BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Dissertation

A MULTI-LAYER APPROACH TO DESIGNING SECURE SYSTEMS: FROM CIRCUIT TO SOFTWARE

by

BOYOU ZHOU

B.S., Southeast University, China, 2013
M.S., Boston University, 2019

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2019

Approved by

First Reader

Ajay J. Joshi, PhD
Associate Professor of Electrical and Computer Engineering

Second Reader

Manuel Egele, PhD
Assistant Professor of Electrical and Computer Engineering

Third Reader

M. Selim Ünlü, PhD
Professor of Electrical and Computer Engineering
Professor of Materials Science and Engineering
Professor of Biomedical Engineering

Fourth Reader

Gianluca Stringhini, PhD
Assistant Professor of Electrical and Computer Engineering

Ignorance is bliss – Thomas Gray (1742)

Acknowledgments

First, I would like to express my gratitude to my advisor, Ajay Joshi. This work would not have been possible without Ajay's guidance and help. I am also thankful to Prof. Manuel Egele, as my co-advisor, for the last three years. He advised and guided most of my works during this process. My regret throughout the entire graduate school was to get to know him sooner. I also appreciate the help from Prof. Selim Unlu, for the guidance in our photonic project. I want to thank the funding agency, NSF and DARPA, to support all my works.

It goes without saying, without support from my parents, Jianhua Zhou, and Qi Jin, this work is not possible. Also, my girlfriend, Weisi Liao, was also incredibly supportive to help me through this process.

I also want to thank my collaborators, Ronen Adato, Aydan Aksoylar, Anmol Gupta, Rasoul Jahanshahi, Negin Zaraee, and Sadulla Canakci. With their collaborations, I could finish the works and publish them in the past few years. I would like to thank Boston University for funding most of my works and DARPA for the Posh open source hardware project (POSH). I also appreciate my colleagues in our group: Leila, Zahra, Yenai, Furkan, Marcia, Saiful, Zafar, Mahmoud, Schuyler, and Chao. I would never forget the life with them, the life sharing all happiness. This goes to the PEAC and CAAD labs too.

A MULTI-LAYER APPROACH TO DESIGNING SECURE SYSTEMS: FROM CIRCUIT TO SOFTWARE

BOYOU ZHOU

Boston University, College of Engineering, 2019

Major Professors: Ajay J. Joshi, PhD
Associate Professor of Electrical and Computer Engineering

Manuel Egele, PhD
Assistant Professor of Electrical and Computer Engineering

ABSTRACT

In the last few years, security has become one of the key challenges in computing systems. Failures in the secure operations of these systems have led to massive information leaks and cyber-attacks. Case in point, the identity leaks from Equifax in 2016, Spectre and Meltdown attacks to Intel and AMD processors in 2017, Cyber-attacks on Facebook in 2018. These recent attacks have shown that the intruders attack different layers of the systems, from low-level hardware to software as a service (SaaS). To protect the systems, the defense mechanisms should confront the attacks in the different layers of the systems. In this work, we propose four security mechanisms for computing systems: (*i*) using backside imaging to detect Hardware Trojans (HTs) in Application Specific Integrated Circuits (ASICs) chips, (*ii*) developing energy-efficient reconfigurable cryptographic engines, (*iii*) examining the feasibility of malware detection using Hardware Performance Counters (HPC).

Most of the threat models assume that the root of trust is the hardware running

beneath the software stack. However, attackers can insert malicious hardware blocks, i.e. HTs, into the Integrated Circuits (ICs) that provide back-doors to the attackers or leak confidential information. HTs inserted during fabrication are extremely hard to detect since their overheads in performance and power are below the variations in the performance and power caused by manufacturing. In our work, we have developed an optical method that identifies modified or replaced gates in the ICs. We use the near-infrared light to image the ICs because silicon is transparent to near-infrared light and metal reflects infrared light. We leverage the near-infrared imaging to identify the locations of each gate, based on the signatures of metal structures reflected by the lowest metal layer. By comparing the imaged results to the pre-fabrication design, we can identify any modifications, shifts or replacements in the circuits to detect HTs.

With the trust of the silicon, the computing system must use secure communication channels for its applications. The low-energy cost devices, such as the Internet of Things (IoT), leverage strong cryptographic algorithms (e.g. AES, RSA, and SHA) during communications. The cryptographic operations cause the IoT devices a significant amount of power. As a result, the power budget limits their applications. To mitigate the high power consumption, modern processors embed these cryptographic operations into hardware primitives. This also improves system performance. The hardware unit embedded into the processor provides high energy-efficiency, low energy cost. However, hardware implementations limit flexibility. The longevity of the IoTs can exceed the lifetime of the cryptographic algorithms. The replacement of the IoT devices is costly and sometimes prohibitive, e.g., monitors in nuclear reactors. In order to reconfigure cryptographic algorithms into hardware, we have developed a system with a reconfigurable encryption engine on the Zedboard platform. The hardware implementation of the engine ensures fast, energy-efficient cryptographic operations.

With reliable hardware and secure communication channels in place, the computing systems should detect any malicious behaviors in the processes. We have explored the use of the Hardware Performance Counters (HPCs) in malware detection. HPCs are hardware units that count micro-architectural events, such as cache hits/misses and floating point operations. Anti-virus software is commonly used to detect malware but it also introduces performance overhead. To reduce anti-virus performance overhead, many researchers propose to use HPCs with machine learning models in malware detection. However, it is counter-intuitive that the high-level program behaviors can manifest themselves in low-level statics. We perform experiments using $2 \sim 3 \times$ larger program counts than the previous works and perform a rigorous analysis to determine whether HPCs can be used to detect malware. Our results show that the False Discovery Rate of malware detection can reach 20%. If we deploy this detection system on a fresh installed Windows 7 systems, among 1,323 binaries, 198 binaries would be flagged as malware.

