View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by University of Waterloo's Institutional Repository

An Attack and a Defence in the Context of Hardware Security

by

Frank Imeson

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2013

© Frank Imeson 2013

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I understand that my thesis may be made electronically available to the public.

Abstract

The security of digital Integrated Circuits (ICs) is essential to the security of a computer system that comprises them. We present an improved attack on computer hardware that avoids known defence mechanisms and as such raises awareness for the need of new and improved defence mechanisms. We also present a new defence method for securing computer hardware against modifications from untrusted manufacturing facilities, which is of concern since manufacturing is increasingly outsourced.

We improve upon time triggered based backdoors, inserted maliciously in hardware. Prior work has addressed deterministic timer-based triggers — those that are designed to trigger at a specific time with probability 1. We address open questions related to the feasibility of realizing non-deterministic timer-based triggers in hardware — those that are designed with a random component. We show that such timers can be realized in hardware in a manner that is impractical to detect or disable using existing countermeasures of which we are aware. We discuss our design, implementation and analysis of such a timer. We show that the attacker can have surprisingly fine-grained control over the time-window within which the timer triggers. From the attacker's standpoint our non-deterministic timer has key advantages over traditional timer designs. For example the hardware footprint is smaller which increases the chances of avoiding detection. Also our timer has a much smaller time-window for which a volatile state needs to be maintained which in turn makes the power reset defence mechanisms less effective.

Our proposed defence mechanism addresses the threat of a malicious agent at the IC foundry who has information of the circuit and inserts covert, malicious circuitry. The use of 3D IC technology has been suggested as a possible technique to counter this threat. However, to our knowledge, there is no prior work on how such technology can be used effectively. We propose a way to use 3D IC technology for security in this context. Specifically, we obfuscate the circuit by lifting wires to a trusted tier, which is fabricated separately. We provide a precise notion of security that we call k-security and point out that it has interesting similarities and important differences from k-anonymity. We also give a precise specification of the underlying computational problems and their complexity and discuss a comprehensive empirical assessment with benchmark circuits that highlight the security versus cost trade-offs introduced by 3D IC based circuit obfuscation.

Acknowledgements

I would like to thank my supervisors Siddharth Garg and Mahesh V. Tripunitara for sharing their insights and knowledge. Also I would like to acknowledge the collaboration between my supervisors and myself on this work, hence the use of “we, our, and us”.

Dedication

I dedicated this thesis to the Internet; without the Internet my perceived intelligence would only be a fraction of what it is.

PS. Spell check has also been very useful in this respect.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The security of computer hardware, particularly digital Integrated Circuits (ICs), is an important aspect of the overall security of computer systems. Of particular concern is the potential for insider-abuse [61] — the insertion of malicious backdoors by trusted insiders that are involved in the manufacturing process of an IC. A backdoor allows an attacker to bypass system-safeguards.

Hardware security has several similarities to software security. However, as recent work points out [33], the compromise of hardware can be more harmful than the compromise of software for two reasons. One is that attack vectors are more persistent, given that hardware is not as easy to patch as software. The other is that hardware is at the lowest level of the computer system, and thereby, a vulnerability in hardware affects the software that runs above it as well.

This thesis presents two new contributions in the context of hardware security: (i) a non-deterministic timer, a new type of trigger used to attack hardware that utilizes random events to evade current defence mechanisms; and (ii) k-security, a defence mechanism that embeds security into the fabrication process by leveraging 3D IC technology and thereby obfuscating the circuit.

The rest of this chapter will introduce the non-deterministic timer and k-security followed by Chapter 2 which introduces the necessary background of hardware and reviews current research on security in the context of our proposed contributions. In Chapter 3 and Chapter 4 we formally introduce the theory and results of the non-deterministic timer and k-security respectively, review strengths and weaknesses of our contributions in Chapter 5 and finally in Chapter 6 we conclude with a summary of our findings.

1.1 Attacks

Hardware attacks compromise the integrity of the IC. Specifically an adversary can design his or her attack to leak information, degrade performance and or modify the logical. To motivate the importance of hardware security consider an orbital satellite with a maliciously compromised IC designed to fail after one year of operation. This one incident, would cost millions of dollars and the execution of the attack is possible due to the multiple opportunities the design and fabrication process allow, as shown in Figure 2.1.

The insertion of a hardware backdoor is a particularly dangerous kind of attack [74, 75]. It can be used by an attacker to affect a part, or even take over the entire system. A backdoor is typically passive until it is triggered. Once triggered, it performs some malicious action such as unauthorized privilege escalation or even destruction of hardware and software components. A trigger is usually one of two types [74]: data-based or time-based. (A combination of the two is also possible.) With the former, an attack is initiated by a cheat-code, a sequence of bits embedded in the data or instruction stream. With the latter, an attack takes place after the elapse of some time, measured, for example, by CPU cycles.

While both data-based and time-based attacks can be destructive, time-based triggers seem to be particularly dangerous because they do not require any cooperation at run-time, whether accidental or intentional, from a higher-level component such as software or a user. With a time-based attack, the attacker typically embeds a timer that triggers the attack as part of the malicious circuitry. As prior work observes [74], such a timer is simple to implement in hardware.

Our proposed attack pertains to timer-based triggers. That is, attack-triggers that are set off by a hardware circuit that implements a timer. We distinguish two types of timers: deterministic and non-deterministic . A deterministic timer triggers at a time of the attacker's choosing with probability 1. A non-deterministic timer does not; it has a random component to it. (A deterministic timer is a special case of a non-deterministic timer.)

Given the potentially devastating effects of timer-based triggers, prior work has considered how their effects can be mitigated. Waksman and Sethumadhavan [75] propose power-cycling the IC as a general countermeasure to timer-based triggers.

Power cycling, in this context, is fine-grained and is not intended to impede the running of higher-level software. Specifically, the execution of instructions is interrupted briefly and some volatile state (e.g., content of registers) is lost as a consequence of power cycling, but the persistent state (e.g., content of caches and of non-volatile memory) is retained. The program counter is saved to and restored from non-volatile memory. Waksman and Sethumadhavan's [75] empirical observation is that such power-cycling every 1,000,000 cycles results in negligible performance overhead.

They propose that such power-cycling be carried out at both the post-fabrication testing and validation stage of the manufacturing process, and in the field. They show that the former is effective in disabling a particular kind of deterministic timer. We show that there exists a deterministic timer that cannot be disabled by power-cycling in the post-fabrication testing and verification stage, but can be rendered useless by power-cycling in the field (see Section 3.1).

More importantly, prior work leaves open issues related to non-deterministic timer-based triggers. It has been conjectured that from the attacker's standpoint, such a trigger may be of limited value given the random component — the attacker has insufficient control over when it triggers [75]. These open issues are the focus of our attack.

We show that an attacker can realize a non-deterministic timer-based trigger in hardware in a manner that is impractical to detect or disable using existing countermeasures of which we are aware. Furthermore, we show that an attacker can have surprisingly fine-grained control over the behaviour of such timers. For example, an attacker can ensure that the timers in all [1] instances of an IC trigger within the same small time-window, i.e., he is able to ensure that the standard deviation of the timer is small around its expectation. Alternately, an attacker can have such timers trigger at different, seemingly random times.

From the standpoint of the attacker, our construction has other appealing features as well. (See Section 5 for a comprehensive discussion.) It is carried out using digital components only, thereby refuting suggestions in

[1] All in this context means a very high percent of ICs will trigger within a window. In other words, since the trigger is non-deterministic there is a very small probability that an IC may trigger outside of the window in question.

some prior work [75] that analog circuitry is necessary to exploit true randomness. (Our construction requires true randomness — see Section 3.3.) The timer is fully synthesizable, and therefore the attacker can insert such a timer at any stage of the design and fabrication process for ICs, given access to Non-Volatile (NV)-Memory. If the attacker inserts the timer in the design stage, simulations with even unbounded resources cannot detect its presence. (Simulation-based testing is customarily part of pre-fabrication testing.)

The size of the non-volatile state we need to maintain, and the corresponding size of the NV-memory we need for the functioning of the timer, is very small: only the state for a counter, i.e., a few bits. The time-window of volatile state between updates to the non-volatile state is so small (two orders of magnitude smaller than the frequency of power cycling that prior work [75] considers) that it is impractical to power-cycle the circuit sufficiently frequently to interrupt it.

Contributions Our research contribution is a design, implementation and thorough analysis of a non-deterministic timer in hardware that can be used to trigger backdoors in ICs. Our work demonstrates that non-deterministic timer-based triggers, with dangerous characteristics, can be feasibly realized by an attacker that has access to the IC manufacturing process. In particular, compared to deterministic timer attacks, which have been the focus of prior study, the non-deterministic timers are more pernicious for the following reasons:

- Non-deterministic timers are immune to the post-fabrication test time power cycling defence proposed by Waksman and Sethumadhavan [75] that is effective against a class deterministic timer attacks.

- Deterministic timers that escape post-fabrication test time defence mechanisms are easily disabled by even infrequent power cycling in the field. On the other hand, non-deterministic timers can be defeated only by very frequent power cycling, at rates high enough to be almost impractical because of the significant performance penalty that they impose. For example, a non-deterministic timer with an expected trigger time of one year is only defeated if the chip is power cycled once every 27 μs, while an equivalent deterministic timer is defeated even if the chip is power cycled once every 50 minutes (see Case Study in Section 3.4).

- Our implementation of the non-deterministic timer requires (perhaps counter-intuitively) fewer logic gates and fewer bits of NV memory than an equivalent deterministic timer implementation. Thus, non-deterministic attacks are more immune to detection techniques based on IC fingerprinting and physical screening.

- Non-deterministic timers that exploit physical random processes (as our attack does) behave very differently in pre-fabrication simulation and validation tests than they do in the field. This fact has important implications for the immunity of the non-deterministic timer attack to pre-fabrication validation testing — the non-deterministic timer never triggers in logic or circuit simulation tests. On the other hand, deterministic timers do trigger and are consequently detected during pre-silicon validation if the simulations are run for a sufficiently large number of clock cycles.

A possible critique of non-deterministic timers is that the trigger time is, by definition, a random variable and not a deterministic value. Non-deterministic timers can therefore trigger, with some probability, during post-fabrication testing, resulting in early detection. We show, however, that by appropriate choice of design parameters, the probability of the non-deterministic timer triggering can be reduced to values as small as 10^{-100} even if each fabricated chip can be tested for 10 days before shipping to the customer. At the same time, we show that the non-deterministic timer can be engineered to trigger within short confidence intervals with high likelihood resulting in very precise, controlled attacks — for example, more than 99.5% of the timers trigger within 10 days of the expected trigger time for an attack designed to trigger in one year (see Case Study in Section 3.3).

As such, our work is intended to alert hardware verification and test engineers to the considerable threat to the safety of digital ICs from non-deterministic timer based attacks, which have previously received little or no attention because of their perceived impracticality. On the contrary, we show that non-deterministic attacks are not only practical, but potentially more pernicious than deterministic timer attacks.

1.2 Countermeasures

Countermeasures with respect to computer hardware security occur in one of two forms deterrents and preventatives. Unfortunately deterrents often present an initial investment of work to the adversary, but allows for a permanent solution with little to no extra cost in the long run. This is due to the fact that adversary is privy to knowledge of the entire circuit, which fact is likely responsible for the absence of more preventative methods. We propose our technique as a true form of prevention which is realized by hiding knowledge of the entire circuit to the adversary by means of 3D integration.

Three-dimensional (3D) integration, an emerging IC manufacturing technology, is a promising technique to enhance the security of computer hardware. A 3D IC consists of two or more independently manufactured ICs that are vertically stacked on top of each other — each IC in the stack is referred to as a *tier*. Interconnections between the tiers are accomplished using vertical metal pillars referred to as through-silicon vias (TSV).

3D IC manufacturing can potentially enhance hardware security since each tier can be manufactured in a separate IC foundry, and vertically stacked in a secure facility. Thus, a malicious attacker at any one foundry has an incomplete view of the entire circuit, reducing the attacker's ability to alter the circuit functionality in a desired manner.

Tezarron, a leading commercial provider of 3D stacking capabilities, has alluded to the enhanced security offered by 3D integration in a white paper [1]. The white paper notes that "A multi-layer circuit may be divided among the layers in such a way that the function of each layer becomes obscure. Assuming that the TSV connections are extremely fine and abundant, elements can be scattered among the layers in apparently random fashion." However, the paper does not provide any formal notion of security for split manufacturing, nor does it propose techniques to quantify security or achieve a certain security level. These are the open challenges that our proposed defence mechanism addresses in this paper.

Our threat model assumes a malicious attacker in an IC foundry who wants to modify the functionality of a digital IC in a specific, targeted manner. Skorobogatovin et al. [61] found an exploit on an Actel FPGA that leaks secret information stored on the chip when a specific cheat code is observed at the JTAG interface. Similarly, the attack proposed by King et al. [41] modifies the state of hardware registers in a processor to raise the privilege level of the attacker.

To effect a targeted attack, an attacker must first identify specific logic gates or wires in the circuit that implement the functionality that he wants to monitor and/or modify; for example, the gate or wire that corresponds to the privilege bit for the privilege escalation attack proposed in [41]. A

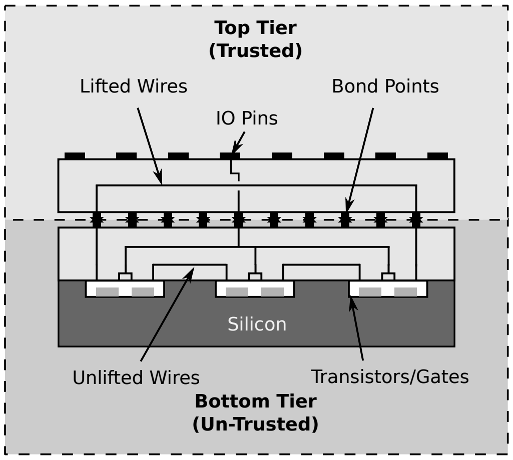


Figure 1.1: A two tier 3D IC. In this instance, the top tier is an interposer, i.e., it only implements metal wires, while the bottom tier has both transistors/gates and wires.

malicious foundry can trivially identify the functionality of every gate and wire in the circuit if it gets to fabricate the entire chip, i.e., if a conventional planar, 2D fabrication process is used. On the other hand, as we show in this paper, 3D integration significantly reduces the ability of an attacker in a malicious foundry to correctly identify gates or wires in the circuit that he wants to attack.

The specific 3D integration technology that we exploit in this work, since it is the only one that is currently in large volume commercial production [15], splits a design into two tiers. The bottom tier consists of digital logic gates and metal wires used to interconnect logic gates. The top tier, also referred to as an *interposer*, only consists of metal wires that provide additional connections between logic gates on the bottom tier.

The bottom tier — this tier is expensive to fabricate since it implements active transistor devices and passive metal — is sent to an external, untrusted foundry for fabrication. This is referred to as the untrusted tier. The top tier implements only passive metal and can be fabricated at lower cost in a trusted fabrication facility. We refer to this tier as the trusted tier.

Assume, for the sake of argument, that all interconnections between logic gates are implemented on the trusted tier, the attacker (who only has access to the untrusted tier) observes only a "sea" of disconnected digital logic gates. From the perspective of the attacker, gates of the same type, for example all NAND gates, are therefore indistinguishable from each other. (Assuming that the relative size or placement of gates reveals no information about interconnections between gates. This is addressed later in the thesis.) Assume also that the attacker wants to attack a specific NAND gate in the circuit, and not just *any* NAND gate. The attacker now has two choices: (a) the attacker could randomly pick one NAND gate to attack from the set of indistinguishable NAND gates, and only succeed in the attack with a certain probability; or (b) the attacker could attack all indistinguishable NAND gates, primarily in cases where the attacker wants to monitor but not modify gates in the circuit, at the expense of a larger malicious circuit and thus, an increased likelihood of the attack being detected. In either instance, the attacker's ability to effect a malicious, targeted attack on the circuit is significantly hindered. We argue that forcing the adversary to either guess or spend more resources then they can afford, while still avoiding detection is a preventative method. We refer to this technique as *circuit obfuscation.*

In general, we define a *k-secure* gate as one that, from the attacker's perspective, cannot be distinguished from $k - 1$ other gates in the circuit. Furthermore, a *k-secure* circuit is defined as one in which each gate is at least *k-secure.*

Contributions In this paper, we make the following contributions:

- We propose a concrete way of leveraging 3D IC technology to secure digital ICs from an active attacker at the foundry. Whereas the use of 3D IC technology has been mooted for security before, we are not aware of prior work like ours that discusses how it can be used.

- We give a precise characterization of the underlying technical problems and identify their computational complexity.

- We propose a meaningful way to characterize security that we call k-security. It has appealing similarities to k-anonymity, but important differences.

- We have devised an approach to addressing our problem of lifting wires, which comprises a greedy heuristic to identify a candidate set of wires to be lifted, and the use of a constraint (SAT) solver to compute k-security.

