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 $\Pr[\mathrm{DIFF_i}(R) = 1] = \frac{1}{2}$ This means that for the pair X_i, X'_i for exactly half of the choices of R, an error will occur on the O_s - i.e., the expected and actual values will differ. Since $\mathrm{DIFF_j}(r_1, ..., r_t) = \mathrm{val}_{X_j|r_1,\ldots,r_d}(O_s) - \mathrm{val}^{\tau}_{X'_j|r_1,\ldots,r_d}(O_s) = \sum_{t=1,\ldots,d} \kappa_t r_t + \epsilon_j$, 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,

$$\left\{\frac{1}{2}\right\} \subseteq \left\{\Pr[\mathrm{val}_{X_i|R}(O_s) = 0], \Pr[\mathrm{val}_{X_j|R}(O_s) = 0]\right\} \subseteq \left\{0, \frac{1}{2}\right\}. \tag{6}$$

The first inclusion is true since $1 \in \{\gamma_{k'}^{X_i}, \gamma_{k'}^{X_j}\}$. 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 $x_1(R) = S(X_i|R)[k']$, $x_2(R) = S(X_j|R)[k']$, $y_1(R) = S^{\tau}(X_i^{\tau}|R)[k']$, $y_2(R) = S^{\tau}(X_j^{\tau}|R)[k']$. Informally speaking, for independent and uniformly random R_i, R_j, Q_i, Q_j the tuple

$$\mathcal{V} = (x_1(R_i), x_1(Q_i), x_2(R_j), x_2(Q_j); y_1(R_i), y_1(Q_i), y_2(R_j), y_2(Q_j))$$

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 $(Y_1, ..., Y_{2z}; Y_1^\tau, ..., Y_{2z}^\tau)$ 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:*

$$\Pr[x_1(R) = y_1(R)] = \frac{1}{2}, \tag{7}$$

$$\Pr[x_2(R) = y_2(R)] \in \left\{0, \frac{1}{2}, 1\right\}, \tag{8}$$

$$\left\{\frac{1}{2}\right\} \subseteq \{\Pr[x_1(R) = 0], \Pr[x_2(R) = 0]\} \subseteq \left\{\frac{1}{2}, 1\right\}, \tag{9}$$

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. [1] 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) $(a = 1 - a') \wedge (b = 1 - b') \wedge (c = 1 - c') \wedge (d = 1 - d')$.

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. Without loss of generality, let $a = 1, b = 0$. Without loss of generality we have two cases $c = 1, d = 1$

[1] This statement was originally proved by me by computer-aided considering several dozen cases; one of the collaborators simplified it, and here is his proof.

or $c = 1, d = 0$. In the first case, if $b' = 0$, 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 $a' = c'$, we get information loss if either b' or d' equals a'. The only way for neither b' nor d' to equal a' is for one of conditions 2 or 3 to hold, 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. 9). 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 Equation 8:

1. $\Pr[x_2(R) = y_2(R)] = 0$: In this case $\Pr[E_2] = 0$. Additionally using eq. 7 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. 7 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. 7 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 information loss at least $\frac{1}{2}$.■

3.5.4 The Compiler

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

Algorithm 7: RANDOMNESSCOMPILER

Data: $F : \mathbb{F}^s \to \mathbb{F}^t, \lambda$
Result: $F_{\mathbf{t},\lambda}, \mathbb{T}_{\mathbf{g}}$

1 Compile circuit F into a gate-covered $F_{\mathbf{g}} : \mathbb{F}^{s+s_g} \to \mathbb{F}^t, \mathbb{T}_{\mathbf{g}}$, by running Algorithm 3 on it
2 Add the COPY gates that route the information loss in the gate-covered circuit to the testing gadget by running Algorithm 6 on the pair $F_{\mathbf{g}}, \mathbb{T}_{\mathbf{g}}$. This procedure gives a circuit with additional t_0 output bits - $F_{\mathbf{t},0} : \mathbb{F}^{s+s_g} \to \mathbb{F}^{t+t_0}$ along with a test set $\mathbb{T}_{\mathbf{g}}$
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 $F_{\mathbf{t},\lambda} : \mathbb{F}^{s+s_g+s_\lambda} \to \mathbb{F}^{t+t_\lambda}$
4 **return** $F_{\mathbf{t},\lambda}, \mathbb{T}_{\mathbf{g}}$

Theorem 11 (Testing Probability of Final Circuit) *On input circuit $F : \mathbb{F}^s \to \mathbb{F}^t$ along with parameter λ, Algorithm 7 outputs a circuit $F_{\mathbf{t},\lambda} : \mathbb{F}^{s+s_g+s_\lambda} \to \mathbb{F}^{t+t_\lambda}$ such that for any tampering τ of $F_{\mathbf{t},\lambda}$ if*

$$\exists X \in \mathbb{F}^s : F^\tau_{\mathbf{t},\lambda}(X||0^{s_g+s_\lambda}) \neq F_{\mathbf{t},\lambda}(X||0^{s_g+s_\lambda})$$

then when observing behaviour of the circuit $F_{\mathbf{t},\lambda}$ on its test set $\mathbb{T}_{\mathbf{t}}$:

- *Either the output is wrong:*

$$\exists X \in \mathbb{T}_{\mathbf{t}} : F^\tau_{\mathbf{t},\lambda}(X||0^{s_\lambda}) \neq F_{\mathbf{t},\lambda}(X||0^{s_\lambda}),$$

- *or the testing gadget detects an inconsistency:*

$$\exists X \in \mathbb{T}_{\mathbf{t}} : \Pr_{R \leftarrow \mathbb{F}^{s_\lambda}} \left[F^\tau_{\mathbf{t},\lambda}(X||R) \neq F_{\mathbf{t},\lambda}(X||R) \right] \geq \frac{1}{2^{2\lambda}}.$$

Proof. By the Proposition 1 we know that either we observe an information loss on the first t bits of the intermediary circuit $F_{\mathbf{t},0}$ or its output is toggled, or we observe information loss on t_0 wires added during Step 2 of the Algorithm 7, or the output is always correct. Any error on the first t bits of the circuit will be detected on at least one query from ($\mathbb{T}_0 \subseteq \mathbb{T}_{\mathbf{t}}$ is the gate-covering set of the $F_{\mathbf{t},0}$ (Proposition 1)). Next, when we append the gadget $G_{n,\lambda,d}$ to the remaining t_0 wires of the construction. Theorem 10 shows 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 $F^{\tau}_{\mathbf{t},\lambda}$ with any tampering τ on its wires we test it by querying it on all the test inputs in $\mathbb{T}_{\mathbf{g}}$ along with uniform random $R \in \mathbb{F}^{s_\lambda}$. 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 $F : \mathbb{F}^s \to \mathbb{F}^t$ with n gates, using the Algorithm 7 with parameter λ. The first step of the Algorithm produces a gate-covered circuit $F_{\mathbf{g}}$ with 5 new input bits and a test set of size 6 and creates a circuit of size $\approx 7n$ gates . 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.

3.6 Summary and discussion

We described two schemes against tampering HTHs – the $\mathfrak{L}$-scheme and the $\mathfrak{R}$-scheme. Their basic properties are compared in Figure 3.6.

Conductivity. The $\mathfrak{R}$-scheme grants resilience against HTHs in a much more general model of non-conductive wires. This model is closer to reality- no reason exists for the adversary not to tamper with a few output wires from a single gate differently. The main reason the $\mathfrak{L}$-scheme does not work in this model is the definition of STRmOR, STRmAND gadgets. For instance, for the STRmOR gadget, the adversary

	F	$\widehat{F}, L \in \mathbb{N}^+$ ($\mathfrak{L}$-scheme)	$F_{\mathbf{t},\lambda}, \lambda \in \mathbb{N}^+$ ($\mathfrak{R}$-scheme)
Number of gates	n	$\leq 12n$	$\leq 23n + o(n)$
Input size	s	$s + 3 + L$	$\leq s + 5 + \lambda \cdot \sqrt[\lambda+1]{4n}$
Output size	t	$t + 2^L$	$\leq t + \sqrt[\lambda+1]{4n}$
Cover/Test size		$4 \cdot 2^L$	$\lvert\mathbb{T}_{\mathbf{t}}\rvert \leq 6$
Probability that adversary wins		0	$2^{-2\lambda}$
Randomness (bits)		0	$\lambda \cdot \sqrt[\lambda+1]{4n}$

Table 2: Summary of the efficiency parameters of the schemes $\mathfrak{L}$ and $\mathfrak{R}$.

could tamper with input wires $I_1, ..., I_m$ by **zero** function so that there would be no information loss on the output wire O (see Definition 9).

Stateful circuits. The solutions we just presented work in the model of *stateless* circuits. Obviously, *stateful* circuits have great interest in cryptography since they can, e.g., store a private key. We could model stateful circuits as directed graphs with cycles – contrary to DAGs – where every cycle contains at least one memory cell. In this model, the solution from $\mathfrak{L}$ looks more promising - recall that the consistency in the lab phase is checked only for auxiliary output wires. In the testing context, we can understand stateful circuits as stateful circuits but with restricted access to some part of its output – the part that goes to the memory gates.

Test set. The two schemes presented in this section also differ in the construction of the test set. Recall that the topology of the circuit $\widehat{F}$ strongly depends on the wire-covering set $\mathbb{T}_{\mathbf{w}}$. More precisely, the STRmOR_i and $\mathsf{STRmAND}_i$ gadgets take as input some subsets of the internal wires of the original circuit $F_{\mathbf{w}}$, depending on the index functions. The test set is small, and the testing procedure is deterministic. The test set in $\mathfrak{R}$-scheme requires randomness.

Derandomization. Note that the construction from $\mathfrak{R}$-scheme can be easily derandomized, affecting a huge (but tractable) test set. For the number of layers of compression $\lambda = \log_2 n$, where n is the number of internal wires, the length of the random vector R is $2\log_2 n$. In this case, a random vector for each layer has length 2. For each $T \in \mathbb{T}_{\mathbf{g}}$ the lab phase can check the consistency for all n^2 possible values of R. Our preliminary calculations show that for each layer, it is sufficient to check not all 4 but only 3 arbitrary values of random vectors to ensure that the tampering

applied to the circuit was dummy.

Industrial issues. There are some industrial issues we want to comment on, even though this thesis is theoretical. Firstly, the resilience proofs for the presented schemes do not rely on the topology of the auxiliary subcircuits we defined – we mean copying and coring trees. They can be realized in a more practically efficient way. Secondly, the subsets of wires taken as input to the $\mathsf{STRmOR}, \mathsf{STRmAND}$ gadgets can be arbitrary. We can even produce more such gadgets because of the geometry of the circuit, which affects a small growth of the number of output wires (one additional bit for each gadget). Thirdly, we believe that the notion of wire-covering and gate-covering sets may be interesting from the perspective of testing devices against unintentional faults.

4 Very Simple Compilers against Total Hardware Trojan Horses[1]

This section is devoted to countermeasures against Total Hardware Trojan Horses that significantly differ from all the others in terms of the specification of the compiled circuit. Recall that a circuit compiler $\mathfrak{C}$ against THTHs for a given circuit F outputs a tuple $(\widehat{F}, \mathbb{T})$ consisting of the description of a compiled circuit and a characterization of a lab phase (see Section 2). The circuit description $\widehat{F}$ consists of a trusted (master) module and some untrusted modules $\mathsf{F}_1, \mathsf{F}_2, ..., \mathsf{F}_n$, whose specification can significantly differ from the input circuit [DFS16; Wah+16]. The dissimilarity in the functionality of the original F and the untrusted submodules $\mathsf{F}_1, \mathsf{F}_2, ..., \mathsf{F}_n$ can be easily understood – the existing solutions use cryptographic primitives such as Multi-Party Computation or Verifiable Computation. For instance, the untrusted and trusted components of the compiled circuit in [Wah+16] play the Prover and Verifier roles of some Verifiable Computation Protocol for original functionality F. We suspect the research objective in [Wah+16; DFS16] was to obtain the best security parameters for a possibly broad class of adversaries. In this section, the starting point of the research is different – we investigate the (reasonably) *simplest* compilers and prove how secure they can and cannot be.

