Secure Outsourcing of Circuit Manufacturing

Giuseppe Ateniese[1], Aggelos Kiayias[2], Bernardo Magri[3], Yiannis Tselekounis[4], and Daniele Venturi[5]

[1] *Stevens Institute of Technology*
[2] *University of Edinburgh*
[3] *Sapienza University of Rome*
[4] *National and Kapodistrian University of Athens*
[5] *University of Trento*

May 28, 2016

Abstract

The fabrication process of integrated circuits (ICs) is complex and requires the use of off-shore foundries to lower the costs and to have access to leading-edge manufacturing facilities. Such an outsourcing trend leaves the possibility of inserting malicious circuitry (a.k.a. hardware Trojans) during the fabrication process, causing serious security issues. Hardware Trojans are very hard and expensive to detect and can disrupt the entire circuit or covertly leak sensitive information.

In this paper, we propose a formal model for assessing the security of ICs whose fabrication has been outsourced to an untrusted off-shore manufacturer. We assume that the IC specification and design are trusted but the fabrication facility(ies) may be untrusted. Our objective is to stop Trojans from releasing sensitive information to the outside while still using its circuitry for day-to-day operations.

We also provide two different methodologies for constructing compilers relying on verifiable computation (VC) schemes and secure multiparty computation (MPC) protocols with certain properties. Suitable VC schemes, with the properties we require, were recently constructed, e.g., by Parno *et al.* (Oakland '13), and by Fiore, Gennaro, and Pastro (CCS '14). Similarly, many MPC protocols readily comply (or can be easily adapted to comply) with our requirements.

By allowing manufacturers to use off-shore fabrication facilities, we ensure a high degree of competition among suppliers, thus providing lower cost without hindering innovation or access to leading-edge microelectronics.

Keywords: hardware Trojans, circuit compilers, fabless manufacturing, verifiable computation.

Contents

1 Introduction

1.1 Background

The fabrication process adopted by the semiconductor industry is fundamentally global, involving several parties that may not be trusted. As a result, integrated circuits (ICs) are vulnerable to so-called hardware Trojans that can compromise or disable critical systems, or covertly leak sensitive information [44, 16, 8]. Analogously to a software Trojan, a hardware Trojan is a *back-door* deliberately added to the circuit to disrupt its operation or disable it when certain events occur. A Trojan can be added to the circuit during the design phase, by some malicious designer, or more often during the manufacturing phase, by some malicious off-shore fabrication facility. A hardware Trojan's objectives may be to modify the functionality of the circuit (e.g., in order to compromise or disable critical systems), modify its specification (e.g., by changing its energy consumption), covertly leak sensitive information (e.g., from a secret memory), or simply disable the entire circuit when instructed to do so [6]. Once the Trojan is inserted into the circuit it can stay activated the entire time, or it can be "triggered" by some event such as a special input to the circuit.

Reliably detecting compromised circuit components through testing and reverse engineering appears to be an impossible task given our current technology [13]. Indeed, all non-destructive testing techniques can easily be circumvented by properly obfuscating embedded Trojans. The U.S. military recognized this threat and started two programs, Trust and IRIS, with the intent of developing techniques and metrics to certify ICs going into weapon systems. The main concern is that advanced weapons may appear to work properly but then switch off in combat or when triggered by some special events. Another stated concern is information leakage, where a malicious component is programmed to leak sensitive information [57].

The U.S. military however currently obtains trusted chips through the DOD Trusted Foundry program which is currently managed by the NSA's Trusted Access Program Office (TAPO). Within this program, a trusted design center and foundry are established through an exclusive partnership with IBM for secure semiconductor fabrication and ASIC services, along with the involvement of several Trusted Suppliers which are accredited by an accreditation authority (DMEA). The intent of the Trusted Foundry program is to provide national security and defense programs with access to ICs from trusted sources. However, a recent report by the U.S. Government Accountability Office (GAO) [46], released in April 2015, found that even though the Trusted Foundry program started in 2004, IBM remained the sole-source supplier for leading-edge technologies meeting the criteria put forth by DOD. GAO's report highlights two main issues: First, it notices that IBM sold its microelectronics fabrication business to a foreign-owned entity (GlobalFoundries). Second, relying on a single source supplier for defense microelectronics hinders competition and thus innovation in this critical area.

In this paper, we propose a new approach to the untrusted fabrication problem. We assume that the IC specification and design are trusted but the fabrication facility is not. Rather than testing or reverse engineering the IC hardware received, we employ it in a controlled environment and continuously verify its operations. Our approach makes sense as long as the controlled environment can be: (i) made cheaply, and (ii) run efficiently. We show how to reach these two goals whenever the main objective is to prevent hardware Trojans from releasing sensitive information.

By allowing manufacturers to use off-shore fabrication facilities, we ensure a high degree of competition among suppliers, thus providing lower cost, improved innovation, and access to leading-edge microelectronics.

1.2 Our Contributions

We put forward a formal framework for assessing security of a circuit whose production has been, in part, outsourced to a set of manufacturers that are not trusted. With such a framework in hand, we give two design methodologies. Our first methodology borrows ideas from the setting of verifiable computing (see, e.g., [30]), while the second one relies on secure multiparty computation (see, e.g., [33]) in the client-server model [7]. A more detailed explanation of our main contributions follows below.

Secure circuit fabrication. Let Γ be the original circuit to be produced. Instead of producing Γ directly, we first "compile" it into a different circuit $\widehat{\Gamma}$ using an efficient, possibly randomized, procedure Φ that we call an *outsourcing compiler*. The compiler Φ takes as input a description of Γ and returns a description of $\widehat{\Gamma}$, together with some auxiliary information specifying how $\widehat{\Gamma}$ can be divided into sub-components, and which of these components can be produced off-shore; the remaining components will be instead built in-house. After all components have been produced, the circuit designer re-assembles the circuit $\widehat{\Gamma}$ (by combining the outsourced components and the components built in-house), which is then initialized with some initial secret memory M_1, and used in the wild.

In order to make sense, the above approach needs to satisfy a few important requirements. The first requirement is that Φ needs to be functionality preserving, meaning that the compiled circuit $\widehat{\Gamma}$ should compute the same functionality as the original circuit Γ (for all possible initial memories M_1, and for all possible inputs). The second requirement is that the effort needed to fabricate the trusted sub-components should be (much) less compared to the effort required to build the original circuit Γ. The third requirement is that Φ should be secure, meaning that, under an acceptable assumption about the manufacturers who construct the outsourced components, the produced circuit $\widehat{\Gamma}$ can be safely used in real-life applications.

Our security definition follows the simulation paradigm, and is inspired by similar definitions in the setting of tamper-proof circuit compilers [39, 28]. We refer the reader to Section 1.3 for a more detailed comparison between the two approaches. In a nutshell, security of Φ is defined by requiring that whatever an adversary can learn by interacting with the fabricated circuit $\widehat{\Gamma}$ (produced following the steps outlined above), can be simulated given only black-box access to the original circuit Γ. This essentially means that, no matter how the outsourced components are maliciously modified (e.g., by inserting a hardware Trojan), using circuit $\widehat{\Gamma}$ is as secure as using the original circuit Γ, and thus, in particular, does not leak sensitive information on the secret memory. See Section 3 for a precise definition.

We also consider a weakening of the above definition, in which the simulator is allowed to receive a short advice (or leakage) on the secret memory M_1. This models a setting where the adversary might be able to learn a short amount of information on the secret memory, but still yields a meaningful security guarantee provided that the original circuit is resilient to such a short leakage. An appealing advantage of this weaker definition is that it might allow for significantly more efficient circuit compilers.

A solution using VC. Next, in Section 4, we show how to construct secure outsourcing compilers that work for *arbitrary* circuits Γ in the setting where *all* outsourcing manufacturers are corrupted. Our compilers generically leverage a verifiable computation (VC) scheme for the function $\mathcal{F}$ implemented by Γ. Verifiable computing (see, e.g., [30]) is a recent paradigm by which resource-constrained clients can delegate the computation of some function $\mathcal{F}$, on (possibly private) input X, to an untrusted (and computationally powerful) server, without the server being able to cheat about the outcome of the computation, and with the property that verifying the server's answer is much more efficient than computing the function from scratch.

Recent breakthrough research on verifiable computing led to nearly practical schemes that work for any function [52, 18]; some schemes additionally preserve the privacy of the inputs on which the function is being computed on [29]. VC schemes satisfying the latter property are called *input-private*.

The main idea of how to use verifiable computing in order to build secure outsourcing compilers is simple enough to describe it here. The entire bulk of the computation will be outsourced to the untrusted fabrication facility, whereas the only circuit components that need to be built in-house are: (i) the component corresponding to the algorithm for encoding the inputs (in case of input-private VC), (ii) the component corresponding to the algorithm run by the client in order to verify correctness of the server's computation, and (iii) the component used to generate fresh random coins as needed for computing the function (in case of randomized functions). Thanks to the nature of VC, the size of the components in (i) and (ii) is independent of the size of the original circuit computing the function. As for the component in (iii), we can use any existing (and trusted) circuitry for generating true random numbers (RNG). A good example is the Intel on-chip hardware random number generator which can be accessed through the RDRAND instruction available on all modern processors [41, 37].[1]

Hence, the effort needed to fabricate the components built in-house is much less (and, in fact, independent) of the effort needed to fabricate the original circuit.

We implement the above idea in two ways, depending on the properties satisfied by the underlying VC scheme, as explained below.

- Our first compiler relies on VC schemes with input-privacy, and achieves our strongest security notion (i.e., no leakage required for the simulation).
- Our second compiler relies on VC schemes without input-privacy, and achieves security provided the original primitive (implemented by the circuit Γ) is resilient against a logarithmic amount of leakage on the private memory.

  Remarkably, any public-key encryption or signature scheme is resilient to such an amount of leakage at the price of a polynomial loss in the concrete security, and recently many leakage-resilient schemes [26, 54, 42, 25, 49, 15, 50, 27] have been constructed, where the concrete security does not degrade with the total amount of tolerated leakage.

[1] Intel's generator relies on unpredictable thermal noise to generates bits that are fed to a cryptographic "conditioner" (AES in CBC-MAC mode) which produces a 256-bit seed that is then passed through a NIST SP800-90A-based pseudorandom generator.

The second compiler additionally relies on a special “self-destruct” feature (which is implemented in one of the components built in-house), meaning that after the first invalid output is ever processed, the entire memory is overwritten. As we show, for this compiler, this is an inherent requirement, in that, without such a property, there exist generic attacks that allow to recover the entire private memory. Moreover, such attacks are undetectable by all polynomial-time (black-box) tests. Our definition of undetectability (see Section 3.2) is similar in spirit to analogous definitions in the context of subversion-resilient cryptography [11, 10].

A solution using MPC. In Section 5, we show how to construct secure outsourcing compilers for arbitrary circuits Γ in the setting where $m \geq 2$ outsourcing manufacturers are available, and a certain unknown subset of them is untrustworthy. This is a strictly stronger assumption compared to the VC setting, nevertheless, opens the possibility for more efficient constructions and stronger availability guarantees. Our compiler utilizes a general client-server secure multi-party computation (MPC) protocol, i.e., a protocol that for any function enables a set of clients to privately communicate their inputs to a set of servers that will perform a computation and return the output to a single designated recipient. We stress that many MPC protocols follow this paradigm (e.g., [21]), while others, as we comment later, can be easily adapted to it.

Given such a protocol, the compiler operates as follows. For a given circuit Γ it produces the MPC protocol implementing it, isolates the client and recipient computation for manufacturing in-house, and outsources each of the other components (representing a server in the MPC protocol) to the untrusted manufacturers. The key points of this compiler construction are as follows: (i) The client and recipient computation are typically quite lightweight; the client, in many protocols, simply performs an encryption or a secret-sharing operation, and the recipient a secret-reconstruction protocol; in either case, the computation is independent of the circuit that is outsourced. (ii) There are MPC protocols that can tolerate up to $m-1$ malicious servers, something we can leverage to argue that if at least one of the outsourcing manufacturer is honest the compiled circuit would be safe for use. Additional properties of the underlying MPC protocol can also be very valuable by our compiler: for instance, if the underlying MPC protocol supports guaranteed output delivery, we can use this guarantee to argue that the final circuit will be resilient to a certain faulty outsourced sub-component. Moreover, if the underlying protocol satisfies the identifiable abort property, cf. [38], we can enable our compiled circuit to partially shutdown an outsourced sub-component that is discovered to be faulty, thus reducing energy consumption. By adapting an efficient protocol, e.g, [22], in the client-server model, we can provide a very efficient compiled circuit (with an overhead linear in the size of the original circuit $|\Gamma|$).

We refer the reader to Section 5 for a more detailed discussion how one can instantiate our compilers using MPC protocols.

A remark. Both our approaches require a partitioning and assembly procedure which must be performed in-house. To lower the cost, trusted and untrusted sub-circuits will likely be diced in their own wafers and packaged independently. Fortunately, recent advanced designs and emerging technologies in the electronics assembly and packaging industry are making the entire process of interconnecting different components more affordable, reliable, and automatic.

1.3 Related Work

Hardware Trojans. *Prevention* of hardware Trojans in ICs is a common practice, that might take place during the design, manufacturing, and post-manufacturing stage [55, 45].

However, since it is not always possible to efficiently prevent Trojans insertion, Trojans *detection* has also been vastly explored [13]; once a Trojan is detected, the circuit can be disposed and not used. Common methodologies used to perform Trojans detection vary from invasive ones (that destroy the IC to examine it inside), to non-invasive ones (where the circuit is executed and compared against a trusted copy of the circuit or against some expected output values). Trojan detection is typically a very expensive and unreliable process, therefore the best practice is usually not to rely on any kind of testing to protect against Trojans.

Explicit *countermeasures* against Trojans also exist, where the objective is to guarantee the functionality or security of the circuit even in the presence of some unknown Trojan. For instance, the so-called "data guards" are designed to prevent a Trojan from being activated and/or to access sensitive data [59]. Another approach is the duplication of logic elements and the division of the sensitive data to independent parts of the circuit [47, 59].