- We have conducted a thorough empirical assessment of our approach on benchmark circuits. We have also conducted a case-study of a DES circuit that illustrates the inability of an attacker to effectively attack circuits secured using 3D IC based obfuscation.

Chapter 2

Background and Related Work

In this chapter, we overview the IC manufacturing process (Section 2.1) and discuss related work (Section 2.2) along two aspects: attacks and countermeasures. Also, we place our work in the context of the related work.

2.1 Design and Fabrication Process

Computer hardware is the realization of digital logic in the form of an integrated circuit (IC) or a chip. In the design phase of an IC, the desired behaviour of the digital logic is first described using a Hardware Description Language (HDL), which is then synthesized into a network of digital logic gates.

The network of interconnected digital logic gates is referred to as a netlist. The gates and wires in the netlist are then placed and routed, respectively, on the surface of the chip, which results in a circuit layout file that is passed on to the IC fabrication facility (foundry). A simplified view of the IC design and fabrication process is shown in Figure 2.1.

Before the design is sent to a foundry for fabrication, it is subjected to pre-fabrication testing and validation. The traditional objective of this phase is to ensure that the design meets functional and performance specifications. The complexity of modern day ICs prohibits the use of formal verification tools to exhaustively verify the entire state-space of the design. Instead, simulation-based testing is used with the goal of covering a significant portion of the state-space.

Fabricated ICs are subjected to post-fabrication testing and validation.

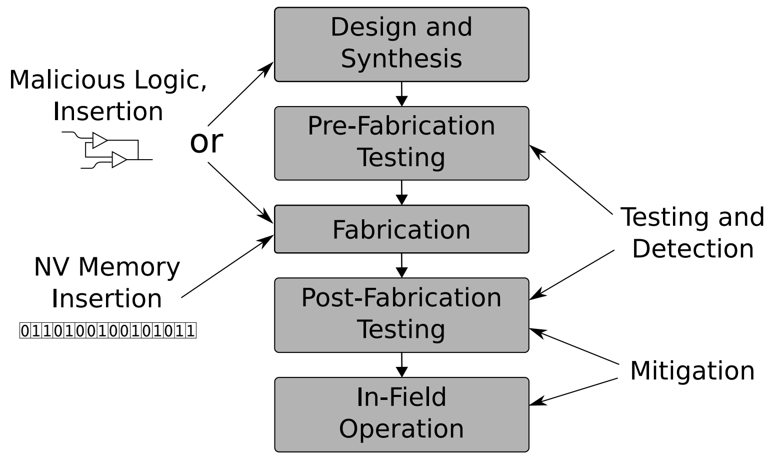


Figure 2.1: The hardware fabrication process. Malicious circuitry may be inserted at either the design and synthesis, or the fabrication phase. The former may be detectable at either the pre- or post-fabrication testing and validation phases. The latter is detectable at the post-fabrication phase only.

Due to cost concerns, only a relatively small portion of the state-space, compared to pre-fabrication testing, can be verified during post-fabrication testing. The traditional purpose of post-fabrication testing is to identify chips that have random, non-malicious defects.

ICs of even moderate complexity can consist of millions of lines of HDL code and millions of gates and wires. In a typical design flow, the HDL code is programmed by teams of programmers in different parts of the world. In addition, portions of the HDL may be purchased from external vendors. Finally, a large majority of IC design companies are *fabless*, i.e., they outsource IC fabrication to a potentially untrusted third-party foundry. The complexity and decentralized nature of the IC design and fabrication processes is the primary reason for the increased threat of malicious hardware being inserted in one or more stages of the design and fabrication flow.

Attacker Model Malicious circuitry can potentially be inserted at the design/synthesis phase or the fabrication phase, or both, of the process shown in Figure 2.1. The former involves an attacker modifying the HDL code or the circuit netlist. The latter involves the attacker modifying the circuit layout at the IC foundry.

Malicious hardware inserted during the design/synthesis stage must avoid detection during the pre- and post-fabrication testing process, while hardware inserted in the fabrication process only needs to escape post-fabrication testing. Although it might seem beneficial to only insert attacks post-fabrication, we note that the attacker has less flexibility in modifying the layout file compared to the HDL code or circuit netlist — once a layout file is sent from the IC vendor to the foundry, any changes in the layout that change the physical dimensions of the chip are immediately observable. Thus, the malicious foundry is restricted to inserting attacks in the "dead space" — unused silicon area between logic gates.

Our work proposes a non-deterministic timer attack that can be inserted in the design or fabrication process of which the exact details are discussed in Chapter 3. Our proposed defence mechanism only addresses the threat from outsourcing the fabrication to an untrusted source of which the details are discussed in Chapter 4.

2.2 Related Work

There is a considerable body of prior work on hardware security. Our work pertains to the design of a non-deterministic timer that can be used to trigger a maliciously inserted hardware backdoor and a defence mechanism that adds security to the fabrication process of ICs in non-trusted facilities. We discuss the related work in the context of both the proposed attack and countermeasure. We also discuss prior research on the technology used in our contributions, namely the use of True Random Number Generators (TRNGs) and 3D IC technology.

2.2.1 Attacks

Broadly, an attack on computer hardware is the modification of the circuitry so that it behaves in a manner not intended by the original designer. The malicious circuitry is typically dormant till it is triggered. Three types of triggers are discussed in prior work [75]: data based, timer based and hybrid triggers. Jin et al. [37] describe the outcome of a competition in which teams of researchers were tasked with inserting hardware attacks in the HDL description of a cryptographic device. The hardware trojans that Jin et al. describe all utilize either a data- or time-trigger to enable the malicious backdoor circuit.

Data-based triggers monitor internal wires of the circuit for specific bit patterns, and trigger when this pattern is detected. King et al. [41] describe an attack that is triggered by the receipt of an unsolicited network packet containing a "magic" byte; this is an example of a data based trigger. On the other hand, ticking time bombs trigger after a certain number of clock cycles have elapsed. (A ticking time bomb is what we call timer-based trigger.)

The number of clock cycles elapsed before the time bomb is triggered can be either deterministic [76, 75] or non-deterministic (as in this work). Wang et al. [76] quantify the hardware resource utilization of a number of different trigger implementations including a deterministic ticking time bomb and a hybrid data and time trigger.

Once triggered, backdoors can either log and transmit sensitive information, modify the circuit functionality, or modify performance specification such as clock frequency and power dissipation [69]. A number of backdoor attacks encompassing these broad categories have been proposed in the literature and we refer the reader to the work of Tehranipoor and Koushanfar for more details [69]. We note that the proposed non deterministic timer based trigger can, in theory, be used to activate any hardware backdoor.

2.2.2 Countermeasures

The most natural phases at which attack-mitigation can occur are the pre- and post-fabrication testing and validation phases of Figure 2.1. Attack mitigation techniques aim to detect or disable malicious circuitry. Indeed, for an attack to be successful, it must evade detection or disablement at these stages.

Hicks et al. [33] propose a pre-fabrication validation technique, referred to as unused circuit identification (UCI), to detect pairs of wires in the design that have the same logic value for all test inputs. The intervening logic between these wires is then flagged as potentially malicious. UCI has since been defeated [65], but we show an alternative exploit that is discussed in more detail in Section 5.1.2.

Banga and Hsiao [10], and Zhang and Tehranipoor [81] propose using equivalence checking and formal verification techniques to check the functionality of a gate level netlist or HDL code against a formal specification. As prior work [75] points out, such techniques are ineffective against timers. Furthermore, as the trigger cannot be activated by external inputs and does not by itself interact with the original design (although the backdoor it triggers presumably does), it is undetected by pre-fabrication equivalence checking and formal verification.

Post-fabrication IC fingerprinting, i.e., measuring the physical characteristics — power dissipation, clock frequency, temperature and electro-magnetic (EM) signatures — of fabricated ICs and comparing them to a gold standard has been proposed as a technique to detect malicious circuitry by a number of prior researchers [3, 6, 38, 54, 78, 80]. These measurements are typically noisy because of inherent variability in the fabrication process and limited precision of measurement instruments. To escape detection, the impact of malicious hardware on the physical characteristics of the IC should be small enough to be statistically indistinguishable from measurement noise.

As we discuss in Section 5.1.4, IC fingerprinting appears to offer the most promise in defending against our non-deterministic timer. However, as we discuss there, some technical challenges need to be addressed before it can be effective.

Waksman and Sethumadhavan [75] propose countermeasures specifically to defend against hardware triggers. In particular, they note that any deterministic timer based trigger must maintain some internal volatile state that can be reset by intermittently power cycling the chip. As we point out in Section 1.1, power cycling, in this context, is a limited operation by which some volatile state such as register- and pipeline-contents are lost, but other state such as cache- and non-volatile memory-contents are retained. Program execution also stops briefly, and the program counter is saved to and restored from non-volatile memory.

They propose also that any on-chip NV-memory used to maintain state can be disabled during post-fabrication testing using a sufficiently large number of such power cycles. We discuss the ineffectiveness of these countermeasures against our timer in Section 3.1. The main issue here is the frequency of power cycling — to disable our timer, it has to be two orders of magnitude more frequent than prior work considers [75], a frequency that is impractical.

Use of TRNGS and NV Memory for Hardware Security Hardware TRNGs have been widely used in cryptographic primitives and security protocols. A number of hardware TRNG implementations have been proposed in the literature — for our proposed non-deterministic attack, we focus on fully-synthesizable implementations that utilize digital logic only and build upon the design suggested by Sunar et al. [67]. Recently, Wang et al. [77] have proposed a non-volatile Flash memory based TRNG that could conceivably be used for a non-deterministic timer attack, but provides lower throughput. In addition, an NV memory based malicious TRNG can be disabled by the post-fabrication test time power cycling defence proposed by Waksman and Sethumadhavan [75]. Finally, we note that the recently proposed TARDIS technique [55] exploits randomness to measure time spent in power off state, but has no direct relationship to our work.

3D Integration for Hardware Security Valamehr et al. [73] also exploit 3D integration capabilities to enhance the security of computer hardware, although in a manner orthogonal to ours. Their proposal involves adding a "control tier" on top of a regular IC to monitor the activity of internal wires in the IC in a cost effective way. By monitoring internal wires on the chip, the control tier is able to detect potentially malicious activity and take appropriate recourse. Adding the monitors vertically on top of the IC to be protected reduces the power and performance cost of monitoring the IC. Another technique along similar lines was proposed by Bilzor [14].

Or technique exploits 3D integration in a different way, i.e., we use it to provide a malicious attacker in an IC foundry with an incomplete view of the circuit netlist, thus deterring the attack. Although the potential for this kind of defence mechanism has been alluded to before by Tezarron in their white paper [71], ours is the first work to address this technique in any consequential way.

Anonymizing Databases and Social Networks Our proposed defence bears relationship to prior work on anonymizing databases and social network graphs, but also has significant differences. A database is k-anonymous if the information for each individual is indistinguishable from $k-1$ other individuals [68] in the database. The notion of k-anonymity for a social network is similar, except that instead of operating on relational data, it operates on a graph. Two individuals in a social network are indistinguishable if their local neighbourhoods are the same [83].

In our setting, the similarity of the local neighbourhood of two gates is only a necessary but not sufficient condition for indistinguishability. This is because the attacker is assumed to have access to the original circuit netlist and an incomplete view of the same netlist, and must thus match *all* gates in the incomplete netlist to gates in the original netlist.

The circuit obfuscation problem also introduces a number of distinct practical issues. These include the additional information that might be conveyed by the circuit layout (for example, the physical proximity of gates), and the role of the number of gate types in the technology library.

Chapter 3

The Non-Deterministic Timer

In this chapter we formalize the exact attack model that pertains to our non-deterministic timer attack, review the previous work on defence mechanisms in Section 3.1. Then we present the theory that allows for a controllable random timer in Section 3.2, in Section 3.3 we show the frame work of our non-deterministic timer on real hardware and in Section 3.4 we analyze the results and costs of the non-deterministic timer in comparison to the deterministic timer.

Attacker Model The proposed non-deterministic timer is comprised of two distinct hardware components, a circuit consisting of digital logic gates (including ring oscillators), and NV-memory. The phase in which each of these malicious components are inserted in the hardware depends on the specific design flow adopted by the IC vendor. For example, in their deterministic timer attack, Waksman and Sethumadhavan [75] assume collusion between a malicious designer that inserts digital logic during the design/synthesis phase and a malicious foundry that inserts NV-memory during fabrication. The same scheme can be used by an attacker to insert the proposed non-deterministic timer attack, as shown in Figure 2.1.

Alternatively, the attack could also be inserted entirely in the design/synthesis phase or entirely in the fabrication phase. An attacker might prefer the former for an IC design and fabrication flow in which CMOS compatible NV-memory technology can be directly instantiated in the design. Such capabilities are increasingly beginning to appear in IC foundry design kits [53, 62]. On the other hand, a fabrication phase only attack is also feasible if sufficient dead space is available on the chip.

In our analysis of the strength of the proposed attack, we have evaluated all relevant pre- and post-fabrication countermeasures against malicious hardware of which we are aware. We show that, regardless of the attack model, a non-deterministic timer can be at least as, if not more pernicious than a deterministic timer.

3.1 Deterministic Timers

In this section, we discuss deterministic timers. In Section 3.1.1, we discuss the state of the art in deterministic timers, and a countermeasure from prior work. In Section 3.1.2, we discuss a new kind of deterministic timer we have conceptualized, and demonstrate that it cannot be disabled in the manner that the timer from Section 3.1.1 can. In Section 3.1.3, we point out that such deterministic timers introduce a trade-off between the size of non-volatile state, and the time-window that volatile state has to be maintained.

3.1.1 State of The Art

A deterministic timer triggers after exactly t clock cycles of chip operation. All deterministic timer implementations of which we are aware, for example the implementations proposed by Waksman and Sethumadhavan [75] and Wang et al. [76], need access to non-volatile state. In a counter based timer, for example, the current count is persistent state.

The simplest deterministic timer stores non-volatile state in on-chip volatile memory. A simple defence against such a timer is to periodically turn off the power to the chip. When the power is restored, the state in the volatile memory is reset to a default value. This is referred to as power cycling. If the number of clock cycles between successive power cycles, t_p, is less than the trigger time, t, then the timer never triggers.

In their work, Waksman and Sethumadhavan [75] consider a more powerful deterministic timer-based trigger that makes use of on-chip NV memory. A drawback of using NV memory is that it has limited write durability. This means that each bit in NV memory can only be written to a limited number of times, after which it loses its ability to store data. Therefore, an attack, to be practical, must be frugal in the number of updates it makes to its non-volatile state.

To address the limited write durability of the NV memory, the timer proposed by Waksman and Sethumadhavan [75] uses the following protocol. The state of the counter is stored in volatile memory. However, if the chip is powered off, the volatile state is transferred to NV memory [1]. It is transferred back to volatile memory when the chip is powered back on. To defend against this attack, that work proposes an elegant defence mechanism. The chip is repeatedly power cycled during post-fabrication testing. As the NV memory is written to once in every power cycle, it burns out if the number of power cycles exceeds its write durability. Thus the timer is rendered ineffective.

3.1.2 Modified Deterministic Timer Protocol

We now discuss a modification of the deterministic timer-based trigger proposed by Waksman and Sethumadhavan [75], which makes the attack immune to repeated power cycling during post-fabrication testing by sacrificing some of the state. In the modified protocol, the NV memory is not written to every time the chip is powered off. Instead, the NV memory is only updated at regular intervals of time, which will drop the current volatile state during a power cycle but will retain the previously stored state.

A simple implementation of the modified timer comprises both a volatile timer and an NV timer that uses m bits of NV memory. The volatile timer triggers after t_v clock cycles, increments the NV timer and resets. The attack is triggered when the NV timer reaches a value k, where $k \leq 2^m - 1$. This protocol is illustrated in Figure 3.1 and 3.2 for clarity. Compared to the deterministic timer of Waksman and Sethumadhavan [75], the NV memory is updated at most k times, regardless of how many times the chip is power cycled. Therefore, the NV memory cannot be burned out during post-fabrication testing. (Of course, we need to ensure that the number of writes to each of the m bits is upper-bounded by the write durability. A sufficient condition is that k is bounded by the write durability.)

While power cycling cannot be used to burn out NV memory in the post-fabrication testing phase, it is still possible to defend against the attack using power cycling in the field. This is another countermeasure proposed by Waksman and Sethumadhavan [75]. In particular, if the number of clock cycles between successive power cycles, $t_p \leq t_v$, the volatile counter will never trigger and the NV memory will never get updated.

[1]This scheme writes to the NV memory the minimum number times to avoid any loss of state. In other words this scheme is the most frugal protocol that always maintains state.