The starting point of the research on Very Simple Compilers is a rigorous division of the computation between the trusted and untrusted components of the device to be protected. Recall that the very first motivation for using the devices produced by untrusted parties is that the ones produced honestly are less efficient. Therefore, we expect the trusted module to perform *very few* operations. However, to achieve meaningful security guarantees, the master module must perform *some* nontrivial computations according to impossibility results from [DFS16]. We can think of *equality check* as the most fundamental nontrivial operation that can be performed on some two objects. It is blind to any properties, structures, or built-in functionalities of the compared entities. The concept of *Very Simple Compilers* is based on this observation. More precisely, we focus on a well-defined class of compilers, such that untrusted subcircuits F_i are expected to behave exactly as the input circuit F. The trusted module treats them in the wild as *oracles*, i.e., it sends input to *some* of them and checks if the received outputs are equal. If so, it outputs the received output; if not – it stops. The untrusted oracles are independently tested in the lab phase. The inputs in the wild and lab phases come from a uniformly random distribution – the

[1]This section is based on the paper [Cha+21], which was presented at the Theory of Cryptography Conference 2021. Section 4.2 is cited for completeness.

adversary that controls the input stream in the wild would be able to activate the THTH with a *cheatcode*. All formal definitions will be given later.

The organization of this section is as follows: firstly, in Section 4.1, we present the result on this topic from [Cha+21] and the possible application of achieved security guarantees. Secondly, in Section 4.2, we cite useful definitions, lemmas, and impossibility results from [Cha+21]. Finally, in Section 4.3, we cite the construction that achieves the best possible security parameters for Very Simple Compilers, along with all the needed proofs. The general idea of the compiler, motivation, and proofs was developed jointly by the authors of [Cha+21]. I wrote down Section 4.3 and resolved some mediocre and minor issues that arrived during that process.

4.1 Results and applications

Here, we want to briefly present both negative and positive results in the field of Very Simple Compilers and possible applications of such security schemes.

4.1.1 (Non)achievable security for Very Simple Schemes

Throughout this section, we work with security defined in terms of a security game $\mathsf{TrojanGame}(\Pi, T, Q)$, which will be formally defined later in Section 4.2. For now, we need to know that T and Q refer to the length of the lab and wild phase, respectively. The achievable security guarantees for Very Simple Compilers are far from making this scheme a black-box building block for arbitrary application. To see that, we need to understand the concept of resilience against THTHs parametrized by real numbers win and wrng. Informally, we call a compiler (like our simple schemes) $(\mathsf{win}, \mathsf{wrng})$-*Trojan resilient*, or simply $(\mathsf{win}, \mathsf{wrng})$-*secure*, if for every Trojan, the probability that it causes the master to output $\geq \mathsf{wrng}$ fraction of wrong outputs without being detected is at most win. The detection of the Trojan can happen in the lab phase (if it does not pass some of the tests) or in the wild phase (to be explained later). In the formal definition, win and wrng are allowed to be a function of the number of test queries T.

Definition 15 ($(\mathsf{win}, \mathsf{wrng})$-Trojan resilience) *For* $\mathsf{win}, \mathsf{wrng} \in [0, 1]$ *an adversary* $(\mathsf{win}, \mathsf{wrng})$-wins *in* $\mathsf{TrojanGame}(\Pi, T, Q)$ *if the master module outputs more than a* wrng *fraction of wrong values without the Trojans being detected with probability greater than* win*, i.e.*

$$\Pr_{\mathsf{TrojanGame}(\Pi,T,Q)}[(\mathsf{detect} = \mathsf{false}) \wedge (Y/Q \geq \mathsf{wrng})] \geq \mathsf{win}.$$

For $\mathsf{win} : \mathbb{N} \to [0,1], \mathsf{wrng} : \mathbb{N} \to [0,1], q : \mathbb{N} \to \mathbb{N}$*, we say that* Π *is* $(\mathsf{win}(T), \mathsf{wrng}(T), q(T))$-Trojan-resilient *(or simply* secure*) if there exists a constant* T_0*, such that for all* $T \geq T_0$ *and* $Q \geq q(T)$ *no adversary* $(\mathsf{win}(T), \mathsf{wrng}(T))$*-wins in* $\mathsf{TrojanGame}(\Pi, T, Q)$*.*

We say Π *is* $(\mathsf{win}(T), \mathsf{wrng}(T))$-Trojan-resilient *if it is* $(\mathsf{win}(T), \mathsf{wrng}(T), q(T))$*-Trojan-resilient for some (sufficiently large) polynomial* $q(T) \in poly(T)$*.*

The paper [Cha+21] presents a discussion on possible meaningful values of the parameters $\mathsf{win}, \mathsf{wrng}$. The paper also gives positive and negative results on the achievable $(\mathsf{win}, \mathsf{wrng})$-Trojan-resilience. Here, we first describe the impossibility results. Very Simple Schemes for the maximal number of tests $\leq T$ allow the adversary to make the device produce $\Omega(1/T) \cdot Q$ wrong outputs in the wild with noticeable probability. It is easy to see if we think of $\mathsf{F}_1 = \mathsf{F}_2 = ...$ that deviate on the $1/T$ fraction of inputs. More precisely[1]:

$$\mathsf{F}_i(x) = \begin{cases} F(x) + 1 \text{ if } X < |\mathbb{F}|^s/T, \\ F(x) \text{ otherwise.} \end{cases}$$

We state it as a lemma below. Its formal proof will be given later, but we can informally argue on it here. Firstly, for uniformly random inputs in the lab phase of length $< T$, the probability of catching the device deviating is $\leq 1 - (1 - 1/T)^T \approx 1 - 1/e$. Secondly, approximately Q/T outputs will be incorrect in the wild for uniformly random inputs.

Lemma 4 (Lower bound for simple schemes) *For any* $c > 0$ *and* $m \in \mathbb{N}$ *there exists a constant* $c' = c'(c, m) > 0$ *such that no* m*-redundant simple scheme* Π_m *is* $(c, c'/T)$*-Trojan-resilient.*

We showed that the above bound is tight. Figure 21 presents a construction for which we prove the theorem below.

Theorem 12 (Optimal security of Π_{12}**)** *For any constant* $c > 0$ *there is a constant* c' *such that the simple construction* Π_{12} *from Figure 21 is* $(c, \frac{c'}{T})$*-Trojan resilient.*

[1]Note that the inputs in this section are denoted with lowercase letters, unlike in the case of tampering HTHs; this is because, in the case of THTHs we never focus on particular bits of the input.

4.1.2 Motivation and possible applications

Very Simple Compilers have many desired properties. First, the trusted module is extremely simple – it only distributes inputs and compares outputs. Secondly, the overhead in the running time is tiny. And lastly, the compiler can use devices that already exist. It is an interesting example of when the security of the existing devices can be improved *after* changing the model.

At the same time, Very Simple Compilers can offer only limited security guarantees, so we must consider whether they have any convincing applications other than improving the security of existing devices. The original motivation for investigating this type of construction was to make it a building block for more advanced ones. To understand one example of such a construction, we must recall the definition of *weak Pseudo Random Function* (weak PRF) - these are Pseudorandom functions that receive random inputs (in particular, their inputs are not controlled by the adversary).

We can think of the following construction of the THTH-resilient weak PRF: it consists of a super master module that xores the outputs of λ instances of $(c, \frac{c'}{T})$-Trojan-resilient weak PRFs. We need to recall a basic fact from probability theory to introduce the intuition behind this idea. Let X, Y be independent random variables over a finite field $\mathbb{F}$ such that X is uniform – then also $X + Y$ is uniform[1]. We can hope that since for each weak PRF, only a tiny fraction of its outputs is wrong, it is unlikely that all of them are wrong; the correct outputs, indistinguishable from random, would mask the incorrect ones, and the output of the whole device would be pseudorandom for random input, even if not correct. Unfortunately, this intuition is neither complete nor entirely accurate. Firstly, the security definition Definition 15 lower-bounds the probability of the average number of errors in the wild phase. We would need a more robust notion, where the error probability is lower-bounded for *every single output*. Secondly, the *correct* outputs from a single subdevice may be, surprisingly, distinguishable from random with noticeable probability. For example, we can think of circuits that deviate for $1/T$ smallest outputs and output 0. Then, correct outputs will never be small, which can be easily distinguished from random by a computationally bounded adversary. Both problems seem to be resolvable, and the manuscript on them is being prepared.

[1]This fact is responsible for the security of the one-time pad encryption.

4.2 Preliminaries[1]

Here, we identify the original circuit F with the desired functionality $\mathcal{F}$. For $m \in \mathbb{N}$, an *m-redundant simple construction* $\Pi_m = (\mathsf{T}^*, \mathsf{M}^*)$ is specified by a master circuit $\mathsf{M}^* : \mathcal{X} \to \mathcal{Y} \cup \{\mathsf{abort}\}$ and a test setup $\mathsf{T}^* : \mathbb{N} \to \{\mathsf{fail}, \mathsf{pass}\}$.[2] The $*$ indicates that they expect access to some *oracles*. The following oracles will be used:

$\mathsf{F}_1, \ldots, \mathsf{F}_m$ — the Trojan circuits that presumably implement the functionality $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$,

F — a trusted implementation of $\mathcal{F}$ or the original circuit F (only available in the test phase), and

$\$$ — a source of random bits (sometimes we will provide the randomness as input instead),

4.2.1 Test and deployment

The construction Π_m, which implements $\mathcal{F}$ in a Trojan-resilient way using the untrusted $\mathsf{F}_1, \ldots, \mathsf{F}_m$, is tested and deployed as follows.

Lab Phase (test): In this first phase we execute $\{\mathsf{pass}, \mathsf{fail}\} \leftarrow \mathsf{T}^{\mathsf{F}_1,\ldots,\mathsf{F}_m,\mathsf{F},\$}(T)$. The input T specifies that each F_i may be queried at most T times. If the output is fail, a Trojan was detected. Otherwise (i.e., the output is pass), we move to the next phase.

Wild Phase (deployment): If the test outputs pass, the F_i's are embedded into the master to get a circuit $\mathsf{M}^{\mathsf{F}_1,\ldots,\mathsf{F}_m,\$} : \mathcal{X} \to \mathcal{Y} \cup \mathsf{abort}$.

4.2.2 Completeness

The completeness requirement states that if every F_i correctly implements $\mathcal{F}$, then the test phase outputs pass with probability 1, and the master truthfully implements the functionality $\mathcal{F}$. That is, for every sequence $x_1, x_2, \ldots, x_q$ (of arbitrary length and potentially with repetitions), we have

$$\Pr[\forall i \in [q] \ : \ y_i = \mathcal{F}(x_i)] = 1 \text{ where } \mathsf{for}\ i = 1\ \mathsf{to}\ q \ : \ y_i := \mathsf{M}^{\mathsf{F}_1,\ldots,\mathsf{F}_m,\$}(x_i)$$

[1] For completeness, we will cite some definitions, intuitions, and theorems from [Cha+21].

[2] We consider much stronger $\mathsf{M}^*, \mathsf{T}^*$ for the lower bounds compared to what we require in the constructions as discussed in Sec. 4.2.5

The reason we define completeness this way and not simply for all x we have $\Pr[\mathsf{M}^{\mathsf{F}_1,\ldots,\mathsf{F}_m,\$}(x) = \mathcal{F}(x)]$ is that the Trojan F_i can be stateful, so the order in which queries are made does matter.

4.2.3 Security of simple schemes

We consider a security game $\mathsf{TrojanGame}(\Pi, T, Q)$ where, for some $T, Q \in \mathbb{Z}$, an adversary $\mathcal{A}$ can choose the functionality $\mathcal{F}$ and the Trojan circuits $\mathsf{F}_1, \ldots, \mathsf{F}_m$. We first run the test phase $\tau \leftarrow \mathsf{T}^{\mathsf{F}_1,\ldots,\mathsf{F}_m,\mathsf{F},\$}(T)$. We then run the wild phase by querying the master on Q iid inputs $x_1, \ldots, x_Q$.

$$\text{for } i = 1, \ldots, Q \ : \ y_i \leftarrow \mathsf{M}^{\mathsf{F}_1,\ldots,\mathsf{F}_m,\$}(x_i).$$

The goal of the adversary is two-fold:

1. They do not want to be caught, if either $\tau = \mathsf{fail}$ or $y_i = \mathsf{abort}$ for some $i \in [Q]$ we say the adversary was detected and define the predicate

$$\mathsf{detect} = \mathsf{false} \iff (\tau = \mathsf{pass}) \wedge (\forall i \in [Q] : y_i \neq \mathsf{abort})$$

2. They want the master to output as many wrong outputs as possible. We denote the number of wrong outputs by $Y \stackrel{\mathsf{def}}{=} |\{i : \ y_i \neq \mathcal{F}(x_i)\}|$.

We can define generic simple schemes.

