Secure Outsourcing of Cryptographic Circuits Manufacturing

Secure Outsourcing of Cryptographic Circuits Manufacturing

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

[1] Stevens Institute of Technology, USA
gatenies@stevens.edu
[2] The University of Edinburgh, United Kingdom & IOHK
akiayias@inf.ed.ac.uk
[3] Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
bernardo.magri@fau.de
[4] The University of Edinburgh, United Kingdom
ytselekounis@ed.ac.uk
[5] Sapienza University of Rome, Italy
venturi@di.uniroma1.it

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 concerns. Hardware Trojans are very hard and expensive to detect and can disrupt the entire circuit or covertly leak sensitive information via a subliminal channel.

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. Our model captures that the IC specification and design are trusted but the fabrication facility(ies) may be malicious. Our objective is to investigate security in an *ideal sense* and follows a simulation based approach that ensures that Trojans cannot release any sensitive information to the outside. It follows that the Trojans' impact in the overall IC operation, in case they exist, will be negligible up to simulation.

We then establish that such level of security is in fact achievable for the case of a single and of multiple outsourcing facilities. We present two compilers for ICs for the single outsourcing facility case relying on verifiable computation (VC) schemes, and another two compilers for the multiple outsourcing facilities case, one relying on multi-server VC schemes, and the other relying on secure multiparty computation (MPC) protocols with certain suitable properties that are attainable by existing schemes.

1 Introduction

The fabrication process adopted by the semiconductor industry is fundamentally global, involving several parties that may not be trusted. As a result, integrated

⋆ Research partly supported by the H2020 project PANORAMIX (# 653497).

circuits (ICs) are vulnerable to so-called hardware Trojans that can compromise or disable critical systems, or covertly leak sensitive information [7]. 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 active 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 [9]. 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 [32].

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 report by the U.S. Government Accountability Office (GAO) [25], 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.

1.1 Previous Work

Inspired by the above considerations, in this work we put forward a formal security model for the problem of utilizing off-shore fabrication facilities for IC manufacturing. Our main motivation is that the setting of secure circuit fabrica-

tion, while being an extremely important practical problem, almost completely lacks theoretical foundations. We discuss a few remarkable exceptions below.

- Seifert and Bayer [31] introduced a very strong security model for the fabrication of Trojan-resilient circuits, where the produced circuit is required to always have the same output as the original circuit; unfortunately, they show how to achieve their definition only 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).
- Recently, Wahby *et al.* [33] introduced a new approach to the problem of defeating hardware Trojans in fabless circuit manufacturing. Their model reflects the fact that IC specification and design are trusted but the fabrication facility is not. Rather than testing or reverse engineering the IC hardware received, which only provides limited security, they consider a class of solutions where the IC's operations are continuously verified.
  In a nutshell, the goal of [33] is to make sure that the produced circuit maintains correctness of the computation, meaning that the output of the circuit is either invalid, or equal to the output of the original circuit. The main drawback is that invalid outputs might be arbitrarily correlated with the secret state of the circuit, which could expose key material in case the produced circuit is a cryptographic circuit. (We will formalize this fact later in the paper.)
- In [14], the authors show how to protect against hardware Trojans using testing-based mechanisms. Their work is based on two existing techniques for Trojan detection, called "input scrambling" and "split manufacturing" [20], for which the authors provide formal models. Hence, they present a generic compiler that transforms any circuit into a new (equivalent) circuit with the following guarantee: Assuming the attacker invokes the circuit q times, and that the device is being tested t times, for $t > q$ uniform on a specific range which is not known to the attacker, the compiled circuit is secure with probability at least $1 - (q/t)^{\ell/2}$, were ℓ is the number of copies of the sub-circuits whose production is outsourced.
  The main limitation is that [14] assumes an a-priori known bound on the number q of interactions between the user and the device; in fact, without such a bound, their construction would require a super-polynomial number of tests. Unfortunately, in many important applications, it is not realistic to assume an upper bound on the value q, and thus it is an important open problem to design a methodology that provides security for an arbitrary polynomial number of interactions between the user/attacker and the device.
- The approach of applying secure distributed computing to defeat hardware Trojans has also been recently explored in [26]. However, this work is more focused on the implementation aspects of this idea, and moreover it assumes that the possibly malicious circuit components run applications that are developed and signed by a trusted software developer.