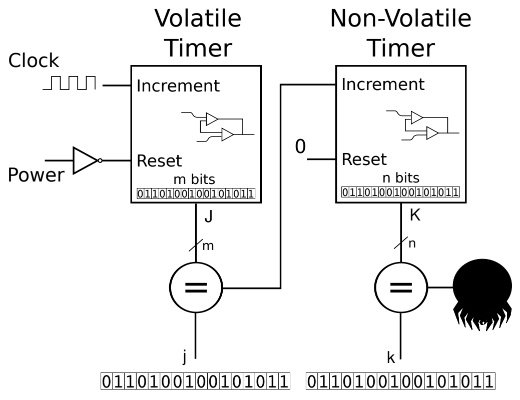


Figure 3.1: Block diagram of a deterministic timer attack that utilizes a volatile timer chained with a non-volatile timer.

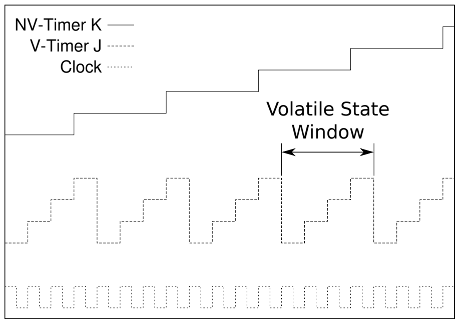


Figure 3.2: Timing diagram for the operation of deterministic timers. The graph shows three signals: the primary clock signal (Clock), the count stored in the volatile state (V-Timer), and the count stored in NV memory (NV-Counter).

We now consider a case study that demonstrates that for realistic attack scenarios, even relatively infrequent power cycling can defend against a deterministic timer attack.

We assume that the attacker wants the trigger to occur after the IC operates for one year in the field. We assume also, based on the characterization of 30-nm NAND Flash technology by Cai et al. [20], that the NV memory can tolerate 10,000 program/erase cycles. Based on these assumptions, the NV memory can be written to at most once every 52 minutes (365 days / 10,000 writes). Therefore, power cycling the chip at any rate faster than once every 52 minutes prevents the timer from going off.

To put this in perspective, Waksman and Sethumadhavan [75] have shown that power cycling complex digital logic as frequently as once a second results in negligible performance impact. This case study illustrates that the limited write durability of NV memory severely limits the effectiveness of deterministic timer based triggers.

3.1.3 NV Memory and Volatile State Window Trade-off

In this section, we continue our discussions of the deterministic timer-based trigger from the previous section. Specifically, we observe that it introduces a trade-off between the size of non-volatile state, and the time-window that volatile state needs to be maintained.

Thus far in our discussions, we have implicitly assumed that binary encoding is used to store data in the NV memory. Thus, the Least Significant Bit (LSB) in the NV memory changes value every time the count is updated. Counting up to a value k results in k writes to the LSB of the NV memory.

To reduce the length of the time-window of volatile state, redundant bits in the NV memory can be used to reduce the number of times each bit in the NV memory is written to while counting up to a value k. However, as we express with the following lemma, it is easy to intuit a lower-bound on the number of bits m required to represent k as a function of k and the write-durability of the NV memory, w.

Lemma 1. *At least one bit of an m bit memory is written to at least $\frac{k}{m}$ times to count up to k. Therefore, $m \geq \frac{k}{w}$.*

We now analyze our scenario from the case study in the previous section. We assume a desired trigger time of one year and an NV memory with a write durability of 10,000 program/erase cycles. We pose the following question: how many bits of NV memory are required that allow it to be written to only once every second? Our intent with this question is to shrink the time-window that volatile state is maintained in an attempt to evade the countermeasure of power-cycling.

If the NV memory is updated every second, it will be written to $\approx 3.154 \times 10^7$ in a year. Based on Lemma 1, at least 3154 bits of NV memory are needed to reduce the volatile state window to one second. This is a $> 200\times$ increase in the number of bits compared to the prior case study where a compact binary encoding was assumed.

NV memory has a different circuit structure compared to on-chip digital logic or volatile memory. Therefore, larger NV memory increases the likelihood of the attack being detected by visually inspecting a small number of fabricated chips. In our proposed non-deterministic timer, we seek to use as few bits of NV memory as possible.

3.2 Non-Deterministic Timer

In this section, we present the design of our non-deterministic timer. Adopting the mindset of the attacker, we have two design goals. One is is to break the apparent trade-off that we discuss in the previous section: between the size of non-volatile storage (NV-memory), and the length of time that volatile state has to be maintained. That is, we want the size of NV-memory to be only the bare minimum. Yet, we would like the time that volatile state must be maintained to be brief as well so that the timer is resistant to countermeasures that erase the volatile state, specifically power cycling. As we discuss in the previous section, this design goal is related to the write durability of the NV memory as well.

Our other design goal is to gain control over when the timer triggers. Our non-deterministic timer is based on random events. So what we want is sufficient control over the expectation and standard deviation of when the timer triggers. We want, with high probability, the timer to trigger at a time of our choosing.

In the next section, we discuss an implementation of our design. The implementation requires that we address some technical challenges of its own. We defer a discussion of those to the next section.

Our timer works as follows. We maintain a counter K to keep track of when the timer should trigger. When K reaches a pre-defined value k, the timer triggers.

We do not use a periodic clock to increment a counter, as in deterministic timers. Instead, we assume access to a source of randomness. (We discuss in the next section how we realize one in practice.) We conduct a series of Bernoulli trials. Each such trial has one of only two outcomes: success or failure. We increment K if and only if we have a successful trial.

We need persistent state to maintain the counter K, up to a value k, only. Each of our Bernoulli trials is independent of the others, and their probabilities are identically distributed. We can choose the probability of success of a trial to be arbitrarily low to increase the expected number of trials between successful events to be arbitrarily large. This way, we can avoid over-utilization of the NV memory.

Other than the counter K, we need to maintain no state. The only actions we perform between increments of K (updates to the non-volatile state) are the Bernoulli trials, and a check of the outcome of each. There is no difference

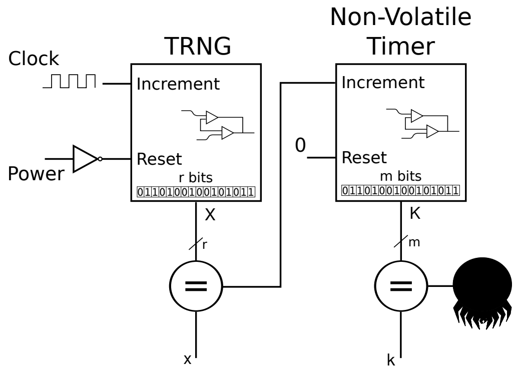


Figure 3.3: Block diagram of a non-deterministic timer attack that utilizes a TRNG chained with a non-volatile timer. The key is compare with the output of the TRNG (see Section 3.3.1).

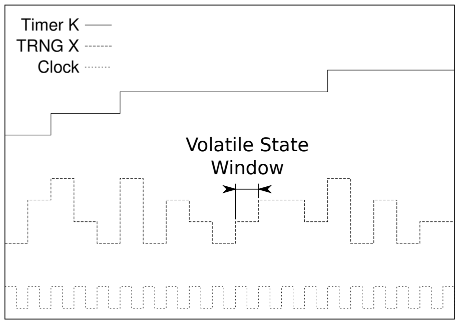


Figure 3.4: Operation of non-deterministic timers. The bottom graph shows the IC's clock signal and the top graph shows the timer's current count. The graph shows three signals: the clock signal (Clock), the output of the TRNG (TRNG X) and the count stored in NV memory (Timer K).

between Bernoulli trials before or after a power-cycle.

Single-event trigger To explain our design in more detail, we first consider the somewhat simpler objective of a single-event trigger. That is, a timer that triggers after one successful Bernoulli trial. If our design is used for a single-event trigger, we can control the expectation, but not the standard deviation.

Let $N \in [1, \infty)$ be the random variable that measures the number of trials up to and including the first success. We consider what the Probability Mass Function (pmf), expectation, variance and standard deviation are for the random variable N. For some value n that N can take, the pmf, denoted $f(n)$ maps n to the probability that $N = n$. The variance measures how widely spread the values that N takes with non-zero probability are, and the standard deviation is the square root of variance. We have the following for the pmf, expectation, variance and standard deviation, respectively, where p is the probability that a Bernoulli trial succeeds. $f(n) = p(1-p)^{n-1}$, $E(N) = \frac{1}{p}$, $V(N) = \frac{1-p}{p^2}$, and, $\sigma \approx \frac{1}{p}$, for small p. For small p, the ratio $\sigma/E(N) \approx 1$ which indicates a wide distribution around the expected value.

As an example let $E(N) = 1000$, $p = 0.001$. In Figure 3.5, we show the corresponding pmf and the Cumulative Distribution Function (cdf). The cdf is the probability that the random variable N takes a value at most n, for various n (the horizontal axis in Figure 3.5). We observe from the cdf that more than half of the timers are likely to go off before 750 trials, which is less than the expected trigger time of 1000 trials.

The single event trigger has another drawback from the perspective of the attacker: it has a high probability of detection during the post-fabrication testing phase. Assume that the testing phase lasts for the first 100 trials. From the cdf in Figure 3.5, we observe that there is a 9.5% chance of a timer being triggered in this phase, thus compromising the attack. Even worse for the attacker, if each fabricated IC is tested, the likelihood that the attack is detected in at least one IC is even higher. If even 100 ICs are tested, the attack is detected with probability 0.999.

We now discuss the multiple event trigger which results in a more favourable distribution of attack time in terms of emulating a deterministic trigger and reducing the likelihood of early detection.

Multiple-event trigger We now analyze the statistics of waiting for k successes, instead of a single success, before triggering the attack. Let N_i be a random variable that represents number of Bernoulli trials between the $(i-1)^{th}$ and i^{th} success. As before, let N represent the total number of trials before k successes. We know that:

$$N = \sum_{i=1}^{k} N_i$$

N_i, for all $i \in [1, K]$, are independent and identically distributed random variables with a distribution that was derived in the preceding discussion on the single event trigger. As a consequence of the Central Limit Theorem [52], the distribution of random variable N will, as $k \to \infty$, tend to a Normal distribution with expectation

$$E(N) = kE(N_i) = \frac{k}{p}$$

and standard deviation

$$\sigma_N = \sqrt{k}\sigma_{N_i} = \frac{\sqrt{k}}{p}$$

We observe that standard deviation as a percentage of the expected number of trials after which the attack triggers depends on the value of k only. In

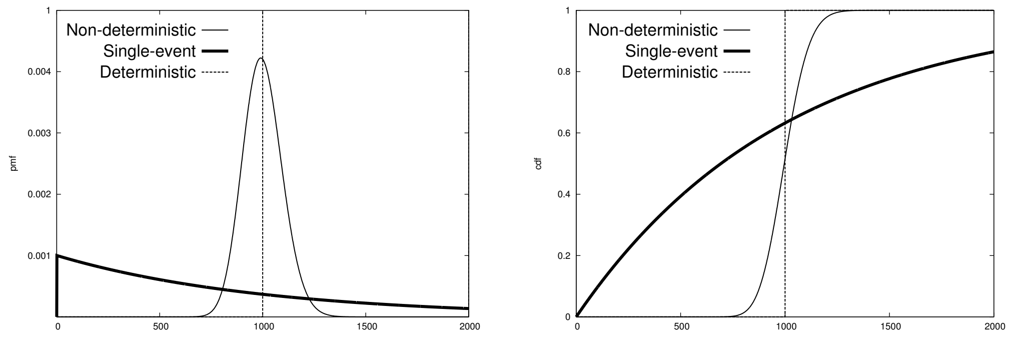


Figure 3.5: The pmfs (left) and cdfs for three different timer designs. Each trigger is designed to have an expectation $E(N) = 1000$. The deterministic timer is shown as dotted, the single event trigger is shown in bold and the multiple event non-deterministic timer is shown as solid, not bold.

particular, $\frac{\sigma_N}{E(N)} = \frac{1}{\sqrt{k}}$. Thus, large values of k result in a narrow distribution around the expected trigger time, while smaller values of k result in a wider distribution. Thus, k is a knob that the attacker can control to effect an attack that either resembles a deterministic attack (large k), or appears to be seemingly random.

Figure 3.5 shows the pmf and cdf of attack time for a multiple event non-deterministic timer ($k = 100$, $p = 0.1$ and $E(N) = 1000$), along with the deterministic and single event trigger distributions for reference. Observe from the figure, that the pmf and cdf of the multiple event timer are, qualitatively, much closer to that of a deterministic timer than that of a single event timer.

In addition, compared to a single event trigger, the multiple event trigger has a much lower likelihood of early detection. As before, if 100 trials are used in post-fabrication testing, the probability that the attack is detected on any given IC is only 10^{-100}. Thus, even if 100,000 ICs are fabricated and each is tested, the attack is detected with probability of only $\approx 10^{-94}$.

We note that, in the discussion so far, we have used the abstract notion of trials as a measure to time. This is easily converted to physical units by noting the time period (in seconds) between successive trials. We discuss this in more detail in the next section.

3.3 Hardware Realization

We have implemented a hardware prototype of our non-deterministic timer on an Altera Straix IV FPGA. In our implementation, we use conventional digital logic components only — logic gates and registers that are readily available in any digital IC fabrication process. The Stratix IV FPGA does not have on-chip NV memory; we use conventional volatile memory blocks instead. In practice, we expect that an NV memory technology, such as NAND Flash, which is compatible with a standard IC fabrication process, will be used to implement the timer.

The non-deterministic timer is programmed in Verilog HDL [72] and is fully synthesizable. We use the Quartus II software suite to synthesize the HDL into a netlist of gates which is then downloaded and programmed on to the FPGA.

The non-deterministic timer comprises two sub-components: the TRNG and the timer logic. The main challenge in the design is to implement a TRNG that satisfies the following properties. First, the TRNG must harness a truly random, physical noise source. A seeded pseudo-random number generator, for example, cannot be used as it restarts from the same seed after every power cycle. A resulting timer would have properties that are very similar to a deterministic timer.

Second, the TRNG must be fully synthesizable and use standard digital logic blocks only. This allows the attack to be inserted in any stage of the design flow, from HDL programming to physical design and fabrication. Finally, the TRNG must have low power dissipation and a small area footprint to avoid detection using IC fingerprinting techniques.

Our implementation of the TRNG is based on the work of Sunar et al. [67], and the subsequent modification due to Wold and Tan [79]. The TRNG exploits oscillator phase noise — small variations in the time period between successive clock ticks of a digital clock generator — to generate a random bit-stream. The oscillators are implemented as rings consisting of an odd number of inverters connected back-to-back. These are referred to as ring oscillators (RO). ROs are commonly used structures in digital ICs. They are used not only for clock generation, but also for in-field monitoring and testing [46]. Therefore, the ROs in the TRNG are unlikely to stand-out as potentially malicious circuitry.

Using this implementation, we conduct experiments with a target trigger time of 24 hours and compare the resulting distribution of trigger time with

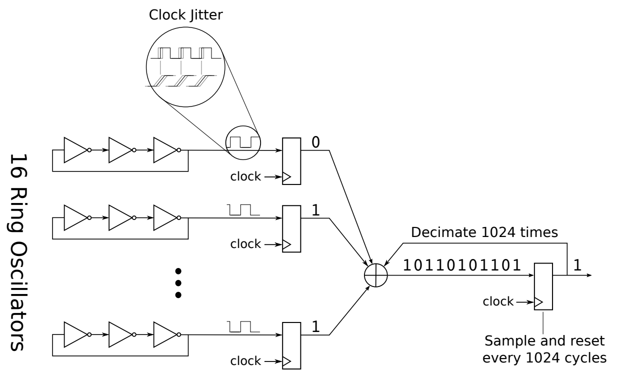


Figure 3.6: Circuit diagram of the TRNG that we implement.

theoretical predictions. We uses hypothesis testing to demonstrate a good match between the theoretical predictions and experimental results.

3.3.1 TRNG Implementation

Figure 3.6 shows a circuit diagram of our TRNG implementation. As mentioned before, the TRNG utilizes ROs, each consisting of a ring of three inverters, that generate noisy clock signals. Here noise refers to the position of the positive and negative edges of the clock with respect to the reference position of an ideal clock and is referred to as phase noise or jitter. Each RO in our implementation consists of three inverters, and there are 16 parallel ROs. Each RO generates an approximately 625 MHz clock signal — the clock frequencies are not identical because of random physical variations between the ROs [67].

The RO clocks are sampled by a 50 MHz system clock using a flip-flop. This results in a stream of bits synchronized to the system clock. In the absence of jitter, the bit-stream at the output of the flip-flop would be pseudo-random in nature. However, in the presence of jitter, any edge of the system clock that aligns sufficiently close to an edge of the RO clock results in a truly random output.

The outputs of the sampling flip-flops are combined together using an exclusive-or (xor) gate. This increases the number of truly random, as opposed to pseudo-random, bits in the output of the xor gate. Finally, the bit-stream at the output of the xor gate is decimated, i.e., each chunk of

1024 bits is xor-ed to produce only one random bit at the output. Decimation guarantees that as long as at least one of the 1024 bits is truly random, the output bit will also be truly random. As shown in Figure 3.6, only a single xor gate is used to perform 1024:1 decimation.