Definition 16 (Generic Simple Scheme) *A* **generic** *simple scheme* $(\mathsf{T}^*, \mathsf{M}^*)$ *treats the outputs of the* F_i *(and for* T^* *additionally* F*) oracles like variables. Concretely, two or more oracles can be queried on the same input, and then one checks if the outputs are identical. Moreover, the master can use the output of an* F_i *as its output.*

In all our simple constructions, the test and master only use the outputs of the F_i (and for the test also F) oracles to check for equivalence. This fact will allow us to consider somewhat restricted adversaries in the security proof. By the following lemma, to prove the security of generic simple schemes, it will be sufficient to consider restricted adversaries that always choose to attack the trivial functionality $\mathcal{F}(x) = 0$ and where the output range of the Trojans is a bit.

Lemma 5 *For any generic simple scheme* Π_m*, assume an adversary* $\mathcal{A}$ *exists that* $(\mathsf{win}, \mathsf{wrng})$*-wins in* $\mathsf{TrojanGame}(\Pi_m, T, Q)$ *and let* $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$, $\mathsf{F}_1, \ldots, \mathsf{F}_m : \mathcal{X} \to \mathcal{Y}$ *denote its choices for the attack. Then there exists an adversary* $\mathcal{A}'$ *who also* $(\mathsf{win}, \mathsf{wrng})$*-wins in* $\mathsf{TrojanGame}(\Pi_m, T, Q)$ *and chooses* $\mathcal{F}' : \mathcal{X} \to \{0,1\}$, $\mathsf{F}'_1, \ldots, \mathsf{F}'_m : \mathcal{X} \to \{0,1\}$ *where moreover* $\forall x \in \mathcal{X} : \mathcal{F}'(x) = 0$.

Proof. $\mathcal{A}'$ firstly runs $\mathcal{A}$ to learn *(i)* the functionality $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$ which it wants to attack and *(ii)* its Trojans $\mathsf{F}_1, \ldots, \mathsf{F}_m$. It then outputs (as its choice of function to attack) an $\mathcal{F}'$ where $\forall x \in \mathcal{X} : \mathcal{F}'(x) = 0$ and, for every $i \in [m]$, it chooses the Trojan F'_i to output 0 if F_i would output the correct value, and 1 otherwise. More formally, $\mathsf{F}'_i(x)$ invokes the original Trojan $y \leftarrow \mathsf{F}_i(x)$ and outputs 0 if $\mathcal{F}(x) = y$ and 1 otherwise.

By construction, whenever one of the F'_i deviates (i.e., outputs 1), the original F_i would also deviate. And whenever the test or master detects an inconsistency in the new construction, they would also have detected an inconsistency with the original $\mathcal{F}$ and F_i.[1] ∎

4.2.4 Lower bounds

By definition, $(\mathsf{win}, \mathsf{wrng})$-security implies $(\mathsf{win}', \mathsf{wrng}')$-security for any $\mathsf{win}' \geq \mathsf{win}$, $\mathsf{wrng}' \geq \mathsf{wrng}$. The completeness property implies that no scheme is $(1, 0)$-secure (as by behaving honestly, an adversary can $(1, 0)$-win). And also no scheme is $(0, 1)$-secure (as $\Pr[E] \geq 0$ holds for every event E). Thus our $(\mathsf{win}, \mathsf{wrng})$-security notion is only interesting if both, win and wrng are > 0. Now we can prove the lower bound from Lemma 4.

Proof (of Lemma 4*).* $\mathcal{A}$ chooses the constant functionality $\mathcal{F}(x) = 0$ with a sufficiently large input domain $|\mathcal{X}| \gg (m \cdot T)^2$ (so that sampling $m \cdot T$ elements at random from $\mathcal{X}$ with or without repetition is basically the same). Now $\mathcal{A}$ samples a random subset $\mathcal{X}' \subset \mathcal{X}$, $|\mathcal{X}'|/|\mathcal{X}| = 1.1 \cdot c'/T$ (for c' to be determined) and then defines Trojans which deviate on inputs from $\mathcal{X}'$

$$\forall i \in [m] \ : \ \mathsf{F}_i(x) = \begin{cases} 1 & \text{if } x \in \mathcal{X}' \text{ (deviate)} \\ 0 & \text{if } x \notin \mathcal{X}' \text{ (correct)} \end{cases}$$

Should the test pass, the master will deviate on each input with probability $1.1 \cdot c'/T$. If we set the number of queries Q large enough, the fraction of wrong outputs will be close to its expectation $1.1 \cdot c'/T$, and thus almost certainly larger than c'/T.

It remains to prove that the test passes with probability $\geq c$. By correctness, the testing procedure $\mathsf{T}^{\mathsf{F}_1,\ldots,\mathsf{F}_m,\mathsf{F},\$}$ must output pass unless one of the total $\leq m \cdot T$ queries it made to the F_is falls into the random subset $\mathcal{X}'$. The probability that no such query is made is at least

$$(1 - 1.1 \cdot c'T)^{m \cdot T}$$

[1] Let us mention that the opposite is not true (it's possible that for some $i \neq j$ we have $\mathsf{F}'_i(x) = \mathsf{F}'_j(x) = 1$, while $\mathsf{F}_i(x) \neq \mathsf{F}_j(x)$). It captures the observation that an adversary who wants to deviate as often as possible without being detected can always deviate to the same value without loss of generality .

and this expression goes to 1 as c' goes to 0. We now choose $c' > 0$ sufficiently small so the expression becomes $> c$. To get a quantitative bound, one can use the well-known inequality $\lim_{T\to\infty}(1-1/T)^T = 1/e \approx 0.367879$. ∎

The (proof of) the previous lemma also implies the following.

Corollary 6 *If a simple scheme* Π_m *is* $(\mathsf{win}(T), \mathsf{wrng}(T))$ *secure with*

1. $\mathsf{win}(T) \in 1 - o(1)$ *then* $\mathsf{wrng}(T) \in o(1/T)$.
2. $\mathsf{wrng}(T) \in \omega(1/T)$ *then* $\mathsf{win}(T) \in o(1)$.

The first item means that if $\mathcal{A}$ wants to ensure the Trojan is only detected with sub-constant probability, then he can only force the master to output a $o(1/T)$ fraction of wrong outputs during deployment. The second item means that if $\mathcal{A}$ wants to deviate on an asymptotically larger than $1/T$ fraction of outputs, it will be detected with a probability going to 1.

Not interesting security for 1-redundant schemes. For $m = 1$ redundant circuits a much stronger lower bound compared to Lemma 4 holds. The following Lemma implies that no 1-redundant scheme is $(\epsilon(T), \delta(T))$-Trojan-resilient for any $\epsilon(T) > 0$ and $\delta(T) = 1/\mathsf{poly}(T)$ (say $\epsilon(T) = 2^{-T}, \delta(T) = T^{-100}$).

Lemma 6 (Lower bound for $m = 1$) *For any 1-redundant scheme* Π_1 *and any polynomial* $p(T) > 0$*, there is an adversary that* $(1, 1 - 1/p(T))$*-wins in the* $\mathsf{TrojanGame}(\Pi_1, T, Q)$ *game for* $Q \geq p(T) \cdot T$.

Proof. Consider an adversary who chooses a "time bomb" Trojan F_1 which correctly outputs $\mathcal{F}(x)$ for the first T queries and also stores those queries, so it can output the correct value if one of those queries is repeated in the future. From query $T+1$, the Trojan outputs wrong values unless given one of the first T queries as input, in which case it outputs the correct value. This Trojan will pass any test, making at most T invocations, while the master will deviate on almost all queries, i.e., all except the first T.

To see why we store the first T queries and do not deviate on them when they repeat in the future, consider a master that stores the outputs it observes on the first T queries so it can later detect inconsistencies. ∎

4.2.5 Efficiency of lower bound vs. constructions

For the lower bounds in the previous section, the *only* restriction on the test $\mathsf{T}^{\mathsf{F}_1,\ldots,\mathsf{F}_m,\mathsf{F},\$}(T)$ is that each F_i can only be queried at most T times. There

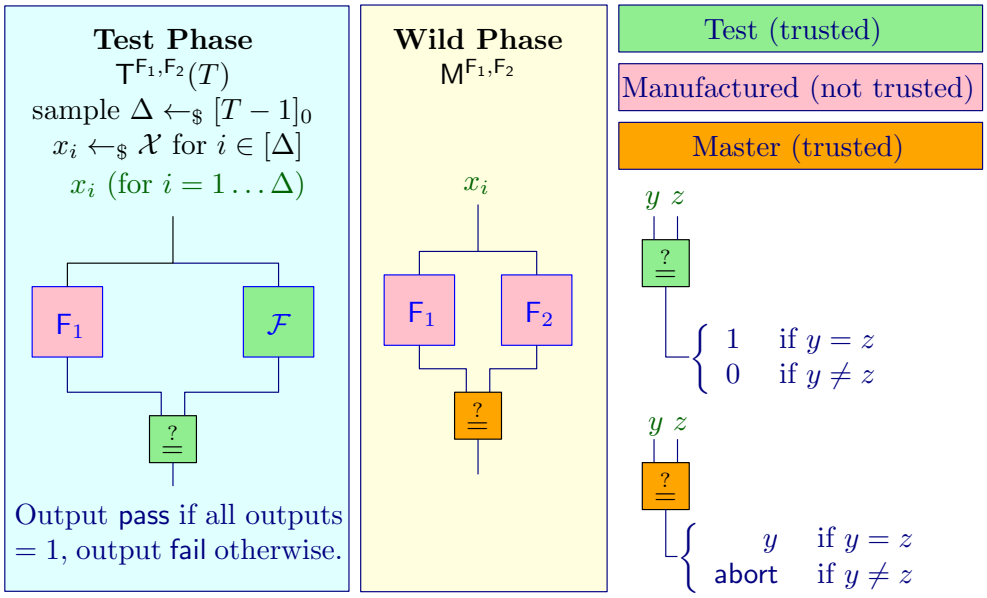


Figure 20: Construction $\Pi_2^\star$ (discussed in Sec. 4.2.8), which is $(c, c'/T)$ secure *for history-independent* Trojans.

are no restrictions on the master $\mathsf{M}^{\mathsf{F}_1,\ldots,\mathsf{F}_m,\$}(\cdot)$ at all. In particular, it can be stateful, computationally unbounded, use an arbitrary amount of randomness, and query the F_is on an unbounded number of inputs (as the Trojan F_is can be stateful, this is not the same as learning the function table of the F_i's).

While the lack of any restrictions strengthens the lower bound, we want our upper bounds, i.e., the actual constructions, to be as efficient (in terms of computational, query, and randomness complexity) and simple as possible, and they will indeed be very simple.

Let us stress that one thing the definition does not allow is the test to pass a message to the master. If we allowed a message of unbounded length to be passed this way, no non-trivial lower bound would hold as T could send the entire function table of $\mathcal{F}$ to M, which then could perfectly implement $\mathcal{F}$. Of course, such a "construction" would go against the motivation for simple schemes where M^* should be much simpler and independent of $\mathcal{F}$. Still, constructions where the test phase sends a short message to the master (say, a few correct input/output pairs of $\mathcal{F}$, which the master could later use to "audit" the Trojans), could be an exciting relaxation to be considered.

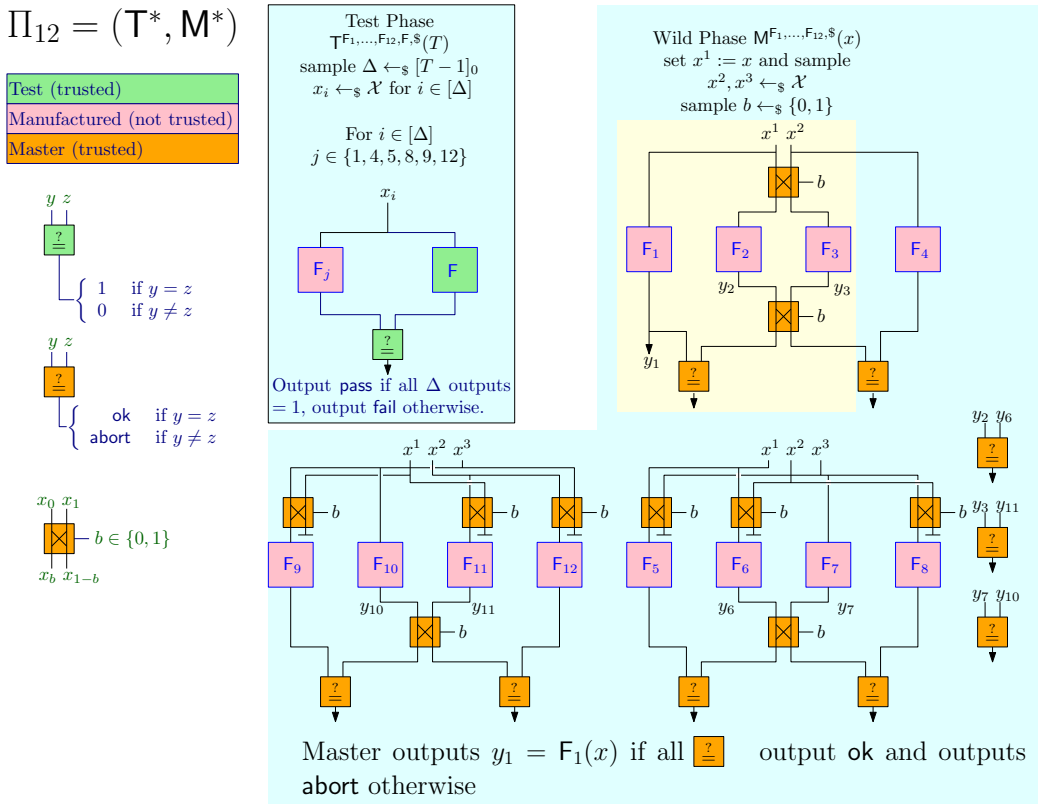


Figure 21: Construction Π_{12} for which we prove optimal Trojan-resilience as stated in Theorem 12. Very informally, the security proof is by contradiction: via a sequence of hybrids, an attack against Π_{12} is shown to imply an attack where the yellow part corresponds to $\Pi_2^\star$ with two history-independent circuits. This attack contradicts the security of $\Pi_2^\star$ as stated in Theorem 13.