To the best of our knowledge, our work is the first introducing a formal model for assessing security of ICs in the presence of arbitrary hardware Trojans. The only exception is [56] that considered an even stronger definition than ours (where the output of the produced circuit must always be the same as that of the original circuit), but could only achieve security for very limited classes of Trojans (i.e., the adversary is allowed to "corrupt" only a small fraction of the gates in each layer of the IC, and a small fraction of the wires connecting different layers).

Concurrent work. Very recently, in work concurrent and independent from ours,Wahby *et al.* [58] also explored the idea of using VC to address the issue of hardware Trojans in fabless circuit manufacturing. The two papers, however, greatly differ in scope and techniques, and, in fact, can be regarded as incomparable as explained below.

First, our goal is to make sure the secret memory of the rebuilt circuit cannot be leaked to an outsider (even in the presence of arbitrary hardware Trojans), whereas [58] aims at the incomparable goal of ensuring *correctness* of the computation (which makes sense also for non-cryptographic functionalities); note that, as we prove, correctness implies security in our sense using self-destruct (up to logarithmic leakage on the memory). Second, our main focus is on precise definitions and proofs in the style of provable security (characterizing which properties are needed for the VC scheme in order for our approach to go through), while [58] addresses the orthogonal question of how to implement the idea of using VC for the problem at hand in practice (providing a concrete ASIC implementation based on an optimized existing VC scheme, and measuring its performances in terms of energy consumption, circuit area, and throughput). Lastly, we also provide a solution to the problem based on MPC.

Tamper-proof circuits. Our main security definition shares similarities with analogous definitions in the context of protecting circuit implementations against tampering attacks. This line of research received considerable attention in the past few years [39, 28, 19, 43, 20].

The main difference between this setting and the one considered in this paper is that tamper-proof circuit compilers are typically used to protect against fault injection [51] and tampering attacks at run-time; such attacks are usually carried out in an adaptive manner, depending on the outcome of previous attempts. Outsourcing compilers, instead, only protect against (non-adaptive) tampering taking place during the circuit fabrication process. Importantly, the latter restriction allows to obtain security against arbitrary modifications, whereas in circuit tampering one has to consider very restricted attacks (e.g., wire tampering [39] or gate tampering [43]).

Subversion. The above type of non-adaptive tampering is, in fact, reminiscent of the setting of subversion attacks against cryptographic primitives and algorithms. Inspired by the recent revelations of Edward Snowden [53, 4, 36], this line of research recently led to constructing several concrete primitives resisting large classes of subversion attacks [11, 23, 24, 3].

In this light, our work could be interpreted as formalizing the security of circuits that might have been subject to subversion during fabrication.

2 Preliminaries

2.1 Notation

For a string x, we denote its length by $|x|$; if S is a set, $|S|$ represents the number of elements in S; for a natural number n, $[n]$ denotes the set $\{1, \ldots, n\}$. When x is chosen randomly in S, we write $x \leftarrow\$ S$. When $\mathcal{A}$ is an algorithm, we write $y \leftarrow \mathcal{A}(x)$ to denote a run of $\mathcal{A}$ on input x and output y; if $\mathcal{A}$ is randomized, then y is a random variable and $\mathcal{A}(x; r)$ denotes a run of $\mathcal{A}$ on input x and randomness r. An algorithm $\mathcal{A}$ is *probabilistic polynomial-time* (PPT) if $\mathcal{A}$ is randomized and for any input $x, r \in \{0,1\}^*$ the computation of $\mathcal{A}(x; r)$ terminates in at most $\text{poly}(|x|)$ steps.

We denote with $\lambda \in \mathbb{N}$ the security parameter. A function $\nu : \mathbb{N} \to [0, 1]$ is negligible in the security parameter (or simply negligible) if it vanishes faster than the inverse of any polynomial in λ, i.e. $\nu(\lambda) = \lambda^{-\omega(1)}$.

The statistical distance between two random variables $\mathbf{Z}$ and $\mathbf{Z}'$ defined over some common set Z is defined as $\Delta(\mathbf{Z}; \mathbf{Z}') = \frac{1}{2} \sum_{z \in Z} |\mathbb{P}[\mathbf{Z} = z] - \mathbb{P}[\mathbf{Z}' = z]|$. We rely on the following lemma (which follows directly from the definition of statistical distance):

Lemma 1. *Let $\mathbf{Z}$ and $\mathbf{Z}'$ be a pair of random variables, and W be an event defined over the probability space of $\mathbf{Z}$ and $\mathbf{Z}'$. Then,*

$$\Delta(\mathbf{Z}; \mathbf{Z}') \leq \Delta(\mathbf{Z}; \mathbf{Z}' | \neg W) + \mathbb{P}[W].$$

For two ensembles $\mathbf{Z} := \{Z_\lambda\}_{\lambda \in \mathbb{N}}$ and $\mathbf{Z}' := \{Z'_\lambda\}_{\lambda \in \mathbb{N}}$, we write $\mathbf{Z} \equiv \mathbf{Z}'$ to denote that the two ensembles are identically distributed. We also write $\mathbf{Z} \approx_c \mathbf{Z}'$ to denote that the ensembles are computationally indistinguishable, i.e. for all PPT distinguishers $\mathcal{D}$ there exists a negligible function $\nu : \mathbb{N} \to [0, 1]$ such that

$$\Delta^{\mathcal{D}}(\mathbf{Z}; \mathbf{Z}') := \left| \mathbb{P}[\mathcal{D}(z) = 1 : \; z \leftarrow\$ \mathbf{Z}] - \mathbb{P}[\mathcal{D}(z) = 1] : \; z \leftarrow\$ \mathbf{Z}' \right| \leq \nu(\lambda).$$

2.2 Circuits

A (Boolean) circuit $\Gamma = (V, E)$ is a directed acyclic graph. The vertices V are logical gates, and the edges E are wires connecting the gates. For the case of *deterministic* circuits, the gates can be of type `AND`, `XOR` and `copy`, where `AND` (resp. `XOR`) have fan-in two and fan-out one, and output the AND (resp. XOR) operation on the input bits; a copy gate, denoted `copy`, simply forwards the input bit into two output wires.

The depth of a circuit is defined as the longest path from an input to an output; the size of a circuit is defined as its total number of gates. Sometimes we explicitly write $\langle \Gamma \rangle$ for the description of the circuit Γ.

A circuit is clocked if it evolves in clock cycles (or rounds). The input and output values of the circuit Γ in clock cycle i are denoted by X_i and Y_i, respectively. A circuit is *probabilistic* if it uses internal randomness as part of its logic. We call such probabilistic logic *randomness gates* and denote them with \$. In each clock cycle \$ outputs a fresh random bit. Additionally, a circuit may contain memory gates. Memory gates, which have a single incoming edge and any number of outgoing edges, maintain state: at any clock cycle, a memory gate sends its current

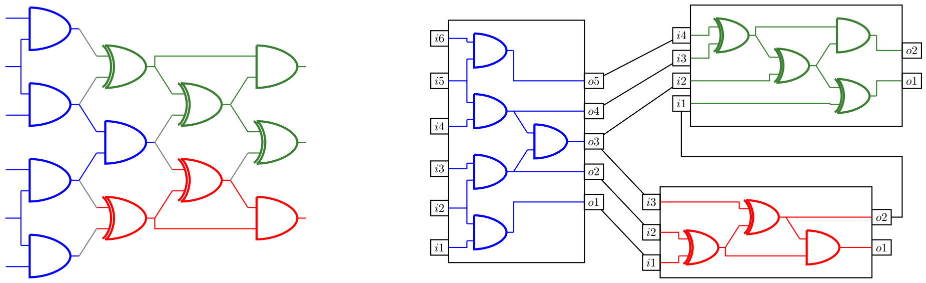

Figure 1: On the left side we present the description of a (compiled) circuit. On the right side the same circuit is represented as three different components. The mapping function $\mathcal{M}$ establishes the connections between the blue component and the green and red components.

state down its outgoing edges and updates it according to the value of its incoming edge. Any cycle in the circuit graph must contain at least one memory gate. The state of all memory gates at clock cycle i is denoted by M_i, with M_1 denoting the initial state. When a circuit is run in state M_i on input X_i, the circuit will output Y_i and the memory gates will be in a new state M_{i+1}. We will denote this by $(Y_i, M_{i+1}) \leftarrow \Gamma[M_i](X_i)$.

3 Secure Circuit Fabrication

In this section we put forward a formal model for assessing security of a (cryptographic) circuit whose production is outsourced to one or more untrusted facilities. We start by recalling the standard notion of connected component of a circuit or graph.

Definition 1 (Component)**.** *A circuit $\Gamma' = (V', E')$ is a (connected) component of circuit $\Gamma = (V, E)$ if $V' \subseteq V$, $E' \subseteq E$ and for all $g_1, g_2 \in V'$ we have that $(g_1, g_2) \in E'$ iff $(g_1, g_2) \in E$.*

Next, we introduce the notion of an outsourcing circuit compiler (or simply compiler). In a nutshell a circuit compiler is an efficient algorithm Φ that takes as input (the description of) a circuit Γ, and outputs (the description of) a compiled circuit $\widehat{\Gamma}$. Additionally, Φ returns a list of sub-components $\widehat{\Gamma}_i$ of $\widehat{\Gamma}$ whose production can be outsourced to one or more external manufacturers, together with the relevant information how to connect those sub-components with the remaining ones (that need to be built in-house) in order to re-assemble the compiled circuit $\widehat{\Gamma}$.

Definition 2 (Outsourcing circuit compiler)**.** *Let Γ be an arbitrary circuit. A (ρ, m)-outsourcing compiler Φ is a* PPT *algorithm $(\widehat{\Gamma}, \mathsf{aux}) \leftarrow \Phi(\Gamma)$, such that the following holds:*

- $\mathsf{aux} := ((\widehat{\Gamma}_1, \ldots, \widehat{\Gamma}_n), \mathcal{M}, (I_1, \ldots, I_m))$, *with $n \in \mathbb{N}$ and $I_j \subseteq [n]$, for $j \in [m]$, mutually disjoint subsets.*
- *$(\widehat{\Gamma}_1, \ldots, \widehat{\Gamma}_n)$ are disjoint (connected) components of $\widehat{\Gamma}$ such that $V = \bigcup_{i \in [n]} V_i$, where $\Gamma_i = (V_i, E_i)$.*
- *$\mathcal{M} : V \times V \rightarrow \{0, 1\}$ is a function such that $\mathcal{M}(v, v') = 1$ iff $v, v' \in V_i, V_j$ for some $i \neq j$ and $(v, v') \in E$.*

We call $\rho := \frac{\sum_{i \in [n] \setminus I_1 \cup \ldots \cup I_m} |\widehat{\Gamma}_i|}{|\Gamma|}$ the outsourcing ratio of the compiler.

Intuitively, in the above definition, the outsourcing ratio ρ represents the fraction of the compiled circuit (w.r.t. the original circuit) that should be built in-house. Note that the sub-components $(\widehat{\Gamma}_i)_{i \in [n]}$ "cover" the entire compiled circuit $\widehat{\Gamma}$ (without overlap), and the mapping function $\mathcal{M}$ specifies how to connect the different components in order to reconstruct $\widehat{\Gamma}$. The sets of indexes $I_j \subseteq [n]$ represents the sub-components whose production will be outsourced to manufacturer $j \in [m]$. See Fig. 1 for a pictorial representation in a simple toy example.

Correctness of an outsourcing compiler demands that the compiled circuit maintains the same functionality of the original circuit.

Definition 3 (Correctness)**.** *We say that an outsourcing compiler Φ is functionality preserving if for all circuits Γ, for all values of the initial memory M_1, and for any set of public inputs $X_1, \ldots, X_q$, the sequence of outputs $Y_1, \ldots, Y_q$ produced by running the original circuit Γ starting with state M_1 is identical to the sequence of outputs produced by running the transformed circuit $\widehat{\Gamma}$ starting with state M_1 (with all but negligible probability over the randomness of the compiler and the randomness of the original and compiled circuit).*

3.1 Security

We define security using the simulation paradigm. Our approach is similar in spirit to previous work on tamper-resilient circuit compilers (see, e.g., [39, 28]). In a nutshell, security is defined by comparing two experiments. In the first experiment, also called the real experiment, the circuit designer compiles the circuit and outsources the production of some of the components in the compiled circuit to a set of m untrusted manufacturer. A subset of size t of the manufacturers are malicious, and controlled by a monolithic adversary $\mathcal{A}$; of course the circuit designer does not know which manufacturers are malicious and which ones are honest. During production, $\mathcal{A}$ is allowed to completely change the outsourced circuit components under its control, whether by adding, removing or changing gates and/or wires. Later, the designer assembles the circuit by re-combining all the components (the outsourced ones and the ones built in-house). Finally $\mathcal{A}$ can access the assembled circuit in a *black-box* way, that is, it can observe inputs/outputs produced by running the assembled circuit (with some initial memory M_1).

In the second experiment, also called the ideal experiment, a simulator is given black-box access to the original circuit (initialized with initial memory M_1). The goal of the simulator is to produce an output distribution which is indistinguishable from the one in the real experiment. In its most general form, our definition allows the simulator to obtain a short leakage on the initial memory. This captures the feature that the adversary in the real experiment could learn at most a short amount of information on the private memory.

Real experiment. The distribution $\mathbf{Real}_{\mathcal{A},\Phi,\mathcal{C},\Gamma,M_1}(\lambda)$ is parameterized by the adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, the set of corrupt manufacturers $\mathcal{C}$, the compiler Φ, and the original circuit Γ with initial memory M_1.