Contents

List of Tables

List of Figures

List of Abbreviations

AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
ASLR	Address-Space-Layout-Randomization
AUC	Area Under Curve
AXI	Advanced eXtensible Interface
CDF	Cumulative Distribution Function
CFG	Control Flow Graph
CFI	Control Flow Integrity
DBI	Dynamic Binary Instrumentation
DEP	Data Execution Prevention
DES	Data Encryption Standard
DPA	Differential Power Analysis
DT	Decision Tree
ECB	Electronic Code Book mode
ECs	Equivalence Classes
GPP	General Purpose Processor
GRUB	GNU GRand Unified Bootloader
HPC	Hardware Performance Counter
HPCs	Hardware Performance Counters
HTs	Hardware Trojans
IoTs	Internet of things
IR	Intermediate Representations
ITLB	Instruction Translation Lookaside Buffer
JIT	Just-In-Time
KNN	K Nearest Neighbors
MLP	Multilayer Perceptron
NN	Neural Net
NSA	National Security Agency
NX	Non-Executable memory
PCA	Principal Component Analysis
PDF	probability density function
POSH	Posh open source hardware project
POSIX	Portable Operating System Interface
RE	Reverse Engineer
RF	Random Forest

RSA	RivestShamirAdleman
Saas	software as a service
SHA	Secure Hash Algorithm
SNR	Signal to Noise Ratio
SoC	System on Chip
SSH	Secure SHell
TSVs	Through Silicon Vias
TTA	Training-and-Testing Approach
VLIW	Very Long Instruction Word

Chapter 1

Introduction

1.1 Background and Motivation

In the last few years, the security of the computing systems has become one of the main concerns in our lives. In 2016, identity theft from Equifax leaked the social security numbers of 300 million people (Equ, 2019); Spectre and Meltdown in 2017 affected all Intel and AMD processors to show the vulnerabilities in the architectural level (Mel, 2019); The leak of information from Facebook in 2018 directly affected millions of user profiles (Fac, 2019). Every year, there were a series of events in the security area involving millions or even billions of people. The previous incidents have shown that the security attacks have reached regimes which cannot be tackled by the existing defense system. Spectre and Meltdown exposed security flaws in computer architecture. The leaks of Facebook user profiles illustrated that the attacks on web applications can have social and political effects. In order to mitigate these failures in the existing systems, there is a pressing need to build secure systems on every layer of the computing systems, from circuits to web applications.

The root of the trust in the defense systems is the silicon running beneath all the software stacks. In general, the fabless design companies, such as Qualcomm, ARM and Broadcom trust the manufacturers of the ICs. The hardware companies fabricate their IC chips overseas, given that overseas IC manufacturers can support a range of technology nodes at much lower cost (AsC, 2019). However, this trust has been broken since the separation of design and manufacturing creates opportunities for hardware

attacks like Hardware Trojan (HT) insertion, IC overbuild, reverse engineering, side-channel analysis, and IC counterfeiting (Tehranipoor and Koushanfar, 2010). Among these attacks, HTs can cause significant economic and social damages. HTs are hardware blocks that are designed to perform malicious operations, e.g. leak secret keys, sabotage the functionality of the chip or launch privilege escalation attacks on the system. The HTs inserted during manufacturing have negligible overheads in performance and power consumption of the overall ICs (Nowroz et al., 2014; Yang et al., 2016). Besides, the attackers design HTs to evade being triggered during functional testing (Wei et al., 2012a). IC design companies need to either monitor the entire manufacturing process or apply costly reverse engineering techniques, including delayering the chip, imaging with Scanning Electron Microscope (SEM) etc. to detect HTs (Tehranipoor and Koushanfar, 2010; Karri et al., 2010). Both delayering and SEM are time-consuming and costly. Hence, there is a pressing need for a detection method, which can detect and locate HTs inserted during fabrication stage in a fast, accurate, and robust manner (Tsoutsos and Maniatakos, 2014).

At the architectural level, the security functionalities, e.g. cryptographic operations, ensure the confidentiality and integrity of the communications. The architectural solutions are deployed using dedicated Application Specific Integrated Circuit (ASIC) units embedded within the processor. For example ARM, Intel, and AVR, all provide hardware accelerated Advanced Encryption Standard (AES), crypto-engines in their ARMv8 (ARM, 2016), x86 (Int, 2016a), and Atmel (AVR, 2016) platforms respectively. These processors can leverage hardware primitives to perform cryptographic operations, such as Advanced Encryption Standard (AES), RivestShamirAdleman (RSA), and Secure Hash Algorithm (SHA). These operations are crucial to the Internet of things (IoT), which handle command and control signals at home and on the factory floors. IoTs are also low-power devices, which cannot easily be

replaced or modified, and are deployed for years at a time. Moreover, the longevity of these IoT devices often surpass the usage of cryptographic algorithm. Besides some IoT devices are deployed at dangerous places where they are dangerous and hard for humans to replace them. In order to achieve flexibility, high performance, and low power cryptographies, we need an energy-efficient configurable cryptography engine for IoT devices.

At the application level, the defense system must detect the malicious processes for protection. Researchers have utilized various analytical methodologies, such as signature analysis and dynamic analysis, for malware detection. All the previous methods have a substantial performance overhead. In order to reduce the overhead, the previous works propose the use of Hardware Performance Counters (HPCs) with machine learning models (Demme et al., 2013; Kazdagli et al., 2016; Wang et al., 2016; Ozsoy et al., 2015; Tang et al., 2014; Khasawneh et al., 2015; Khasawneh et al., 2017; Singh et al., 2017). HPCs are hardware blocks that count micro-architectural events. However, it is counter-intuitive that the high-level program behaviors can manifest themselves in low-level statics. In order to show the feasibility of HPC malware detection, the researchers must evaluate HPC malware detection rigorously before the real-life deployments.