4.2.6 Our results and conjectures

Our main technical result is Theorem 12 – construction of a simple scheme that matches the lower bound from Lemma 4. Of course, for any constant $c > 0$, the constant c' in the theorem below must be larger than in Lemma 4, so there is no contradiction.

Let us shortly compare the security we achieve with the more costly solutions based on verifiable computation (VC) [Wah+16; Ate+18] and multiparty computation (MPC) [DFS16] discussed in the introduction. We can consider $(\mathsf{win}, \mathsf{wrng})$-security as in Definition 15 also, for the VC and MPC construction, here one would need to change the $\mathsf{TrojanGame}(\Pi, T, Q)$ from Section 4.2.3 to allow the Trojans F_i to

implement a different functionality than the target $\mathcal{F}$ (for VC, one needs to compute an extra succinct proof, for MPC, the Trojans implement the players in an MPC computation). For VC, there is no test (so $T = 0$) and only one $m = 1$ Trojan, and for MPC and VC, we can drop the requirement that the inputs are iid.

4.2.7 Stateless Trojans

In our security definition, we put no restriction on the Trojans F_i provided by the adversary (other than being *digital* hardware Trojans as discussed in the introduction); in particular, the F_i's can have arbitrary complex evolving state while honestly manufactured circuits could be stateless. For the following few paragraphs, we consider a variant of our security definition (Def. 15) where the adversary is only allowed to choose *stateless* Trojan circuits F_i. Note that the lower bound from Lemma 4 still holds because we only considered stateless F_i's to prove it. An extremely simple 1-redundant construction matches the lower bound when the adversary is only allowed to choose stateless Trojans.

Consider a construction $\Pi_1 = (\mathsf{T}^*, \mathsf{M}^*)$ where the master is the simplest imaginable: it just forwards inputs/outputs to/from its oracle; if F_1 is stateless, this means $\mathsf{M}^{\mathsf{F}_1}(\cdot) = \mathsf{F}_1(\cdot)$. The test $\mathsf{T}^{\mathsf{F}_1,\mathsf{F},\$}(T)$ queries F_1 and the trusted F on T random inputs and outputs fail iff there is a mismatch.

Claim 1 (Optimal security for 1-redundant scheme for *stateless* Trojans) *For any constant $c > 0$, there is a constant $c' > 0$ such that Π_1 is $(c, c'/T)$-Trojan resilient if the adversary is additionally restricted to choose a stateless Trojan.*

Proof. If wrng' denotes the fraction of inputs on which the Trojan F_1 differs from the specification $\mathcal{F}$ (both chosen by an adversary $\mathcal{A}$, note that wrng' is only well defined here as F_1 is stateless), then wrng' must satisfy $c > (1 - \mathsf{wrng}')^T$ if the adversary wants to (c, wrng)-win for any wrng, as otherwise already the test catches the Trojan with probability $(1 - \mathsf{wrng}')^T > c$. For $c > (1 - \mathsf{wrng}')^T$ to hold $\mathsf{wrng}' \in \Omega(1/T)$, in particular, $\mathsf{wrng}' \geq c'/T$ for some $c' > 0$. ∎

4.2.8 History-independent Trojans

A notion of in-between general (stateful) Trojans and stateless Trojans will play a central role in our security proof. We say a trojan F_i is *history-independent* if its only state is a counter which is incremented by one on every invocation, so its answer to the i'th query can depend on the current index i, but not on any inputs it saw in the past.

We observe that Lemma 6 stating that no 1-redundant simple scheme can be secure still holds if we restrict the choice of the adversary to history-independent Trojans as the "time-bomb" Trojan used in the proof is history-independent. We will show a 2-redundant construction $\Pi_2^\star$ that achieves optimal security against history-independent Trojans.

Theorem 13 (History-Independent Security of $\Pi_2^\star$) *For any constant $c > 0$, there is a constant $c' = c'(c) > 0$ such that $\Pi_2^\star$ from Figure 20 is $(c, c'/T)$-Trojan resilient if the adversary is additionally restricted to choose a history-independent Trojans.*

The technical Lemma 7 we prove and which implies this theorem, actually implies a stronger statement: for any positive integer k, the above holds even if we relax the security notion and declare the adversary a winner as long a Trojan is detected by the test or master at most $k-1$ times. Note that this notion coincides with the standard notion for $k = 1$.

Lemma 7 *For any constant $c > 0$ and positive integer k, there exists a constant c', and integer T_0 and polynomial $q(.)$ such that no adversary $\mathcal{A}$ exists that only chooses* history-independent Trojans *and that for any*

$$T \geq T_0 \quad , \quad Q \geq q(T)$$

can $(c, c'/T)$*-k-win* $\mathsf{TrojanGame}(\Pi_2^\star, T, Q)$.

4.2.9 Proof outline

The proof of our main Theorem 12 stating that Π_{12} is optimally Trojan-resilient is done in two steps. As just discussed, we first prove the security of $\Pi_2^\star$ against *history-independent* Trojans. In a second step, we reduce the security of Π_{12} against general Trojans to the security of $\Pi_2^\star$ against history-independent Trojans. We outline the main ideas of the two parts below.

Part 1: security of $\Pi_2^\star$ against history-independent Trojans (Theorem 13, Lemma 7). $\Pi_2^\star$ is a very simple scheme where the test $\mathsf{T}^{\mathsf{F}_1,\mathsf{F}_2,\mathsf{F},\$}$ just queries F_1 on a random number $\Delta, 0 \leq \Delta < T$ of inputs and checks if they are correct (the test ignores F_2). The master $\mathsf{M}^{\mathsf{F}_1,\mathsf{F}_2,\$}(x)$ queries $y \leftarrow \mathsf{F}_1(x)$ and $y' \leftarrow \mathsf{F}_2(x)$ on x and aborts if they disagree.

In the proof of Lemma 7 we define p_i and q_i as the probability that F_1 and F_2 outputs a wrong value in the ith query on a random input, respectively. As $\mathsf{F}_1, \mathsf{F}_2$

are history independent, this is well defined as this probability only depends on i (but not previous queries).

Let the variable Φ_Δ denote the number of times the Trojans will be detected, conditioned on the random number of test queries Δ. Note that

$$\mathbb{E}[\Phi_\Delta] = \sum_{i=1}^{Q+\Delta} |p_i - q_{i-\Delta}|, \tag{10}$$

where Q is the number of queries to the master, and we use the convention $q_i = 0$ for $i < 0$. In this sum, the first Δ terms account for the test and the last Q terms for the wild-phase. Moreover, let Y_Δ denote the number of times F_1 deviates (and thus the master outputs a wrong value); its expectation is

$$\mathbb{E}[Y_\Delta] = \sum_{i=\Delta+1}^{Q+\Delta} p_i$$

To prove Trojan-resilience of $\Pi_2^\star$ as stated in Lemma 7 boils down to proving that, for most Δ, whenever the probability of $\Phi_\Delta = 0$ (i.e., the Trojan is not detected) is constant, the fraction of wrong outputs Y_Δ/Q must be "small" (concretely, $O(1/T)$).

The core technical result establishing this fact is Lemma 8.

Lemma 8 *For any $q_1, \ldots, q_z \in [0,1]$, (defining $q_i = 0$ for $i \leq 0$) and integers Δ', τ where $0 \geq \Delta', \tau \geq 0$, if $p_1 = p_2 = \ldots = p_\tau = 0$ then*

$$\sum_{\Delta \in \{\Delta', \Delta'+\tau\}} \sum_{i=1}^{z} |p_i - q_{i-\Delta}| \geq \tau \cdot \bar{p} \tag{11}$$

Unfortunately, this Lemma only establishes this fact for the expectation, i.e., whenever $\mathbb{E}[\Phi_\Delta]$ is small, also $\mathbb{E}[Y_\Delta]$ is small. Here is where we use the fact that the $\mathsf{F}_1, \mathsf{F}_2$ are history independent: in the history independent case Φ_Δ and Y_Δ can be written as the sum of independent boolean variables, so using a Chernoff bound, it follows that their actual value will be close to their expectation with high probability.

It is instructive to see why, for example, setting $p_i = q_i = \delta$ for some fixed $\delta > 0$ does not contradict Theorem 13. To contradict it, the fraction of wrong outputs (which here is δ) must be $\omega(1/T)$. In this case, $\mathbb{E}[\Phi_\Delta] = \Delta \cdot \delta = \omega(\Delta/T)$, which to contradict the lemma must be at least constant, which in turn means $\Delta \in o(T)$ must hold. As $\Delta, 0 \geq \Delta < T$ is uniform, it's $o(T)$ with $o(1)$ probability, but for a contradiction, we also need this probability to be constant.

Part 2: reducing the security of Π_{12} against general Trojans to the security of $\Pi_2^\star$ against history independent Trojans (Theorem 12). While the random shift Δ makes $\Pi_2^\star$ secure against history-independent attacks (like time-bombs, where a Trojan starts deviating after some fixed number of queries), it succumbs to cheat codes: as the master always queries $\mathsf{F}_1, \mathsf{F}_2$ on the same inputs, a Trojan can specify some set of trigger inputs, and after receiving such a trigger the Trojans will deviate forever. The Trojans will likely not be detected during testing by making the fraction of triggers' inputs sparse (a $1/T$ fraction will survive testing with constant probability).

To prevent such coordination via the inputs, for the construction Π_{12} inputs are somewhat randomly assigned to the different Trojans. In particular, as emphasized in the yellow area in Figure 21, the F_1 is always queried on the input x, and then the random bit b determines whether F_2 or F_3 are queried on x. If an input x were to trigger the Trojans always to deviate, F_1 and one of F_2 and F_3 would be triggered, say it's F_2. But now, as soon as F_3 is queried in a future round, the master will abort as F_1 will deviate, but F_3 will not (except if this query also happens to be a trigger, which is unlikely as triggers must be sparse to survive the testing phase).

The observations above show why a particular attack does not work on Π_{12}. But we want proof of security against all possible Trojans. Our proof proceeds by a sequence of hybrids, where we start with assuming a successful attack on Π_{12}, and then, by carefully switching some circuits and redefining them by hard-coding "fake histories", we arrive at a hybrid game where there is still a successful attack, but now the circuits in the yellow area correspond to two the $\Pi_2^\star$ construction instantiated with history-independent Trojans, but such an attack contradicts our security proof for $\Pi_2^\star$ as stated in Lemma 7.

4.3 An optimal Very Simple Compiler

In this section, we define a scheme Π_{12} and show that the lower bound for achievable security for very simple schemes (shown in Lemma 4) is asymptotically tight. Our proof is constructive – the analysis of our Π_{12} construction shows that it is $(c, \frac{c'}{T})$-Trojan resilient for suitable constants.

Our Π_{12} scheme operates with three independent input streams and one independent bit stream. Every circuit in Π_{12} on each query receives one of the three inputs and produces an output. The master circuit then checks the consistency of the outputs, i.e., verifies if there is no mismatch between any pair of circuits receiving the same input.