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. Our security definition implies that using the produced circuit in the wild leaks no information on its secrets. Additionally, the adversarial model we consider does not assume any a-priori bound on the number of executions, and allows the manufacturer(s) to make arbitrary modifications to the outsourced components. In essence, our security model captures any attack in which the backdoored circuit communicates with the user/attacker through the input/output gates of the produced circuit. (This includes digital and analog Trojans, but not hidden antennas as considered in [14].)

With such a framework in hand, we give several design methodologies that achieve our definition with different tradeoffs in terms of security, efficiency, and underlying assumptions. Thus, our work establishes the theoretical feasibility of utilizing off-shore fabrication facilities for IC manufacturing. 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 manufacture the trusted sub-components should be (much) less compared to the effort required to manufacture 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 [22]. 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), us-

ing 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.

Case study I: Single manufacturer. 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. Similarly to [33], our compiler generically leverages a VC scheme for the function $\mathcal{F}$ implemented by Γ. Recent breakthrough research on verifiable computation led to nearly practical schemes that work for any function [29,10]; some schemes additionally preserve the privacy of the inputs on which the function is being computed on [15]. VC schemes satisfying the latter property are called *input-private*.

The main idea of how to use verifiable computation in order to build secure outsourcing compilers is simple enough to describe it here. The fabrication of the chips that perform the entire bulk of 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 [19].

Our compiler relies on VC schemes with input-privacy, and achieves our strongest security notion (i.e., no leakage required for the simulation).

Case study II: Multiple manufacturers. 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 malicious. This is a strictly stronger assumption compared to the setting of a single manufacturer, nevertheless, as we show, it opens the possibility for more efficient constructions and stronger availability guarantees.

We present an outsourcing compiler utilizing a general client-server secure multiparty 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., [12]), while others, as we comment later, can be easily adapted to it.

Given such a protocol, the compiler operates in the following way (see also Section 5.1). 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. [21], we can enable our compiled circuit to switch-off an outsourced sub-component that is discovered to be faulty (or malicious), thus reducing energy consumption.

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 [30,24]. However, since it is not always possible to efficiently prevent Trojans insertion, Trojans *detection* has also been vastly explored [9]; 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 [34]. Another approach is the duplication of logic elements and the division of the sensitive data to independent parts of the circuit [27,34].

Tamper-proof circuits. Our main security definition shares similarities with analogous definitions in the context of protecting circuits against tampering attacks [11]. The main difference between this setting and the one considered in our paper is that tamper-proof circuit compilers are typically used to protect against fault injection [28] 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 [22] or gate tampering [23]).

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 [18], this line of research recently led to constructing several concrete primitives resisting large classes of subversion attacks [8,3]. 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 $\mathrm{poly}(|x|)$ 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 $\Delta(\mathbf{Z};\mathbf{Z}') = \frac{1}{2}\sum_{z\in Z} |\mathbb{P}[\mathbf{Z}=z] - \mathbb{P}[\mathbf{Z}'=z]|$. 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 $\Delta^{\mathcal{D}}(\mathbf{Z};\mathbf{Z}') := |\mathbb{P}[\mathcal{D}(z)=1 :\ z \leftarrow\!\!\$\, \mathbf{Z}] - \mathbb{P}[\mathcal{D}(z)=1] :\ z \leftarrow\!\!\$\, \mathbf{Z}'| \le \nu(\lambda)$.

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

2.2 Circuits

A (Boolean) circuit $\Gamma = (V, E)$ is a directed 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 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. *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 on 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. *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.

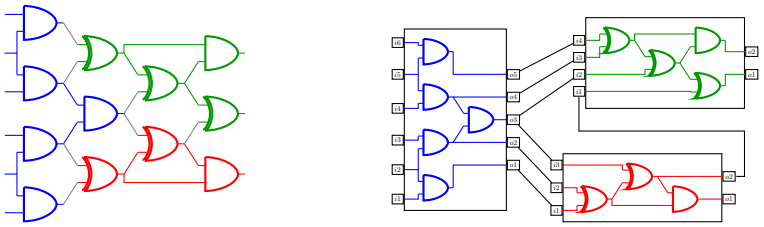

Fig. 1: On the left side we present the description of a (compiled) circuit. On the right side the same circuit is represented as three different components. The mapping function $\mathcal{M}$ establishes the connections between the components.

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 $j \in [m]$.

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

Definition 3. *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).*

For randomized functionalities we require the output distributions of the original and the compiled circuits, to be statistically close.