In this work, we construct the solutions towards the threat models on each computing layer, hardware, architectural and software layer. We present the thesis statement as following:

Constructing a defense mechanism on the same computing layer as a security threat can protect the computing system effectively and efficiently.

Therefore, we propose a multi-level approach to securing computing systems. Our approaches include the following solutions for securing computing systems:

- **HT Detection using backside imaging**: We propose a novel optical method, where we image the integrated circuit chip from the backside without powering it up or delayering it (Chapter 2). Using our method, any replacements, modifications or re-arrangements of gates to add HTs can be easily detected through comparisons between the simulated optical response and backside imaged measurements. We introduce the idea of a noise-based detection method as our testing method to have higher HT detection rates in different testbenches. To further improve the robustness of our method, we strategically place high reflectance fill cells in the designs. Our imaging method provides high-resolution, non-destructive and rapid means to detect HTs inserted during fabrication. We evaluate our approach using various hardware blocks where the HTs can occupy less than 0.1% of the total area or consist of fewer than 3 gates. In addition, we analyze our method with different magnitudes of noise, process variations, detection window sizes, and resolutions. To increase the difference in the reflection among various gates, we also engineer nano-antenna structures into the gate pairs. These nano-antennas are unique to illumination polarizations, frequencies and angles. By implementing the nano-antennas, we can improve the reflectance signature by 300%.

- **Flexible Hardware Implementations of Cryptographies**: We propose to use FPGA as the reconfigurable substrate for cryptographic operations in IoT devices (Chapter 3). We demonstrate our proposed approach on the Zedboard platform, which has two ARM cores and a Zynq FPGA. The implemented cryptographic algorithms include symmetric cryptography, asymmetric cryptography, and secure hash functions. We also integrate our cryptographic engines in the SSL library to inherit the software support for block cipher modes implemented by OpenSSL. Our approach shows that the FPGA-based recon-

figurable cryptographic components consume 1.8× ∼ 4033× less energy and run 1.6× ∼ 2983× faster than software implementation. At the same time, the FPGA implementation of cryptographic operations are more flexible compared to custom hardware implementations of cryptographic components.

- **Malware Detection using Hardware Performance Counters with machine learning**: We identify the prevalent unrealistic assumptions and the insufficient analysis used in prior works that leverage HPCs for malware detection (Chapter 4). We perform thorough experiments with a program count that exceeds prior works (Demme et al., 2013; Kazdagli et al., 2016; Wang et al., 2016; Tang et al., 2014; Singh et al., 2017) by a factor of 2× ∼ 3×, and the number of experiments in cross-validations that are 3 orders of magnitude more than previous works. We divide the train-and-test dataset similar to what prior works have done, as well as, in a realistic setting where testing programs are not in the training programs. We compare the effects of this choice on the quality of the machine learning models. Finally, to facilitate reproducibility, and enable future researchers to easily compare their experiments with ours, we make all code, data, and results of our project publicly available under an open-source license: `https://github.com/bu-icsg/Hardware_Performance_Counters_Can_Detect_Malware_Myth_or_Fact`

1.2 Related Work

In this work, we present the works related to HT detection, cryptographic engines, malware detection using HPCs with machine learning and CFI hardware enforcement.

1.2.1 HT Detection using Backside Imaging

Three different HT detection approaches have been proposed for preventing HT insertion and detecting any inserted HT: protective designs before fabrication, HT detection after fabrication, and side channel detection method. A number of pre-fabrication protection designs have been proposed on both circuits and architecture levels. At the circuit level, the separation of fabrication, gate level obfuscations, and limitations of gate usage have been proposed to protection designs. Separation of fabrication (Vaidyanathan et al., 2014a; Vaidyanathan et al., 2014b; Valamehr et al., 2013) proposes that the device layer of IC is fabricated by one trusted vendor, while other layers are fabricated by untrusted vendors at a lower cost. The trusted vendor use high cost Through Silicon Vias (TSVs) to connect different layers of logic. HARPOON (Chakraborty and Bhunia, 2009) provides netlist-level obfuscation, which can be integrated into the synthesis of Soc designs. Other techniques, (Roy et al., 2008; Baumgarten et al., 2010) and (Xiao et al., 2014; Xiao et al., 2015) also provide protective designs by replacing the filler cells with functional cells. That way there are no empty spaces for inserting HTs. Bao et al. proposed to both prevent and detect HTs in (Bao et al., 2015), by limiting the useable gate types in the standard library. All these proposed techniques either use a high-cost method or sacrifice area in order to protect designs from HTs. On the architecture level, logic state scrambling, power signatures in potential tampered logics, and circuitry encoding are the main techniques in protecting designs from HTs. ODETTE (Banga and Hsiao, 2011) protects the circuits from HTs by increasing the number and variations of the design. VITAMIN (Banga and Hsiao, 2009b) utilizes an inverted voltage scheme that aims to activate some targeted Trojans with a higher triggering rate. The power consumption of HTs will become prominent in the power analysis. Linear Complement Dual Codes (Ngo et al., 2015) is designed to encode the instructions inside the circuitry

and to be resistant to error injection and side channel analysis. These methods potentially increase the complexity of logic designs, and they also increase the power consumptions of the overall system.