Digital Trojans mainly employ two strategies: time bombs (where time is mea-

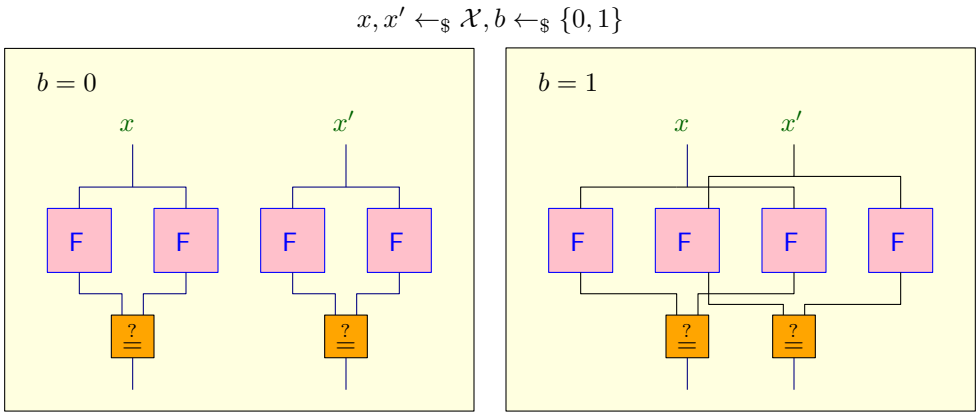


Figure 22: In a given group of circuits, depending on the value of b, the leftmost and rightmost circuits (outer circuits) are paired with the circuits in the middle (inner circuits). Circuits in a pairing are given the same input, and their outputs are checked for equivalence.

sured in the number of usages) and cheat codes (as a part of the input). To counter these strategies, Π_{12} desynchronizes the circuits in two ways. First, some circuits are tested in the test phase for a randomly chosen time (already employed in the Π_2^* scheme). This effectively makes it difficult for the time bomb THTHs to coordinate the time in which they start deviating. In Π_{12}, half of the circuits are tested for Δ times where Δ is a random variable with uniform distribution on $\{0, 1, 2, ..., T-1\}$.

The second method of desynchronization involves using the value of the input mentioned above bit to *alternate* the way inputs are distributed among the circuits. Consequently, cheat code THTHs are rendered ineffective as only a subset of the circuits shares the same input. Moreover, at any given time, a circuit never *knows* which alternating state it is in (i.e., it does not know whether its output would be compared with deviating circuits or not).

The main building block of the Π_{12}-scheme is a group of four circuits: two outer ones and two inner ones (see Figure 22). On each query, every group of circuits receives two inputs – the first is given to the outer circuit on the left and the second to the outer circuit on the right. Additionally, a fresh decision/alternation bit b is sampled in every step. According to its value, these two inputs are given to the inner circuits. Π_{12} consists of three such groups. Crosschecks are performed whenever two distinct circuits receive the same input (both within a group and among groups).

The proof that the construction Π_{12} is Trojan-resilient starts by assuming that it is not secure, goes via a hybrid argument, and leads to a contradiction with the

security of Π_2^* construction. We change the construction slightly in every hybrid by swapping some pairs of circuits, arguing that the adversary's advantage does not change much between successive hybrids. For the final hybrid, we show that the modified construction contains $\Pi_2^\star$ as a sub-construction. It turns out that any adversary who wins with reasonable good probability in the final hybrid can be used to build an adversary who breaks the security of $\Pi_2^\star$, which is a contradiction with Theorem 13.

4.3.1 The Π_{12} scheme

We will now define our Π_{12} construction. It is illustrated in Figure 21 (see page 88). We view our 12-circuit construction as three groups of four circuits each. Group 1 consists of circuits $\mathsf{F}_1, \ldots, \mathsf{F}_4$, group 2 consists of $\mathsf{F}_5, \ldots, \mathsf{F}_8$, and group 3 consists of $\mathsf{F}_9, \ldots, \mathsf{F}_{12}$. In the beginning, the three independent and identically distributed sequences of inputs are sampled. Moreover, an independent sequence of bits is sampled (it is used to alternate the inputs' distribution in the wild). For every query in the wild, the construction performs two steps: (i) the *querying step*, where the inputs are distributed to all the 12 circuits depending on the value of the corresponding bit, and (ii) the *cross-checking step*, where the master circuit checks the consistency of the outputs of the circuits who receive the same inputs.

Now, we can take a closer look at our construction. There are three pairs of circuits that share the same input throughout the game regardless of the value of the random bit (see Figure 21). For instance, the circuit pairs $(\mathsf{F}_2, \mathsf{F}_6)$, $(\mathsf{F}_3, \mathsf{F}_{11})$ and $(\mathsf{F}_7, \mathsf{F}_{10})$ share the exact same inputs throughout the game. The outer two circuits within each circuit group (circuits F_i for $i \equiv 0, 1 \bmod 4$) are uniquely paired with exactly one of the middle circuits, i.e., given the same input, depending on the value of the random bit b_i sampled by the master circuit at each step of the game. For instance, in circuit group 1, if $b_i = 0$, F_1 is paired with F_2 and both circuits are given x_i^1 as input, and F_4 is paired with F_3 and both are given x_i^2 as input. After the querying phase, the master cross-checks the circuits' output that share the same input streams. If any cross-checks in any round fail, then the master aborts and the adversary loses.

We now provide a more detailed description of the construction as follows:

test: In the test phase, $\mathsf{T}^{\mathsf{F}_1,\cdots,\mathsf{F}_{12},\mathsf{F}}(T)$ queries $\mathsf{F}_1, \mathsf{F}_4, \mathsf{F}_5, \mathsf{F}_8, \mathsf{F}_9$ and F_{12} on Δ random and independent inputs $x_i^1, x_i^4, x_i^5, x_i^8, x_i^9$ and x_i^{12} respectively and checks if the outputs of the corresponding circuits are correct by comparing them with the trusted F (recall that Δ is random and uniform on $\{0, 1, ..., T-1\}$).

master: The master samples three independent input streams $\vec{x_1} = (x_1^1, x_2^1, x_3^1, \cdots), \vec{x_2} = (x_1^2, x_2^2, x_3^2, \cdots), \vec{x_3} = (x_1^3, x_2^3, x_3^3, \cdots .)$ and an independent bit string $\vec{b} = (b_1, b_2, \cdots)$. The operation of the master circuit is split into two phases: (i) the querying phase and (ii) the cross-checking phase.

Querying step. For all $i \in [Q]$, it queries the functions $\mathsf{F}_1, \mathsf{F}_2 \cdots, \mathsf{F}_{12}$ as follows:

1. If $b_i = 0$,
   - The functions $\mathsf{F}_1, \mathsf{F}_2, \mathsf{F}_5, \mathsf{F}_6$ get x_i^1 as input,
   - The functions $\mathsf{F}_3, \mathsf{F}_4, \mathsf{F}_{11}, \mathsf{F}_{12}$ get x_i^2 as input,
   - The functions $\mathsf{F}_7, \mathsf{F}_8, \mathsf{F}_9, \mathsf{F}_{10}$ get x_i^3 as input.
2. if $b_i = 1$,
   - The functions $\mathsf{F}_1, \mathsf{F}_3, \mathsf{F}_9, \mathsf{F}_{11}$ get x_i^1 as input,
   - The functions $\mathsf{F}_2, \mathsf{F}_4, \mathsf{F}_6, \mathsf{F}_8$ get x_i^2 as input,
   - The functions $\mathsf{F}_5, \mathsf{F}_7, \mathsf{F}_{10}, \mathsf{F}_{12}$ get x_i^3 as input

Cross-checking step. For all $i \in [Q]$, the master circuit pairwise compares the outputs of the circuits that receive the same inputs as follows:

- if $b_i = 0$, check if
  - $\mathsf{F}_1(x_i^1) \stackrel{?}{=} \mathsf{F}_2(x_i^1)$, $\mathsf{F}_5(x_i^1) \stackrel{?}{=} \mathsf{F}_6(x_i^1)$, and $\mathsf{F}_2(x_i^1) \stackrel{?}{=} \mathsf{F}_6(x_i^1)$.
  - $\mathsf{F}_3(x_i^2) \stackrel{?}{=} \mathsf{F}_4(x_i^2)$, $\mathsf{F}_{11}(x_i^2) \stackrel{?}{=} \mathsf{F}_{12}(x_i^2)$, and $\mathsf{F}_3(x_i^2) \stackrel{?}{=} \mathsf{F}_{11}(x_i^2)$,
  - $\mathsf{F}_7(x_i^3) \stackrel{?}{=} \mathsf{F}_8(x_i^3)$, $\mathsf{F}_9(x_i^3) \stackrel{?}{=} \mathsf{F}_{10}(x_i^3)$, and $\mathsf{F}_7(x_i^3) \stackrel{?}{=} \mathsf{F}_{10}(x_i^3)$,
- if $b_i = 1$, check if
  - $\mathsf{F}_1(x_1^i) \stackrel{?}{=} \mathsf{F}_3(x_i^1)$, $\mathsf{F}_9(x_i^1) \stackrel{?}{=} \mathsf{F}_{11}(x_i^1)$, and $\mathsf{F}_3(x_i^1) \stackrel{?}{=} \mathsf{F}_{11}(x_i^1)$.
  - $\mathsf{F}_2(x_i^2) \stackrel{?}{=} \mathsf{F}_4(x_i^2)$, $\mathsf{F}_6(x_i^2) \stackrel{?}{=} \mathsf{F}_8(x_i^2)$, and $\mathsf{F}_2(x_i^2) \stackrel{?}{=} \mathsf{F}_6(x_i^2)$,
  - $\mathsf{F}_5(x_i^3) \stackrel{?}{=} \mathsf{F}_7(x_i^3)$, $\mathsf{F}_{10}(x_i^3) \stackrel{?}{=} \mathsf{F}_{12}(x_i^3)$, and $\mathsf{F}_7(x_i^3) \stackrel{?}{=} \mathsf{F}_{10}(x_i^3)$,

If, at any round, any of the checks fail, the master outputs abort, and the adversary loses.

Output. If all the checks succeed in the cross-checking phase, the master outputs the output of the circuit F_1, i.e., $\vec{y} = \mathsf{F}_1(\vec{x^1})$ as the output of Π_{12}.

4.3.2 Security of Π_{12}

In this section, we prove Theorem 12, which states that the construction presented in Sec. 4.3.1 is $(c, \frac{c'}{T})$-secure for an appropriate choice of constants c and c'. More precisely, we show that the security of the 2-circuit construction from Sec. 4.2.8 can be reduced to the security of the 12-circuit construction presented above. Before proceeding with the proof, we introduce some useful definitions and notations.

History hardcoded circuits and plaits. We observe that the notation $\mathsf{F}(x)$ for *stateful* circuits is ambiguous since its value also depends on the *history* of queries to F (which is not provided as a parameter). We can thus assume that each F is associated with some stream $\mathbf{x} = (x_1, x_2, ...)$ and that $\mathsf{F}(x_i) := \mathsf{F}(x_i|x_1, x_2, ..., x_{i-1})$. This notation uniquely describes the i-th query to F given the stream $\mathbf{x}$.

However, for our proof, we will need a slightly different notion called *history-hardcoded circuits*. Given any stateful circuit F and two arbitrary streams $\mathbf{x} = (x_1, x_2, x_3, ...)$ and $\mathbf{w} = (w_1, w_2, w_3, ...)$, we say $\mathsf{F}^{\mathbf{x}}$ is an $\mathbf{x}$-history-hardcoded circuit if at the i-th query it *hardcodes* the stream values $x_1, \ldots, x_{i-1}$ as its history and takes w_i from the stream $\mathbf{w}$ as the input to query i. Thus $\mathsf{F}^{\mathbf{x}}$ on the i-th query with input w_i returns the value: $\mathsf{F}^{\mathbf{x},i}(w_i) = \mathsf{F}(w_i|x_1, x_2, ..., x_{i-1})$ and on the $i+1$-th query returns $\mathsf{F}^{\mathbf{x},i+1}(w_{i+1}) = \mathsf{F}(w_{i+1}|x_1, x_2, ..., x_{i-1}, x_i)$. We call the stream $\mathbf{x}$ the *hardcoded history stream* and $\mathbf{w}$ the *input stream*.

For a random variable $\mathbf{X}$ which takes values from $\{X_1, X_2, ...\}$ and a circuit F we define another random variable $\mathsf{F}^{\mathbf{X}}$ as follows. Its value for $\mathbf{X} = \mathbf{x}$ is simply $\mathsf{F}^{\mathbf{x}}$. We will call this random variable an $\mathbf{X}$-*history-hardcoded* circuit. Note that as long as $\mathsf{F}^{\mathbf{X}}$ receives inputs from a stream $\mathbf{W}$ *independent* from $\mathbf{X}$, we can say that $\mathsf{F}^{\mathbf{x}}$ is a *history-independent* circuit.