In Section 3.4, we use the National Institute of Standards and Technology (NIST) statistical test suite for random number generators to validate the quality of the random bit-stream that we obtain from our TRNG implementation and show that it passes all NIST mandated tests. At least 16 parallel ROs and 1024:1 decimation is required to pass all NIST tests. Designs with fewer parallel ROs or lesser decimations failed one or more tests.

3.3.2 Timer Design

The random bit-stream from the TRNG is used as input to the timer logic which we have also implemented on the FPGA. The timer collects r random bits from the TRNG output and compares them with an r bit value that we call a key.

If the bit-stream is truly random, the match succeeds with probability $p = \frac{1}{2^r}$. If there is a match, the count in the NV memory is incremented and the attack is triggered when the count reaches k.

As each random bit take 1024 clock cycles to generate and r bits are matched in each Bernoulli trial, the volatile state window, t_v, for the non-deterministic timer is $1024r$ clock cycles. In other words, power cycling the chip at a rate faster than once every $1024r$ clock cycles ensures that the non-deterministic timer never triggers. We show, however, that in practice, the volatile state window is small. For practical attack scenarios, defending against the attack requires the chip to be power cycled once every 16,000–32,000 clock cycles.

To put the performance impact of power cycling every 16,000 clock cycles in perspective, we note that for a processor running at 1 GHz, 16,000 clock cycles corresponds to only 16 μs of time. Putting an Intel mobile processor in sleep state (i.e., powering down the chip) incurs a latency of $30\mu s$ to transition back to active mode [50]. This latency is over and above any additional performance overheads incurred because of the power cycle, for example because of loss of processor state. Thus the performance impact of power cycling every 16,000 clock cycles is prohibitive and makes such a defence infeasible to implement in practice.

The choice of r and k is governed by the following factors:

- The expected trigger time, t, which is, $t = \frac{k}{p} = k2^r$.
- The standard deviation of trigger time, σ_t, which is, $\sigma_t = \frac{\sqrt{k}}{p} = \sqrt{k}2^r$.
- Assuming a binary encoding of the count stored in NV memory, k should be less than the NV memory write durability, w. Therefore, $k \leq w$.
- The volatile state window, t_v. For our implementation, $t_v = 1024r$.

Case Study: A Non-deterministic Timer As a basis for comparison, we study the realization of a non-deterministic timer attack which triggers (in expectation) after one year. As before, we assume an NV memory write durability of 10,000 program/erase cycles. The same specifications were used for the deterministic timer study in Section 3.1.2.

Assuming a 1 GHz system clock, the attack can be implemented using $r = 27$ ($p = 1/2^r$) and $k = 8498$. The attack has an expected trigger time of one year, a standard deviation of ≈ 3.96 days, a volatile state window of only $27.6\mu s$, and requires 14 bits of NV memory. Even if every fabricated chip is tested for 10 days, the probability of the timer triggering during post-fabrication testing is only 10^{-100}. If $100,000$ chips are tested, the probability of at least one timer triggering is still only 10^{-94}.

As we discuss in Section 3.1.3, the same attack implemented with a deterministic timer has a volatile state window of 52 minutes for the same amount of NV writes, or requires at least a 114 MBits of NV memory for the same volatile window.

3.4 Results

We discuss our experimental results obtained from the FPGA prototype of the non-deterministic timer in three parts. We first evaluate the quality of the random bit-streams produced by our TRNG implementation using the NIST test suite. Next, we present data from a series of attacks in which we programmed the non-deterministic timer prototype to trigger after 24-hours. Finally, we discuss the hardware resource utilization of the non-deterministic timer and compare with the resource utilization of a 32-bit RISC processor implemented on the same FPGA.

3.4.1 TRNG

A number of statistical tests have been proposed to quantify the randomness of a bit-stream, including the tests proposed by Knuth [42], the diehard tests [49] and the NIST test suite [64]. The NIST suite makes use of 15 tests, including a number of tests proposed in prior literature, that check various properties of the random bit-stream. Recent work on random number generation using Flash memory [77] has used the NIST test suite for validation, which is what we use in this paper.

The tests are run on a 1 Gbit bit-stream which is divided into 1000 chunks of 1 Mbit each. The NIST test suite performs a hypothesis test — in this case the null hypothesis being that the sequence of bits is indeed drawn from a TRNG — on each chunk and computes the corresponding P-value. The test suite then reports two metrics that we tabulate in Table 3.1: the proportion of chunks that pass the hypothesis test (Proportion), and the uniformity of P-values over all chunks (P-value). A bit-stream passes the NIST tests if the proportion of chunks that pass the test are greater than 90%, and the P-value is greater than 0.0001 for all tests.

NIST Sub-Test	P-Value	Proportion
Frequency	0.536606	0.9904
Block Frequency	0.766441	0.9904
CumulativeSums	0.406370	0.9904
Runs	0.259361	0.9879
Longest Run	0.842729	0.9855
Rank	0.265811	0.9976
FFT	0.662496	0.9759
Non-Overlapping Template	0.799050	0.9855
Overlapping Template	0.380906	0.9928
Universal	0.526790	0.9904
Approximate Entropy	0.450891	0.9928
Random Excursions	0.162606	0.9841
Random Excursions Variant	0.711601	1.0000
Serial	0.402058	0.9855
Linear Complexity	0.707813	0.9976

Table 3.1: NIST test results for TRNG implementation (16 ROs, 3 wide, 1024 decimation). (See Section 3.4.1.)

Our TRNG implementation generates one random bit every 1024 clock cycles. At the 50 MHz FPGA clock, this corresponds to a throughput of 48.8

Kbits/second. For a practical implementation of the attack in a digital IC with a 1 GHz clock, for example, the throughput will be ≈ 1 Mbits/second.

3.4.2 Non-deterministic Timer

We expect real timer based attacks on ICs to have an expected trigger time in the order of months or even years. However, to obtain statistically significant data, we need to repeat the same experiment many times. Therefore, it was impractical for us, given limited FPGA resources, to experiment with month or year long attacks.

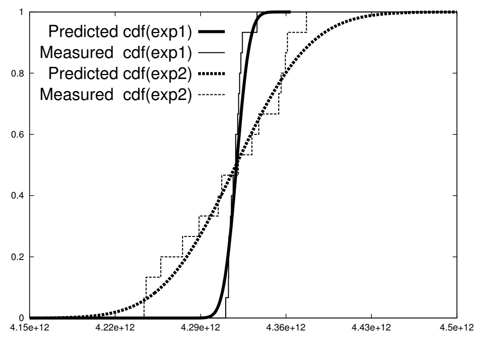


Figure 3.7: Measured and predicted cdfs for the timer experiments.

To validate the proposed attack, we ran experiments in which the expected trigger time is set to 24 hours. To illustrate the ability of the attacker to control pmf and cdf of trigger time, we ran two experiments with the same expected trigger time but different standard deviations. In Experiment 1, the standard deviation is set to 2.84 minutes to emulate a more deterministic attack, while in Experiment 2, the standard deviation is set to 16 minutes. Each experiment is repeated 15 times to provide enough data for a statistical hypothesis test.

The results of these experiments are shown in Figure 3.7 above, and Table A.1 in the Appendix. The hypothesis test results validate that the sample data is consistent with the distributions predicted analytically.

3.4.3 Hardware Resource Utilization

Table 3.2 shows the FPGA resources used by the non-deterministic timer and, for comparison, the resource utilization of an equivalent deterministic

	LEs	REGs	ELC	NV-Bits
TigerMIPS	12379	4578	148548	–
D-Timer	164	15	1983 (1.33%)	104
ND-Timer	99	58	1246 (0.84%)	13

Table 3.2: FPGA resource utilization of an non-deterministic and deterministic timer (ND-Timer and D-Timer respectively), and a 32-bit RISC processor. Resources are measured using logic elements (LE), registers (REG), and equivalent logic cells (ELC), where $ELC = 12 \times LE + REG$.

timer and a 32-bit RISC processor [51]. The number of logic elements (LE) and registers (REG) are obtained directly from FPGA synthesis results, while the equivalent logic cells (ELC) represent an estimate of the gate counts of these designs were they to be implemented in a custom digital IC [15].

The non-deterministic timer utilizes less than 1% of the hardware resources of a simple RISC processor. In fact, the non-deterministic timer also utilizes fewer logic resources (and NV bits) than an equivalent deterministic timer. This is because the D-Timer requires two internal counters, one each to track volatile and NV state, while the non-deterministic timer only requires (besides the RO based TRNG) a comparator for key matching and a single counter for the NV state.

3.4.4 Non- vs. Deterministic Timers

As we discuss in Section 3.1.3, a deterministic timer introduces a trade-off between the size of the NV memory required and the volatile state window which is resolved by the non-deterministic timer. In Figure 3.8, we compare the amount of memory required by the deterministic and non-deterministic timers, both volatile and non-volatile, as a function of the expected trigger time of the attack. We keep the volatile state window the same for both timers; i.e., they have the same susceptibility to a defence based on power cycling. The amount of NV memory required by the deterministic timer increases rapidly as the expected trigger time of the attack increases.

Similarly, Figure 3.9 shows the volatile state window for the deterministic and non-deterministic timers as a function of the expected trigger time of the attack. We keep the size of the NV memory the same for both timers. We observe that the deterministic timer is significantly easier to defend using power cycling compared to the non-deterministic timer.

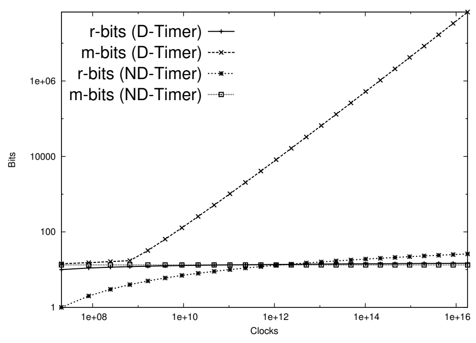


Figure 3.8: The volatile and NV memory sizes, with the same volatile state window, as a function of the expected trigger time for deterministic and non-deterministic timers.

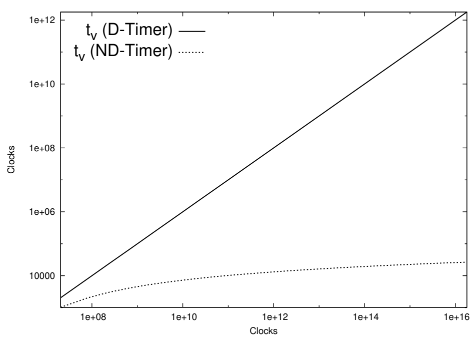


Figure 3.9: The volatile state window, with the same NV memory size, as a function of the expected trigger time for deterministic and non-deterministic timers.

Chapter 4

3D-Security

This chapter first reviews 3D IC manufacturing in Section 4.1, which is a sufficient technology to enable the hiding of wires, then we formalize the specific attack model that k-security defends against in Section 4.2, formally introduce the notion of k-security in Section 4.3, address additional steps needed in the design/fabrication process to preserve k-security in Section 4.4 and finally we review the costs incurred by imposing k-security on a benchmark circuit in Section 4.5.

4.1 3D IC Design and Fabrication

In this section, we overview the IC manufacturing process in the specific context of 3D integration.

Digital ICs consist of a network of inter-connected digital logic gates. Digital logic gates are built using complementary metal-oxide-semiconductor (CMOS) transistors. In a conventional planar/2D IC, CMOS transistors, and by extension digital logic gates, lie in a single layer of silicon. In addition, there are several layers of metal wires used to inter-connect the gates.

3D integration enables the vertical stacking of two or more planar ICs. Each IC in the vertical stack is referred to as a tier. Vertical interconnects (TSVs) are provided to allow the transistors and metal wires in each tier to connect to each other.

The initial motivation for 3D integration came from the potential reduction in the average distance between logic gates — in a 3D IC, the third, vertical dimension can be used to achiever a tighter packing of logic gates [9].

However, a number of issues, including high power density, temperature and cost, have plagued high volume, commercial availability of logic-on-logic 3D ICs [26].

A more practical 3D IC technology that has been demonstrated in a commercial product (a Xilinx FPGA [15]) is shown in Figure 1.1. It consists of two tiers. The bottom tier contains both transistors/gates and metal wires, while the top tier, the interposer, contains only metal wires. The two tiers are interfaced using uniformly spaced metallic bond-points. TSVs make use of these bond-points to provide connections between wires in the top and bottom tiers. This technology has also been referred to as 2.5D integration [27].

Since the bottom tier consists of CMOS transistors, it is fabricated at one of the few foundries worldwide with advanced lithographic capabilities at high cost. The top tier, i.e., the interposer, only contains passive metal and can be fabricated at significantly reduced cost [47].

Figure 4.1 shows a 3D IC design flow with appropriate modifications for security. The design flow begins with the design specified using a hardware description language (HDL), which is then synthesised to a net-list of gates. The types of gates allowed in the gate net-list are specified in a technology library.

In the **wire lifting** stage, the edges (or wires) in the net-list that are to be implemented on the top tier are selected. These are referred to as **lifted wires**. The rest of the net-list, implemented on the bottom tier, is referred to as the **unlifted net-list** and consists of unlifted gates and unlifted wires.

The unlifted gates are then placed on the surface of the bottom tier, i.e., the (x, y) co-ordinates for each gate are selected. Unlifted wires are routed using the bottom tier metal layers. Two bond-points are selected for every lifted wire; one each for the two gates that the wire connects. The gates are connected to the corresponding bond-points. Finally, lifted wires are routed between pairs of bond-points in the top tier using the top tier routing resources.

Finally, the two tiers are fabricated at separate foundries. The chips from the two foundries are vertically stacked to create the final 3D IC chip that is shipped to the vendor.

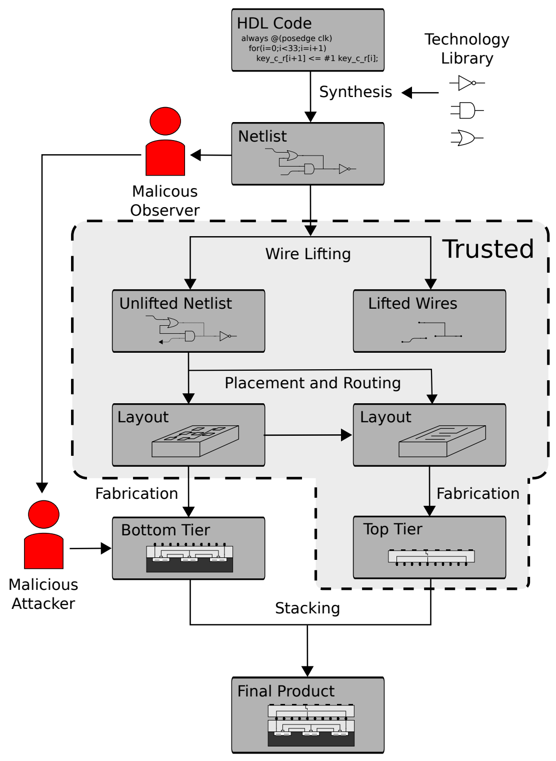

```
always @(posedge clk)
  for(i=0;i<33;i=i+1)
    key_c_r[i+1] <= #1 key_c_r[i];
```

Figure 4.1: Secure 3D IC design and fabrication flow.

4.2 Attack Model

The attack model that we address for our notion of security is that of a **malicious attacker** in the foundry. This attack model has been commonly used in the hardware security literature because of the serious threat it presents [36]. We further strengthen the attack by assuming a **malicious observer** in the design stage, working in collusion with malicious attacker in the foundry[1]. The malicious observer has full knowledge of the circuit as it goes through the design process, but can not effect any changes. The malicious attacker in the foundry can, on the other hand, effect changes in the circuit layout before the chip is fabricated.

To defend against this attack, the following steps of the design and fabrication flow are assumed to be secure, i.e., executed by a trusted party: (a) the wire lifting, placement and routing steps in the design, and (b) the fabrication of the top tier (therefore also referred to as the trusted tier).

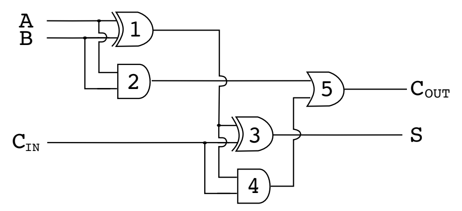
Discussion Two aspects of the attack and defence models deserve further mention. First, we note that the attack model described above subsumes a number of other practically feasible attack models. It is stronger than a malicious attacker in the foundry working by himself. It is also stronger than a malicious attacker in the foundry with partial design knowledge — for example, the attacker is likely to know the functionality and input/output behaviour of circuit he is attacking (an ALU or a DES encryption circuit, *etc.*). Providing the attacker with the precise circuit net-list can only strengthen the attack.