Post-fabrication detection includes reverse engineer (RE), timing and power analysis, and data monitoring during runtime. The generic detection method after fabrications is to reverse engineering the fabricated ICs. Scanning Electron Microscope (SEM) and FIB (Focused Ion Beam) (Tehranipoor and Koushanfar, 2010) have been proposed for HT Detection. However, the approaches are high in cost and low in speed. The process of RE the fabricated chip is only affordable to large semiconductor companies (Tehranipoor and Koushanfar, 2010; Karri et al., 2010). FANCI (Waksman et al., 2013) and FIGHT (Sullivan et al., 2014), flag possible HT wires, which reduces the complexity in reverse engineering. To detect HTs without reverse engineering the IC chips timing analyses and power analyses methods have been proposed. Timing-based analysis (Li and Lach, 2008; Jin and Makris, 2008; Exurville et al., 2015) detects the delay changes in the circuitry for HT Detection. Dynamic power detection, such as the sustained vector technique, measures the dynamic power consumption difference in the circuitry and identifies the gates that may contain Trojans (Banga and Hsiao, 2009a; Salmani et al., 2012; Wilcox et al., 2015). Such time and power based analysis (Potkonjak et al., 2009a; Wei et al., 2010; Wei and Potkonjak, 2012; Wei et al., 2012b; Rad et al., 2008; Alkabani and Koushanfar, 2009; Potkonjak et al., 2009b; Wei and Potkonjak, 2012) are not suitable at newer technology nodes, because the impact of process variations is larger than the impact by the HTs on power and delays. This introduces overhead in the performance in order to detect HTs.

Compared to electrical methods, side channel detection utilizes physical properties of ICs instead of electrical properties, which can help detect HTs that can hide from electrical tests. Emission measurements (Song et al., 2011; Stellari et al., 2014)

and thermal analysis (Nowroz et al., 2014; Forte et al., 2013) are good examples of side channel analysis. Emission measurements utilize the long wavelength imaging techniques resulting the low-resolution imaging. Thermal analysis requires the powering on the logic, which introduces the engineering of testing infrastructures. All these techniques are either not as fast, easy or high in resolution as our technique.

1.2.2 Cryptographic Engines for IoT Devices

Many previous works have implemented various algorithms on the FPGA to show the performance improvements, energy savings in these FPGA implementations. The FPGA implementation of AES was first shown to have benefits on both performance improvements and energy savings in 2004 (Rouvroy et al., 2004). The performance reached a speed of 204Mbps stream in AES encryption. Since then, FPGAs were explored as a potential solution to cryptographic applications for low power devices. Good et al. proposed a design of AES, which use only on 8-bit data-path in all the operations in AES (Good and Benaissa, 2006). This design made the entire AES fit on small programmable areas. Hamalainen et al. presented an upgraded version with the cost of using 3.1k gates (Hamalainen et al., 2006). With the increasing area of FPGA, more and more resources could be used for cryptographic designs. One 128-bit AES block operation consists of key expansions and SubBytes operations. Key expansions increase the key length for further data processing in SubBytes operations. SubBytes is a 16-by-16 matrix operated with a minimum unit of 8-bit for permutations and substitutions during AES encryption/decryption. SubBytes operations consist of substitutions of cells, shifts of rows, a mix of columns and exclusive or with the keys, which are operations to increase the pseudorandomness of the output text. AES encryption/decryption usually consists of $10 \sim 14$ rounds of SubBytes operations according to the required security level. With fully integrated SubBytes operations in hardware, AES has much higher efficiency, as AES needs fewer cycles to

finish the operations. Bulens et al. first put one full round of SubBytes operations in AES on FPGA in 2008 (Bulens et al., 2008). With pipelined designs, the plain text can be continuously streamed into the cryptographic units for encryption/decryption to achieve high throughput. Repetitive use of same hardware block cannot have the same high performance as pipelined design do since inputs in each round depend on results of previous round operations. Good et al. showed the designs from the fastest to the smallest, which can provide different design points on different hardware platforms, in order to achieve a trade-off between speed and area (Good and Benaissa, 2005). It achieved the speed of 358Mbps compared to encryption/decryption speed of 397Mbps in our implementation without streaming throughput. Today, modern FPGAs have enough programmable logic that can fit the expansion of 10 to 14 rounds of SubBytes operations. Hoang et al. showed a fast implementation of AES with a key size of 128-bit, with look-up table of SubByteses (Hoang et al., 2012). There are also partial and dynamic reconfigurable implementations for more flexible solutions on FPGA (Granado-Criado et al., 2010). As partial configurations are designed for high-end FPGA applications, low-energy FPGA has not yet widely adopted the partial configurations for their applications. Side channel attacks use power, time or other physical properties to break cryptographic algorithms. Once discovered, it is possible to prevent a given side channel attack by modifying the implementation of cryptographic algorithms. Robust designs, such as (Oswald et al., 2005; Shah et al., 2010), are resistant to Differential Power Analysis (DPA) leaking AES keys in their implementation. However, robustness and DPA resistant designs come at a higher cost in performance or energy consumption. Other cryptographic algorithms, such as Hummingbird (Fan et al., 2010) and Whirlpool (Pramstaller et al., 2006), have also been implemented on FPGA. Recent works have shown that FPGA can be an ideal solution for modern cryptographic computation. Saarinen et al. im-

plemented Authenticated Encryption with Associated Data (AEAD), AES-GCM on Zedboard (Saarinen, 2014). Saarinen et al. also showed one variant implementation of Whirlpool (Saarinen and Brumley, 2014). These works contain detailed explanations of implementation and performance analysis, but lack of power and energy analysis and multi-crypto-engines implementation.

1.2.3 Malware Detection using Hardware Performance Counters with Machine Learning

Malware detection is the process of detecting malicious programs. Many previous works commonly utilize *sub-semantic features* in malware detection (Demme et al., 2013; Kazdagli et al., 2016; Wang et al., 2016; Ozsoy et al., 2015; Tang et al., 2014; Khasawneh et al., 2015; Khasawneh et al., 2017; Singh et al., 2017). Ozsoy et al. defined the term *sub-semantic features* as "micro-architectural information about an executing program that does not require modeling or detecting program semantics" (Ozsoy et al., 2015). These sub-semantic features are often aggregated in the HPCs to record the statistics of low-level events. All these previous works have several drawbacks to various extent. We categorize the drawbacks that we observed in the following classes.