We emphasize that when the hardcoded history stream equals the actual input stream, the history-hardcoded circuit returns the same results as the original stateful circuit receiving the same input stream. In other words:

$$\mathsf{F}(X_i) = \mathsf{F}^{\mathbf{X},i}(X_i), \tag{12}$$

for all $i \in \mathbb{N}$ with probability 1.

Another idea exploited in our construction is the concept of *alternating* inputs depending on the values of random bits. We will express this idea using the notion of $\mathbf{b}$-*plaits*, where $\mathbf{b}$ is a stream of random bits. A $\mathbf{b}$-plait of two streams $\mathbf{a}^0$ and $\mathbf{a}^1$ is a new stream $(\mathbf{a}^0\mathbf{a}^1)_{\mathbf{b}}$, where its i-th value is either a_i^0 from stream $\mathbf{a}^0$ or a_i^1 from stream $\mathbf{a}^1$ depending on the i-th value of the *decision* stream $\mathbf{b}$. More precisely:

$$(\mathbf{a}^0\mathbf{a}^1)_{\mathbf{b}} = (a_1^{b_1}, a_2^{b_2}, a_3^{b_3}, ...)$$

In our construction, only one decision stream used for every plait; therefore, the $\mathbf{b}$ will be omitted for simplicity. Thus, to express the plait of two streams $\mathbf{a}^0$, $\mathbf{a}^1$, we will simply write $\mathbf{a}^0\mathbf{a}^1$. A plait of two identical streams of, say, $\mathbf{s}$ will be written as $\mathbf{s}$ rather than $\mathbf{ss}$.

Similarly to $\mathbf{b}$-plaits of streams, we can define the plaits of history-hardcoded circuits. Let $\mathsf{G}_0^{\mathbf{x}^0}$ be an $\mathbf{x}^0$-history-hardcoded circuit and $\mathsf{G}_1^{\mathbf{x}^1}$ be an $\mathbf{x}^1$-history-hardcoded circuit. We say $(\mathsf{G}_0^{\mathbf{x}^0}\mathsf{G}_1^{\mathbf{x}^1})_{\mathbf{b}}$ is $\mathbf{b}$-*plait* for $\mathsf{G}_0^{\mathbf{x}^0}, \mathsf{G}_1^{\mathbf{x}^1}$ iff

$$(\mathsf{G}_0^{\mathbf{x}^0}\mathsf{G}_1^{\mathbf{x}^1})_{\mathbf{b}}^{\mathsf{i}}(\mathsf{x}) = \mathsf{G}_{\mathsf{b}^{\mathsf{i}}}^{\mathbf{x}^{\mathsf{b}_{\mathsf{i}}},\mathsf{i}}(\mathsf{x}). \tag{13}$$

Note that the plaited circuit $(\mathsf{G}_0^{\mathbf{x}^0}\mathsf{G}_1^{\mathbf{x}^1})_{\mathbf{b}}$ can be expressed as a function of $\mathsf{G}_0, \mathsf{G}_1$ and streams $\mathbf{x}^0, \mathbf{x}^1$. Looking ahead, this notion of plaited circuits will be crucial in our final reduction of the security of Π_{12} to $\Pi_2^\star$

Finally, we define an operation on history-hardcoded circuits in the context of our construction:

$\mathsf{Swap}(\mathsf{F}^{\mathbf{x}}, \mathsf{G}^{\mathbf{t}})$: Given two history hardcoded circuits $\mathsf{F}^{\mathbf{x}}$ and $\mathsf{G}^{\mathbf{t}}$ in our construction, this operation physically exchanges the positions of both circuits. That is, that $\mathsf{F}^{\mathbf{t}}$ physically replaces $\mathsf{G}^{\mathbf{x}}$ and vice versa. Swapped circuits keep their histories, but since they change their place in the construction, they now receive potentially different inputs and are crosschecked with different circuits.

An important notion related to the Swap operation, which we will exploit in a proof, is a *red edge*. We say there is a *red edge* in the k-th query between two history hardcoded circuits $\mathsf{F}^{\mathbf{x}}$ and $\mathsf{G}^{\mathbf{t}}$ iff after performing the $\mathsf{Swap}(\mathsf{F}, \mathsf{G})$ operation there is a change in either of the outputs of the swapped circuits on the k-th query compared to the outputs of the circuits if the Swap operation was not performed. The notion of swaps and red edges would be used in our proof to show that modifying the original Π_{12} construction by some Swap operations does not change much the security parameters.

Now, given these definitions, we are ready to present an intuition that lies behind our construction. The readers can ask the authors, "But why 12 circuits?". The reason is that it is hard to perform any direct proof for history-dependent circuits; things become too complicated. However, we have Theorem 13 for 2 history-*independent* circuits. Therefore, we must find a reduction between these two cases. The main goal is to design the crosscheck for history-dependent circuits so that when the adversary makes the Trojans *more history-dependent*, the construction becomes *more secure*. For our Π_{12} construction, this statement looks valid – thanks to the alternating random bit, you never know which of some two circuits will receive specific input. If these two circuits are, informally speaking, *very* history-dependent and

have independent histories, there is a high probability that, on the given input, they would answer differently. Thanks to crosschecks, the master may detect such inconsistency with high probability. To make a practical advantage of this remark, we need to perform many Swap operations and analyze the behavior of various parameters describing our construction. We were able to handle such design and analysis for 12-circuit construction.

Now, we will give a more detailed description of intuition. As written a few lines before, the main idea of the proof is to reduce the construction, which consists of (possibly) history-dependent circuits, to $\Pi_2^\star$. $\Pi_2^\star$ consists of 2 history-independent circuits (alternatively speaking – pairs of circuits with different hardcoded histories, independent of the inputs they receive). The Swap operation $\mathsf{Swap}(\mathsf{F}^{\mathbf{x}}, \mathsf{G}^{\mathbf{t}})$ is legit whenever either one of the conditions holds – the circuits F and G are engaged in the cross-checking process as pictured in the Figure 22 (e.g. circuits F_1 and F_4 or circuits F_6 and F_7 in the Figure 23 (Hyb_0) or the circuits received the same inputs before performing any swaps (e.g. circuits F_2 and F_7 swapped in Hyb_2 which are placed at the positions of F_2 and F_6 from Hyb_0 in the Figure 23). Now, the main idea of the proof is that by performing a series of Swap operations on the setting with 3 rows of 4-circuit groups, we can end up with a setting Hyb_2 that contains 2 pairs of history-independent circuits at the place of cross-checked circuit pairs (F_1, F_2) and (F_3, F_4). We need just 1 Swap operation in the middle row to have history-independent circuits in the place of F_1 and F_4, but for F_2 (and F_2), we will need an additional input stream that goes with a new row. We are now ready to proceed to the proof of Theorem 12.

Proof of Theorem 12. The proof of Theorem 12 proceeds in two parts. We ultimately want to prove a reduction from the security of Π_{12} to that of $\Pi_2^\star$. Nevertheless, recall in Lemma 7 the security of $\Pi_2^\star$ crucially depends on history-independent circuits. Thus, the first part of our proof constructs a sequence of three hybrids, Hyb_0, Hyb_1, Hyb_2, to get a pair of history-independent circuits, $\mathsf{F}_4^{\mathbf{2}}$ and $\mathsf{F}_7^{\mathbf{3}}\mathsf{F}_{10}^{\mathbf{3}}$, in the final hybrid. Hyb_0 is the original construction. To get from Hyb_0 to Hyb_1, we perform the Swap operation on the following pairs of circuits in Hyb_0: $(\mathsf{F}_1^{\mathbf{1}}, \mathsf{F}_4^{\mathbf{2}})$; $(\mathsf{F}_6^{\mathbf{12}}, \mathsf{F}_7^{\mathbf{3}})$; $(\mathsf{F}_{10}^{\mathbf{3}}, \mathsf{F}_{11}^{\mathbf{21}})$. To get from Hyb_1 to Hyb_2, we perform the Swap operation on the following pairs of circuits in Hyb_1: $(\mathsf{F}_2^{\mathbf{12}}, \mathsf{F}_7^{\mathbf{3}})$; $(\mathsf{F}_3^{\mathbf{21}}, \mathsf{F}_{10}^{\mathbf{3}})$ (refer to Figure 23). Note that in the final hybrid Hyb_2, it is crucial that $\mathsf{F}_4^{\mathbf{2}}$ and $\mathsf{F}_7^{\mathbf{3}}\mathsf{F}_{10}^{\mathbf{3}}$ are not just history independent, but also take in the same inputs from input stream $\mathbf{1}$ regardless of the value of the random bit ($\mathsf{F}_7^{\mathbf{3}}\mathsf{F}_{10}^{\mathbf{3}}$ takes inputs from stream $\mathbf{1}$ due to the definition of the plaited circuit in (13)). This will be necessary for the second part of our proof which uses $\mathsf{F}_4^{\mathbf{2}}$ and $\mathsf{F}_7^{\mathbf{3}}\mathsf{F}_{10}^{\mathbf{3}}$ in the final hybrid as the two history independent circuits needed for the $\Pi_2^\star$

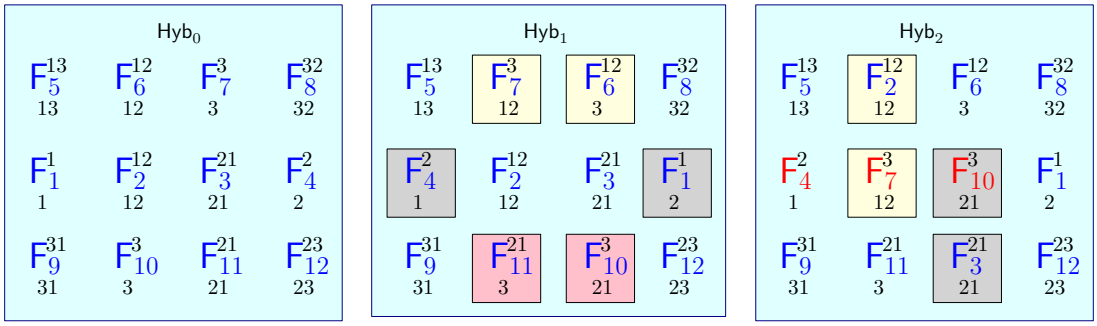


Figure 23: Hybrids with the circuits and corresponding plaited hardcoded history and input streams (above and below each circuit in black, respectively). In Hyb_2, $\mathsf{F}_4^{\mathbf{2}}$ and the plaited circuit $\mathsf{F}_7^{\mathbf{3}}\mathsf{F}_{10}^{\mathbf{3}}$ (in red) are history independent.

construction and uses the $\Pi_2^\star$ construction with these circuits as a subroutine.

Proof. For a given $\mathsf{F}_1, ..., \mathsf{F}_{12}$, we can define some random variables as follows. Let $\phi_{\mathsf{F}_j^{\mathbf{A}};\mathbf{B}}$ be the total number of queries, where $\mathsf{F}_j^{\mathbf{A}}$ gets input from a stream $\mathbf{B}$ and has a mismatch with any other circuit getting input from the same stream in this query. We will refer to random variables related to the i-th hybrid by adding a superscript i. For example, $\phi^0_{\mathsf{F}_1^{\mathbf{1}};\mathbf{2}} = 0$, since in Hyb_0 no crosschecks are made between $\mathsf{F}_1^{\mathbf{1}}$ and the circuits receiving inputs from stream $\mathbf{2}$. Let Φ be the total number of mismatches detected by the master circuit. Recall from Sec. 4.2.3 that Y is the total number of mistakes the master circuit makes. The probability space of these random variables is the set of all choices of a stream of random bits $\mathbf{b}$ and streams of random inputs $\mathbf{1}, \mathbf{2}, \mathbf{3}$ and a number of tests Δ.