1. $(\widehat{\Gamma}, \mathsf{aux}) \leftarrow \Phi(\Gamma)$: In the first step, the description of the original circuit Γ is given as input to the compiler Φ; the compiler outputs the description of the compiled circuit $\widehat{\Gamma}$ plus the auxiliary information $\mathsf{aux} := ((\widehat{\Gamma}_1, \ldots, \widehat{\Gamma}_n), \mathcal{M}, (I_1, \ldots, I_m))$ which is used to specify how the compiled circuit is split into sub-components, how the different sub-components are connected (via the mapping function $\mathcal{M}$), and the subset of sub-components whose production is outsourced to each manufacturer (in the index sets I_j, for $j \in [m]$).
2. $(\{\widehat{\Gamma}'_i\}_{i \in I}, \tau) \leftarrow \mathcal{A}_0(1^\lambda, \{\langle \widehat{\Gamma}_i \rangle\}_{i \in I}, \langle \Gamma \rangle, \langle \widehat{\Gamma} \rangle)$: The adversary is given as input the description of the components from the index set $I = \cup_{j \in \mathcal{C}} I_j$, the description of the original circuit Γ, the description of the compiled circuit $\widehat{\Gamma}$, and returns the modified components along with some value τ that may contain some auxiliary state information.

3. $\widehat{\Gamma}' := (\widehat{V}', \widehat{E}')$: The compiled circuit $\widehat{\Gamma}'$ is rebuilt by replacing the components $(\widehat{\Gamma}_i)_{i \in I}$ with the modified components $(\widehat{\Gamma}'_i)_{i \in I}$, and by connecting the different components as specified by the mapping function $\mathcal{M}$.
4. $\mathcal{A}_1^{\widehat{\Gamma}'[M_1](\cdot)}(1^\lambda, \tau)$: Adversary $\mathcal{A}_1$, with auxiliary information τ, is given oracle access to the rebuilt circuit $\widehat{\Gamma}'$ with compiled private memory M_1.

Simulation. The distribution $\mathbf{Ideal}_{\mathcal{S},\mathcal{A},\Phi,\mathcal{C},\Gamma,M_1,\ell}(\lambda)$ is parametrized by the simulator $\mathcal{S}$, the adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, the compiler Φ, the set of corrupt manufacturers $\mathcal{C}$, the original circuit Γ with initial memory M_1, and some value $\ell \in \mathbb{N}$.

1. $f \leftarrow \mathcal{S}(1^\lambda, \langle\Gamma\rangle, \Phi, \mathcal{A}, \mathcal{C}, \ell)$: Given as input a description of the original circuit, of the compiler and of the adversary, the subset of corrupt manufacturers, and the parameter $\ell \in \mathbb{N}$, the simulator specifies an arbitrary polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^\ell$.
2. $\mathcal{S}^{\mathcal{A},\Gamma[M_1](\cdot)}(1^\lambda, L)$: The simulator takes as input leakage $L = f(M_1)$, and is given oracle access to adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ and to the original circuit Γ with private memory M_1. We remark that the simulator is restricted to be fully black-box. In particular, $\mathcal{S}$ only accesses the modified sub-components returned by $\mathcal{A}_0$ in a black-box way (i.e., without knowing their description).

Definition 4 (Security). *We say that a (ρ, m)-outsourcing circuit compiler Φ is (ℓ, t)-secure if the following conditions are met.*

(i) Non-triviality: $\rho < 1$*, for sufficiently large values of* $\lambda \in \mathbb{N}$.

(ii) Simulatability: *For all* $\mathcal{C} \subseteq [m]$ *of size at most* t *and for all PPT adversaries* $\mathcal{A}$*, for all circuits* Γ*, and for all initial values of the memory* $M_1 \in \{0,1\}^*$*, there exists a simulator* $\mathcal{S}$ *with running time* $\mathrm{poly}(|\mathcal{A}|, |\Gamma[M_1]|)$ *such that*

$$\{\mathbf{Real}_{\mathcal{A},\Phi,\mathcal{C},\Gamma,M_1}(\lambda)\}_{\lambda \in \mathbb{N}} \approx_c \{\mathbf{Ideal}_{\mathcal{S},\mathcal{A},\Phi,\mathcal{C},\Gamma,M_1,\ell}(\lambda)\}_{\lambda \in \mathbb{N}}.$$

We observe that the above definition is only interesting for small values of ℓ (as, e.g., it becomes trivial in case $\ell = |M_1|$). Also notice that the non-triviality condition demands that the ratio between the size of the sub-components of the compiled circuit built in-house, and the size of the original circuit, should be less than one. This is necessary, as otherwise a manufacturer could simply produce the entire circuit by itself, without the help of any off-shore facility. Clearly, the smaller ρ the better, as this means that a large fraction of the original circuit production can be outsourced.

3.2 Undetectability

We formally define what it means for an adversarial strategy to be undetectable by all black-box polynomial-time tests. Informally, this means that it is hard to distinguish the output of the original circuit from the output of the compiled circuit (after the outsourced sub-components have been maliciously modified). Importantly, the latter has to hold even if the testing strategy knows the initial content of the private memory and the description of all sub-components in the compiled circuit.

For simplicity, we give the definition in the case of a single manufacturer (i.e., $m = 1$); a generalization to the case $m \geq 2$ is immediate. The formalization borrows ideas from similar definitions in the setting of subversion and algorithm-substitution attacks [11, 10].

Definition 5 (Undetectability)**.** *Let Φ be an outsourcing circuit compiler, and Γ be a circuit. We say that an adversary $\mathcal{A}$ is undetectable for Γ w.r.t. Φ if for all* PPT *algorithms* Test *there exists a negligible function $\nu : \mathbb{N} \to [0, 1]$ such that, for all initial values of the memory M_1, we have that $\mathbb{P}[\mathsf{Test}\ wins] \leq 1/2 + \nu(\lambda)$ in the following game:*

1. *The challenger picks $b \leftarrow\!\!\$\ \{0, 1\}$, runs $(\widehat{\Gamma}, (\widehat{\Gamma}_1, \ldots, \widehat{\Gamma}_n), \mathcal{M}, I) \leftarrow \Phi(\Gamma)$, and returns $(M_1, \langle\Gamma\rangle, \langle\widehat{\Gamma}\rangle, (\langle\widehat{\Gamma}_1\rangle, \ldots, \langle\widehat{\Gamma}_n\rangle), \mathcal{M}, I)$ to* Test*.*
2. *Let $\widehat{\Gamma}'$ be the circuit implicitly defined by the sub-components $\{\widehat{\Gamma}_i\}_{i\in[n]\setminus I} \cup \{\widehat{\Gamma}'_i\}_{i\in I}$ together with the mapping function $\mathcal{M}$, where $\{\widehat{\Gamma}'_i\}_{i\in I} \leftarrow \mathcal{A}(1^\lambda, \{\langle\widehat{\Gamma}_i\rangle\}_{i\in I}, \langle\Gamma\rangle, \langle\widehat{\Gamma}\rangle)$.*
3. *Algorithm* Test*, can ask polynomially many queries of the type X_i. Upon input such query, the answer from the challenger depends on the value of the bit b:*
   - *In case $b = 0$, the output is Y_i where $(Y_i, M_{i+1}) \leftarrow\!\!\$\ \Gamma[M_i](X_i)$.*
   - *In case $b = 1$, the output is Y_i where $(Y_i, M_{i+1}) \leftarrow\!\!\$\ \widehat{\Gamma}'[M_i](X_i)$.*
4. *Algorithm* Test *outputs a bit b', and wins iff $b' = b$.*

4 Compilers based on VC

In this section we build secure outsourcing compilers that work for any circuit, in the presence of a *single* malicious manufacturer. The compilers are based on any verifiable computation (VC) scheme (satisfying certain properties) for the function computed by the underlying circuit.

We start by recalling the basic definitions for VC schemes in Section 4.1. In Section 4.2 we describe our first compiler, which requires a VC scheme satisfying input privacy. In Section 4.3 we describe our second compiler, which can be instantiated with non-input-private VC schemes; our second compiler requires that once the first invalid output is produced, the compiled circuit overwrites its entire memory with the all-zero string and "self-destructs." As we show, this restriction is necessary. Finally, in Section 4.4 we discuss concrete instantiations of our compilers, based on state-of-the-art research on verifiable computing.

4.1 Prelude: Verifiable Computation

A verifiable computation scheme allows for a client to outsource the computation of a function to a (untrusted) server; the server produces a proof of correctness along with the output of the function. The client checks the correctness proof to decide whether the output provided by the server is accepted or not.

Definition 6 (Verifiable Computation)**.** *Let $\mathcal{F}$ be a function. A VC scheme $\mathcal{VC}$ = (*KeyGen*,* ProbGen*,* Compute*,* Verify*) for function $\mathcal{F}$ consists of the algorithms described below.*

- $(SK, PK) \leftarrow \mathsf{KeyGen}(\mathcal{F}, \lambda)$: *The (randomized) key generation algorithm takes as input the function $\mathcal{F}$ and the security parameter λ, and outputs a public key PK and a secret key SK.*
- $(\Sigma_X, VK_X) \leftarrow \mathsf{ProbGen}_{SK}(X)$: *The (randomized) problem generation algorithm takes as input the value X and uses the secret key SK to compute an encoding Σ_X of X and a* secret *verification key VK_X.*
- $\Sigma_Y \leftarrow \mathsf{Compute}_{PK}(\Sigma_X)$: *The (deterministic) compute algorithm takes as input the encoded value Σ_X and uses the public key PK to compute an encoding of $Y = \mathcal{F}(X)$.*
- $Y \leftarrow \mathsf{Verify}_{SK}(VK_X, \Sigma_Y)$: *The (deterministic) verify algorithm takes as input the verification key VK_X and the value Σ_Y; it uses the secret key SK and VK_X to compute a value $Y \in \{0, 1\}^* \cup \{\bot\}$, where symbol $\bot$ denotes that the algorithm rejects the value Σ_Y.*

A typical VC scheme needs to satisfy some properties that we formalize below.

Correctness. A VC scheme is correct if the $\mathsf{ProbGen}$ algorithm produces problem instances that allow for a honest server to successfully compute values Σ_Y such that $Y = \mathcal{F}(X)$.

Definition 7 (Correctness for VC schemes)**.** *Let $\mathcal{VC}$ be a VC scheme for some function $\mathcal{F}$. We say that $\mathcal{VC}$ is correct if for all values X the following holds:*

$$\mathbb{P}\left[Y = \mathcal{F}(X) : \begin{array}{c} (SK, PK) \leftarrow \mathsf{KeyGen}(\mathcal{F}, \lambda); (\Sigma_X, VK_X) \leftarrow \mathsf{ProbGen}_{PK}(X) \\ \Sigma_Y \leftarrow \mathsf{Compute}_{PK}(\Sigma_X); Y \leftarrow \mathsf{Verify}_{SK}(VK_X, \Sigma_Y) \end{array} \right] = 1.$$

Soundness. A VC scheme is sound if no malicious server can "trick" a client into accepting an incorrect output, i.e, some value Y such that $Y \neq \mathcal{F}(X)$. We require this to hold even in the presence of so-called verification queries [29].

Definition 8 (Soundness for VC schemes)**.** *Let $\mathcal{VC}$ be a VC scheme for some function $\mathcal{F}$. We say that $\mathcal{VC}$ is sound if for all* PPT *adversaries $\mathcal{A}$ there exists some negligible function $\nu : \mathbb{N} \to [0, 1]$ such that $\mathbb{P}[\mathcal{A}\ wins] \leq \nu(\lambda)$ in the following game.*

1. *The challenger runs $(SK, PK) \leftarrow \mathsf{KeyGen}(\mathcal{F}, \lambda)$ to obtain the secret key SK and the public key PK, and sends PK to $\mathcal{A}$.*
2. *Adversary $\mathcal{A}$ can make the following two types of queries to the challenger, that can be carried out polinomially many times in any order and in an adaptive way.*
   - *(i) Adversary $\mathcal{A}$ can specify an input X_i; the challenger computes $(VK_i, \Sigma_i) \leftarrow \mathsf{ProbGen}_{SK}(X_i)$ and sends Σ_i to $\mathcal{A}$.*
   - *(ii) Adversary $\mathcal{A}$ can specify verification queries $(i, \hat{\Sigma})$; the challenger computes $Y = \mathsf{Verify}_{SK}(VK_i, \hat{\Sigma})$ and returns 1 if $Y \neq \perp$, otherwise returns 0.*
3. *Eventually, adversary $\mathcal{A}$ will output a pair (i^*, Σ^*); we say that $\mathcal{A}$ wins iff $Y^* \neq \mathcal{F}(X_{i^*})$ and $Y^* \neq \perp$, such that $Y^* = \mathsf{Verify}_{SK}(VK_{i^*}, \Sigma^*)$.*

Input privacy. A VC scheme is input-private if no server can learn the input value X that the function is being computed on.

Definition 9 (Input-privacy for VC schemes)**.** *Let $\mathcal{VC}$ be a VC scheme for some function $\mathcal{F}$. We say that $\mathcal{VC}$ is input private if for all* PPT *adversaries $\mathcal{A}$ there exists some negligible function $\nu : \mathbb{N} \to [0, 1]$ such that $\mathbb{P}[\mathcal{A}\ wins] \leq \nu(\lambda)$ in the following game.*

1. *The challenger runs $(SK, PK) \leftarrow \mathsf{KeyGen}(\mathcal{F}, \lambda)$ to obtain the secret key SK and the public key PK, and sends PK to $\mathcal{A}$.*
2. *Adversary $\mathcal{A}$ can make the following two types of queries to the challenger, that can be carried out polinomially many times in any order and in an adaptive way.*
   - *(i) Adversary $\mathcal{A}$ can specify an input X_i; the challenger computes $(VK_i, \Sigma_i) \leftarrow \mathsf{ProbGen}_{SK}(X_i)$ and sends Σ_i to $\mathcal{A}$.*
   - *(ii) Adversary $\mathcal{A}$ can specify verification queries $(i, \hat{\Sigma})$; the challenger computes $Y = \mathsf{Verify}_{SK}(VK_i, \hat{\Sigma})$ and returns 1 if $Y \neq \perp$, otherwise returns 0.*
3. *Adversary $\mathcal{A}$ chooses two values X_0 and X_1 and sends them to the challenger.*
4. *The challenger samples a random bit $b \leftarrow\!\!\$\ \{0, 1\}$ and computes $(VK^*, \Sigma^*) \leftarrow \mathsf{ProbGen}_{SK}(X_b)$ forwarding Σ^* to $\mathcal{A}$.*
5. *Adversary $\mathcal{A}$ can still specify to the challenger the queries described above, including special verification queries for the verification key VK^*.*
6. *Finally, $\mathcal{A}$ outputs a bit b'; we say that $\mathcal{A}$ wins if and only if $b = b'$.*

Outsourceability. A VC scheme is outsourceable if the time to encode the input plus the time to run a verification is smaller than the time to compute the function itself.