I Dynamic Binary Instrumentation (DBI)

II Virtual Machines (VMs)

III Division of Data By Traces (TTA1 in § 4.4.3)

IV No Cross-Validations or Insufficient Validations

V Few Data Samples

Besides HPCs, sub-semantic features can be extracted with dynamic binary instrumentation (DBI) tools such as Intel's Pin (Patil et al., 2004; Luk et al., 2005), QEMU (Bellard, 2005), Valgrind (Nethercote and Seward, 2007), or DynamoRIO (Dyn, 2017). Khasawneh, Ozsoy et al. use Pin to monitor the instructions executed on vir-

tual machines in their experimental setup (Ozsoy et al., 2015; Khasawneh et al., 2015; Khasawneh et al., 2017). Though DBI can extract sub-semantic features that are not available from HPCs, DBI introduces a substantial amount of performance overhead and is thus not suited to run in an *always-on*, online protection setting, which is the default use-case for current anti-malware suites. We denote the drawbacks of DBI as **Drawback I** in Table 1.1.

While DBI is not feasible in online detection systems, other methods in sampling HPCs can incur inaccurate measurements. A plethora of previous works that uses HPCs for malware detections run the evaluated programs on VMs (Demme et al., 2013; Ozsoy et al., 2015; Khasawneh et al., 2015; Khasawneh et al., 2017; Singh et al., 2017). While VMs provide significant benefits to analyze unknown programs (e.g., strong isolation guarantees), HPCs are limited and shared resource between the host and all VMs. Thus, virtualizing HPCs is a challenge in itself (Serebrin and Hecht, 2011). The measured HPC values obtained in VM are substantially different from HPC values obtained from the bare-metal environment that real-life users have. To make matters worse, evasive malware can detect whether it is running in a VM and ceases to exhibit malicious behavior (Kirat et al. (Kirat et al., 2014)). These observations motivate our experimental setup (§4.3) to run all experiments on bare-metal systems. We label the use of VM in the experimental setups as **Drawback II** in Table 1.1.

Due to inaccurate HPC measurements (Weaver and McKee, 2008), previous works (Demme et al., 2013; Kazdagli et al., 2016; Wang et al., 2016; Tang et al., 2014) choose to maximize the measuring granularity by using HPCs without time-multiplexing. Recall that as modern CPUs only have 6 (AMD) or 4 (Intel) registers for HPCs, malware detection methods must select the events from more than 100 available micro-architectural events (130 in AMD Bulldozer and 196 in Intel Skylake). Previous

Paper	Tool Choice	Experimental Setups		Event Choice	Data Division		Cross Validation				Machine Learning Models					# of Programs	Opensource	
	Drawback I: DBI (Pin or QEMU)	**Drawback II**: Virtual Machine	Bare-metal Machine	Quantative Selection of Events	**Drawback III**: Data Divided By Traces (TTA1 in § 4.4.3)	Data Divided By Samples (TTA2 in § 4.4.3)	**Drawback IV**: No Cross-validation	**Drawback IV**: $60-20-20\%$ Data Division	10-fold Cross-validation	1,000 10-fold Cross-validations	DT	RF	KNN	NN	Ensemble Model (a collection of models)	**Drawback V**: Fewer than 1,000 programs	Release of Data and Codes to Public	Number of Drawbacks
(Demme et al., 2013)	◇	◇	●	◇	●	◇	●	◇	◇	◇	●	●	●	●	◇	●	◇	3
(Kazdagli et al., 2016)	◇	◇	●	◇	●	●	◇	◇	●	◇	●	●	●	◇	◇	●	◇	2
(Wang et al., 2016)	◇	●	◇	●	●	◇	●	◇	◇	◇	◇	◇	◇	◇	◇	●	◇	4
(Tang et al., 2014)	◇	●	◇	●	◇	●	●	◇	◇	◇	◇	◇	◇	◇	◇	●	◇	3
(Singh et al., 2017)	◇	●	◇	●	●	◇	◇	●	◇	◇	●	◇	◇	◇	◇	●	◇	4
(Ozsoy et al., 2015)	●	●	◇	◇	◇	●	◇	●	◇	◇	◇	◇	◇	●	◇	◇	◇	2
(Khasawneh et al., 2015)	●	●	◇	◇	◇	●	◇	●	◇	◇	◇	●	◇	◇	●	◇	◇	3
(Khasawneh et al., 2017)	●	●	◇	◇	◇	●	◇	●	◇	◇	◇	●	◇	◇	●	◇	◇	3
⋆	◇	◇	●	●	●	●	◇	◇	◇	●	●	●	●	●	●	◇	●	-

Table 1.1: Comparison between various previous works: Rows are various works in HPC-based malware detection and columns are design choices. The alternative shaded and white background represents different categories of tool/setup/model in malware detection using HPCs. Red texts highlight drawbacks, and black texts express the suggested tool/setup/model from this work. Solid dots (●) indicate the use of that tool/setup/model (column) by the reference (row), and hollow dimonds (◇) indicate the non-use of that tool/setup/model by the reference. Star (⋆) is our work. Our work avoids the drawbacks discussed in the table, and quantitatively analyzes how these drawbacks lead to the conclusion that HPCs can reliably detect hardware.

works (Demme et al., 2013; Kazdagli et al., 2016; Ozsoy et al., 2015; Khasawneh et al., 2015; Khasawneh et al., 2017) have not provided a numerical analysis of how micro-architectural events are selected. After the selection of events, we need to track these events and transform the measured HPC values to examples in machine learning models, i.e. feature extraction. Dividing examples into training and testing datasets for machine learning models are training-and-testing split. Previous works (Demme et al., 2013; Kazdagli et al., 2016; Wang et al., 2016; Singh et al., 2017) have training-and-testing split based on the examples (TTA1 in § 4.4.3) that the testing dataset can have the same examples produced by programs in the training dataset. However, in real life, it is unlikely that the offline training dataset can include all the malware that a user might encounter. We mark the use of data division based on examples as **Drawback III** in Table 1.1.