We prove our statement by contradiction. To this end, we assume that

$$\begin{gathered}\exists_{c>0}\forall_{c'>0,T_0\in\mathbb{N},q\in poly}\exists_{T>T_0,Q>q(T)}\exists_{\mathcal{A}} \text{ such that} \\ \mathcal{A}\left(c,\tfrac{c'}{T}\right)\text{–wins} \quad \mathsf{TrojanGame}(\Pi_{12},T,Q)\end{gathered} \tag{14}$$

Therefore for some c and for all c' there exists an infinite set $\mathcal{T} \subset \mathbb{N}$ such that for every $t \in \mathcal{T}$ there exists an infinite set $\mathcal{Q}_t \subset \mathbb{N}$ such that for every $t \in \mathcal{T}, z \in \mathcal{Q}_t$ there exists an adversary $\mathcal{A} = \mathcal{A}(c, c', z, t)$ such that the following formula is true:

$$\Pr\left[\Phi^0 = 0 \quad \text{and} \quad Y^0 \geq c' \cdot \left(\frac{z}{t}\right)\right] \geq c. \tag{15}$$

Now we will look what happens to inequality (15) as we move through each hybrid:

Hyb_0 : Hybrid 0 corresponds to the original construction due to equality (12). Hence, the probability that the adversary $\mathcal{A}(c, c', z, t)$ wins in this hybrid is precisely that

in Equation (15).

Hyb_1: In this hybrid we simply perform three Swap operations on the following pairs of circuits: $(\mathsf{F}^{\mathbf{1}}_1, \mathsf{F}^{\mathbf{2}}_4)$; $(\mathsf{F}^{\mathbf{12}}_6, \mathsf{F}^{\mathbf{3}}_7)$; $(\mathsf{F}^{\mathbf{3}}_{10}, \mathsf{F}^{\mathbf{21}}_{11})$.

Claim 2 $\Pr\left[\phi^1_{\mathsf{F}^{\mathbf{2}}_4,\mathbf{1}}, \phi^1_{\mathsf{F}^{\mathbf{3}}_7,\mathbf{12}}, \phi^1_{\mathsf{F}^{\mathbf{3}}_{10},\mathbf{21}} \leq k \wedge \Phi^0 = 0 \wedge Y^1 \geq c' \cdot (\frac{z}{t}) - 3k\right] \geq c - 3 \cdot 2^{-k}$.

Proof. After the first two swaps, two disjoint events may occur: either the number of the red edges exceeds some value k, or it stays small. We study these two cases below:

- Huge_k–any number of red edges between $(\mathsf{F}^{\mathbf{1}}_1, \mathsf{F}^{\mathbf{2}}_4)$, $(\mathsf{F}^{\mathbf{12}}_6, \mathsf{F}^{\mathbf{3}}_7)$, $(\mathsf{F}^{\mathbf{3}}_{10}, \mathsf{F}^{\mathbf{21}}_{11})$ is greater than k. We would like to show that

$$\Pr\left[\mathsf{Huge}_\mathsf{k} \wedge (\Phi^0 = 0)\right] \leq 3 \cdot 2^{-k}. \tag{16}$$

  We will firstly consider the red edges between $\mathsf{F}^{\mathbf{1}}_1, \mathsf{F}^{\mathbf{2}}_4$. For this purpose, we define an event huge_k indicating that the number of red edges between $\mathsf{F}^{\mathbf{1}}_1, \mathsf{F}^{\mathbf{2}}_4$ is greater than k. We consider the following game that shows that, in this case, the $\mathcal{A}$ is easily caught cheating. G lasts for z steps with a player P: at the beginning streams $\mathbf{1}', \mathbf{2}', \mathbf{b}$ are sampled uniformly and independently from $\mathcal{X}^z, \mathcal{X}^z, \{0,1\}^z$, respectively. The i-th step has the following substeps:

  1. $\mathbf{1}'_i, \mathbf{2}'_i$ are sampled uniformly and independently from $\mathcal{X}$ and revealed to P.
  2. P decides if this step is *red*, i.e. if she wants to make a guess.
  3. If this step is red, P makes a guess, i.e. outputs a bit d_i.
  4. b_i is sampled uniformly and independently from $\{0,1\}$ and revealed to P.
  5. If P made a guess, we say P is correct iff $d_i = b_i$.

  P wins, if she made at least k guesses and all of them were correct.

  Obviously, the probability of winning is not greater than 2^{-k} for any P. On the other hand, we can analyze what happens if the adversary employs the circuits $\mathsf{F}_1, \mathsf{F}_4$ from the original experiment to play the game for him.

  The strategy of the player P in game G is following: $\mathsf{F}^{\mathbf{12}}_2, \mathsf{F}^{\mathbf{21}}_3$ in the i-th step get inputs $\mathbf{1}'_i, \mathbf{2}'_i$, respectively. The step is red if there is a red edge in the original experiment, i.e., if:

$$\mathsf{F}^1_1(\mathbf{1}'_i) \neq \mathsf{F}^{\mathbf{2}}_4(\mathbf{1}'_i) \vee \mathsf{F}^1_1(\mathbf{2}'_i) \neq \mathsf{F}^{\mathbf{2}}_4(\mathbf{2}'_i).$$

We do not analyze the way d_i is chosen, since $\Pr\left[d_i = b_i\right] = \frac{1}{2}$. For such a strategy P wins the game with probability at least $\Pr\left[\mathsf{huge_k} \wedge (\Phi^0 = 0)\right]$, therefore

$$\Pr\left[\mathsf{huge_k} \wedge (\Phi^0 = 0)\right] \leq 2^{-k}.$$

The same conclusion works for pairs $(\mathsf{F}_6^{\mathbf{12}}, \mathsf{F}_7^{\mathbf{3}}), (\mathsf{F}_{10}^{\mathbf{3}}, \mathsf{F}_{11}^{\mathbf{21}})$, therefore by the bound on the sum of the probabilities we can conclude that the inequality (16) holds.

- $\mathsf{Tiny_k}$–in this case every number of red edges between $(\mathsf{F}_1^{\mathbf{1}}, \mathsf{F}_4^{\mathbf{2}})$, $(\mathsf{F}_6^{\mathbf{12}}, \mathsf{F}_7^{\mathbf{3}})$, $(\mathsf{F}_{10}^{\mathbf{3}}, \mathsf{F}_{11}^{\mathbf{21}})$ is not greater than k. 3 Swap operations performed in Hyb_1 will not change the value Y^1 by more than $3k$ and the behavior of e.g. $\mathsf{F}_4^{\mathbf{2}}$ differs from $\mathsf{F}_1^{\mathbf{1}}$ only on up to k queries, i.e.

$$\phi^1_{\mathsf{F}_4^{\mathbf{2}},\mathbf{1}}, \phi^1_{\mathsf{F}_7^{\mathbf{3}},\mathbf{12}}, \phi^1_{\mathsf{F}_{10}^{\mathbf{3}},\mathbf{21}} \leq k \tag{17}$$

$$|Y^0 - Y^1| \leq 3k. \tag{18}$$

Now we can transform inequality (15)

$$\begin{aligned}
c - 3 \cdot 2^{-k} \leq \; & \Pr\left[\Phi^0 = 0 \wedge Y^0 \geq c' \cdot (\tfrac{z}{t})\right] - 2^{-(k+1)} = \\
= \; & \Pr\left[\mathsf{Tiny_k} \wedge \Phi^0 = 0 \wedge \mathsf{Y}^0 \geq \mathsf{c}' \cdot (\tfrac{\mathsf{z}}{\mathsf{t}})\right] + \\
& + \Pr\left[\mathsf{Huge_k} \wedge \Phi^0 = 0 \wedge \mathsf{Y}^0 \geq \mathsf{c}' \cdot (\tfrac{\mathsf{z}}{\mathsf{t}})\right] - 2^{-(\mathsf{k}+1)} \leq \\
\leq \; & \Pr\left[\mathsf{Tiny_k} \wedge \Phi^0 = 0 \wedge \mathsf{Y}^0 \geq \mathsf{c}' \cdot (\tfrac{\mathsf{z}}{\mathsf{t}})\right] \leq \\
\leq \; & \Pr\left[\phi^1_{\mathsf{F}_4^{\mathbf{2}},\mathbf{1}}, \phi^1_{\mathsf{F}_7^{\mathbf{3}},\mathbf{12}}, \phi^1_{\mathsf{F}_{10}^{\mathbf{3}},\mathbf{21}} \leq k \wedge \Phi^0 = 0 \wedge Y^1 \geq c' \cdot (\tfrac{z}{t}) - 3k\right].
\end{aligned}$$

□

Hyb_2 : In this hybrid we simply perform two Swap operations on the following pairs of circuits: $(\mathsf{F}_2^{\mathbf{12}}, \mathsf{F}_7^{\mathbf{3}})$; $(\mathsf{F}_3^{\mathbf{21}}, \mathsf{F}_{10}^{\mathbf{3}})$.

Claim 3

$$\Pr\left[(\phi^2_{\mathsf{F}_4^{\mathbf{2}},\mathbf{1}} \leq 3k) \wedge (Y^2 \geq c'(\frac{z}{t}) - 5k)\right] \geq c - 3 \cdot 2^{-k}. \tag{19}$$

Proof. Every Swap operation performed in Hyb_2 changes the value of $Y^2, \phi^2_{\mathsf{F}_4^{\mathbf{2}},\mathbf{1}}$ by at most k (since inequality (17) holds). The inequality is explicit. □

Claim 4 *For every $k \in \mathbb{N}$ and every adversary $\mathcal{A}$ who $(c, \frac{c'}{t})$-wins (Π_{12}, T, Q) – $\mathsf{TrojanGame}$ there exists an adversary $\mathcal{A}'$ who $(c - 3 \cdot 2^{-k}, \frac{c'}{t} - \frac{5k}{z})$-$(3k+1)$-wins the game $\mathsf{TrojanGame}(\Pi_2^\star, T, Q)$.*

We want to conclude that the above statement contradicts Lemma 7. So we want to show, that our construction implies this incorrect statement.

$$\begin{aligned}&\exists_{\tilde{c}>0}\forall_{\tilde{c'}>0,T_0\in\mathbb{N},q\in poly}\exists_{T>T_0,Q>q(T)}\exists_{\widetilde{\mathcal{A}}} \text{ such that}\\ &\widetilde{\mathcal{A}} \quad (\tilde{c},\tfrac{\tilde{c'}}{T})-\text{wins} \quad \mathsf{TrojanGame}(\Pi_2^\star,T,Q).\end{aligned} \tag{20}$$

Let $k = 2 + \log(\frac{1}{c})$ and $\tilde{c} = c - 3 \cdot 2^{-k} = \frac{c}{4} > 0$. Choose $\tilde{c'}$ arbitrarily and let $c' = \cdot\tilde{c'}$. Let $\widetilde{\mathcal{T}} = \mathcal{T}$. Let

$$\widetilde{\mathcal{Q}_t} = \{z \in \mathcal{Q}_t : z > t\left(\frac{5k}{\tilde{c'}} + 1\right)\}.$$

Obviously $\widetilde{\mathcal{Q}_t}$ is infinite. As a result, for every $q \in poly$, there exists $z \in \widetilde{\mathcal{Q}_t}$ such that $z > q(t)$.

Now we can construct the adversary $\widetilde{\mathcal{A}}$, which would break the security of $\Pi_2^\star$, which leads us to contradiction. Thanks to the analysis of the hybrids, we know that for $\mathcal{A}$ the inequality (19) holds. Define the circuits $\widetilde{\mathsf{F}_1}, \widetilde{\mathsf{F}_2}$ given to $\widetilde{\mathcal{A}}$ in the following way:

$$\widetilde{\mathsf{F}_1} = \mathsf{F}_4^{\mathbf{2}}, \quad \widetilde{\mathsf{F}_2} = \mathsf{F}_7^{\mathbf{3}}\mathsf{F}_{10}^{\mathbf{3}},$$

where the latter is a $\mathbf{b}$-plait (as defined in Equation (13)). Actual values of streams $\mathbf{2}, \mathbf{3}, \mathbf{b}$ are sampled uniformly and independently by $\widetilde{\mathcal{A}}$, and hardcoded in $\widetilde{\mathsf{F}_1}, \widetilde{\mathsf{F}_2}$. Obviously $\widetilde{\mathsf{F}_1}, \widetilde{\mathsf{F}_2}$ are history independent, therefore $\widetilde{\mathcal{A}}$ meets the requirements for the $\Pi_2^\star$ scheme.