Definition 10 (Outsourceability for VC schemes)**.** *A VC scheme can be outsourced if it allows efficient generation and efficient verification. This means that for any X and any Σ_Y the time (or circuit size) required for $\mathsf{ProbGen}_{SK}(X)$ plus the time (or circuit size) required for $\mathsf{Verify}_{SK}(VK, \Sigma_Y)$ is $o(T)$, where T is the time (or circuit size) required to compute $\mathcal{F}(X)$.*

VC without input-privacy. The above definitions can be adapted to cast VC schemes without input-privacy, i.e schemes where the server is allowed to learn the input value X used by the client. For such VC schemes, algorithm $\mathsf{ProbGen}$ returns the value X in the clear along with the secret verification key VK_X; the correctness, soundness and outsourceability definitions can easily be adapted to this setting.

4.2 First Compiler

In this section we construct an outsourcing circuit compiler by using a VC scheme that satisfies the properties of correctness, soundness, input-privacy and outsourceability. Let Γ be a circuit. The idea is to invoke a VC scheme for the function $\mathcal{F}$ corresponding to the functionality computed by Γ. The compiled circuit will consist of four main components $\widehat{\Gamma}_{\mathsf{ProbGen}}$, $\widehat{\Gamma}_{\mathsf{Compute}}$, $\widehat{\Gamma}_{\mathsf{Verify}}$, and $\widehat{\Gamma}_{\$}$. The first three components are the circuit representations of the algorithms $\mathsf{ProbGen}$, $\mathsf{Compute}$ and Verify corresponding to the underlying VC scheme; such components hard-wire keys (SK, PK) generated using algorithm KeyGen. The fourth component samples the random coins R_i to be used during each invocation of the circuit.

The production of component $\widehat{\Gamma}_{\mathsf{Compute}}$ will then be outsourced to a single untrusted facility, whereas all other components are built in-house (as their implementation needs to be trusted). Notice that the implementation of algorithm KeyGen can be thought of as a pre-processing stage that runs only once (and could be carried out in software).

An important observation is that the size of circuit $\widehat{\Gamma}_{\mathsf{Verify}}$ and $\widehat{\Gamma}_{\mathsf{ProbGen}}$ is independent, and much smaller, than the size of circuit $\widehat{\Gamma}_{\mathsf{Compute}}$. As discussed in the introduction, the size of $\widehat{\Gamma}_{\$}$ can also be considered to be constant (consisting only of a few gates). We describe our first compiler below in more details.

The compiler $\Phi^1_{\mathcal{VC}}$. Let Γ be a circuit, and $\mathcal{VC} = (\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ be a VC scheme for the function $\mathcal{F}$ implemented by Γ. Our first compiler is depicted in Fig. 2, and can be described as follows.

1. First run $(SK, PK) \leftarrow \mathsf{KeyGen}(\mathcal{F}, \lambda)$ once, obtaining the pair of keys (SK, PK).
2. In case the original circuit is randomized, let $\widehat{\Gamma}_{\$}$ be a circuit outputting random coins R_i (as needed in each invocation of the circuit).
3. Define a component for each function $\mathsf{ProbGen}$, $\mathsf{Compute}$ and Verify of the $\mathcal{VC}$ scheme as explained below.

   - $\widehat{\Gamma}_{\mathsf{ProbGen}}$: This component embeds the secret key SK, and it takes three inputs; the input X_i, the (current) private memory M_i, and random coins R_i (in case the function $\mathcal{F}$ is randomized). It implements function $\mathsf{ProbGen}_{SK}(X_i||M_i||R_i)$, that produces two outputs: an encoding Σ_{X_i,M_i,R_i}, and a verification key VK_{X_i,M_i,R_i}.
   - $\widehat{\Gamma}_{\mathsf{Compute}}$: This component embeds the public key PK, and it takes as input the encoding Σ_{X_i,M_i,R_i}. It implements function $\mathsf{Compute}_{PK}(\Sigma_{X_i,M_i,R_i})$, that produces the encoding $\Sigma_{Y_i,M_{i+1}}$ of $(Y_i, M_{i+1}) = \mathcal{F}(X_i, M_i; R_i)$ as output.

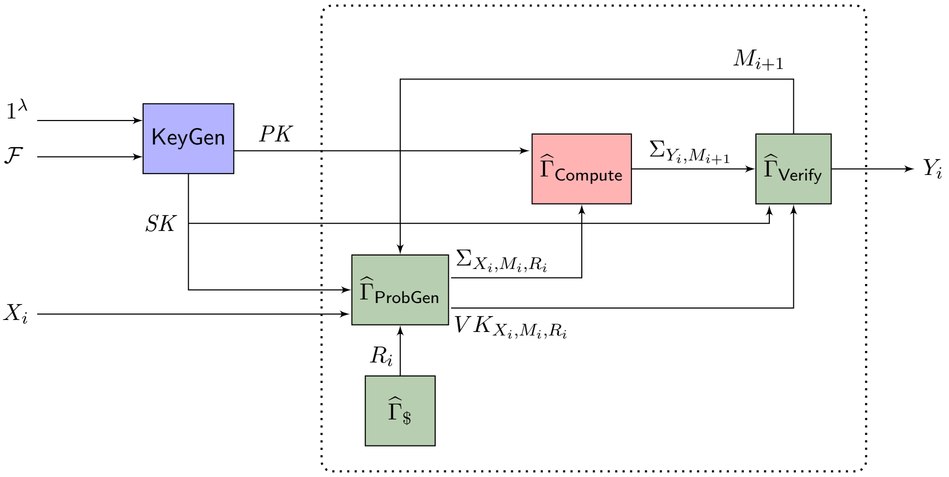

Figure 2: The description of compiler $\Phi^1_{\mathcal{VC}}$. The green parts (i.e., $\widehat{\Gamma}_{\mathsf{ProbGen}}$, $\widehat{\Gamma}_{\mathsf{Verify}}$, and $\widehat{\Gamma}_{\$}$) need to be built in-house, while the production of the red part (i.e., $\widehat{\Gamma}_{\mathsf{Compute}}$) can be outsourced; the blue part (i.e., KeyGen) is built only once (not necessarily in hardware). The dotted line depicts the circuit boundaries.

- $\widehat{\Gamma}_{\mathsf{Verify}}$: This component embeds the secret key SK, and it takes two inputs; the encoding $\Sigma_{Y_i,M_{i+1}}$ and the verification key VK_{X_i,M_i,R_i}. It implements function $\mathsf{Verify}_{SK}(VK_{X_i,M_i,R_i}, \Sigma_{Y_i,M_{i+1}})$, to produce the output $Y_i \in \{0,1\}^* \cup \{\perp\}$, and eventually update the circuit private memory to M_{i+1}.

4. The output of $\Phi^1_{\mathcal{VC}}$ is defined as follows. The first output is a (description of the) compiled circuit $\widehat{\Gamma}$ as depicted in Fig. 2. The auxiliary information aux consists of the components $\widehat{\Gamma}_{\mathsf{ProbGen}}, \widehat{\Gamma}_{\mathsf{Compute}}, \widehat{\Gamma}_{\mathsf{Verify}}$ and $\widehat{\Gamma}_{\$}$, the mapping function $\mathcal{M}$ that describes the physical connections between such components (i.e., the arrows in Fig. 2), and the index set $I = \{2\}$ specifying the component $\widehat{\Gamma}_{\mathsf{Compute}}$ as a candidate for outsourcing.

The theorem belowstates that the compiler from Fig. 2 satisfies our strongest security notion (i.e., Definition 4 with $\ell = 0$), provided that the underlying VC scheme is correct, sound, input-private, and outsourceable.

Theorem 1. *Let Γ be an arbitrary circuit and let $\mathcal{VC}$ be a verifiable computation scheme for the function $\mathcal{F}$ computed by Γ, satisfying the properties of correctness, soundness, input-privacy and outsourceability. Then the compiler $\Phi^1_{\mathcal{VC}}$ is a correct, $(0,1)$-secure $(o(1),1)$-outsourcing circuit compiler.*

Proof idea. We give an intuition for the security proof. Correctness of the compiler and the fact that $\rho = o(1)$ follow immediately, respectively, from the correctness and the outsourceability of the underlying VC scheme. As for security, we need to build a simulator $\mathcal{S}$ that is able to "fake" the real experiment for all adversaries $\mathcal{A}$, for all circuits Γ, and for all initial memory values M_1. The simulator runs compiler $\Phi^1_{\mathcal{VC}}$ upon input Γ, forwards the circuit component $\widehat{\Gamma}_{\mathsf{Compute}}$ to $\mathcal{A}$ obtaining a modified component $\widehat{\Gamma}'_{\mathsf{Compute}}$, and re-assembles the compiled circuit $\widehat{\Gamma}'$ plugging together all the required components. Thus, upon input a query X_i from $\mathcal{A}$, the simulator simply runs $\widehat{\Gamma}$ upon input X_i and using some fixed memory (e.g., the all-zero string); if the output is invalid, $\mathcal{S}$ answers the query with $\perp$, and otherwise it answers the query by using black-box access to the original circuit.

Intuitively, by soundness of the underlying VC scheme, whenever the output of $\widehat{\Gamma}[M_i](\cdot)$ is not $\perp$, such value must be equal to the output of the function $\mathcal{F}(\cdot, M_i)$. On the other hand, the fact that the output is valid or not must be independent of the actual memory used for the computation, as otherwise one could break the input-privacy property of the VC scheme. With this in mind, on can show the indistinguishability between the real and the simulated experiment using a hybrid argument.

Proof. We start by showing that the outsourcing ratio parameter ρ of the compiler $\Phi^1_{\mathcal{VC}}$ is always smaller than 1, for sufficiently large values of the security parameter λ, thus meeting the non-triviality condition.

Claim 1. $\rho = o(1)$.

Proof. The non-triviality requirement from Definition 4 states that

$$\rho = \frac{|\widehat{\Gamma}_{\mathsf{ProbGen}}| + |\widehat{\Gamma}_{\mathsf{Verify}}| + |\widehat{\Gamma}_{\$}|}{|\Gamma|} < 1.$$

The claim is that $\rho = o(1)$, i.e. $\lim_{\lambda\to\infty} \frac{\rho}{1} = 0$. By the outsourceability property of the $\mathcal{VC}$ scheme we know that $|\widehat{\Gamma}_{\mathsf{ProbGen}}| + |\widehat{\Gamma}_{\mathsf{Verify}}| = o(|\Gamma|)$, i.e. $\lim_{\lambda\to\infty} \frac{|\widehat{\Gamma}_{\mathsf{ProbGen}}|+|\widehat{\Gamma}_{\mathsf{Verify}}|}{|\Gamma|} = 0$. By substituting the expression for ρ, and by using the fact that the size of $\widehat{\Gamma}_{\$}$ can assumed to be constant, we obtain:

$$\lim_{\lambda\to\infty} \frac{(|\widehat{\Gamma}_{\mathsf{ProbGen}}| + |\widehat{\Gamma}_{\mathsf{Verify}}| + |\widehat{\Gamma}_{\$}|)/|\Gamma|}{1} = 0.$$

This shows that function ρ converges to 0. As the size of each component is monotonously increasing with the security parameter, for sufficiently large λ, the outsourcing ratio ρ will always be smaller than 1, as desired. □

Claim 2. *The compiler $\Phi^1_{\mathcal{VC}}$ satisfies correctness.*

Proof. The correctness of the compiler $\Phi^1_{\mathcal{VC}}$ follows immediately from the correctness property of the underlying VC scheme. □

We proceed to prove security of $\Phi^1_{\mathcal{VC}}$. We need to build a simulator $\mathcal{S}$ that is able to "fake" experiment **Real** for all adversaries $\mathcal{A}$, for all circuits Γ, and for all initial memory values M_1. A description of the simulator follows.

- Run the compiler $\Phi^1_{\mathcal{VC}}(\Gamma)$ supplying (a description of) the original circuit Γ; the output is (a description of) the compiled circuit $\widehat{\Gamma}$, and the auxiliary information $\mathsf{aux} := ((\widehat{\Gamma}_{\mathsf{ProbGen}}, \widehat{\Gamma}_{\mathsf{Compute}}, \widehat{\Gamma}_{\mathsf{Verify}}, \widehat{\Gamma}_{\$}), \mathcal{M}, \{2\})$.
- The description of component $\widehat{\Gamma}_{\mathsf{Compute}}$ is sent to adversary $\mathcal{A}$ together with the descriptions of Γ and $\widehat{\Gamma}$.
- Adversary $\mathcal{A}$ produces the component $\widehat{\Gamma}'_{\mathsf{Compute}}$ (that may be malicious) and sends it to the simulator; the circuit $\widehat{\Gamma}'$ is assembled using the components $(\widehat{\Gamma}_{\mathsf{ProbGen}}, \widehat{\Gamma}'_{\mathsf{Compute}}, \widehat{\Gamma}_{\mathsf{Verify}}, \widehat{\Gamma}_{\$})$, via the mapping function $\mathcal{M}$.
- Upon input X_i from $\mathcal{A}$, run $(Y_i, \widetilde{M}_{i+1}) \leftarrow \widehat{\Gamma}'[0^\mu](X_i)$, where $\mu := |M_1|$; if $Y_i = \perp$ then forward $\perp$ to $\mathcal{A}$, otherwise query X_i to oracle $\Gamma[M_1](\cdot)$ and forward the output to $\mathcal{A}$.