We observe that there is no cross-validation in some of the previous works (Demme et al., 2013; Wang et al., 2016; Tang et al., 2014), while other works (Ozsoy et al., 2015; Khasawneh et al., 2015; Khasawneh et al., 2017; Singh et al., 2017) present insufficient cross-validation, i.e. not every example in the dataset is validated. None of these works report standard deviations of detection rates with cross-validations. Without a substantial amount of cross-validation, we cannot assert the reproducibility of detection rates, since a model can have its high detection rates with specific training and testing datasets. We refer to no cross-validation or insufficient validations as **Drawback IV** in Table 1.1.

Previous works reported their results with double decimal precision (Demme et al., 2013). However, double decimal precision require at least 100 experiments in testing. With 10-fold cross-validation in the experiments, the total number of programs (benignware and malware) should be more than 1,000 programs. Thus, at least 1,000 programs are required to evaluate the machine learning models within numer-

ical rounding error of less than 1%. There are several works that use fewer than 1,000 programs leading to over-generalization (training and testing with insufficient cross-validation), or over-interpretation of the results (comparisons beyond rounding errors) (Demme et al., 2013; Kazdagli et al., 2016; Wang et al., 2016; Tang et al., 2014; Singh et al., 2017). This insufficient number of programs in the experiments is **Drawback V** in Table 1.1.

In addition to the drawbacks of the previous works, we found that there is no public access to their data or codes, which makes it hard to perform a direct comparison and examinations of the methods applied in these works.

We present all the tools/setups/models in various previous works in Table 1.1. In Table 1.1, rows are various works in HPC-based malware detection and columns are design choices of the tools/setups/models. The alternative shaded and white background represents different categories of tool/setup/model in malware detection using HPCs. Red texts highlight drawbacks and black texts express the suggested tool/setup/model from this work. Solid dots (•) indicate the use of that tool/setup/model (column) by the reference (row), and hollow diamonds (⋄) indicate the non-use of that tool/setup/model by the reference. Star ($\star$) is our work. The last column counts the drawbacks of the corresponding work. Table 1.1 shows that there are at least 2 drawbacks in each work.

1.3 Completed Research

The following work was completed as part of my Ph.D. Program:

- **Hardware Trojan Detection using Backside Optical Imaging:** In this work, we proposed to use backside optical imaging for HT detection. We first engineered fill cells to have maximum reflectance compared to standard cells. These highly reflective cells formed watermark. Any shifts, modifications to

these cells could be detected using backside imaging. By mapping the gate response to the locations and orientations of all the gates in the layout, we generated an optical response of the entire chip as the "golden response". We applied correlation between imaged results and generated "golden reference" to determine whether the chip has HTs (Chapter 2).

We then extended our work to create a library of responses of all the cells in the standard cell library. We illuminated multiple locations on each gate according to the size of the gates to have a unique response for each gate. We improved the accuracy and robustness of our method by using noise-based detection method in the comparisons between "golden reference" and imaged results. We also did process variation analysis and optimized the detection method.

Our approach does not require costly devices as low throughput techniques, such as delayering the metal layers and SEM. Compared to the traditional electrical detection method, our method images the layouts of ICs, which does not depend on the tests that are able to trigger the HTs. The ICs under test do not need to be powered up or calibrate for testing environments, such as thermal detection method. At the same time, backside imaging presents high-resistance against process variations during manufacturing. The infrared light imaging produces images with a resolution at the gate level. No previous imaging methods have been able to achieve.

- **High-Performance Low Energy Implementation of Cryptographic Algorithms on a Programmable SoC for IoT Devices:** In this work, we implemented various kinds of cryptographies on a programmable SoC for IoT devices. We measured the performance, power and energy consumption, and reconfigurability of our implementation. We implemented the same cryptographic algorithms in software and compared performance, power and energy to evaluate

the improvements achievable through hardware implementations. Our implementation showed that the Zedboard platform provided high performance, low energy consumption and reconfigurability for IoT devices (Chapter 3).

Our proposed FPGA implementation can replace the implementations in the existing library (OpenSSL), which enables performance boost and energy savings in various block cipher modes.

- **Malware Detection using Hardware Performance Counters with Machine Learning:** We identify the prevalent unrealistic assumptions and the insufficient analysis used in prior works that leverage HPCs for malware detection (Chapter 4). We perform thorough experiments with a program count that exceeds prior works (Demme et al., 2013; Kazdagli et al., 2016; Wang et al., 2016; Tang et al., 2014; Singh et al., 2017) by a factor of $2\times \sim 3\times$, and the number of experiments in cross-validations that is 3 orders of magnitude more than previous works. We harvest the HPC traces from 1,924 programs, 962 benignware and 962 malware, on our experimental setups. We achieve an F1-score (a metric of detection rates) of 83.39%, 84.84%, 83.59%, 75.01%, 78.75%, and 14.32% for Decision Tree (DT), Random Forest (RF), K Nearest Neighbors (KNN), Adaboost, Neural Net (NN), and Naive Bayes, respectively. We cross-validate our models 1,000 times and the F1-score of models in DT, RF, KNN, Adaboost, NN, and Naive Bayes is 80.22%, 81.29%, 80.22%, 70.32%, 35.66%, and 9.903%, respectively. To show how fragile the HPC malware detection system is, we show that one benignware (Notepad++) infused with malware (ransomware) cannot be detected by HPC-based malware detection. To ease the reproducibility and advance the community's efforts to assess the utility of HPC-based malware detection, we release all the code and data produced for this work under an open-source license.