Now we can bound a random variable $\widetilde{\Phi}$ – the number of queries in a $(\Pi_2^\star, T, Q)-$ $\mathsf{TrojanGame}$ where the adversary is caught on deviating. If $\phi^2_{\mathsf{F}_4^{\mathbf{2}},\mathbf{1}} \leq 3k$, then $\widetilde{\Phi} \leq 3k$, since if in the i-th query there was any inconsistency between $\widetilde{\mathsf{F}_1}, \widetilde{\mathsf{F}_2}$, there must had been a mismatch between $\mathsf{F}_4^{\mathbf{3}}$ and any other circuit receiving the same input. Which concludes with:

$$\begin{aligned}\tilde{c} = c - 3 \cdot 2^{-k} &\leq \Pr\left[(\phi^2_{\mathsf{F}_4^{\mathbf{2}},\mathbf{1}} \leq 3k) \wedge (Y^2 \geq c'(\tfrac{z}{t}) - 5k)\right]\\ &\leq \Pr\left[\widetilde{\Phi} \leq 3k \wedge (\widetilde{Y} \geq c'(\tfrac{z}{t}) - 5k)\right] \leq\\ &\leq \Pr\left[\widetilde{\Phi} \leq 3k \wedge (\widetilde{Y} \geq \tilde{c'}(\tfrac{5k}{\tilde{c'}} + 1) - 5k = \tilde{c'})\right].\end{aligned}$$

Note that $\widetilde{Y}$ is the number of mistakes made by the master circuit in the $\mathsf{TrojanGame}(\Pi_2^\star, T, Q)$, and the last inequality comes from $z > t\left(\frac{5k}{c'} + 1\right)$. We conclude, that that there exists $\tilde{c}$, such that for *every* $\tilde{c'}$ there exists $\widetilde{\mathcal{A}}$ who $(\tilde{c}, \frac{\tilde{c'}}{t})$-$(3k+1)$-wins $\mathsf{TrojanGame}(\Pi_2^\star, T, Q)$. It contradicts Lemma 7, which ends the proof. ∎

Conclusions

The natural approach to real-world digital security problems is to review existing advanced, well-known, or simply efficient cryptographic tools and adjust one of them. In the context of Hardware Trojan Horses, we can observe it, e.g., in paper [Wah+16] where authors use Verifiable Computation Protocol CMT [CMT12] – to protect the functionality $\mathcal{F}$, the trusted module performs the actions of the Verifier, and the untrusted module performs the actions of the Prover. The CMT protocol was chosen because it does not require the Prover to perform any cryptographic or non-local operations, unlike many other verifiable computation protocols. The authors of [Wah+16] admit that the efficiency of their solution is based on a vast technological gap between trusted and untrusted manufacturers because the Verifier needs to perform approximately 10^5-10^7 times more operations than simply evaluating the original functionality to be protected. This case is a perfect example of the main problem when using well-founded cryptographic tools as building blocks for HTH-resilience schemes – many are too heavy. Recall that the primary motivation for research on countermeasures on HTHs is the real-world economic scheme where production is more profitable when outsourced. Therefore, the overhead size or efficiency of the HTH-resilince scheme must remain small; the trusted module should be especially simple – if it performs advanced operations to protect $\mathcal{F}$, why do not simply ask it to compute $\mathcal{F}$?

However, cryptography does not fail in the presence of Hardware Trojan Horses, and we believe this thesis makes it evident. Challenged by real-life digital security problems, we did not ask what cryptography could do for us but what we could do for cryptography. Therefore, our research was focused on developing new techniques and ideas, such as wire- and gate-covering sets, information loss, and history-hardcoded circuits. We also went in-depth with understanding the damage that can be done by constrained (tampering) HTHs, and we did not hesitate to make our consideration low-level and focus on single gates and wires of the infected circuit. Last but not least, we always considered the *original* real-world motivation for our research, so we investigated lightweight solutions, keeping our results provable, not only theoretical.

References

[ADF16] Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. "Circuit Compilers with O (1/\log (n)) O (1/log (n)) Leakage Rate". In: *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*. Springer. 2016, pp. 586–615.

[Ate+18] Giuseppe Ateniese, Aggelos Kiayias, Bernardo Magri, Yiannis Tselekounis, and Daniele Venturi. "Secure Outsourcing of Cryptographic Circuits Manufacturing". In: *ProvSec*. Ed. by Joonsang Baek, Willy Susilo, and Jongkil 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.

[Bai+23a] Mirza Ahad Baig, Suvradip Chakraborty, Stefan Dziembowski, Małgorzata Gałązka, Tomasz Lizurej, and Krzysztof Pietrzak. "Efficiently Testable Circuits". In: *ITCS - Innovations in Theoretical Computer Science*. 2023.

[Bai+23b] Mirza Ahad Baig, Suvradip Chakraborty, Stefan Dziembowski, Małgorzata Gałązka, Tomasz Lizurej, and Krzysztof Pietrzak. *Efficiently Testable Circuits without Conductivity*. 2023.

[Bam+14] Tobias Bamert, Christian Decker, Roger Wattenhofer, and Samuel Welten. "Bluewallet: The secure bitcoin wallet". In: *Security and Trust Management: 10th International Workshop, STM 2014, Wroclaw, Poland, September 10-11, 2014. Proceedings 10*. Springer. 2014, pp. 65–80.

[Bań03] Mirosław Bańko. *Już koniami czy jeszcze końmi*. https://sjp.pwn.pl/poradnia/haslo/Juz-koniami-czy-jeszcze-konmi;4729.html. Access: 15.04.2023. 2003.

[Bar+01a] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. "On the (im) possibility of obfuscating programs". In: *Advances in Cryptology—CRYPTO 2001: 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19–23, 2001 Proceedings*. Springer. 2001, pp. 1–18.

[Bar+01b] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. "On the (Im)possibility of Obfuscating Programs". In: 2001, pp. 1–18. DOI: 10.1007/3-540-44647-8_1.

[BM92] Steven Michael Bellovin and Michael Merritt. “Encrypted key exchange: Password-based protocols secure against dictionary attacks”. In: (1992).

[BT18] Swarup Bhunia and M Tehranipoor. “The Hardware Trojan War”. In: *Cham„ Switzerland: Springer* (2018).

[BA04] Michael Bushnell and Vishwani Agrawal. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. Vol. 17. Springer Science & Business Media, 2004.

[Cha+21] Suvradip Chakraborty, Stefan Dziembowski, Małgorzata Gałązka, Tomasz Lizurej, Krzysztof Pietrzak, and Michelle Yeo. “Trojan-resilience without cryptography”. In: *Theory of Cryptography Conference*. Springer. 2021, pp. 397–428.

[Cha+98] JT-Y Chang, Chao-Wen Tseng, C-MJ Li, Mike Purtell, and Edward J McCluskey. “Analysis of pattern-dependent and timing-dependent failures in an experimental test chip”. In: *Proceedings International Test Conference 1998 (IEEE Cat. No. 98CH36270)*. IEEE. 1998, pp. 184–193.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. “Practical verified computation with streaming interactive proofs”. In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. 2012, pp. 90–112.

[23] *Cyber security Market*. https://www.fortunebusinessinsights.com/industry-reports/cyber-security-market-101165. Access: 04.06.2023. 2023.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. “Unifying leakage models: from probing attacks to noisy leakage.” In: *Advances in Cryptology–EUROCRYPT 2014: 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings 33*. Springer. 2014, pp. 423–440.

[DFS16] Stefan Dziembowski, Sebastian Faust, and François-Xavier Standaert. “Private Circuits III: Hardware Trojan-Resilience via Testing Amplification”. In: *ACM CCS*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM, 2016, pp. 142–153. DOI: 10.1145/2976749.2978419.

[Efr+22] Klim Efremenko, Bernhard Haeupler, Yael Tauman Kalai, Pritish Kamath, Gillat Kol, Nicolas Resch, and Raghuvansh R. Saxena. "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, pp. 582–594. ISBN: 9781450392648. DOI: 10.1145/3519935.3520007. URL: https://doi.org/10.1145/3519935.3520007.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. "Electromagnetic analysis: Concrete results". In: *Cryptographic Hardware and Embedded Systems—CHES 2001: Third International Workshop Paris, France, May 14–16, 2001 Proceedings 3*. Springer. 2001, pp. 251–261.

[Gar+16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. "Candidate indistinguishability obfuscation and functional encryption for all circuits". In: *SIAM Journal on Computing* 45.3 (2016), pp. 882–929.

[Gol+05] Oded Goldreich et al. "Foundations of cryptography–a primer". In: *Foundations and Trends® in Theoretical Computer Science* 1.1 (2005), pp. 1–116.

[HBM18] Yuanwen Huang, Swarup Bhunia, and Prabhat Mishra. "Scalable test generation for Trojan detection using side channel analysis". In: *IEEE Transactions on Information Forensics and Security* 13.11 (2018), pp. 2746–2760.

[Imp95] Russell Impagliazzo. "A personal view of average-case complexity". In: *Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference*. IEEE. 1995, pp. 134–147.

[Ish+06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David A. Wagner. "Private Circuits II: Keeping Secrets in Tamperable Circuits". In: *EUROCRYPT*. Ed. by Serge 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.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. "Private Circuits: Securing Hardware against Probing Attacks". In: *CRYPTO*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Springer, 2003, pp. 463–481. DOI: 10.1007/978-3-540-45146-4_27. URL: https://doi.org/10.1007/978-3-540-45146-4_27.

[KLR12] Yael Tauman Kalai, Allison B. Lewko, and Anup 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.

[KM58] Edward L Kaplan and Paul Meier. "Nonparametric estimation from incomplete observations". In: *Journal of the American statistical association* 53.282 (1958), pp. 457–481.

[KLM97] Dan Kleitman, Tom Leighton, and Yuan Ma. "On the design of reliable Boolean circuits that contain partially unreliable gates". In: *Journal of Computer and System Sciences* 55.3 (1997), pp. 385–401.

[Koc96] Paul C Kocher. "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems". In: *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*. Springer. 1996, pp. 104–113.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. "Differential Power Analysis". In: *CRYPTO*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 388–397. DOI: 10.1007/3-540-48405-1_25. URL: https://doi.org/10.1007/3-540-48405-1_25.

[Luc22] Nan Tian Lucie Béraud-Sudreau Ana Assis. "Trends in World Military Expenditure". In: (2022).

[Mad23] Chisom Maduonuorah. "Crypto Bankruptcies". In: *Milkroad.com* (2023).

[Mas16] B. Mastroianni. *These were the 25 worst passwords of 2015*. https://www.cbsnews.com/news/these-were-the-25-worst-passwords-of-2015/. Access: 16.12.2023. 2016.

[MT00] Edward J McCluskey and Chao-Wen Tseng. "Stuck-fault tests vs. actual defects". In: *Proceedings International Test Conference 2000 (IEEE Cat. No. 00CH37159)*. IEEE. 2000, pp. 336–342.

[Mie+13] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. "Zerocoin: Anonymous distributed e-cash from bitcoin". In: *2013 IEEE Symposium on Security and Privacy*. IEEE. 2013, pp. 397–411.

[Nak+08] Satoshi Nakamoto et al. “Bitcoin”. In: *A peer-to-peer electronic cash system* 21260 (2008).

[Pag02] Dan Page. “Theoretical use of cache memory as a cryptanalytic side-channel”. In: *Cryptology ePrint Archive* (2002).

[Pus+22] Endres Puschner, Thorben Moos, Steffen Becker, Christian Kison, Amir Moradi, and Christof Paar. “Red Team vs. Blue Team: A Real-World Hardware Trojan Detection Case Study Across Four Modern CMOS Technology Generations”. In: *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society. 2022, pp. 763–781.

[QS01] Jean-Jacques Quisquater and David Samyde. “Electromagnetic analysis (ema): Measures and counter-measures for smart cards”. In: *Smart Card Programming and Security: International Conference on Research in Smart Cards, E-smart 2001 Cannes, France, September 19–21, 2001 Proceedings*. Springer. 2001, pp. 200–210.

[RR18] Jordan Robertson and Michael Riley. “The Big Hack: How China Used a Tiny Chip to Infiltrate U.S. Companies”. In: *Bloomberg* (2018).

[Sch15] Bruce Schneier. “What’s Next in Government Surveillance”. In: *The Atlantic* (2015).

[Sha48] Claude Elwood Shannon. “A mathematical theory of communication”. In: *The Bell system technical journal* 27.3 (1948), pp. 379–423.

[Sha23] Anurakti Sharma. “Spy, trojan horse, or lift: Why US is in panic mode over giant Chinese cargo cranes”. In: *Times Now* (2023).

[TSZ13] Mohammad Tehranipoor, Hassan Salmani, and Xuehui Zhang. *Integrated Circuit Authentication: Hardware Trojans and Counterfeit Detection*. Springer Publishing Company, Incorporated, 2013. ISBN: 3319008153, 9783319008158.

[Wah+16] Riad S. Wahby, Max Howald, Siddharth Garg, Abhi Shelat, and Michael Walfish. “Verifiable ASICs”. In: *IEEE SP*. IEEE Computer Society, 2016, pp. 759–778. DOI: 10.1109/SP.2016.51.

[Woo+14] Gavin Wood et al. “Ethereum: A secure decentralised generalised transaction ledger”. In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32.