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., [22]). 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 manufacturers. 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 (reasonable) scenario where 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 parameterized 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. *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, there exists a simulator $\mathcal{S}$ with running time $\mathrm{poly}(|\mathcal{A}|, |\Gamma|)$, such that for all initial values of the memory $M_1 \in \{0,1\}^*$, $\{\mathbf{Real}_{\mathcal{A},\Phi,\mathcal{C},\Gamma,M_1}(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{Ideal}_{\mathcal{S},\mathcal{A},\Phi,\mathcal{C},\Gamma,M_1,\ell}(\lambda)\}_{\lambda\in\mathbb{N}}$.*

In the above definitions the adversary is allowed to modify each $\widehat{\Gamma}_i$ arbitrarily, i.e., there is no restriction on the edges and nodes of $\widehat{\Gamma}'_i$, as long as the input and output gates enable connectivity with the remaining components. We also allow arbitrary modifications of the circuit memory (cf. Remark 1). Observe that, the above definition is only interesting for small values of ℓ (as, e.g., it becomes trivial in case $\ell = |M_1|$). 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 is, the better, as this means that a large fraction of the original circuit production can be outsourced.

4 Case Study I: Single Manufacturer

In this section we study secure outsourcing compilers that work for any circuit, in the presence of a *single* malicious manufacturer. In Section 4.1 we describe our compiler, that is based on any verifiable computation (VC) scheme (satisfying some properties) for the function computed by the underlying circuit.

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

- Correctness: The ProbGen algorithm produces problem instances that allow for a honest server to successfully compute values Σ_Y such that $Y = \mathcal{F}(X)$.
- Soundness: 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 [15].
- Input privacy: No server can learn the input value X that the function is being computed on.
- Outsourceability: The time to encode the input plus the time to run a verification is smaller than the time to compute the function itself.

The reader is deferred to the full version [2] of this paper for a more thorough treatment of the definitions for VC schemes.

4.1 Compiler based on Input-Private VC

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 ProbGen, 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. Let $\widehat{\Gamma}_{\mathsf{Memory}}$ be a circuit component consisting only of memory gates, as needed by the original circuit Γ, storing the initial value of the private memory M_1.
3. Let $\widehat{\Gamma}_{\$}$ be a circuit outputting random coins $\widehat{R}_i$ (as needed in each invocation of the compiled circuit).
4. Define a component for each function ProbGen, 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 $\widehat{R}_i := R_i || R'_i$. It implements function $\mathsf{ProbGen}_{SK}(X_i||M_i||R_i; R'_i)$, 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 an encoding. It implements the 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.
   - $\widehat{\Gamma}_{\mathsf{Verify}}$: 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 out-

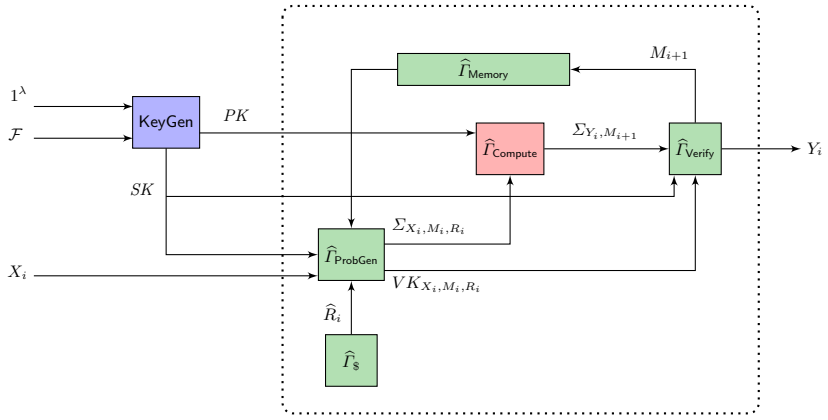

Fig. 2: The description of compilers $\Phi^1_{\mathcal{VC}}$. The green components (i.e., $\widehat{\Gamma}_{\mathsf{ProbGen}}$, $\widehat{\Gamma}_{\mathsf{Verify}}$, and $\widehat{\Gamma}_{\$}$) need to be built in-house, while the production of the red component (i.e., $\widehat{\Gamma}_{\mathsf{Compute}}$)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

put $Y_i \in \{0,1\}^* \cup \{\perp\}$, and eventually update the circuit private memory to M_{i+1}.

5. 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 $\widehat{\Gamma}_{\mathsf{ProbGen}}, \widehat{\Gamma}_{\mathsf{Compute}}, \widehat{\Gamma}_{\mathsf{Verify}}, \widehat{\Gamma}_{\mathsf{Memory}}$, 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.