Second, the steps in the design and fabrication process that are assumed to be trusted are also relatively easy to perform in a secure manner, compared to the untrusted steps. Wire lifting and placement/routing (in the design stage) are performed using automated software tools, the former based on algorithms that we propose in this paper, and the latter using commercially available software from electronic design automation (EDA) vendors. In comparison, writing the HDL code is manually intensive, time-consuming and costly. Furthermore, only the top tier is fabricated in a trusted foundry. The top tier only consists of passive metal wires that are inexpensive to fabricated compared to the active CMOS transistors and metal wires in the untrusted, bottom tier.

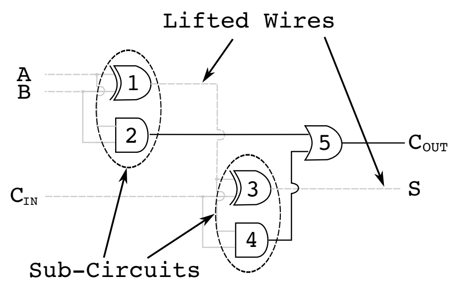
[1]Note that 3D IC based circuit obfuscation cannot, and is not intended to, defend against malicious attackers in the design stage who can alter the HDL or circuit net-list.

4.3 Problem Formulation

In this section, we formulate the circuit obfuscation problem that we address in this paper as a problem in the context of directed graphs. We begin by discussing the example circuit for a full adder that we show in Figure 4.2.



(a) Original Circuit



(b) Obfuscated Circuit

Figure 4.2: Full adder circuit and it's obfuscated version. Grey wires are lifted.

Example As we mention in Section 4.2, in the most powerful attack model we consider, an attacker is in possession of two pieces of information: the originally designed (complete) circuit, and the layout of the circuit that is sent to the foundry for fabrication, which we call the unlifted netlist. The latter results from wires having been lifted from the former. The attacker's objective is to map the latter to the former. Specifically, he seeks a one-to-one bijective mapping of gates in the circuit after wires have been lifted, to the complete circuit.

From the defender's perspective, then, for the circuit in Figure 4.2(a), the question is which wires he should lift. Assume that we lift the wires that are showed greyed out in Figure 4.2(b); that is, the wires $A \rightarrow \{1, 2\}$, $B \rightarrow \{1, 2\}$, $\mathsf{C_{IN}} \rightarrow \{3, 4\}$, $1 \rightarrow \{3, 4\}$ and $3 \rightarrow \mathsf{C_{OUT}}$.

Then, the two possible sub-circuits, one which comprises the gates $\{1, 2\}$, and the other which comprises $\{3, 4\}$ cannot be distinguished from one another based on their connectivity. More specifically, at the level of individual gates, gate 1 cannot be distinguished from 3, and 2 cannot be distinguished from 4. Gate 5, on the other hand, is distinguishable from every other gate as it is the only OR gate in the circuit. We assume also that the inputs and outputs, $A, B, \mathsf{C_{IN}}, \mathsf{C_{OUT}}$ can be distinguished as well.

One may argue that gate 1 is distinguishable from 3 based on the layout (placement). That is, the layout may leak information about the mapping of gates from the unlifted netlist to gates in the original circuit. This is certainly true, and we address this in Section 4.4 with layout-anonymization.

The security of the circuit comes from the indistinguishability property achieved by lifting wires, formally the security level is quantified by the existence of isomorphisms (mappings) between gates in the unlifted netlist and distinct gates in the original netlist. We discuss below the exact notion of security we adopt. Informally, an approach is to simply count the number of such isomorphisms. In our example, we have 4 such mappings, and the attacker is unable to distinguish the correct mapping from the others. In this case, we would quantify the security level as 4. It is possible that in many settings, such a notion of security is effective.

However, the above notion may be seen as too permissive. It can be argued that given the fact that across all mappings, gate 5 is mapped uniquely, we have no security (i.e., security of 1). Indeed, this is the notion of security we adopt — the minimum across all possible mappings of each gate. This notion is more restrictive than the first, and is intended to capture the intuition that the attacker is unable to distinguish even a single gate.

4.3.1 Formulation as a Graph Problem

We now formulate our problem as a graph problem.

Preliminaries A circuit can be perceived as a directed graph — gates are vertices, and wires are edges. The direction of an edge into or out of a vertex indicates whether it is an input or output wire to the gate that corresponds to the vertex. If G is a graph, we denote its set of vertices as $V[G]$, and its set of edges as $E[G]$. Each vertex in the graph is associated with a color that is used to distinguish types of gates (e.g, AND and OR) from one another. Consequently, a graph G is a 3-tuple, $\langle V, E, c\rangle$, where V is the set of vertices, E the set of edges and the function $c\colon V \to \mathbb{N}$ maps each vertex to a natural number that denotes its color. Given a graph G, we sometimes denote its set of vertices, edges and colouring function as $V[G], E[G]$ and $c[G]$ respectively.

For example, the circuit in Figure 4.2 and its unlifted portion can be represented by the graphs in Figure 4.3.

Intuition If we perceive the circuits as graphs, then our underlying prob-

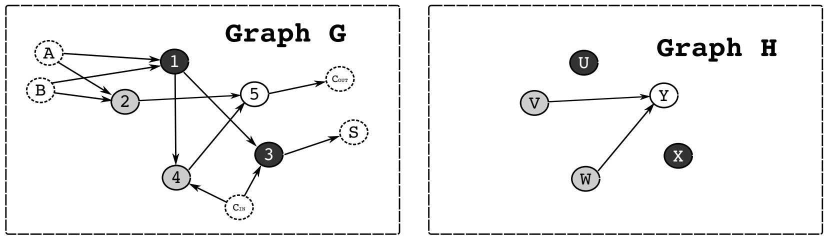


Figure 4.3: Full adder graphs: G is the full graph representation of the full adder circuit, H is the remaining graph after wires have been lifted.

lem corresponds to a kind of subgraph isomorphism. Given two graphs $G_1 = \langle V_1, E_1, c_1 \rangle, G_2 = \langle V_2, E_2, c_2 \rangle$, we say that G_1 is isomorphic to G_2 if there exists a one-to-one bijective mapping $\phi \colon V_1 \to V_2$ such that $\langle u, v \rangle \in E_1$ if and only if $\langle \phi(u), \phi(v) \rangle \in E_2$ and $c_1(u) = c_2(\phi(u)), c_1(v) = c_2(\phi(v))$. That is, if we rename the vertices in G_1 according to ϕ, we get G_2. A specific such mapping ϕ is called an isomorphism.

We say that $G_1 = \langle V_1, E_1, c_1 \rangle$ is a subgraph of $G_2 = \langle V_2, E_2, c_2 \rangle$ if $V_1 \subseteq V_2$, and $\langle u, v \rangle \in E_1$ only if $\langle u, v \rangle \in E_2$. We say that G_1 is subgraph isomorphic to G_2 if a subgraph of G_1 is isomorphic to G_2. For example, in Figure 4.3, a candidate subgraph isomorphism, ϕ, is $\phi(1) = U, \phi(2) = V, \phi(3) = X, \phi(4) = W, \phi(5) = Y$.

Let G be the graph that represents the original circuit with all wires, and H the graph of the circuit after wires have been lifted. Then, the attacker knows that G is subgraph isomorphic to H. What he seeks is the correct mapping of vertices in G to H (or vice versa). This is equivalent to him having reconstructed the circuit, and now, he can effect his malicious modifications to the circuit that corresponds to H.

From the defender's standpoint, therefore, what we seek intuitively is that there be several subgraph isomorphisms between G and H. As we mention in Section 1, this then gives the kind of security in a k-anonymity sense — the attacker cannot be sure which of the mappings is the correct one, and therefore is able to reconstruct the circuit with probability $1/k$ only. As we mention there and discuss in more detail in the related work Section, though our notion of security has similarities to k-anonymity, there are important differences, and we call it k-security instead.

There is a cost to security, which is the cost of removing edges (i.e., lifting wires). We assume that there is some function that, given a set of edges, outputs the cost.

Overall problem The overall problem of a defender, then, is to construct a suitable H from G that gives him a k that is sufficiently large, while incurring a cost of at most η. That is, he seeks to identify a set of edges $E' \subseteq E[G]$ such that $\alpha(G, E') \leq \eta$ and $\sigma(G, E') \geq k$, where α is the cost function, σ is the function that computes security, and η and k are this cost and security thresholds, respectively. Henceforth, we use the acronym DAG for Directed Acyclic Graph.

Definition 1 (Lifting)**.** *Lifting* $= \{\langle G, \eta, k\rangle : G$ *is a DAG, and there exists* $E' \subseteq E[G]$ *such that* $\alpha(G, E') \leq \eta$ *and* $\sigma(G, E') \geq k\}$*.*

The above definition defines a language that corresponds to a decision problem that we seek. The decision problem Lifting is posed as a problem of existence — whether there exists a set of edges E' that can be lifted. We assume access to oracles that compute α and σ. We assume that α can be computed efficiently; we adopt the concrete function $\alpha(G, E') = |E'|$ in our approach in Section 4.4. We discuss the computation of σ below.

We point out that if Lifting can be solved efficiently, given access to oracles that compute α and σ, we can identify a set of edges E' to be lifted efficiently as well. That is, given oracles that decide Lifting, and compute α and σ, an edge-set to be lifted can be computed in polynomial-time. The reason is that Lifting $\in$ **NP** if we assume oracle access to α, σ, and an E' is a certificate for an instance in Lifting.

Our overall problem, then is to devise a decision procedure for Lifting. To do so, we first discuss how we instantiate σ below, and then consider the computational complexity of deciding Lifting.

k-security We now specify our notion of security. As we mention above, it is related to subgraph isomorphism. We begin by specifying a subgraph isomorphism problem that is relevant to our context.

Definition 2 (Circuit-Subiso)**.** *Circuit-Subiso* $= \{\langle G, H, u, v\rangle : G$ *and* H *are DAGs such that* $|V[G]| = |V[H]|$, $u \in V[G]$, $v \in V[H]$, *and there exists* $R \subseteq E[G]$ *and a mapping* $\phi \colon V[G] \to V[H]$ *such that* ϕ *is an isomorphism for the graph* $\langle V[G], E[G] - R, c[G]\rangle$ *to* H, *and* $\phi(u) = v\}$*.*

The above definition is a special case of the well known subgraph isomorphism problem [30]. Circuit-Subiso expresses the decision problem with input two DAGs G, H and two vertices $u \in V[G]$ and $v \in V[H]$. We need to determine whether that input has the properties the above definition expresses: G and H have the same number of vertices, and if we remove some edges from G, we have a graph that is isomorphic to H such that under that isomorphism, u is mapped to v.

Based on Circuit-Subiso above, we are now able to specify our notion of k-security.

Definition 3 (k-security)**.** *Given a DAG G, $u \in V[G]$ and $E' \subseteq E[G]$, let the set $S = \{\langle G, H, u, v\rangle : H = \langle V[G], E[G] - E', c[G]\rangle, v \in V[G]$ and $\langle G, H, u, v\rangle \in$ Circuit-Subiso$\}$.*

We say that $\langle G, E'\rangle$ is k-secure if $\min\limits_{u \in V[G]} |S| \geq k$.

As we indicate above, we denote as $\sigma(G, E')$ the maximum k-security we are able to achieve with G, E'.

Given G, u, E', the set S in the above definition identifies those vertices v from which u can map in a subgraph isomorphism. For every u, there is at least one such v, which is u itself. The question is whether other v's exist, with which we can confuse the attacker in his attempts to distinguish u. With the minimization, we take the smallest number of such v's across all u's (vertices) in G.

In Figure 4.3, for example, we know that $\min |S|$ evaluates to 1, because node 5 can be mapped to itself only. The corresponding sizes of the sets S for nodes 1, 2, 3 and 4, however, are 2. The reason is that each can be mapped either to itself, or to another node.

Computational complexity We now consider the computational complexity of computing the maximum k-security, σ, and of deciding Lifting. For the former, we consider a corresponding decision problem, k-Security-Dec $= \{\langle G, E', k\rangle : G$ is a DAG, $E' \subseteq E[G]$ and $k \in [1, |V[G]|]$, such that $\langle G, E'\rangle$ is k-secure$\}$. That is, k-Security-Dec is the problem of determining whether lifting E' from G gives us a k-security of k.

We point out that if we have an oracle that decides k-Security-Dec, then we can compute the maximum k-security we can get by lifting E' from G using binary search on k. That is, the problem of computing σ is easy if deciding k-Security-Dec is easy.

Theorem 1. *k-Security-Dec $\in$ **NP**-complete.*

To prove the above theorem, we need to show that k-Security-Dec is in **NP**, and that it is **NP**-hard. For the former, we need to present an efficiently (polynomial-size) certificate that can be verified efficiently. Such a certificate is k mappings each of which is a sub-isomorphism, for each pair of vertices $u, v \in V[G]$. Each such mapping can be encoded with size $O(|V[G]|)$, and there are $k|V[G]|^2$ such mappings, and therefore the certificate is efficiently-sized. The verification algorithm simply checks that each mapping is indeed a sub-isomorphism, which can be done efficiently.

To show that k-Security-Dec is **NP**-hard, it suffices that we show that Circuit-Subiso is **NP**-hard. We present the proof in the Appendix with a reduction from a special case of subgraph isomorphism that is known to be **NP**-complete.

As for Lifting, in our earlier discussions, we assumed oracle access to the cost-function for lifting edges, α and the function for computing k-security, σ. As we mention above, we assume that α is efficiently computable; a simple choice is $\alpha(G, E') = |E'|$. That is, the cost of lifting a set of n edges is n. Other choices are also possible, and leave an investigation into more meaningful choices for α as future work.

To compute σ efficiently, we need access to an oracle that can decide problems in **NP**, because k-Security-Dec is **NP**-complete. Apart from the difficulty of determining σ given a set E', we have also the problem of determining an appropriate candidate set E' to decide Lifting. Without assuming oracle access to σ, we also observe that Lifting $\in$ **NP**. The certificate is simply the set of wires to remove E' and the certificate used for k-Security-Dec, which with the addition of E' can still be efficiently encoded with size $O(|V[G]|^2)$ and thus is still efficiently sized.

We suspect that Lifting is **NP**-complete but we leave this proof as topic for future work. In fact we previously believed Lifting was not in **NP** which explains why the rest of this work treats these two languages as separate problems.

4.4 Approach

Having considered the computational complexity of the problem that underlies our work in the previous section, in this section, we propose a concrete approach for it. As our discussions in the prior section reveal, there are two parts to the solution: (a) computing the maximum k-security for $\langle G, E' \rangle$, given the graph G that represents the complete circuit, and, (b) choosing the set E'.

We propose an approach for each in this section. For the problem of computing security, we employ constraint-solving. We discuss this in Section 4.4.1. For the problem of choosing E', we propose a greedy heuristic. We discuss that in Section 4.4.2. We conclude with some practical considerations, specifically, scalability and layout-anonymization in Section 4.4.3 .

4.4.1 Computing Security

As shown in Section 4.3, the problem of determining the security level of circuit G, given the unlifted netlist H is **NP**-complete. Given the relationship of the problem to subgraph isomorphism, a natural approach to solving this problem would be to use graph (sub)isomorphism algorithms proposed in literature — of these, the VF2 algorithm [24] has been empirically shown to be the most promising [29]. However, in our experience, VF2 does not scale for circuits with > 50 gates (more on scalability in Section 4.4.3).

Instead, motivated by the recent advances in the speed and efficiency of SAT solvers, we reduce the sub-isomorphism problem to a SAT instance and use an off-the-shelf SAT solver to decide the instance.

Reduction to SAT Given graphs G and H, we define a bijective mapping ϕ from the vertex set of H to the vertex set of G as follows: Boolean variable ϕ_{ij} is true if and only if vertex $q_i \in H$ maps to a vertex $r_j \in G$. Here $V[G] = \{r_1, r_2, \ldots, r_{|V[G]|}\}$ and $V[H] = \{q_1, q_2, \ldots, q_{|V[H]|}\}$

We now construct a Boolean formula that is true if and only if graphs G and H are sub-isomorphic for the mapping ϕ. We will construct the formula in parts.

First, we ensure that each vertex in G maps to only vertex in H:

$$F_1 = \prod_{i}^{|V[H]|} \sum_{j}^{|V[G]|} \left(\phi_{i,j} \prod_{k \neq j}^{|V[G]|} \neg \phi_{i,k} \right)$$

and vice-versa:

$$F_2 = \prod_{j}^{|V[G]|} \sum_{i}^{|V[H]|} \left(\phi_{i,j} \prod_{k \neq i}^{|V[H]|} \neg \phi_{k,j} \right)$$

Finally we need to ensure that each edge in H maps to an edge in G. Let $E[H] = \{e_1, e_2, \ldots, e_{|E[H]|}\}$ and $E[G] = \{f_1, f_2, \ldots, f_{|E[G]|}\}$. Furthermore, let $e_k = \langle r_{src(e_k)}, r_{dest(e_k)} \rangle \in E[H]$ and $f_k = \langle q_{src(f_k)}, q_{dest(f_k)} \rangle \in E[G]$. This condition can be expressed as follows:

$$F_3 = \prod_{k}^{|E[H]|} \sum_{l}^{|E[G]|} \phi_{src(e_k),src(f_l)} \wedge \phi_{dest(e_k),dest(f_l)}$$

The formula F that is input to the SAT solver is then expressed as a conjunction of the three formulae above: $F = F_1 \wedge F_2 \wedge F_3$. The formula F has $O(|V[H]||V[G]|)$ variables and $O(|E[H]||E[G]|)$ clauses.