Let us write $\mathbf{R}$ and $\mathbf{S}$ for the distribution of the random variables in experiment **Real** and **Ideal** of Definition 4. Recall that these variables are parametrized by adversary $\mathcal{A}$, simulator $\mathcal{S}$, initial memory M_1, compiler $\Phi^1_{\mathcal{VC}}$, and circuit Γ, but we omit explicitly writing all these parameters to simplify the exposition.[2] We consider a new experiment $\mathbf{R}'$ that is exactly the same as $\mathbf{R}$, except that for all queries X_i such that $(Y'_i, M_{i+1}) \leftarrow \widehat{\Gamma}'[M_i](X_i)$ with $Y'_i \neq \bot$, the experiment computes $(Y_i, M_{i+1}) \leftarrow \mathcal{F}(X_i, M_i)$ and outputs Y_i.

The claim below shows that experiment $\mathbf{R}$ and $\mathbf{R}'$ are computationally indistinguishable.

Claim 3. *For all* PPT *adversaries $\mathcal{A}$, and for all* PPT *distinguishers $\mathcal{D}$, there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that $\Delta^{\mathcal{D}}(\mathbf{R}; \mathbf{R}') \leq \nu(\lambda)$.*

Proof. Let $q \in \text{poly}(\lambda)$ be the number of input queries asked by $\mathcal{A}$ to its oracle, and let us define an event W that becomes true whenever there is some index $i^* \in [q]$ such that in experiment $\mathbf{R}'$ we have $Y_{i^*} \neq Y'_{i^*}$. Then, by Lemma 1, we have that $\Delta^{\mathcal{D}}(\mathbf{R}; \mathbf{R}') \leq \Delta(\mathbf{R}; \mathbf{R}' | \neg W) + \mathbb{P}[W]$. Note that the distributions $\mathbf{R}$ and $\mathbf{R}'$, conditioned on event W *not* happening are exactly the same. We proceed to show that the probability of event W is negligible.

Let us assume that there exists some circuit Γ, some initial memory M_1 and a PPT adversary $\mathcal{A}$ provoking event W with non-negligible probability. We build a PPT adversary $\mathcal{A}'$ that uses $\mathcal{A}$ in order to break the soundness property of the underlying VC scheme (cf. Definition 8).

<u>Adversary $\mathcal{A}'(M_1, \mathcal{F}, \Gamma, \Phi)$:</u>

1. Receive PK from the challenger, where $(SK, PK) \leftarrow \mathsf{KeyGen}(\mathcal{F}, \lambda)$.
2. Run $(\widehat{\Gamma}, M_1, \mathsf{aux}) \leftarrow \Phi(\Gamma, M_1)$, and send (the description of) circuit components $(\widehat{\Gamma}_{\mathsf{Compute}}, \Gamma, \widehat{\Gamma})$ to $\mathcal{A}$. Adversary $\mathcal{A}$ produces the modified component $\widehat{\Gamma}'_{\mathsf{Compute}}$ and sends it to $\mathcal{A}'$.
3. For all $i \in [q]$, upon input query X_i from $\mathcal{A}$, do the following.
   - Sample random coins R_i, create the string $Z_i := X_i || M_i || R_i$ and send it to the challenger as an encoding query. In response the challenger computes $(VK_{Z_i}, \Sigma_{Z_i}) \leftarrow \mathsf{ProbGen}_{SK}(Z_i)$ and sends back the encoding Σ_{Z_i} to $\mathcal{A}'$.
   - Run $\Sigma_{Y'_i, M_{i+1}} = \widehat{\Gamma}'_{\mathsf{Compute}}(PK, \Sigma_{Z_i})$ and send a verification query $(i, \Sigma_{Y'_i, M_{i+1}})$ to the challenger; the answer from the challenger is a verification bit d. If $d = 0$, then reply with $\bot$, otherwise compute $(Y_i, M_{i+1}) = \mathcal{F}(M_i, X_i; R_i)$ and reply with Y_i.
4. Finally, pick a random $i^* \in [q]$ and output the pair $(i^*, \Sigma_{Y'_{i^*}})$.

For the analysis, we note that the above simulation is perfect. In particular the outputs seen by $\mathcal{A}$ retain exactly the same distribution as in experiment $\mathbf{R}'$. Now $\mathcal{A}$ provokes event W with non-negligible probability, so there exists some index $i^* \in [q]$ such that Y_{i^*} is different from Y'_{i^*} where $(Y'_{i^*}, M_{i^*+1}) \leftarrow \widehat{\Gamma}'[M_{i^*}](X_{i^*})$. Since $\mathcal{A}'$ guesses the right index i^* with probability $1/q$, we obtain that $\mathcal{A}'$ wins the soundness game with non-negligible probability $\frac{1}{q} \cdot \mathbb{P}[W]$. This contradicts the soundness property of the underlying VC scheme, and thus concludes the proof of the claim. □

Claim 4. *For all* PPT *adversaries $\mathcal{A}$, and for all* PPT *distinguishers $\mathcal{D}$, there exists a negligible function $\nu' : \mathbb{N} \to [0,1]$ such that $\Delta^{\mathcal{D}}(\mathbf{R}'; \mathbf{S}) \leq \nu'(\lambda)$.*

[2] We also omit to mention the set of corrupt manufacturers $\mathcal{C}$, as in this case there is a single malicious manufacturer.

Proof. Let $q \in \text{poly}(\lambda)$ be the number of input queries asked by $\mathcal{A}$ to its oracle. For an index $i \in [q]$ consider the hybrid experiment $\mathbf{H}_i$ that answers the first i queries as in $\mathbf{R}'$ and all the subsequent queries as in $\mathbf{S}$. We note that experiments $\mathbf{H}_{i-1}$ and $\mathbf{H}_i$ only differ in how the output is computed in position i, and that $\mathbf{H}_0 \equiv \mathbf{S}$ and $\mathbf{H}_q \equiv \mathbf{R}'$.

We now show that for all circuits Γ, all initial memories M_1, and all PPT adversaries $\mathcal{A}$, each pair of adjacent hybrids $\mathbf{H}_{i-1}$ and $\mathbf{H}_i$ are computationally indistinguishable. Fix some $i \in [0, q]$, and assume that there exists some circuit Γ, some initial memory M_1, a PPT adversary $\mathcal{A}$, and a PPT distinguisher $\mathcal{D}$ that distinguishes between the pair of hybrids $\mathbf{H}_{i-1}$ and $\mathbf{H}_i$. We build an adversary $\mathcal{A}'$ that uses $(\mathcal{A}, \mathcal{D})$ in order to break the input-privacy property of the underlying VC scheme (cf. Definition 9).

Adversary $\mathcal{A}'(M_1, \mathcal{F}, \Gamma, \Phi, i)$:

1. Receive PK from the challenger, where $(SK, PK) \leftarrow \mathsf{KeyGen}(\mathcal{F}, \lambda)$.
2. Run $(\widehat{\Gamma}, \mathsf{aux}) \leftarrow \Phi(\Gamma)$, and send (the description of) circuit components $(\widehat{\Gamma}_{\mathsf{Compute}}, \Gamma, \widehat{\Gamma})$ to $\mathcal{A}$. Adversary $\mathcal{A}$ produces the modified component $\widehat{\Gamma}'_{\mathsf{Compute}}$ and sends it to $\mathcal{A}'$.
3. Upon input query X_j from $\mathcal{D}$, such that $j \neq i$, answer as follows:
   - Sample random coins R_j. If $j < i$ create the string $Z_j := X_j || M_j || R_j$, else create the string $Z_j := X_j || 0^\mu || R_j$.
   - Send Z_j to the challenger as an encoding query. In response the challenger computes $(VK_{Z_j}, \Sigma_{Z_j}) \leftarrow \mathsf{ProbGen}_{SK}(Z_j)$ and sends Σ_{Z_j} to $\mathcal{A}'$.
   - Run $\Sigma_{Y_j, M_{j+1}} = \widehat{\Gamma}'_{\mathsf{Compute}}(PK, \Sigma_{Z_j})$, and send a verification query $(j, \Sigma_{Y_j, M_{j+1}})$ to the challenger.
   - The challenger replies with a decision bit d; if $d = 0$ then return $\perp$, otherwise compute $(Y_j, M_{j+1}) = \mathcal{F}(M_j, X_j; R_j)$ and return Y_j.
4. Upon input query X_i from $\mathcal{A}$, answer as follows:
   - Sample random coins R_i, and create two strings $Z_0^* := X_i || M_i || R_i$ and $Z_1^* := X_i || 0^\mu || R_i$.
   - Send Z_0^* and Z_1^* to the challenger as challenge inputs for the input-privacy game; the challenger replies with Σ_{Z^*} (which corresponds to the encoding of either Z_0^* or Z_1^*).
   - Run $\Sigma_{Y^*} = \widehat{\Gamma}'_{\mathsf{Compute}}(PK, \Sigma_{Z^*})$ and send a verification query (i, Σ_{Y^*}) to the challenger; the challenger replies with a decision bit d. If $d = 0$ then return $\perp$, otherwise compute $(Y_i, M_{i+1}) = \mathcal{F}(M_i, X_i; R_i)$ and return Y_i.
5. Finally, output whatever $\mathcal{D}$ outputs.

For the analysis, we note that the above simulation is perfect. In particular, depending on Z^* being either an encoding of Z_0^* or Z_1^* the view of $(\mathcal{A}, \mathcal{D})$ is identical to the view in either experiment $\mathbf{H}_{i-1}$ or $\mathbf{H}_i$. Hence, $\mathcal{A}'$ retains the same advantage as $(\mathcal{A}, \mathcal{D})$ which contradicts input-privacy of the underlying VC scheme. We conclude that there exist negligible functions $\nu', \nu'' : \mathbb{N} \to [0, 1]$ such that

$$\Delta^{\mathcal{D}}(\mathbf{R}'; \mathbf{S}) \leq \sum_{i=1}^{q} \Delta^{\mathcal{D}}(\mathbf{H}_{i-1}; \mathbf{H}_i) \leq q \cdot \nu'(\lambda) \leq \nu''(\lambda),$$

as desired. □

The statement now follows by Claim 3 and Claim 4, and by the triangle inequality, as

$$\Delta^{\mathcal{D}}(\mathbf{R}; \mathbf{S}) \leq \Delta^{\mathcal{D}}(\mathbf{R}; \mathbf{R}') + \Delta^{\mathcal{D}}(\mathbf{R}'; \mathbf{S}) \leq \nu(\lambda) + \nu'(\lambda).$$

This finishes the proof. □

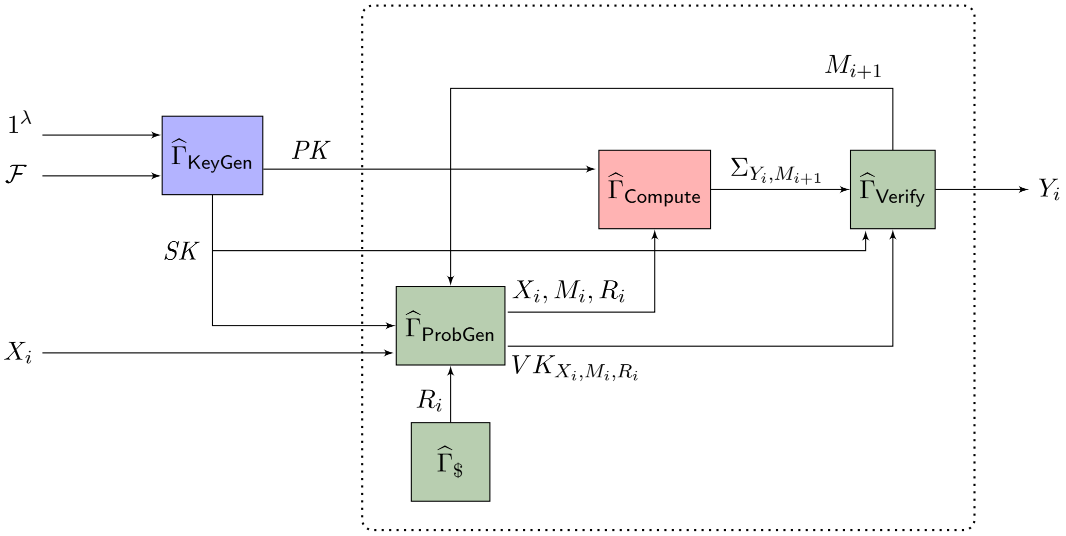
M_{i+1}
1^λ
$\mathcal{F}$
$\widehat{\Gamma}_{\mathsf{KeyGen}}$
PK
$\widehat{\Gamma}_{\mathsf{Compute}}$
$\Sigma_{Y_i,M_{i+1}}$
$\widehat{\Gamma}_{\mathsf{Verify}}$
Y_i
SK
X_i, M_i, R_i
$\widehat{\Gamma}_{\mathsf{ProbGen}}$
X_i
VK_{X_i,M_i,R_i}
R_i
$\widehat{\Gamma}_{\$}$

Figure 3: The description of compiler $\Phi^2_{\mathcal{VC}}$. Notice that component $\widehat{\Gamma}_{\mathsf{ProbGen}}$ does not need to hide its input, and that component $\widehat{\Gamma}_{\mathsf{Verify}}$ implements the self-destruct feature. The dotted line depicts the circuit boundaries.