Chapter 2

Hardware Trojan Detection Using Backside Imaging

2.1 Introduction

Integrated circuits (ICs) are the root of the trust in computing systems. Secure mechanisms rely on the the trust on the hardware underneath software stacks. However, the trust is broken due to fragmented supply chain of IC manufacturing. The large demands for IC chips has led to the globalization of the IC chip supply chain. Over the past two decades, IC design and manufacturing has become increasingly distributed across the world (Tehranipoor and Koushanfar, 2010). The standard IC chip production process consists of specification, design, fabrication, testing and packaging phases (Dar, 2019; Chakraborty et al., 2009; Karri et al., 2010). Many U.S. companies still design their IC chips locally. However, given that overseas manufacturing can support a range of technology nodes at a significantly lower cost, these companies prefer to fabricate their IC chips overseas.

Highly fragmented and distributed production brings efficiency and productivity to IC design and fabrication (The, 2019). However, given the pervasive use of the IC chips in both commercial and military domains, it is mandatory to ensure that the security of these chips has not been compromised. During the different phases of IC production, the ICs face threats from HTs (Karri et al., 2010; Tehranipoor and Koushanfar, 2010; Love et al., 2012; Waksman and Sethumadhavan, 2011; Hicks et al.,

2010), IP privacy (Roy et al., 2008), IC chip overbuilding (Chakraborty and Bhunia, 2009), reverse engineering (Torrance and James, 2011; electroiq.com, 2019; DARPA, 2019; iPh, 2019; How, 2019; Rev, 2019), side-channel analysis (Kocher et al., 1999; Schlösser et al., 2012; Koeune and Standaert, 2005; Koeune and Standaert, 2005; Genkin et al., 2014; Rohatgi, 2009) and IC counterfeiting (Rostami et al., 2014). HTs are malicious modifications or insertions of unwanted circuitry into the chip designs to sabotage the functionality or leak secret information. IP privacy discloses the intellectual properties of IC designs and causes economic damages to the IC company. Foundries can overbuild ICs and sell them to make illegal profits. Reverse engineering recovers designs of the chips to steal intellectual property. Side channel analysis focuses on using physical properties other than electrical signals on the chip to extract secret information from ICs. IC counterfeiting refers to fabricating unqualified products but labeling them as products from other companies, which causes indirect loss to companies that originally design the chip. Among all these threats, HT attacks are the worst attacks in hardware security. Therefore, our approach targets detecting the HTs inserted during fabrication.

The HT attacks can directly cause information leakage, system compromise and failures. HTs can control, modify, disable or monitor the IC chips (Rostami et al., 2014). Common HT insertion approaches are insertions of the malicious IP cores, design modifications, and layout modifications during the fabrication phase (Rostami et al., 2014). We can detect the HTs inserted using the first two approaches with functional verifications at different stages of the IC chip design process. In the third type of HT insertion, the attackers reverse engineer the physical layout and modify the design during manufacturing. The HTs inserted during fabrication have a very small impact on the delay and power consumption of the overall IC chip. Besides, HTs inserted during fabrication are designed to have an extremely low triggering rate

to hide them from standard tests. Unless manufacturing is closely monitored, the IC design company needs to use expensive reverse engineering techniques like delayering the chip, SEM imaging, and etc. to check for HTs. Hence, there is a pressing need for a new detection method, which can detect and locate HTs inserted during fabrication in a fast, accurate, and robust manner (Tsoutsos and Maniatakos, 2014).

We propose a novel optical imaging method that can rapidly and accurately detect HTs inserted during the fabrication. In our proposed approach, we use backside imaging to detect HTs inserted during the fabrication. We measure the optical responses from post-fabricated ICs, and compare the results to the responses from FDTD simulations performed on the chip. We can detect any compromises to the chip by identifying differences between the generated responses and the measured results. To this end, we have completed the following work:

- *Backside Imaging using near-IR light*

  We propose a new approach to generate a backside image of an IC chip, which we call the "golden reference", using the detailed physical layout of the IC chip and FDTD simulations of individual standard cells. We model the process of backside imaging of the IC chips by FDTD simulations. We compose the reference responses by mapping individual gate FDTD responses to the locations of standard cells and interpolate the response image. Any inconsistencies between measurements and reference response correspond to changes in the designs.

- *Noise-based Detection Method*

  We implement a noise-based detection method that compares the imaged data and "golden reference". This method performs better than our previously proposed correlation-based detection method. The noise -based detection exhibits robustness against noise, and is effective across different testbenches.

- *Engineering High Reflectance Fill Cells*

We engineer fill cells as high reflectance cells and strategically insert them before placing the cells of the digital designs to increase the contrast of imaging responses. We demonstrate that the insertion of extra fill cells improves the HT detection accuracy against noise.

- *Evaluations using Various Hardware Blocks*

  We show the effectiveness of our approach through a variety of testbenches. We use standard *Cadence* tool flows in *Nangate* 45nm technology to synthesize and place & route the circuits. The area of inserted HT vary from 0.1% to 12% of the total chip sizes. We use our method to detect the HTs in these testbenches under different signal to noise ratios. We also analyze the effects of process variations, different resolutions in imaging, and different detection window frame sizes on detection rates with different noise levels.

2.2 Threat Model

In this work, we assume that the attacker can modify, shift or replace the fill cells in order to accommodate the malicious hardware blocks in the victim's design. The attacker can get access to the GDSII files used for fabrication, but cannot change the RTL, gate level designs or any designs before the victim generates the GDSII files. We trust the IP blocks in the designs from any third party to be HT free.