4.4.2 Wire Lifting Procedure

To determine a candidate set of edges, E', to lift, we employ a greedy heuristic. Our heuristic is shown as Algorithm 1.

```
$E' \leftarrow E[G]$;
while $|E'| > 0$ do
    $s \leftarrow 0$;
    foreach $e \in E'$ do
        $E' \leftarrow E' - \{e\}$;
        if $\sigma(G, E') > s$ then
            $s \leftarrow \sigma(G, E')$;
            $e_b \leftarrow e$;
        $E' \leftarrow E' \cup \{e\}$;
    if $s < k$ then return $E'$;
    $E' \leftarrow E' - \{e_b\}$;
return $E'$;
```

Algorithm 1: lift_wires(G, k)

In our heuristic, we begin with the best security we can achieve. This occurs when we lift every edge in $E[G]$; that is, we set E' to $E[G]$ at the start in Line 1. We then progressively try to remove edges from E'. We do this if not lifting a particular edge e still gives us sufficient security.

That is, we iterate while we still have candidate edges to add back (Line 2). If we do, we identify the "best" edge that we can add back. We identify this as e_b. The best edge is the one that gives us the best security level if removed from E'. If even the best edge e_b cannot be removed from E', then we are done (Line 10).

The heuristic does not necessarily yield an optimal set of edges. The reason is that we may greedily remove an edge e_1 from E' in an iteration of the above algorithm. And in later iterations, we may be unable to remove edges e_2 and e_3. Whereas if we had left e_1 in E', we may have been able to remove both e_2 and e_3. Note that removing as many edges from E' is good, because our cost is monotonic in the size of E' (set of edges being lifted).

4.4.3 Practical Considerations

From a graph-theoretic perspective, the wire lifting procedure outlined provides a set of wires to lift that guarantees a certain security level. However, two practical consideration merit further mention — the scalability of the proposed techniques to "large" circuits, and the security implication of the attacker having access to the layout of H, as opposed to just the netlist.

Scalability Although the SAT based technique for computing security scales better than the VF2 algorithm, we empirically observe that it times out for circuits with > 1000 gates. To address this issue, we propose a circuit partitioning approach that scales our technique to larger circuits of practical interest. We note that circuit partitioning is, in fact, a commonly used technique to address the scalability issue for a large number of automated circuit design problems.

Algorithm 2 is a simplified description of the partitioning based wire lifting procedure. The function *partition*(G) recursively partitions the vertex set of the graph into P mutually exclusive subsets and returns subgraphs $\{G_1, G_2, \ldots, G_P\}$ of size such that they can be tractably solved by the SAT based greedy wire lifting procedure. The final set of lifted wires includes the union of all wires that cross partitions, and those returned by P calls to Algorithm 1. We have used this technique to lift wires from circuits with as many as 35000 gates (see Section 5.2.1).

```
$\{G_1, G_2, \ldots, G_P\} \leftarrow partition(G)$
$E_R \leftarrow E - \bigcup_{i\in[1,P]} E_i$
for $i \in [1, P]$ do
    $E_R \leftarrow E_R \bigcup lift\_wires(G_i, s_{req})$
return $E_R$
```

Algorithm 2: lift_wires_big(G, s_{req})

Layout anonymization We have, so far, assumed that the unlifted circuit H is a net-list corresponding to the unlifted gates and wires. However, in practice, the attacker observes a layout corresponding to H, from which he reconstructs the net-list of H. We therefore need to ensure that the layout does not reveal any other information to the attacker besides the reconstructed net-list.

Existing commercial layout tools place gates on the chip surface so as to minimize the average distance, typically measured as the Manhattan distance, between all connected gates in the circuit netlist. Thus, if the complete circuit G is used to place gates, the physical proximity of gates will reveal some information about lifted wires — gates that are closer in the bottom tier are more likely to be connected in the top tier. The attacker can use this information to his advantage.

Instead of using the net-list G to place gates, we instead use the net-list H. Since this net-list does not contain *any* lifted wires, these wires do not have any impact on the resulting placement. Conversely, we expect the physical proximity of gates to reveal no information about hidden wires in the top tier. In Section 4.5, we empirically validate this fact. However, anonymizing the layout with respect to the hidden wires does result in increased wire-length between gates, which has an impact on circuit performance. This impact is also quantified in Section 4.5.

4.5 Results

We conduct our experimental study to investigate security-cost trade-offs obtained from the proposed techniques. All experimental results are obtained using the c432 circuit from the ISCAS-85 benchmark suite [17] (a 27-channel bus interrupt controller) which has ≈ 200 gates which is physically implemented with IBM 0.13μ technology. For 3D integration, bond points are assumed to be spaced at a pitch of $4\mu m$, allowing for one bond-point per $16\mu m^2$. This is consistent with the design rules specified in the Tezzaron $0.13\mu m$ technology kit.

Circuit synthesis was performed using the Berkeley SIS tool [60]. Placement and routing is performed using Cadence Encounter. Finally, we used miniSAT as our SAT solver [63].

4.5.1 Security-Cost Tradeoffs

Figure 4.4 graphs the security level for the c432 circuit as a function of $E[H]$, the number of unlifted wires in the untrusted tier. $E[H] = 0$ corresponds to a scenario in which *all* wires are lifted, while $E[H] = E[G]$ corresponds to a case in which all wires are implemented in the untrusted tier.

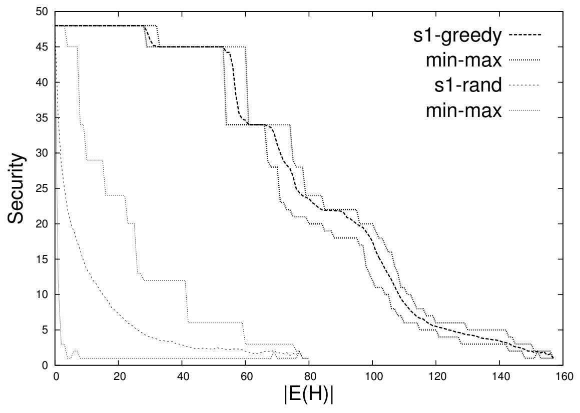


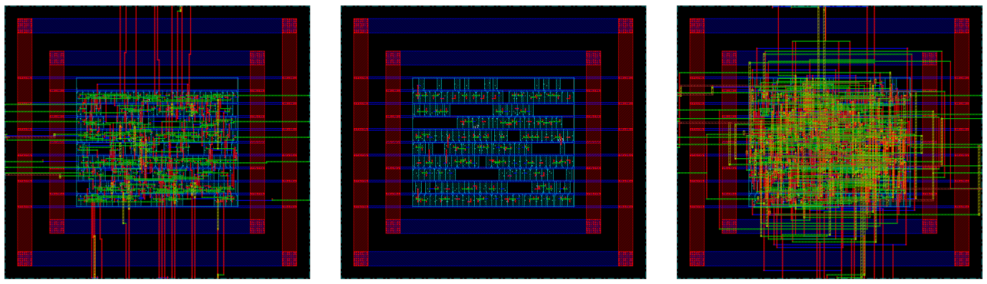
Figure 4.4: Maximum, average and minimum security levels for the c432 circuit using the proposed greedy wire lifting procedure and random wire lifting.

Proposed Vs. Random Wire Lifting Figure 4.4 compares the proposed greedy wire lifting technique with a baseline technique in which wires are lifted at random. In both cases, we show the maximum, average and minimum security achieved by these techniques over all runs.

Observe that greedy wire lifting provides significantly greater security compared to random wire lifting. With 80 unlifted wires, the greedy solution results in a 23-secure circuit, while *all* random trials resulted in 1-secure (equivalently, completely insecure) circuits.

Number of Lifted Edges Vs. Security Figure 4.4 reveals that, for c432, at least 145 of the 303 ($\approx 47\%$) wires must be lifted to get any meaningful degree of security. If any fewer wires are lifted, circuit obfuscation provides no security at all. However, once more than this minimum number of wires is lifted, the security offered increases quite rapidly.

Another observation that merits mention are the plateaus in security level, for example between $E[H] = 30$ and $E[H] = 55$. In other words, in some



(a) Original Circuit (b) Bottom Tier of 8-Secure Circuit (c) Top Tier of 8-Secure Circuit

Figure 4.5: Layout of c432 without any lifting (left), and the bottom (middle) and top (right) tiers of an 8-secure version of c432. Green and red lines correspond to metal wires.

cases, wires can be retained in the untrusted tier without any degradation in security.

Impact of Layout Anonymization Figure 4.5 shows three layouts for the c432 circuit. The far left corresponds to the original 1-secure c432 circuit without any wire lifting. The other two layouts correspond to the top and bottom tiers of an 8-secure version of c432 with $\approx$ 66% lifted wires. Of particular interest is the wire routing in the trusted top tier — because the placement of the corresponding gates in the untrusted bottom tier have been anonymized, the lifted wires are routed seemingly randomly. This is in stark contrast to the wire routing in the original circuit that is far more structured.

Figure 4.6 shows the histogram of wire lengths for the three layouts shown in Figure 4.5. Note that, in the original 1-secure circuit, a large majority of wires are short; in other words, connected gates are placed closer together. Wire lengths on the bottom untrusted tier of the 8-secure circuit also skew towards shorter values — however, these wires are already observable to the attacker and he gains no additional information from their lengths. On the other hand, the wire length distribution of the top tier is more evenly spread out. This reflects that fact that the physical proximity of gates in the bottom tier reveals very little information about the lifted wires. Note that the the distribution of wire lengths in the top tier reflects the fact that the distribution of Manhattan distance between randomly placed points on a plane is triangular.

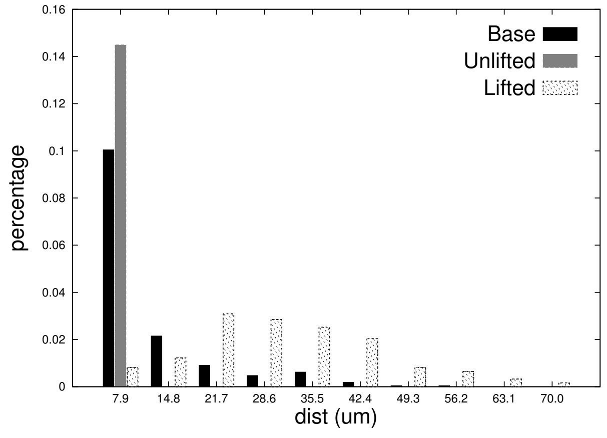


Figure 4.6: Comparison of the c432 circuit wire lengths the original 1-secure circuit and the bottom and top tiers of the 8-secure circuit.

Area, Delay and Power Cost Area, delay (inversely proportional to clock frequency) and power consumption are important metrics of circuit performance. 3D integration based circuit obfuscation introduces overheads on all three metrics.

The area of a 3D circuit is determined by the larger of two areas: the area consumed by the standard cells in the bottom tier, and the area consumed by the bond-points required to lift wires to the top tier. The bond-point density is limited by technology (1 bond-point per $16\mu m^2$ in our case) and therefore more lifted wires correspond to increased area.

Delay and power are strong functions of wire length, since increased wire length results in increased wire parasitics (capacitance and resistance). Layout anonymization results in increased wire length as we have already noted.

Table 4.1 shows the area, power and delay for the c432 circuit for different security levels. Compared to the original circuit, the 8-secure circuit has 1.6× more power consumption, 1.8× larger delay, and about 3× more area.

Choice of Technology Library The technology library determines the type of gates that are allowed in the circuit netlist. Diverse technology libraries with many different gate types allow for more optimization, but also hurt security. Figure 4.7 shows the security levels achievable for c432 for five different technology libraries with between three and seven gates.

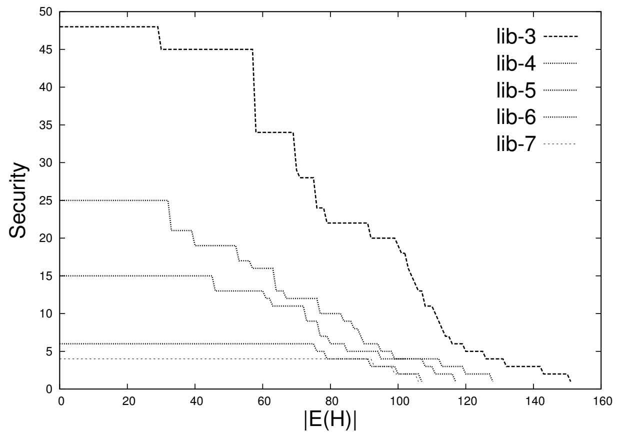


Figure 4.7: Comparison of the obtainable security levels for the c432 circuit mapped with different technology libraries.

Table 4.1: Power, delay, wire length and area analysis for different levels of security on the c432 circuit. 1* is the base circuit with no wires lifted and 48* has all of the wires lifted.

Security	Power Ratio	Delay Ratio	Total Wire Length (μm)	Total Area (μm^2)
1*	1.00	1.00	2739	1621
2	1.54	1.73	6574	4336
4	1.55	1.76	7050	4416
8	1.61	1.82	8084	4976
16	1.62	1.86	8161	5248
24	1.71	1.98	9476	6048
32	1.73	1.99	9836	6368
48*	1.92	2.14	13058	8144

Table 4.2: Technology libraries used for the experiment in Figure 4.7. lib-x corresponds to a library with x different gate types.

Library	max(S_1)	$\|V(G)\|$	$\|E(G)\|$
lib-3	48	209	303
lib-4	24	181	271
lib-5	13	169	259
lib-6	7	165	252
lib-7	4	159	246

Chapter 5

Discussion

In this chapter we discuss alternative countermeasures, costs, precautions and notions of security with respect to our contributions.

5.1 Non-Deterministic Timer

As mentioned before, countermeasures to detect or disable malicious circuits can be deployed during both pre- and post-fabrication testing and validation. We discuss our attack in the context of these countermeasures.

5.1.1 Pre-fabrication Simulation and Validation

Pre-fabrication simulations or, when possible, formal verification can be used to detect differences between a circuit and its behavioral specification. However, logic simulation tools only follow discrete event semantics and do not model physical phenomena such as noise. From the perspective of the proposed non-deterministic timer attack, this is advantageous because the output of the TRNG remains steady at logic zero for the entire length of the simulation never resulting in a key match (see Figure 5.1). Thus, the attack will never trigger in simulation, even if the circuit is simulated for any number of clock cycles.

Deterministic timer attacks do not have the same distinguishing feature — the deterministic timer attack does trigger in simulation, if the circuit is simulated for a sufficiently large number of clock cycles.

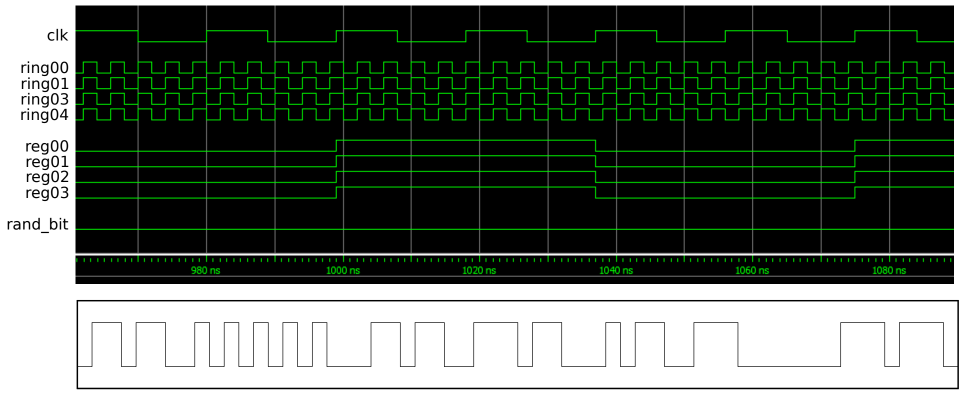


Figure 5.1: A test simulation done in Modelsim is shown as the top diagram for a sample non-deterministic -timer using four ring oscillators `ringXX`. Note how all the ring oscillators are in phase without any clock jitter. All the ring sampling registers `regXX` are also in phase and thus the decimation of a even number of rings will always be $0 =$ `rand_bit`. The bottom diagram shows a typical pattern for `rand_bit` gathered from the FPGA itself.

5.1.2 Logic Signature Detection

Hardware attacks that exhibit unique signatures during logic simulations not exhibited by regular circuits might be more susceptible to detection during pre-fabrication logic simulations. For example, the UCI approach [33] attempts to identify unused circuits, and therefore potentially malicious, circuits in the netlist by flagging pairs of wires that have an input output relationship, but always hold the same logic value in simulation.