4.3 Second Compiler

In this section we construct an outsourcing circuit compiler by using any VC scheme that satisfies the properties of correctness, soundness and outsourceability. The construction follows the same ideas of compiler $\Phi^1_{\mathcal{VC}}$ (cf. Section 4.2), with two main differences. First, as we rely on a VC scheme without input-privacy, the component $\widehat{\Gamma}_{\mathsf{ProbGen}}$ now outputs the values X_i, M_i, R_i in the clear. Second, the component $\widehat{\Gamma}_{\mathsf{Verify}}$ needs to implement a special "self-destruct" feature: The first time the component returns the special symbol $\perp$, the private memory is overwritten with the all-zero string.

As we argue later in this section, the self-destruct feature is *necessary*, in that, without such a feature, generic attacks against our compiler are possible, possibly exposing the entire private memory in an undetectable manner.

The compiler $\Phi^2_{\mathcal{VC}}$. Let Γ be a circuit, and $\mathcal{VC} = (\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ be a VC scheme for the function $\mathcal{F}$ implemented by Γ. The description of the compiler $\Phi^2_{\mathcal{VC}}(\Gamma)$ can be found in Fig. 3.The theorem belowestablishes that such a compiler is secure, provided that the original circuit to be produced is resilient to a logarithmic (in the security parameter) amount of leakage on its private memory.

Theorem 2. *Let Γ be an arbitrary circuit and let $\mathcal{VC}$ be a verifiable computation scheme for the function $\mathcal{F}$ computed by Γ, satisfying the properties of correctness, soundness and outsourceability. Then the compiler $\Phi^2_{\mathcal{VC}}$ is a correct* $(\log(q)+1, 1)$*-secure* $(o(1), 1)$*-outsourcing circuit compiler, where q is the number of oracle queries asked by adversary $\mathcal{A}$ in Definition 4.*

Proof idea. We give an intuition for the security proof. As for our first compiler, correctness of the compiler and the fact that $\rho = o(1)$ follow readily from the correctness and the outsourceability properties of the underlying VC scheme. As for security, we need to build a simulator $\mathcal{S}$ that is able to "fake" the real experiment for all adversaries $\mathcal{A}$, for all circuits Γ, and for all initial memory values M_1. The simulator $\mathcal{S}$ is allowed to define an arbitrary polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^{\log(q)+1}$ that leaks a logarithmic amount of information regarding the initial private memory M_1. Intuitively, the function f hard-wires the randomness for running the entire real experiment with adversary $\mathcal{A}$; note that this randomness includes the randomness for the compiler, and the random coin tosses of $\mathcal{A}$. The output of the function consists of: (i) a predicate abort which tells whether there was some round where the output of circuit $\widehat{\Gamma}'$ differs from the output of the original circuit Γ; (ii) the index q^* corresponding to the round where self-destruct was triggered.

In case $\mathsf{abort} = 1$ the simulator $\mathcal{S}$ simply gives up and aborts. Otherwise, it emulates the real experiment with the same randomness hard-wired in the function f, answering all the queries from $\mathcal{A}$ up until round $q^* - 1$ using black-box access to the original circuit, whereas all further queries are answered with $\perp$. On the one hand, this is a perfect simulation as long as $\mathsf{abort} = 0$. On the other hand, the soundness property of the underlying VC scheme ensures that $\mathsf{abort} = 0$ with all but a negligible probability. The proof follows.

Proof. The proof of the non-triviality and the correctness condition are similar to the proofs of Claim 1 and Claim 2 respectively, and are therefore omitted. We proceed to prove the security of $\Phi^2_{\mathcal{VC}}$. We need to build a simulator $\mathcal{S}$ that is able to "fake" experiment **Real** for all adversaries $\mathcal{A}$, for all circuits Γ, and for all initial memory values M_1. The simulator $\mathcal{S}$ is allowed to define an arbitrary polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^{\log(q)+1}$ that leaks a logarithmic amount of information regarding the initial private memory M_1. The function f is defined as follows.

- Sample all random coins for experiment **Real**, including the random coins $\rho_{\mathcal{A}}$ of adversary $\mathcal{A}$, and the random coins ρ_Φ of the compiler Φ.
- Function f takes as input the memory M_1 and the sampled random coins $\rho = (\rho_{\mathcal{A}}, \rho_\Phi)$; note that f can completely simulate experiment **Real** using the memory M_1 and the sampled randomness ρ.
- For all $i \in [q]$ function f runs simultaneously the circuits $\widehat{\Gamma}'[M_i](X_i)$ and $\Gamma[M_i](X_i)$ using the initial memory M_1 and adversary $\mathcal{A}$'s randomness to sample the inputs X_i's.
- Define the event W which becomes true in case there exists some index $i^* \in [q]$ such that $\widehat{\Gamma}'[M_{i^*}](X_{i^*}) \neq \Gamma[M_{i^*}](X_{i^*})$.
- The outputs of function f are
  - $\mathsf{abort} \in \{0,1\}$: a predicate that is 1 if event W happens;
  - $q^* \in [q]$: the index corresponding to the round in which a self-destruct is triggered (if any).

The claim below shows that the probability of event W happening is negligible.

Claim 5. *For all* PPT *adversaries $\mathcal{A}$, the probability that f returns* $\mathsf{abort} = 1$ *is negligible.*

Proof. Let us assume that there exists some circuit Γ, some memory M_1, and a PPT adversary $\mathcal{A}$ that provokes an $\mathsf{abort} = 1$ with non-negligible probability. We build a PPT adversary $\mathcal{A}'$ that uses $\mathcal{A}$ to break the soundness property of the underlying VC scheme (cf. Definition 8). The description of $\mathcal{A}'$ follows.

Adversary $\mathcal{A}'(M_1, \mathcal{F}, \Gamma, \Phi)$:

1. Receive PK from the challenger, where $(SK, PK) \leftarrow \mathsf{KeyGen}(\mathcal{F}, \lambda)$.
2. Run $(\widehat{\Gamma}, \mathsf{aux}) \leftarrow \Phi(\Gamma)$ and send (the description of) circuit components $\widehat{\Gamma}_{\mathsf{Compute}}$, Γ and $\widehat{\Gamma}$ to $\mathcal{A}$. Adversary $\mathcal{A}$ produces the modified component $\widehat{\Gamma}'_{\mathsf{Compute}}$ and sends it to $\mathcal{A}'$.
3. For all $i \in [q]$, upon input query X_i answer as follows.

- Sample random coins R_i, define the string $Z_i := X_i||M_i||R_i$, and run $\Sigma_{Y_i,M_{i+1}} = \widehat{\Gamma}'_{\mathsf{Compute}}(PK, Z_i)$.
- Send a verification query $(i, \Sigma_{Y_i,M_{i+1}})$ to the challenger, receiving back a verification bit d. If $d = 0$ then return $\perp$, and set $M_{i+1} := 0^\mu$; otherwise, compute $(Y_i, M_{i+1}) = \mathcal{F}(M_i, X_i; R_i)$ and return Y_i.

4. Finally, pick a random $i^* \in [q]$ and output the pair $(i^*, \Sigma_{Y_{i^*},M_{i^*+1}})$.

For the analysis, note that the above simulation is perfect. In particular, the first time a verification query is rejected, the reduction returns $\perp$ and overwrites the private memory with the all-zero string. By definition of the event W, we know that there exists an index $i^* \in [q]$ such that the value Y_{i^*} corresponding to the encoding $\Sigma_{Y_{i^*},M_{i^*+1}}$ is different from the value Y_{i^*} computed via $\mathcal{F}(M_{i^*}, X_{i^*})$.

Since $\mathcal{A}'$ guesses the right index i^* with probability $1/q$, we conclude that $\mathcal{A}'$ breaks the soundness property of the underlying VC scheme with non-negligible probability $\frac{1}{q} \cdot \mathbb{P}[W]$. This concludes the claim proof. □

We proceed to describe how the simulator $\mathcal{S}$ uses the obtained leakage in order to fake the distribution in the real experiment (when run with the same randomness ρ initially sampled by the simulator to define the leakage function f).

- After getting the auxiliary input (abort, q^*), check if $\mathsf{abort} = 1$ and in this case stop with output "simulation failed." Otherwise, proceed to the next step.
- Upon input query X_i, such that $i < q^*$, simply forward the query X_i to oracle $\Gamma[M_1](\cdot)$ and output the answer Y_i obtained from the oracle.
- Upon input query X_i, such that $i = q^*$, output $\perp$ and initialize the memory $\widetilde{M}_i := 0^\mu$.
- Upon input query X_i, such that $i > q^*$, run $(Y_i, \widetilde{M}_{i+1}) = \widehat{\Gamma}'[\widetilde{M}_i](X_i)$ and output Y_i.

The claim below shows that, whenever $\mathsf{abort} = 0$, the above simulation is perfect.

Claim 6. *Whenever* $\mathsf{abort} = 0$*, for all circuits* Γ*, all initial memories* M_1*, and for all* PPT *adversaries* $\mathcal{A}$*, the output produced by the above simulator* $\mathcal{S}$ *is identically distributed to the output of the experiment* $\mathbf{Real}_{\mathcal{A},\Phi,\Gamma,M_1}(\lambda)$ *(using the same randomness as sampled by* $\mathcal{S}$*).*

Proof. Notice that the simulator implements the self-destruct feature (overwriting the private memory with the all-zero string) at round q^*; this is exactly what happens in the real experiment (conditioning on the randomness ρ used to define the leakage function being the same as the one used in the experiment). Moreover, all queries before round q^* are answered by running the original circuit Γ. Again, this is a perfect simulation (as $\mathsf{abort} = 0$). The claim follows. □

Let us write $\mathcal{R}$ for the randomness space of experiment $\mathbf{Real}_{\mathcal{A},\Phi,\Gamma,M_1}(\lambda)$.[3] For $\rho \in \mathcal{R}$ let us write $\mathbf{Real}_{\mathcal{A},\Phi,\Gamma,M_1}(\lambda)[\rho]$ to denote the outcome of the real experiment when using the randomness ρ; similarly, let us write $\mathbf{Ideal}_{\mathcal{S},\mathcal{A},\Phi,\Gamma,M_1,\ell}(\lambda)[\rho]$ for the outcome of the ideal experiment when using the randomness ρ. Whenever $\mathsf{abort} = 0$, by the above claim, we have that for any ρ:

$$\mathbf{Real}_{\mathcal{A},\Phi,\Gamma,M_1}(\lambda)[\rho] = \mathbf{Ideal}_{\mathcal{S},\mathcal{A},\Phi,\Gamma,M_1,\ell}(\lambda)[\rho].$$

On the other hand, by Claim 5, for a random ρ we have that $\mathsf{abort} = 1$ happens only with a negligible probability. It follows that for all PPT distinguishers $\mathcal{D}$ there exists a negligible function $\nu : \mathbb{N} \to [0, 1]$ such that

$$|\mathbb{P}[\mathcal{D}(\mathbf{Real}_{\mathcal{A},\Phi,\Gamma,M_1}(\lambda)) = 1] - \mathbb{P}[\mathcal{D}(\mathbf{Ideal}_{\mathcal{S},\mathcal{A},\Phi,\Gamma,M_1,\ell}(\lambda)) = 1]| \leq \nu(\lambda),$$

[3] In what follows we omit to parametrize the experiments by the set $\mathcal{C}$ of corrupt manufacturers, as we are considering the case of a single malicious manufacturer.

finishing the proof. □

Necessity of self-destruct. We show that the self-destruct feature is *necessary* for the security of compiler $\Phi^2_{\mathcal{VC}}$ by presenting an *undetectable* attack (as per Definition 5) against the circuit $\widehat{\Gamma}'$ produced by compiler $\Phi^2_{\mathcal{VC}}$. The attack, which is described in details in Fig. 4, works for a large class of circuits, and leaks the entire initial private memory M_1 embedded in the compiled circuit.

Theorem 3. *Let Γ be any circuit with input size $n = \omega(\log \lambda)$ and let $\Phi^2_{\mathcal{VC}}$ be the compiler from Fig. 3 without the self-destruct capability. Then, the attack $\mathcal{A}^*$ described in Fig. 4 is undetectable for Γ w.r.t. $\Phi^2_{\mathcal{VC}}$, and leaks the entire initial private memory M_1.*

Proof. The second part of the statement follows directly by observing that knowledge of the trapdoor information τ allows to learn the value M_1 with overwhelming probability.

We proceed to show undetectability. Let $\mathbf{G}$ be the undetectability game described in Definition 5, where the adversary $\mathcal{A}$ is chosen to be adversary $\mathcal{A}^*$ from Fig. 4. Consider the game $\mathbf{G}_0$, an identical copy of game $\mathbf{G}$ when $b = 0$, and consider the game $\mathbf{G}_1$ an identical copy of game $\mathbf{G}$ when $b = 1$. Abusing notation, let us write $\mathbf{G}_0$ and $\mathbf{G}_1$ for the distribution of the random variables corresponding to Test's view in games $\mathbf{G}_0$ and game $\mathbf{G}_1$ respectively. For an index $i \in [q]$ consider the hybrid game $\mathbf{H}_i$ that answers the first i queries as in $\mathbf{G}_0$ and all the subsequent queries as in $\mathbf{G}_1$. We note that game $\mathbf{H}_{i-1}$ and $\mathbf{H}_i$ only differ in position i, and that $\mathbf{H}_0 \equiv \mathbf{G}_1$ and $\mathbf{H}_q \equiv \mathbf{G}_0$.

We claim that, for all $i \in [q]$, it holds $\mathbf{H}_{i-1} \approx_c \mathbf{H}_i$. Fix some index i, and define the event W that the i-th query X_i happens to be equal to the secret value τ embedded in the modified component $\widehat{\Gamma}_{\mathsf{Compute}}$ (as described in Fig. 4). By Lemma 1, $\Delta(\mathbf{G}_0; \mathbf{G}_1) \leq \Delta(\mathbf{G}_0; \mathbf{G}_1 | \neg W) + \mathbb{P}[W]$. Clearly, conditioned on event W *not* happening, the distributions of game $\mathbf{G}_0$ and $\mathbf{G}_1$ are identical; this is because in such a case the modified component $\widehat{\Gamma}'_{\mathsf{Compute}}$ behaves exactly like the original component $\widehat{\Gamma}_{\mathsf{Compute}}$.