Remark 1 (On outsourcing memory gates.). In the compiler depicted in Fig. 2, $\widehat{\Gamma}_{\mathsf{Memory}}$ is being built in-house. In order to outsource private memory to a potentially malicious manufacturer we can modify the above compiler as follows: instead of storing in $\widehat{\Gamma}_{\mathsf{Memory}}$ the value M_i in plaintext, we store $C \leftarrow \mathsf{AE}_{SK'}(M_i)$, where C is the encryption of M_i using a symmetric, semantically secure authenticated encryption scheme, with secret key SK'. Moreover, $\widehat{\Gamma}_{\mathsf{ProbGen}}$ is modified such that when receiving the private memory value C, it first decrypts it using SK' and then executes the original circuit $\widehat{\Gamma}_{\mathsf{ProbGen}}$ on the resulting plaintext. We also substitute $\widehat{\Gamma}_{\mathsf{Verify}}$ so that it outputs the encryption of M_{i+1}, under SK'. This modification enables the simulator to execute the circuit using the all-zeros bit-string as the initial memory value, and security follows by the semantic security of the encryption scheme. Finally, whenever the decryption of C gives $\perp$ the circuit output is $\perp$.

The theorem below, whose proof appears in the full version [2] of the paper, states 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, one can show the indistinguishability between the real and the simulated experiments using a hybrid argument.

5 Case Study II: Multiple Manufacturers

In this section we focus on outsourcing compilers in the presence of multiple manufacturers, 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 $t \leq m - 1$).

Our solution, described in Section 5.1, is based on client-server multi-party computation protocols.

5.1 Compiler based on MPC

In this section we present our compiler based on a client-server multi-party computation (MPC) protocol. The reader is referred to the full version [2] of this paper for a formal definition of client-server MPC.

The compiler $\Phi_{\Pi_\mathcal{F}}$. 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 $\Pi_\mathcal{F}$ having as a sub-circuit $\widehat{\Gamma}_{\mathsf{Memory}}$, which is a circuit consisting only of memory gates, as needed by the original circuit Γ. During initialization, $\widehat{\Gamma}_{\mathsf{Memory}}$ stores the initial private memory value, M_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 .
  - 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 5 (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 in the full version [2] of this paper:

Theorem 2. *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.*

6 Concrete Instantiations

In this section we propose several instantiations for the compilers analyzed in the previous sections, highlighting several possible tradeoffs between security, efficiency, and underlying hardness assumptions.

Compiler	Reference	t	s_{in}	s_{out}	No Self-Destruct	Leakage	Assumption
§ 4.1	[15]	–	$O(n+v)$	$O(s \log s)$	✓	0	KoE + FHE
§ 5.1	[13]	$m-1$	$O(dm)$	$O(sm+m^3)$	✓	0	SHE
	[36]	$m-1$	$O(dm^2)$	$O(sm^2 \cdot \lambda / \log s)$	✓	0	OT

Table 1: Comparing our compilers in terms of security, efficiency, and hardness assumptions. We write s, n, v for the size, number of inputs and number of outputs of the original circuit Γ, respectively; as usual m denotes the number of servers of which up to t might be corrupted (note that $t = m$ corresponds to the case of a single manufacturer). The values s_{in} and s_{out} denote, respectively, for the sizes of the components built in house and the size of the outsourced components; d denotes the number of multiplications in Γ. KoE stands for "Knowledge of Exponent" assumptions, FHE for "Fully-Homomorphic Encryption", OT for "Oblivious Transfer" and SHE for "Somewhat Homomorphic Encryption". The first (colored) row represents the compiler with a single outsourcing facility ($m = 1$), while the remaining rows represent the compiler with multiple outsourcing facilities ($m \geq 2$).

6.1 Case Study I

The area of verifiable computation has a long history in the cryptographic literature [4,16] . We refer the reader to the excellent survey by Walfish and Blumberg [35] for a thorough introduction. By now, several schemes and models for the problem of outsourcing computation are known (see, among others, [1]). Below, we focus only on single server VC schemes suitable for the single manufacturer compiler.

Input privacy. For the compiler of Section 4.1, 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 constructed by Fiore, Gennaro, and Pastro [15]

6.2 Case Study II

We describe below a few possible instantiations for the multiple manufacturers compilers of Section 5.