UCI has been defeated [65], and the technique in that work can be used by us as well to defeat UCI. Broadly, the technique is to identify an equivalent circuit that is not identified by UCI as unused. An alternative approach is to modify the functionality of the attack to perform a similar or equivalent function that avoids the signature detection scheme. In the case of our non-deterministic timer, we are able to modify the non-random behaviour in simulation while still preserving the random behaviour at run time. This allows us to defeat the UCI logic signature based technique as we indicate in Figure 5.2.

As the figure indicates, we can have the TRNG output independent random values in each sample, a dynamic key can be used to match against

```
case(state)
      ⋮
  SAMPLE :
  begin
    key = count[7:0];
    if (rand[7:0] == key)
    begin
      count <= count + 1;
    end
    state <= DECIMATE;
  end
```

```
case(state)
    ⋮
  SAMPLE :
  begin
    if (count < 5 || key == 0)
        key <= key + 1;
    if (rand[7:0] == key)
    begin
      count <= count + 1;
    end
    state <= DECIMATE;
  end
```

Figure 5.2: Two possible dynamic choices are shown for the key. The current value of the count (left) and some function of the count and the current value of the key (right). Adopting a non-constant key defeats logic-signature based countermeasures such as UCI.

the random TRNG output instead of a fixed, static key. The dynamic key sequence can be arbitrary and has no impact on the statistics of the trigger time when deployed in the field.

We can see the key as being generated by a (simple) Finite State Machine (FSM). It may use the current NV state (counter) as input, as we do in the examples in Figure 5.2. Any FSM that results in at least one and at most $K-1$ key matches with the TRNG output *in simulation* (in this case, when the key value equals 0) is a candidate implementation and results in between one to $K-1$ updates to NV state. Thus, the circuit is no longer dormant in simulation. FSM obfuscation techniques [44] can be used to further disguise the attack.

The resulting timer never triggers during simulations (as before), is undetected by dormant/unused circuit identification techniques like UCI, can exhibit a wide range of logic signatures, and has exactly the same statistical properties as the original attack shown in Figure 3.3 when deployed in the field.

In Figure 5.2, we show two possible alternatives that the key is chosen — a simple function of the current count that is maintained in the the NV memory, and a simple function of the count and the current key. The reason for adopting a non-constant key is to defeat logic-signature based countermeasures such as UCI [33].

The simulation that UCI performs will see the timer circuitry as active, and not unused, and therefore will not flag it as potentially malicious. The reason we present two non-constant possibilities for the key is that the two have different logic signatures, and therefore, more general than UCI, logic-signature based countermeasures are defeated.

5.1.3 Hardware Signature Detection

The digital logic blocks used by the non-deterministic timer, i.e., comparators, adders and FSMs, are commonly used in digital ICs and are unlikely to be flagged as malicious circuitry. In addition to digital logic, the non-deterministic timer uses two other components — ROs and NV memory.

Due to their versatility and simplicity, ROs are commonly used in digital ICs for a variety of purposes which include on-chip monitoring of temperature [39], process [13] and device aging [40], implementing replica timing circuits [48], and random number generation for cryptographic primitives and security protocols [18]. In the context of replica timing circuits, for example, there would be one RO for every critical path on the chip, resulting in thousands of on-chip ROs. Our work illustrates that these ubiquitous circuit blocks can be exploited for malicious intent, and raises the bar on test and validation teams to verify the functionality of each RO. However we note that ROs are not the only source of randomness that can be exploited which in turn further raises the bar for signature detection.

The NV memory can be inserted either post-fabrication by a malicious foundry working in collusion with a malicious designer (the attack model assumed by Waksman and Sethumadhavan [75]), or in the design phase if a CMOS compatible NV memory technology is used. In the former case, destructive chip testing can reveal the presence of NV memory, but this is a time consuming and expensive process that can take up to a week and cost up to $\$250,000$ per chip [45].

For chips in which NV memory is directly instantiated in the design phase, an attacker can either attempt to make use of existing benign NV bits or add additional bits for malicious purposes. In this context, it is interesting to note that the first known instance of malicious circuitry detected in a commercial product exploited the available NV registers in a military grade FPGA to implement its attack [61].

5.1.4 IC Fingerprinting

IC fingerprinting [3, 6, 38, 54, 78, 80] appears to offer the most promise in detecting our kind of timer. However, the small footprint may pose a challenge to such detection techniques.

As we discuss in Section 3.4, the resource utilization of an non-deterministic timer attack is lower than that of an equivalent deterministic timer attack, both in terms of logic gates and and number of NV bits used. The non-deterministic timer uses less than 1% of the logic resources used by a simple RISC processor — modern day digital ICs can consist of tens or even hundreds of such processors. Finally, the estimated gate count of the non-deterministic timer attack, 1246, compares well to the "stealthy" attack proposed by King et al. [35] that used 1341 logic gates. Nonetheless, in theory at least, none of the malicious hardware attacks of which we are aware is entirely immune to IC fingerprinting. We leave a thorough examination of the practicality of such techniques to low-footprint malicious circuits such as ours as a topic for future work.

5.1.5 Cryogenic Operation

Cooling a chip to low temperatures reduces thermal and phase noise (our source of randomness), potentially preventing the timer from triggering. Cryogenic cooling of ICs to temperatures as low as 4 K has been demonstrated for specialized applications where cost is not a concern [31]. However, the cost of cryogenic cooling in the field is prohibitive and thus impractical as a defence mechanism for consumer ICs. Furthermore, cryogenic testing during the post-fabrication validation and testing phase is not useful for early detection since the attack is disabled at low temperatures.

5.2 k-security

5.2.1 Case Study: DES Circuit

We use a DES encryption circuit obtained from `opencores.org` to demonstrate that applicability of our techniques, including circuit partitioning based wire lifting, to larger circuits. The DES circuit takes as input a fixed-length string of plain-text and transforms the string into cipher text using 16 rounds of obfuscation, as shown in the block-level circuit diagram in Figure 5.3.

The original, 1-secure implementation of DES that we synthesized has ≈ 35000 logic gates, which results in an intractable SAT instance. However, using recursive circuit partitioning, we are able to lift wires to obtain a 64-secure implementation. We note that a security level of 16 is obtained in the first few rounds of partitioning by removing only 13% of the wires, i.e., all wires that lie between successive DES rounds. This is because the circuit description of each DES round is identical — thus, once the wires between the rounds have been removed, each round can be confused for any other round. The final 64-secure implementation has only 30% of the wires unlifted, and consumes $2.38\times$ the area of the original 1-secure circuit.

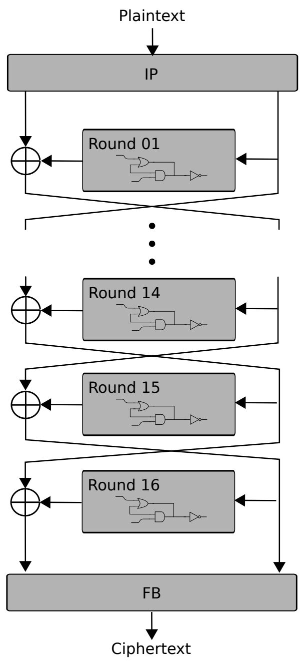


Figure 5.3: Block diagram of the DES encryption circuit.

Attack Scenario Boneh et al. [16] have shown that specific bits in a DES implementation are particularly susceptible to fault attacks. For example, if the attacker is able to insert an attack such that the LSB output of the 14^{th} round is stuck at logic zero, the secret key can be recovered using as few as two messages.

Figure 5.4 shows how such an attack might be effected using a trigger (we do not address here how this trigger may be activated) and three additional gates in an insecure (or 1-secure) circuit. When the trigger is set, the output is set to zero, but is equal to the correct value when the trigger is at logic zero.

Now, assume that wire lifting is performed to make the circuit 64-secure. Given the set of lifted wires, we note that the LSB of the 14^{th} round is, in fact, 256-secure, i.e., there are 255 other gates in the circuit that are indistinguishable from the LSB of the 14^{th} round.

The attacker now has two choices. he can either attack one of the 256 options, which will only succeed for 1 out of 256 choices, or he can choose to carry out a multiplexed attack on all 256 gates. This is shown in Figure 5.4. In this attack, the trigger transmits a sequence of 8-bits that identify which of the 256 signals the attacker wants to attack. These 8-bits feed an 8:256 demultiplexer that generates individual triggers for each of the 256 signals that are indistinguishable.

The attacker can now iteratively insert attacks in each gate one at a time and conceivably determine which iteration actually corresponds to the LSB of the 14^{th} round. However, in doing so, the attacker incurs two costs: (i) the modified attack circuit now requires 1280 gates instead of just 3, a $420\times$ overhead; (ii) the attacker would require, in the worst case an extra 255 messages to recover the key.

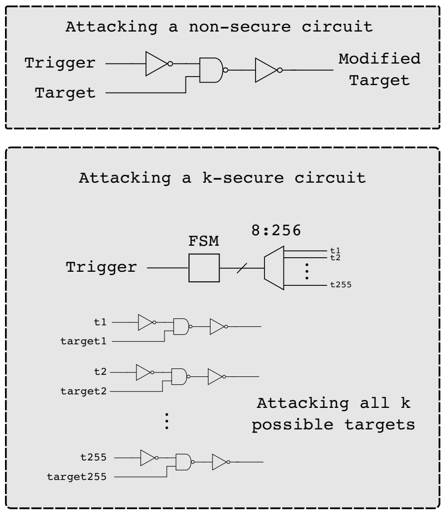

Attacking a non-secure circuit
Trigger
Target
Modified Target
Attacking a k-secure circuit
FSM
8:256
Trigger
t1
t2
t255
target1
target2
target255
Attacking all k possible targets

Figure 5.4: Attack scenarios of 1-secure and k-secure circuits.

5.2.2 Hiding Wire Selections

As stated in Section 4.1 the selection process of the hidden wire set needs to be secure. Specifically in some instances if the adversary were to know which wires were lifted, the obfuscation of the gates may be compromised even though the circuit remains k-secure. As an example revealing the hidden wire set for the graph in Figure 5.5(a) would allow for the proper identification of vertex 1 and 2. However knowing the removed wire set for the graph in Figure 5.5(b) would not compromise the k-security of two since vertex 1 can be confused with vertex 3 and vertex 2 can be confused with vertex 4.

A valid question arises from the use of a greedy algorithm: can an adversary derive any useful information about which wires are lifted by having knowledge about the greedy selection process? We do not have the answer to that exact question for Algorithm 1, since there is a best order as to how the edges are chosen. However a simple modification to the algorithm will randomly select wires to add to the non-lifted set if k-security is still preserved. This results of this algorithm agree with the results obtained in Section 4.5 but due to the random approach does not reveal any useful information about the selection process.

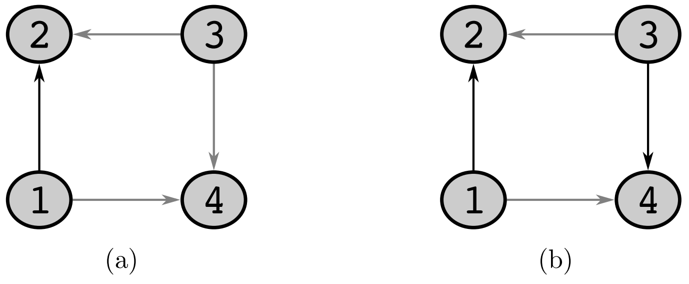


Figure 5.5: Revealing the removed edge set (grey edges) would comprise k-security for the left graph but not the right graph.

5.2.3 An Alternative Notion of Security

An alternative approach that the adversary could take is to find one isomorphism out of all the possible isomorphism uniformly at random and assume that this isomorphism is the correct mapping. Given this approach a meaningful notion of security would be to compute the probability of a gate being correctly identified. The complexity of this problem is thought to be more difficult then k-security since every possible isomorphism would conceivably need to be iterated to compute the probability.

Consider the following scenario where the defender chooses the hidden wire set to minimize the chance of a random isomorphism correctly mapping one or more gates, could the adversary then use this meta knowledge about the selection process to help identify the hidden wire set? Further thought could be given to this notion of security if it is believed that the adversary will behave in the above manner.

Chapter 6

Conclusions

We have investigated how realistic the threat from non-deterministic timer-based triggers realized in hardware is. This is an open issue in existing work on hardware security. We have designed, implemented and thoroughly analyzed such a timer. We have shown that an attacker can realize such a timer feasibly. It has a small footprint and is therefore resistant to countermeasures such as IC fingerprinting. It also affords the attacker several appealing properties. For example, he is able to control the time-window within which the timer triggers with surprisingly granularity. Also, the timer is practical and effective with only a few bits of non-volatile storage, and a small time-window in which we need to maintain volatile state.

We have also proposed the use of 3D integration circuit technology to enhance the security of digital ICs via circuit obfuscation. The specific 3D technology we exploit allows gates and wires on the bottom tier, and only metal wires on the top. By implementing a subset of wires on the top tier, which is manufactured in a trusted fabrication facility, we obfuscate the identity of gates in the bottom tier, thus deterring malicious attackers.

We introduced a formal notion of security for 3D integration based circuit obfuscation and characterized the complexity of computing security under this notion. We proposed practical approaches to determining the security level given a subset of lifted wires, and of identifying a subset of wires to lift to achieve a desired security level. Our experimental results on the c432 and DES benchmark circuits allow us to quantify the power, area and delay costs to achieve different security levels.

An obvious question is how does k-security protect against a non-deterministic timer attack done from the foundry. The answer is that k-security will not impede the adversary from constructing the non-deterministic timer since the only wires he/she needs to identify correctly are the power lines. However in addition to inserting the timer the adversary will also insert some malicious changes which will be difficult to implement due to k-security.

We conclude with the overall observation that non-deterministic timer-based triggers pose a significant threat to the security of digital ICs and as shown in the DES circuit case study, 3D IC based circuit obfuscation can significantly reduce the ability of an attacker to carry out an effective attack.

Appendix A

Non-Deterministic Timer

A.1 Additional Data

Clock Cycles	Elapsed at Time	T	$E(T)$
4310601695232	23:56:52	1	2.05
4312852611072	23:57:37	2	3.07
4312979800064	23:57:39	3	3.15
4313141313536	23:57:42	4	3.24
4314477805568	23:58:09	5	3.96
4315219066880	23:58:24	6	4.44
4316670607360	23:58:53	7	5.27
4318743527424	23:59:34	8	6.71
4318743527424	23:59:34	9	6.71
4320404357120	24:00:08	10	7.86
4321153515520	24:00:23	11	8.44
4321940324352	24:00:38	12	9.00
4323020423168	24:01:00	13	9.66
4324269916160	24:01:25	14	10.49
4336227237888	24:05:24	15	14.58

Clock Cycles	Elapsed at Time	T	$E(T)$
4243684638720	23:34:33	1	0.84
4245332606976	23:35:06	2	0.90
4257367736320	23:39:07	3	1.44
4275229016064	23:45:04	4	2.64
4288744374272	23:49:34	5	3.88
4304409657344	23:54:48	6	5.62
4307356008448	23:55:47	7	5.97
4320939245568	24:00:18	8	7.62
4332075483136	24:04:01	9	8.99
4337718509568	24:05:54	10	9.67
4354048360448	24:11:20	11	11.42
4355995942912	24:11:59	12	11.59
4359474970624	24:13:09	13	11.91
4361189523456	24:13:43	14	12.06
4376770297856	24:18:55	15	13.20

Table A.1: 15 non-deterministic timers set to trigger in 24 hours with a standard deviation of 2.84 minutes (left) and 16 minutes (right). The resulting distributions of the number of timers that finish by a certain time were tested against the expected distribution with the χ^2 test, and yielded P-values of 0.878 (left) and 0.985 (right). The random variable T is the number of finished timers. (See Section 3.2.)

Appendix B

k-security

B.1 Proof of Circuit-Subiso is NP-hard

To prove that Circuit-Subiso is **NP**-hard, we first introduce two intermediate languages Circuit-Subiso' and Sub-Iso-9.

Definition 4 (Sub-Iso-9)**.** *Sub-Iso-9* $= \{\langle G, H\rangle$: *G and H be directed acyclic graphs and H is further restricted to be a directed tree. Then there exists a sub-graph of G that is isomorphic to H*$\}$.

Sub-Iso-9 is known to be **NP**-hard [30].

Definition 5 (Circuit-Subiso')**.** *Circuit-Subiso'* $= \{\langle G, H\rangle$: *G and H are DAGs such that* $|V[G]| = |V[H]|$, *and there exists* $R \subseteq E[G]$ *and a mapping* $\phi : V[G] \to V[H]$ *such that* ϕ *is an isomorphism for the graph* $\langle V[G], E[G] - R, c[G]\rangle$ *to H*.