On the other hand, if $|\tau| = \omega(\log \lambda)$, the probability of event W is negligible. We conclude

Let Γ be any circuit with input size of n bits, and consider the compiled circuit $\widehat{\Gamma}$ produced by running $\Phi^2_{\mathcal{VC}}(\Gamma)$. Define the following adversarial strategy, aimed at building a modified circuit component $\widehat{\Gamma}'_{\mathsf{Compute}}$ that leaks the entire initial private memory M_1 when given some secret trapdoor information.

$\underline{\mathcal{A}^*(X_i, M_i, R_i)}$:

1. Choose a random value τ, such that $|\tau| = n$, and store it in a memory location.
2. Upon the first run, duplicate and store the content of memory M_1 into another memory location, and initialize a counter $j := 0$.
3. Upon input a tuple (X_i, M_i, R_i) such that $X_i \neq \tau$, output $\Sigma_{Y_i} = \mathsf{Compute}_{PK}(X_i, M_i, R_i)$. Otherwise, behave as follows:
   - If $M_1[j] = 0$, let Σ_{Y_i} be the all-zero string.
   - Else, compute $\Sigma_{Y_i} = \mathsf{Compute}(X_i, M_i, R_i)$.
   - Update counter $j \leftarrow (j + 1) \bmod |M_1|$ and output Σ_{Y_i}.

Figure 4: Undetectable attack against compiler $\Phi^2_{\mathcal{VC}}$ (without the self-destruct capability).

that for all PPT distinguishers $\mathcal{D}$ there exists a negligible function $\nu : \mathbb{N} \to [0, 1]$ such that

$$\Delta^{\mathcal{D}}(\mathbf{G}_0; \mathbf{G}_1) \leq \sum_{i=1}^{q} \Delta^{\mathcal{D}}(\mathbf{H}_{i-1}; \mathbf{H}_i) \leq \nu(\lambda).$$

□

4.4 Concrete Instantiations

The first work dealing explicitly with the problem of outsourcing of computation was the paper by Gennaro *et. al.* [30], where Yao's garbled circuit protocol were adapted for constructing the first VC scheme. Since then, many new schemes and models for verifiable computing were put forward [2, 1, 9, 14, 40, 12]. VC schemes are also a natural application of succinct non-interactive arguments of knowledge (SNARKs), in which proofs can be verified much faster and independently of the complexity of the prover.

For the setting where a single untrusted server performs the function computation, the only technique known how to achieve input-privacy is via fully homomorphic encryption (FHE) [32]. (In other settings, for instance if there are two servers one of which is assumed to be trusted [1], one-way functions are sufficient for achieving input-privacy.)

First compiler. For the compiler of Section 4.2, we need a VC scheme satisfying both soundness and input-privacy (in the presence of verification queries). The only known schemes meeting these requirements are the ones recently constructed by Fiore, Gennaro, and Pastro [29] (relying on FHE).

Second compiler. For the compiler of Section 4.3, we need a VC scheme satisfying soundness (in the presence of verification queries), but no input-privacy is required. Therefore, we can instantiate this compiler using more efficient schemes based on SNARKs [48, 31]. To the best of our knowledge, "Pinocchio" [52] is one of the fastest such schemes (for computing arbitrary functions); it relies on quadratic span programs [31], and some other optimizations in order to achieve its (near) practicality. A more recent work [18], "Geppetto" further reduces the overhead of producing proofs, which could reduce the size of component $\widehat{\Gamma}_{\mathsf{Compute}}$ in our compiler.

Extensions. While the compilers described in this section rely on non-interactive VC schemes, it is easy to generalize our constructions to also work with *interactive* VC. The difference is that the communication pattern between the components $\widehat{\Gamma}_{\mathsf{Compute}}$ and $\widehat{\Gamma}_{\mathsf{Verify}}$ would consist of multiple sequential messages, before the values Y_i and M_{i+1} are produced. VC schemes of this type were designed, e.g., in [34, 17, 35]. See also [58] for an excellent discussion on how to implement those schemes in hardware.

5 Compiler based on MPC

In this section we present a compiler which is based on *multi-party computation* (MPC) techniques, aiming to improve the efficiency of the resulting circuit at the expense of achieving security in the weaker model where there are $m \geq 2$ manufacturers, a t-fraction of which is malicious (for some threshold $t \leq m - 1$). Before presenting our compiler, we first revisit the core ideas of MPC, and then we give a generic definition for MPC protocols in the *client-server* model, along the lines of [7].

5.1 MPC in the Client-Server Model

In MPC we consider p parties, where each party P_i, for $1 \leq i \leq p$, possesses an input X_i and they all wish to jointly compute the tuple $(Y_1, \ldots, Y_p) = \mathcal{F}(X_1, \ldots, X_p)$, where P_i receives Y_i. In the client-server model, the parties are divided into two categories: the parties that provide inputs and wish to receive the output of the computation (clients), and those performing the computation (servers). A *t-private* MPC protocol ensures that any adversary who controls up to t servers cannot leak any information related to the private inputs of the clients, besides the information that can be inferred by inspecting the output of the computation, and regardless of the number of corrupted clients. In our compiler the circuit corresponding to the code executed by the servers will be outsourced to a number of possibly malicious manufacturers, that may apply arbitrary modifications against the circuit components. Thus, we require MPC protocols that are secure against active (malicious) attackers. Additional properties that might be useful are the following:

1. Correctness: The protocol computes the correct output.
2. Output delivery: It guarantees that the honest parties will receive the output of the computation.
3. Fairness: If at least one party learns the output, then, all the parties learn the output.
4. Identifiable abort: Whenever the protocol terminates due to an abort message, it is guaranteed that at least one malicious party will be caught (this might be useful for deactivating adversarial circuit components).

The general idea behind the compiler is the following. Let Γ be a circuit implementing some functionality $\mathcal{F}$, and let $\Pi_{\mathcal{F}}$ be a t-private MPC protocol realizing the function $\mathcal{F}$. Then, assuming the number of malicious manufacturers is at most $t < m$, the circuit $\widehat{\Gamma}$ will implement the code of $\Pi_{\mathcal{F}}$, and each $\widehat{\Gamma}_i$ will implement the code of the i-th server. Below we define the protocol framework that we are going to use for the rest of this section. The idea is to describe any MPC protocol using its next message function, denoted as Next.

Definition 11 (r-round protocols). *Let C, S be sets of probabilistic interactive Turing machines, with cardinalities p, m, respectively. An r-round protocol Π for p clients and m servers is a tuple $(C, S, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Next})$, where $\mathsf{Next} = (\mathsf{Next}_1, \ldots, \mathsf{Next}_m)$, described as follows.*

- Setup: *Each client computes $(X_i^1, \ldots, X_i^m) \leftarrow \mathsf{Enc}(X_i)$, and sends X_i^j to the server indexed by j. Let $\mathbf{in}^j := (X_1^j, \ldots, X_p^j)$, and $\tau_j := 0$ (we assume that the network is fully connected, still the properties of the communication channel depend on the instantiation).*
- Computation: *For $i \in [r]$:*
  - *If $j \neq r$, for $j \in [m]$ execute $(o_1^j, \ldots, o_m^j, \tau_j') \leftarrow \mathsf{Next}_j(\mathbf{in}^j, \tau_j)$, send o_k^j, $k \neq j$, to the server with index k. Set $\mathbf{in}^j = (o_j^1, \ldots, o_j^m)$, and $\tau_j = \tau_j'$.*
  - *If $i = r$, for $j \in [m]$ execute $o^j \leftarrow \mathsf{Next}_j(\mathbf{in}^j, \tau_j)$, and send o_j to Dec.*
- Output: *Execute $(Y_1, \ldots, Y_p) \leftarrow \mathsf{Dec}(o_1, \ldots, o_m)$, and send Y_j to the client with index j.*

For any function $\mathcal{F}$, the protocol computing $\mathcal{F}$ will be denoted by $\Pi_{\mathcal{F}}$.

Informally, in the first step of the protocol execution, the clients encode their inputs, as it is prescribed by Enc, and then the main computation begins. The code executed by the servers at each round is defined by the function Next (the next message function). Hence, in the j-th round, server S_i computes Next_i upon the outputs and the state information τ produced by the other servers in round $j-1$. One can also consider deterministic next message functions, assuming the randomness is given as input in each round. Below, we formally define *correctness* and *privacy* for MPC protocols.

Definition 12 (Correctness). *Let $\mathcal{F}$ be a p-party functionality. We say that Π realizes $\mathcal{F}$ with perfect (resp., statistical) correctness if for any input $(X_1, \ldots, X_p)$, the probability that the output delivered to the i-th client during the protocol execution, is different from Y_i, is 0 (resp., negligible in λ), where $(Y_1, \ldots, Y_p) = \mathcal{F}(X_1, \ldots, X_p)$ for $i \in [p]$.*

Definition 13 ((t, m)-Privacy). *Let λ be the security parameter, p be the number of parties (clients) and m be the number of servers, and let $\mathcal{A}$ be an adversary that may corrupt any set of parties $I_c \subseteq [p]$, and servers $I_s \subset [m]$, of cardinality at most t. We say that the protocol Π realizes $\mathcal{F}$ with (t, m)-privacy if there exists a* PPT *algorithm $\mathcal{S}$ such that for all sufficiently large $\lambda \in \mathbb{N}$,*

$$\mathsf{View}_{I_s, I_c}(\lambda, X_1, \ldots, X_p) \approx_c \mathcal{S}(\lambda, I_c, I_s, , (X_i, Y_i)_{i \in I_c})$$

where $\mathsf{View}_{I_s, I_c}(\lambda, X_1, \ldots, X_p)$ denotes the joint view of the servers and clients in I_s and I_c, respectively, within an execution of the protocol upon inputs $X_1, \ldots, X_p$, and $(Y_1, \ldots, Y_p) = \mathcal{F}(X_1, \ldots, X_p)$.

The main idea behind the above definition is that the view of the attacker during the protocol execution can be computed based on its own input and output only.

5.2 The Compiler

Let Γ be a circuit implementing the function $\mathcal{F}(M_1, \cdot)$, where for any X and $i \in \mathbb{N}$, we have $(Y, M_{i+1}) = \mathcal{F}(M_i, X)$. Let $\Pi_{\mathcal{F}} = (C, S, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Next})$ be an r-round protocol realizing the function $\mathcal{F}$, over a set of m servers with a single client. The compiler produces $(\widehat{\Gamma}, \mathsf{aux}) \leftarrow \Phi_{\Pi_{\mathcal{F}}}(\Gamma)$, where

- $\widehat{\Gamma}$ is the circuit that implements $\Pi_{\mathcal{F}}$ (depicted in Figure 5 for the case $m = 2$ and $p = 1$).
- $\mathsf{aux} = ((\widehat{\Gamma}_1, \ldots, \widehat{\Gamma}_{m+2}), \mathcal{M}, (I_1, \ldots, I_m))$, where
  - $\widehat{\Gamma}_{m+1} = \widehat{\Gamma}_{\mathsf{Enc}}$ and $\widehat{\Gamma}_{m+2} = \widehat{\Gamma}_{\mathsf{Dec}}$, i.e., the circuits $\widehat{\Gamma}_{m+1}$ and $\widehat{\Gamma}_{m+2}$ implement the encoder, Enc, and the decoder Dec, of $\Pi_{\mathcal{F}}$, respectively.
  - For $i \in [m]$, $\widehat{\Gamma}_i$ is the circuit that implements the code of the i-th server, for the entire execution of $\Pi_{\mathcal{F}}$ (r-rounds). Those circuits can be implemented in a straightforward way using the next message function Next_i (cf. the sub-components $\widehat{\Gamma}_1$ and $\widehat{\Gamma}_2$ in Figure 5).
  - The mapping function $\mathcal{M}$ describes the physical connections between the circuits described above, and I_j, for $j \in [m]$, specifies the components that will be outsourced to the manufacturer with index j. In our case $I_j = \{j\}$.
  - In case the original circuit is randomized, in addition to the components described above, Φ also outputs a circuit $\widehat{\Gamma}_{\$}$ producing random coins R_i (as needed in each invocation of the circuit).

Our construction must be non-trivial (cf. Definition 4), thus the underlying protocol Π must satisfy the following outsourceability property.

Definition 14 (Outsourceability of procotols). *A protocol $\Pi = (C, S, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Next})$ that realizes the function $\mathcal{F}$ can be outsourced if it satisfies the following condition: The circuit computing the encoding and decoding procedures $(\mathsf{Enc}, \mathsf{Dec})$ must be smaller than the circuit computing the function $\mathcal{F}$.*

We prove the following result.

Theorem 4. *Let $\mathcal{F}$ be any function, and let $\Pi_{\mathcal{F}}$ be a (t, m)-private MPC protocol for $\mathcal{F}$, satisfying the correctness and outsourceability properties. Then, the compiler $\Phi_{\Pi_{\mathcal{F}}}$ is a correct, $(0, t)$-secure, $(o(1), m)$-outsourcing circuit compiler.*

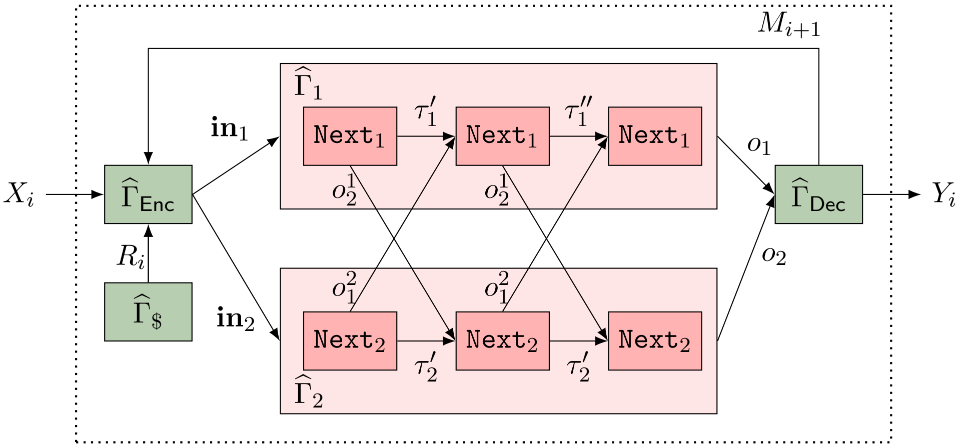

Figure 5: The MPC compiler for the case of $m = 2$ outsourcing facilities (from which *one* can be untrusted). The components $\widehat{\Gamma}_1$ and $\widehat{\Gamma}_2$ can be outsourced, while the connectivity between them and the remaining components are built in-house. The dotted line depicts the circuit boundaries.