Client-server MPC. 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 depend on the number of parties, as in the case of [17] (where the same holds for decoding). An explicit (t, m)-private protocol is given in [12], 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 [13] 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 s, with the encoder requiring only a linear number of operations in m and the number of multiplications of the original circuit. The efficiency of the pre-processing stage can be further improved [5]. This construction does not guarantee output delivery, but it is secure against adversaries that corrupt up to $m - 1$ sub-components.

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. Giuseppe Ateniese, Aggelos Kiayias, Bernardo Magri, Yiannis Tselekounis, and Daniele Venturi. Secure outsourcing of circuit manufacturing. Cryptology ePrint Archive, Report 2016/527, 2016. https://eprint.iacr.org/2016/527.
3. Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. In *ACM CCS*, pages 364–375, 2015.
4. László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *ACM STOC*, pages 21–31, 1991.
5. Carsten Baum, Ivan Damgård, Tomas Toft, and Rasmus Winther Zakarias. Better preprocessing for secure multiparty computation. In *ACNS*, pages 327–345, 2016.
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. 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.
8. Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In *CRYPTO*, pages 1–19, 2014.
9. Shivam Bhasin and Francesco Regazzoni. A survey on hardware trojan detection techniques. In *IEEE ISCAS*, pages 2021–2024, 2015.
10. 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.
11. Dana Dachman-Soled and Yael Tauman Kalai. Securing circuits and protocols against $1/\text{poly}(k)$ tampering rate. In *TCC*, pages 540–565, 2014.
12. Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *CRYPTO*, pages 378–394, 2005.
13. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662, 2012.
14. Stefan Dziembowski, Sebastian Faust, and François-Xavier Standaert. Private circuits III: hardware trojan-resilience via testing amplification. In *ACM CCS*, pages 142–153, 2016.
15. Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In *ACM CCS*, pages 844–855, 2014.

16. Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.
17. 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.
18. Glenn Greenwald. No place to hide: Edward Snowden, the NSA, and the U.S. surveillance state. *Metropolitan Books*, May 2014.
19. Mike Hamburg, Paul Kocher, and Mark Marson. Analysis of Intel's Ivy Bridge digital random number generator. Technical report, Cryptography Research, Inc., 03 2012.
20. Frank Imeson, Ariq Emtenan, Siddharth Garg, and Mahesh V. Tripunitara. Securing computer hardware using 3d integrated circuit (IC) technology and split manufacturing for obfuscation. In *USENIX Security Symposium*, pages 495–510, 2013.
21. Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In *CRYPTO*, pages 369–386, 2014.
22. Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits II: keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308–327, 2006.
23. Aggelos Kiayias and Yiannis Tselekounis. Tamper resilient circuits: The adversary at the gates. In *ASIACRYPT*, pages 161–180, 2013.
24. Eric Love, Yier Jin, and Yiorgos Makris. Enhancing security via provably trustworthy hardware intellectual property. In *IEEE HOST*, pages 12–17, 2011.
25. Marie A. Mak. Trusted Defense Microelectronics: Future Access and Capabilities Are Uncertain. Technical report, United States Government Accountability Office, 10 2015.
26. Vasilios Mavroudis, Andrea Cerulli, Petr Svenda, Dan Cvrcek, Dusan Klinec, and George Danezis. A touch of evil: High-assurance cryptographic hardware from untrusted components. In *ACM CCS*, pages 1583–1600, 2017.
27. David R. McIntyre, Francis G. Wolff, Christos A. Papachristou, and Swarup Bhunia. Dynamic evaluation of hardware trust. In *IEEE HOST*, pages 108–111, 2009.
28. Martin Otto. *Fault Attacks and Countermeasures*. PhD thesis, University of Paderborn, Germany, 2006.
29. 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.
30. Miodrag Potkonjak. Synthesis of trustable ics using untrusted CAD tools. In *DAC*, pages 633–634, 2010.
31. 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.
32. Brian Sharkey. Trust in Integrated Circuits Program. Technical report, DARPA, 03 2007.
33. Riad S. Wahby, Max Howald, Siddharth J. Garg, Abhi Shelat, and Michael Walfish. Verifiable asics. In *IEEE S&P*, pages 759–778, 2016.
34. Adam Waksman and Simha Sethumadhavan. Silencing hardware backdoors. In *IEEE Symposium on Security and Privacy*, pages 49–63, 2011.
35. Michael Walfish and Andrew J. Blumberg. Verifying computations without reexecuting them. *Commun. ACM*, 58(2):74–84, 2015.
36. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In *ACM CCS*, 2017.