Theorem 2. *Circuit-Subiso*$'$ $\in$ **NP**-*hard.*

Proof. Given a string $\langle G, H\rangle$, we compute $G' = G$. Let V_R be a set of new vertices such that $V_R \cap V[H] = \emptyset$ and $|V_R| = ||V[G]| - |V[H]||$. If G passes the DAG test and H is a directed tree then compute $V[H'] \leftarrow V[H] \cup V_R$ and $E[H'] \leftarrow E[H]$. Otherwise construct H' such that $|V[H']| = |V[G]| + 1$. Now $\langle G', H'\rangle \in$ Circuit-Subiso$'$ if and only if $\langle G', H'\rangle \in$ Sub-Iso-9. □

Theorem 3. *Circuit-Subiso* $\in$ **NP**-*hard.*

Proof. Given a string $\langle G, H \rangle$, we add a new vertex v_x such that $v_x \notin V[G] \cup V[H]$. Then construct $V[G'] \leftarrow V[G] \cup \{v_x\}, V[H'] \leftarrow V[H] \cup \{v_x\}$, $E[G] \leftarrow E[G], E[H'] \leftarrow E[H]$. Now $\langle G', H', v_x, v_x \rangle \in$ Circuit-Subiso if and only if $\langle G', H' \rangle \in$ Circuit-Subiso$'$. $\square$

References

[1] 3D-ICs and Integrated Circuit Security. Technical report, Tezzarron Semiconductors, 2008.

[2] Altera Stratix V. http://www.altera.com/devices/fpga/stratix-fpgas/stratix-v/stxv-index.jsp, 2013.

[3] J. Aarestad, D. Acharyya, R. Rad, and J. Plusquellic. Detecting trojans through leakage current analysis using multiple supply pad I_DDQs. *Information Forensics and Security, IEEE Transactions on*, 5(4):893–904, 2010.

[4] B.J. Abcunas. *Evaluation of random number generators on FPGAs.* PhD thesis, Worcester Polytechnic Institute, 2004.

[5] S. Adee. The hunt for the kill switch. *Spectrum, IEEE*, 45(5):34 –39, may 2008.

[6] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar. Trojan detection using ic fingerprinting. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 296–310. IEEE, 2007.

[7] Yousra Alkabani and Farinaz Koushanfar. N-variant ic design: Methodology and applications. In *Proceedings of the 45th annual Design Automation Conference*, pages 546–551. ACM, 2008.

[8] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*, volume 1. Cambridge University Press Cambridge, UK, 2009.

[9] Kaustav Banerjee, Shukri J Souri, Pawan Kapur, and Krishna C Saraswat. 3-d ics: A novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration. *Proceedings of the IEEE*, 89(5):602–633, 2001.

[10] M. Banga and M.S. Hsiao. Trusted rtl: Trojan detection methodology in pre-silicon designs. In *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on*, pages 56–59. IEEE, 2010.

[11] A. Baumgarten, M. Steffen, M. Clausman, and J. Zambreno. A case study in hardware trojan design and implementation. *International Journal of Information Security*, 10(1):1–14, 2011.

[12] G.T. Becker, A. Lakshminarasimhan, L. Lin, S. Srivathsa, V.B. Suresh, and W. Burelson. Implementing hardware trojans: Experiences from a hardware trojan challenge. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pages 301–304. IEEE, 2011.

[13] Manjul Bhushan, Anne Gattiker, Mark B Ketchen, and Koushik K Das. Ring oscillators for cmos process tuning and variability control. *Semiconductor Manufacturing, IEEE Transactions on*, 19(1):10–18, 2006.

[14] Michael Bilzor. 3d execution monitor (3d-em): Using 3d circuits to detect hardware malicious inclusions in general purpose processors. In *Proceedings of the 6th International Conference on Information Warfare and Secuirty*, page 288. Academic Conferences Limited, 2011.

[15] I. Bolsens. 2.5d ics: Just a stepping stone or a long term alternative to 3d? Keynote Talk at 3-D Architectures for Semiconductor Integration & Packaging Conference, 2011.

[16] Dan Boneh, Richard DeMillo, and Richard Lipton. On the importance of checking cryptographic protocols for faults. In *Advances in CryptologyEUROCRYPT97*, pages 37–51. Springer, 1997.

[17] F. Brglez. Neutral netlist of ten combinational benchmark circuits and a target translator in fortran. In *Special session on ATPG and fault simulation, Proc. IEEE Int. Symp. Circuits and Systems, June 1985*, pages 663–698, 1985.

[18] Marco Bucci, Lucia Germani, Raimondo Luzzi, Alessandro Trifiletti, and Mario Varanonuovo. A high-speed oscillator-based truly random number source for cryptographic applications on a smart card ic. *Computers, IEEE Transactions on*, 52(4):403–409, 2003.

[19] J. Burns. Tsv-based 3d integration. 2011.

[20] Y. Cai, E.F. Haratsch, O. Mutlu, and K. Mai. Error patterns in mlc nand flash memory: Measurement, characterization, and analysis. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*, pages 521–526. IEEE, 2012.

[21] S. Callegari, R. Rovatti, and G. Setti. Embeddable adc-based true random number generator for cryptographic applications exploiting nonlinear signal processing and chaos. *Signal Processing, IEEE Transactions on*, 53(2):793–805, 2005.

[22] Rajat Subhra Chakraborty, Seetharam Narasimhan, and Swarup Bhunia. Hardware trojan: Threats and emerging solutions. In *High Level Design Validation and Test Workshop, 2009. HLDVT 2009. IEEE International*, pages 166–171. IEEE, 2009.

[23] Z. Chen, X. Guo, R. Nagesh, A. Reddy, M. Gora, and A. Maiti. Hardware trojan designs on basys fpga board. *Embedded System Challenge Contest in Cyber Security Awareness Week, CSAW*, 2008.

[24] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. Performance evaluation of the vf graph matching algorithm. In *Image Analysis and Processing, 1999. Proceedings. International Conference on*, pages 1172–1177. IEEE, 1999.

[25] Shuaifu Dai, Tao Wei, Chao Zhang, Tielei Wang, Yu Ding, Zhenkai Liang, and Wei Zou. A framework to eliminate backdoors from response-computable authentication. In *IEEE Symposium on Security and Privacy*, pages 3–17, 2012.

[26] W Rhett Davis, John Wilson, Stephen Mick, Jian Xu, Hao Hua, Christopher Mineo, Ambarish M Sule, Michael Steer, and Paul D Franzon. Demystifying 3d ics: the pros and cons of going vertical. *Design & Test of Computers, IEEE*, 22(6):498–510, 2005.

[27] Yangdong Deng and Wojciech Maly. 2.5 d system integration: a design driven system implementation schema. In *Design Automation Conference, 2004. Proceedings of the ASP-DAC 2004. Asia and South Pacific*, pages 450–455. IEEE, 2004.

[28] V. Fischer and M. Drutarovskỳ. True random number generator embedded in reconfigurable hardware. *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 103–125, 2003.

[29] Pasquale Foggia, Carlo Sansone, and Mario Vento. A performance comparison of five algorithms for graph isomorphism. In *Proceedings of the 3rd IAPR TC-15 Workshop on Graph-based Representations in Pattern Recognition*, pages 188–199, 2001.

[30] M.R. Gary and D.S. Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.

[31] G. Gildenblat, L. Colonna-Romano, D. Lau, and DE Nelsen. Investigation of cryogenic cmos performance. In *Electron Devices Meeting, 1985 International*, volume 31, pages 268–271. IEEE, 1985.

[32] Christophe Giraud and Hugues Thiebeauld. A survey on fault attacks. *Smart Card Research and Advanced Applications VI*, pages 159–176, 2004.

[33] Matthew Hicks, Murph Finnicum, Samuel T. King, Milo M. K. Martin, and Jonathan M. Smith. Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically. In *IEEE Symposium on Security and Privacy*, pages 159–172, 2010.

[34] D.E. Holcomb, W.P. Burleson, and K. Fu. Power-up sram state as an identifying fingerprint and source of true random numbers. *Computers, IEEE Transactions on*, 58(9):1198–1210, 2009.

[35] C.C. Hsieh, C.Y. Wu, and T.P. Sun. A new cryogenic cmos readout structure for infrared focal plane array. *Solid-State Circuits, IEEE Journal of*, 32(8):1192–1199, 1997.

[36] Cynthia E Irvine and Karl Levitt. Trusted hardware: Can it be trustworthy? In *Proceedings of the 44th Annual Design Automation Conference*, pages 1–4. ACM, 2007.

[37] Y. Jin, N. Kupp, and Y. Makris. Experiences in hardware trojan design and implementation. In *Hardware-Oriented Security and Trust, 2009. HOST'09. IEEE International Workshop on*, pages 50–57. IEEE, 2009.

[38] Y. Jin and Y. Makris. Hardware trojan detection using path delay fingerprint. In *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, pages 51–57. IEEE, 2008.

[39] Chan-Kyung Kim, Jae-Goo Lee, Young-Hyun Jun, Chil-Gee Lee, and Bai-Sun Kong. Cmos temperature sensor with ring oscillator for mobile dram self-refresh control. *Microelectronics Journal*, 38(10):1042–1049, 2007.

[40] Tae-Hyoung Kim, Randy Persaud, and Chris H Kim. Silicon odometer: An on-chip reliability monitor for measuring frequency degradation of digital circuits. *Solid-State Circuits, IEEE Journal of*, 43(4):874–880, 2008.

[41] S.T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou. Designing and implementing malicious hardware. In *Proceedings of the 1st USENIX Workshop on Large-scale Exploits and Emergent Threats*, pages 1–8. USENIX Association, 2008.

[42] D.E. Knuth. *The art of computer programming.* addison-Wesley, 2006.

[43] P. Kohlbrenner and K. Gaj. An embedded true random number generator for fpgas. In *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 71–78. ACM, 2004.

[44] F. Koushanfar and Y. Alkabani. Provably secure obfuscation of diverse watermarks for sequential circuits. In *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on*, pages 42–47, june 2010.

[45] J. Kumagai. Chip detectives [reverse engineering]. *Spectrum, IEEE*, 37(11):43 –48, nov 2000.

[46] C. Lamech, R.M. Rad, M. Tehranipoor, and J. Plusquellic. An experimental analysis of power and delay signal-to-noise requirements for detecting trojans and methods for achieving the required detection sensitivities. *Information Forensics and Security, IEEE Transactions on*, 6(3):1170–1179, 2011.

[47] John H Lau. The most cost-effective integrator (tsv interposer) for 3d ic integration system-in-package (sip). ASME, 2011.

[48] Seongsoo Lee and Takayasu Sakurai. Run-time voltage hopping for low-power real-time systems. In *Proceedings of the 37th Annual Design Automation Conference*, pages 806–809. ACM, 2000.

[49] G. Marsaglia. Diehard: a battery of tests of randomness. *See http://stat. fsu. edu/ geo/diehard. html*, 1996.

[50] D. Meisner, B.T. Gold, and T.F. Wenisch. Powernap: eliminating server idle power. *ACM Sigplan Notices*, 44(3):205–216, 2009.

[51] Simon Moore and Gregory Chadwick. http://www.cl.cam.ac.uk/teaching/.

[52] A. Papoulis and R.V. Probability. *Stochastic processes*, volume 3. McGraw-hill New York, 1991.

[53] Betty Prince. Trends in scaled and nanotechnology memories. In *Non-Volatile Memory Technology Symposium, 2005*, pages 7–pp. IEEE, 2005.

[54] R. Rad, J. Plusquellic, and M. Tehranipoor. A sensitivity analysis of power signal methods for detecting hardware trojans under real process and environmental conditions. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 18(12):1735–1744, 2010.

[55] Amir Rahmati, Mastooreh Salajegheh, Dan Holcomb, Jacob Sorber, Wayne P Burleson, and Kevin Fu. Tardis: Time and remanence decay in sram to implement secure protocols on embedded devices without clocks. In *21st USENIX Security Symposium (USENIX Security 2012)*, 2012.

[56] T. Reece and W.H. Robinson. Hardware trojans: The defense and attack of integrated circuits. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pages 293–296. IEEE, 2011.

[57] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, DTIC Document, 2001.

[58] H. Salmani and M. Tehranipoor. Layout-aware switching activity localization to enhance hardware trojan detection. *Information Forensics and Security, IEEE Transactions on*, 7(1):76–87, 2012.

[59] D. Schellekens, B. Preneel, and I. Verbauwhede. Fpga vendor agnostic true random number generator. In *Field Programmable Logic and Applications, 2006. FPL'06. International Conference on*, pages 1–6. IEEE, 2006.

[60] Ellen M Sentovich, Kanwar Jit Singh, Cho Moon, Hamid Savoj, Robert K Brayton, and Alberto Sangiovanni-Vincentelli. Sequential circuit design using synthesis and optimization. In *Computer Design: VLSI in Computers and Processors, 1992. ICCD'92. Proceedings., IEEE 1992 International Conference on*, pages 328–333. IEEE, 1992.

[61] S. Skorobogatov and C. Woods. Breakthrough silicon scanning discovers backdoor in military chip. *Cryptographic Hardware and Embedded Systems–CHES 2012*, pages 23–40, 2012.

[62] Seung-Hwan Song, Ki Chul Chun, and Chris H Kim. A logic-compatible embedded flash memory featuring a multi-story high voltage switch and a selective refresh scheme. In *VLSI Circuits (VLSIC), 2012 Symposium on*, pages 130–131. IEEE, 2012.

[63] Niklas Sorensson and Niklas Een. Minisat v1. 13-a sat solver with conflict-clause minimization. *SAT*, 2005:53, 2005.

[64] J. Soto. Statistical testing of random number generators. In *Proceedings of the 22nd National Information Systems Security Conference*, volume 10, page 12. NIST Gaithersburg, MD, 1999.

[65] C. Sturton, M. Hicks, D. Wagner, and S.T. King. Defeating uci: Building stealthy and malicious hardware. In *IEEE Symposium on Security and Privacy*, pages 64–77, 2011.

[66] G.E. Suh and S. Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th annual Design Automation Conference*, pages 9–14. ACM, 2007.

[67] B. Sunar, W.J. Martin, and D.R. Stinson. A provably secure true random number generator with built-in tolerance to active attacks. *Computers, IEEE Transactions on*, 56(1):109–119, 2007.

[68] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[69] M. Tehranipoor and F. Koushanfar. A survey of hardware trojan taxonomy and detection. *Design & Test of Computers, IEEE*, 27(1):10–25, 2010.

[70] M. Tehranipoor, H. Salmani, X. Zhang, X. Wang, R. Karri, J. Rajendran, and K. Rosenfeld. Trustworthy hardware: Trojan detection and design-for-trust challenges. *Computer*, 44(7):66–74, 2011.

[71] Tezzaron. 3D-ICs and integrated circuit security. Technical report, 02 2008.

[72] D.E. Thomas and P.R. Moorby. *The Verilog® Hardware Description Language*, volume 2. Springer, 2002.

[73] Jonathan Valamehr, Mohit Tiwari, Timothy Sherwood, Ryan Kastner, Ted Huffmire, Cynthia Irvine, and Timothy Levin. Hardware assistance for trustworthy systems through 3-d integration. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 199–210. ACM, 2010.

[74] A. Waksman and S. Sethumadhavan. Tamper evident microprocessors. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 173–188. IEEE, 2010.

[75] A. Waksman and S. Sethumadhavan. Silencing hardware backdoors. In *IEEE Symposium on Security and Privacy*, pages 49–63, 2011.

[76] X. Wang, S. Narasimhan, A. Krishna, T. Mal-Sarkar, and S. Bhunia. Sequential hardware trojan: Side-channel aware design and placement. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pages 297–300. IEEE, 2011.

[77] Yinglei Wang, Wing kei Yu, Shuo Wu, G. Malysa, G.E. Suh, and E.C. Kan. Flash memory for ubiquitous hardware security functions: True random number generation and device fingerprints. In *IEEE Symposium on Security and Privacy*, pages 33–47, 2012.

[78] S. Wei, S. Meguerdichian, and M. Potkonjak. Malicious circuitry detection using thermal conditioning. *Information Forensics and Security, IEEE Transactions on*, 6(3):1136–1145, 2011.

[79] K. Wold and C.H. Tan. Analysis and enhancement of random number generator in fpga based on oscillator rings. *International Journal of Reconfigurable Computing*, 2009:4, 2009.

[80] F. Wolff, C. Papachristou, S. Bhunia, and R.S. Chakraborty. Towards trojan-free trusted ics: Problem analysis and detection scheme. In *Design, Automation and Test in Europe, 2008. DATE'08*, pages 1362–1365. IEEE, 2008.

[81] X. Zhang and M. Tehranipoor. Case study: Detecting hardware trojans in third-party digital ip cores. In *Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on*, pages 67–70. IEEE, 2011.

[82] X. Zhang, N. Tuzzio, and M. Tehranipoor. Red team: Design of intelligent hardware trojans with known defense schemes. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pages 309 –312, oct. 2011.

[83] Bin Zhou and Jian Pei. Preserving privacy in social networks against neighborhood attacks. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 506–515. IEEE, 2008.