Proof idea The correctness property of $\Phi_{\Pi_{\mathcal{F}}}$ follows directly by the correctness property of $\Pi_{\mathcal{F}}$. By the outsourceability property of the $\Pi_{\mathcal{F}}$, we have that $|\mathsf{Enc}| + |\mathsf{Dec}| = o(|\Gamma|)$, which implies that $\lim_{\lambda\to\infty} \frac{|\mathsf{Enc}|+|\mathsf{Dec}|}{|\Gamma|} = 0$. Having this in mind we can prove that the ratio ρ converges to 0, as λ goes to infinity.

Towards contradiction assume $\Phi_{\Pi_{\mathcal{F}}}$ is not $(0, t)$-secure and recall that the number of manufacturers is m. By definition, there exist a set of corrupted manufacturers $\mathcal{C} \subseteq [m]$ of size at most t, a PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, a circuit Γ, and initial memory $M_1 \in \{0, 1\}^*$, such that for all PPT simulators $\mathcal{S}$ with running time $\text{poly}(|\mathcal{A}|, |\Gamma[M_1]|)$, there exists a PPT distinguisher $\mathcal{D}$ that distinguishes between the real and ideal experiments of Definition 4. This informally means that we cannot simulate the output of $\mathcal{A}$ after interacting with the compiled circuit, while we are given black-box access to the original circuit, and the modified sub-components. The goal is to build a PPT attacker $\mathcal{A}'$ against the (t, m)-privacy of $\Pi_{\mathcal{F}}$, i.e., we need to define an attacker $\mathcal{A}'$ corrupting up to t servers, whose view is not simulatable in a black-box way. The protocol $\Pi_{\mathcal{F}}$ that $\Phi_{\Pi_{\mathcal{F}}}$ is based on, consists of a single client, i.e., the one that gives input to the circuit, and m servers. Now, given an adversary $\mathcal{A}$ for $\Phi_{\Pi_{\mathcal{F}}}$ we build an adversary $\mathcal{A}' = (\mathcal{A}'_0, \mathcal{A}'_1)$ against $\Pi_{\mathcal{F}}$ as follows:

- $\mathcal{A}'_0$ controls t servers, that execute the code defined by $\mathcal{A}_0$. Recall that $\mathcal{A}_0$ is given a set of circuit components $\widehat{\Gamma}_i$ of cardinality at most t, and produces a set of possibly modified circuits $\widehat{\Gamma}'_i$.
- $\mathcal{A}'_1$ controls the single client that defines the protocol input, and its code is identical to the code of $\mathcal{A}_1$.

Notice that, after the modifications made by $\mathcal{A}$, the attacker interacts with the circuit $\widehat{\Gamma}'$ initialized with private memory M_1. On the other hand, the above defined adversary $\mathcal{A}'$ runs in the protocol $\Pi_{\mathcal{F}'}$, where $\mathcal{F}'$ is implemented by the circuit $\widehat{\Gamma}'$ having private memory M_1. Clearly the output of $\mathcal{A}'$ cannot be simulated in a black-box way, as we have already assumed that the interaction between $\mathcal{A}$ and the compiled circuit $\widehat{\Gamma}'$ cannot, and thus the (t, m)-privacy property of $\Pi_{\mathcal{F}}$ breaks.

5.3 Concrete Instantiations

Many MPC protocols satisfy the outsourceability property, as the values that feed the main computation, i.e., the output of the encoder, are independent of the function that is being evaluated, and mostly depends on the number of parties, as in the case of [33] (where the same holds for decoding). An explicit (t, m)-private protocol is given in [21], for $t < m/2$, in which there is a pre-processing phase that can be implemented by the encoder, with running time independent of the function that is being evaluated. The construction uses secure point-to-point and broadcast channels, that can be implemented directly between the components, and besides privacy it also guarantees output delivery.

We can also easily adapt the SPDZ protocol [22] to the client-server setting. The SPDZ protocol requires a pre-processing phase that is performed by the parties, and that will feed the encoder circuit who will perform the actual encoding (which is only a linear operation). The complete protocol requires a linear number of public-key operations in the circuit size $|\Gamma|$, with the encoder requiring only a linear number of operations in m. The efficiency of the pre-processing stage can be further improved [5]. This construction does not guarantee output deliver, but it is secure against adversaries that corrupt up to $m - 1$ sub-components.

Finally, the construction of [38] can also be adapted to the client-server scenario and additionally provides security with identifiable aborts.

6 Conclusion and Open Problems

We put forward a simulation-based security definition for assessing security of ICs whose fabrication has (partially) been outsourced to an untrusted off-shore manufacturer. Importantly, the size of the components built in-house must be much smaller than the size of the original IC. Our definition implies a strong guarantee, essentially saying that no matter how the manufacturer modified the outsourced IC sub-components, using the re-assembled IC in the wild cannot leak sensitive information about the private memory.

With such a framework in hand, we proposed three compilers that meet our security definition, and work for any circuit. Our constructions generically leverage a VC scheme or an MPC protocol for the function underlying the produced IC.

There are several interesting open questions related to our work. First, it might be interesting to explore variations of our model, for example by considering the case where there are several (non-colluding) manufacturers involved in the fabrication process. In such a case, it might be possible to obtain significant efficiency improvements, e.g., by relying on special VC schemes already suitable for similar settings [1].

Second, one could try to instantiate our compilers with specialized VC schemes that are tailored for specific functionalities. Although there are already some schemes with this feature—e.g., [29] constructs VC schemes tailored for (multi-variate) polynomials and linear combinations—to the best of our knowledge, there is no concrete VC scheme for verifying the computation of a specific cryptographic functionality (such as AES).

Third, it would be interesting to explore different approaches in order to build compilers meeting our security definition without relying on verifiable computing or multiparty computation.

References

[1] Prabhanjan Ananth, Nishanth Chandran, Vipul Goyal, Bhavana Kanukurthi, and Rafail Ostrovsky. Achieving privacy in verifiable computation with multiple servers - without

FHE and without pre-processing. In *PKC*, pages 149–166, 2014.

[2] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP*, pages 152–163, 2010.

[3] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. In *ACM CCS*, pages 364–375, 2015.

[4] James Ball, Julian Borger, and Glenn Greenwald. Revealed: how US and UK spy agencies defeat internet privacy and security. *Guardian Weekly*, September 2013.

[5] Carsten Baum, Ivan Damgård, Tomas Toft, and Rasmus Zakarias. Better preprocessing for secure multiparty computation. Cryptology ePrint Archive, Report 2016/048, 2016. http://eprint.iacr.org/.

[6] Mark Beaumont, Bradley Hopkins, and Tristan Newby. Hardware trojans — prevention, detection, countermeasures (a literature review). Technical report, Australian Government Department of Defence, 07 2011.

[7] Donald Beaver. Commodity-based cryptography (extended abstract). In *ACM STOC*, pages 446–455, 1997.

[8] Georg T. Becker, Francesco Regazzoni, Christof Paar, and Wayne P. Burleson. Stealthy dopant-level hardware trojans: extended version. *J. Cryptographic Engineering*, 4(1):19–31, 2014.

[9] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In *ASIACRYPT*, pages 134–153, 2012.

[10] Mihir Bellare, Joseph Jaeger, and Daniel Kane. Mass-surveillance without the state: Strongly undetectable algorithm-substitution attacks. In *ACM CCS*, pages 1431–1440, 2015.

[11] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In *CRYPTO*, pages 1–19, 2014.

[12] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: verifying program executions succinctly and in zero knowledge. In *CRYPTO*, pages 90–108, 2013.

[13] Shivam Bhasin and Francesco Regazzoni. A survey on hardware trojan detection techniques. In *IEEE ISCAS*, pages 2021–2024, 2015.

[14] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ICS*, pages 326–349, 2012.

[15] Elette Boyle, Gil Segev, and Daniel Wichs. Fully leakage-resilient signatures. *J. Cryptology*, 26(3):513–558, 2013.

[16] Rajat Subhra Chakraborty, Seetharam Narasimhan, and Swarup Bhunia. Hardware trojan: Threats and emerging solutions. In *IEEE HLDVT*, pages 166–171, 2009.

[17] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Innovations in Theoretical Computer Science*, pages 90–112, 2012.

[18] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 253–270, 2015.

[19] Dana Dachman-Soled and Yael Tauman Kalai. Securing circuits against constant-rate tampering. In *CRYPTO*, pages 533–551, 2012.

[20] Dana Dachman-Soled and Yael Tauman Kalai. Securing circuits and protocols against $1/\mathrm{poly}(k)$ tampering rate. In *TCC*, pages 540–565, 2014.

[21] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *CRYPTO*, pages 378–394, 2005.

[22] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662, 2012.

[23] Jean Paul Degabriele, Pooya Farshim, and Bertram Poettering. A more cautious approach to security against mass surveillance. In *FSE*, pages 579–598, 2015.

[24] Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart. A formal treatment of backdoored pseudorandom generators. In *EUROCRYPT*, pages 101–126, 2015.

[25] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Efficient public-key cryptography in the presence of key leakage. In *ASIACRYPT*, pages 613–631, 2010.

[26] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.

[27] Antonio Faonio, Jesper Buus Nielsen, and Daniele Venturi. Mind your coins: Fully leakage-resilient signatures with graceful degradation. In *ICALP*, pages 456–468, 2015.

[28] Sebastian Faust, Krzysztof Pietrzak, and Daniele Venturi. Tamper-proof circuits: How to trade leakage for tamper-resilience. In *ICALP*, pages 391–402, 2011.

[29] Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In *ACM CCS*, pages 844–855, 2014.

[30] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[31] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *EUROCRYPT*, pages 626–645, 2013.

[32] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[33] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *ACM STOC*, pages 218–229, 1987.

[34] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *ACM STOC*, pages 113–122, 2008.

[35] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27, 2015.

[36] Glenn Greenwald. No place to hide: Edward Snowden, the NSA, and the U.S. surveillance state. *Metropolitan Books*, May 2014.

[37] Mike Hamburg, Paul Kocher, and Mark Marson. Analysis of Intel's Ivy Bridge digital random number generator. Technical report, Cryptography Research, Inc., 03 2012.

[38] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In *CRYPTO*, pages 369–386, 2014.

[39] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits II: keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308–327, 2006.

[40] Thomas P. Jakobsen, Jesper Buus Nielsen, and Claudio Orlandi. A framework for outsourcing of secure computation. In *CCSW*, pages 81–92, 2014.

[41] Benjamin Jun and Paul Kocher. The Intel random number generator. Technical report, Cryptography Research, Inc., 04 1999.

[42] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.

[43] Aggelos Kiayias and Yiannis Tselekounis. Tamper resilient circuits: The adversary at the gates. In *ASIACRYPT*, pages 161–180, 2013.

[44] Lang Lin, Markus Kasper, Tim Güneysu, Christof Paar, and Wayne Burleson. Trojan side-channels: Lightweight hardware trojans through side-channel engineering. In *CHES*, pages 382–395, 2009.

[45] Eric Love, Yier Jin, and Yiorgos Makris. Enhancing security via provably trustworthy hardware intellectual property. In *IEEE HOST*, pages 12–17, 2011.

[46] Marie A. Mak. Trusted Defense Microelectronics: Future Access and Capabilities Are Uncertain. Technical report, United States Government Accountability Office, 10 2015.

[47] David R. McIntyre, Francis G. Wolff, Christos A. Papachristou, and Swarup Bhunia. Dynamic evaluation of hardware trust. In *IEEE HOST*, pages 108–111, 2009.

[48] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[49] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. *SIAM J. Comput.*, 41(4):772–814, 2012.

[50] Jesper Buus Nielsen, Daniele Venturi, and Angela Zottarel. Leakage-resilient signatures with graceful degradation. In *PKC*, pages 362–379, 2014.

[51] Martin Otto. *Fault Attacks and Countermeasures.* PhD thesis, University of Paderborn, Germany, 2006.

[52] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 238–252, 2013.

[53] Nicole Perlroth, Jeff Larson, and Scott Shane. N.S.A. able to foil basic safeguards of privacy on web. *The New York Times*, September 2013.

[54] Krzysztof Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, pages 462–482, 2009.

[55] Miodrag Potkonjak. Synthesis of trustable ics using untrusted CAD tools. In *DAC*, pages 633–634, 2010.

[56] Jean-Pierre Seifert and Christoph Bayer. Trojan-resilient circuits. In Al-Sakib Khan Pathan, editor, *Securing Cyber-Physical Systems*, chapter 14, pages 349–370. CRC Press, Boca Raton, London, New York, 2015.

[57] Brian Sharkey. Trust in Integrated Circuits Program. Technical report, DARPA, 03 2007.

[58] Riad S. Wahby, Max Howald, Siddharth J. Garg, Abhi Shelat, and Michael Walfish. Verifiable ASICs. *IACR Cryptology ePrint Archive*, 2015:1243, 2015.

[59] Adam Waksman and Simha Sethumadhavan. Silencing hardware backdoors. In *IEEE Symposium on Security and Privacy*, pages 49–63, 2011.