2.3 Backside Imaging of IC Chips

We propose a dramatically different approach to detect HTs inserted in IC chips during the fabrication phase. Metals in IC chips are strongly reflective to near-IR light, while silicon is transparent to light at those wavelengths. Backside optical imaging of the fabricated chip enables us to extract the full standard cell layout of the chip with the watermarks, which in turn can be validated to detect any modifications

to the IC layout (Figure 2·1). In addition to the original metal lines of standard cells, we also embed a maximal amount of metal in the M1 layer in the fill cells during the design stage to increase the total reflectance of the design to improve detection accuracy. This strong reflected pattern forms the signature of chip design which will be used for HT detection.

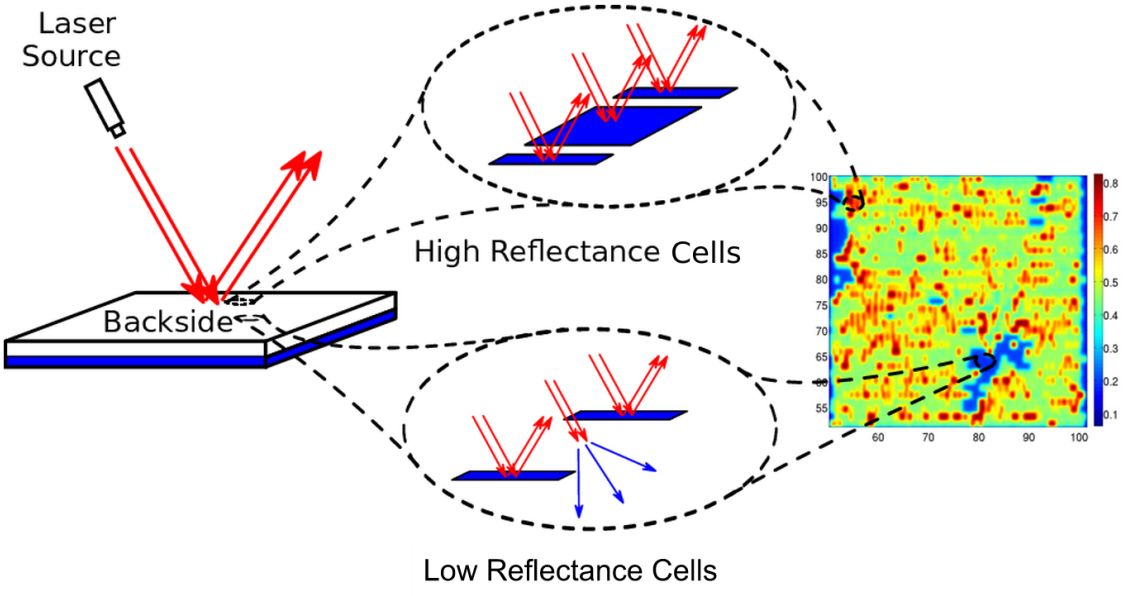


Figure 2·1: Backside Imaging of IC chips

Our optical measurements are highly modular and independent of the multitude of connections in the full IC chip. In addition, the physical principles behind the implementation of our watermarking scheme are also highly distinct from previous approaches. Although, like the first PUF (Pappu et al., 2002), it is an optical method that utilizes embedded scatters, their intended design and functionality are fundamentally different. The scattering in (Pappu et al., 2002) was explicitly designed to be random and impossible to predict and replicate. In contrast, determinism is essential to the functionality of our approach. We embed the watermark during the design phase and we can determine the optical response before fabrications. Our designed

signature must be predictable within our analytical range.

2.3.1 Optical Imaging Method

As the opaque metal layers prohibit front side imaging, backside imaging of integrated circuits is a well-established technique for failure analysis (Ippolito et al., 2004; Kindereit et al., 2007; Köklü and Unlü, 2010). Bright field images at near-infrared (IR) wavelengths (e.g. $\lambda \sim 1 - 2\,\mu$m) can be used for passive measurements for fault detection and localization, such as for inspecting the fidelity of the metal wires (Ippolito et al., 2005). In addition to passive measurements, the active functionality of the circuit can also be probed via techniques such as thermal imaging (Ippolito et al., 2004), which records power dissipation via heat generation or laser-voltage imaging (LVI) (Kindereit et al., 2007), which records the switching response of the transistors.

As integrated circuits shrink in size, a key challenge in these imaging techniques is to obtain sufficient spatial resolution to resolve each individual structure in each gate. The diffraction limit imposes a fundamental restriction on the maximum spatial frequency that can be imaged using conventional optical systems which limits the resolution. The resolution of an optical system can be determined by the size of the impulse response of the system which takes the form of an Airy function (Novotny and Hecht, 2006),

$$I(\rho) \propto \left[2\frac{J_1(2\pi\rho)}{2\pi\rho}\right]^2 \tag{2.1}$$

where $\rho = NAr/(M\lambda)$ is the image space coordinate. The size of the impulse response is

$$\Delta x = 0.61\frac{\lambda}{\mathrm{NA}} \tag{2.2}$$

Here, λ is the wavelength of the light in free space and NA is the numerical aperture ($\mathrm{NA} = n \sin\theta$) of the system, where n is the refractive index of the material in which the light propagates. J_1 is Bessel Function of the First kind.

Due to their high NA capability (~ 3.4), complex solid immersion lenses provide a high resolution for fault analysis of integrated circuits (Köklü and Unlü, 2010). In this work, we can eliminate the need for the high resolution and can rapidly and accurately detect malicious tampering and the presence of an HT at relatively low NA's (~ 0.8).

Our approach is based on the fact that for low NA's in the near-IR, we can achieve impulse response functions with widths on the order of the gate size in 45 nm or lower technology nodes (see Equation (2.2)). NA's of 0.14, 0.42 and 0.5 correspond to spot sizes of approximately 4.6 μm, 1.5 μm and 1.3 μm, respectively at $\lambda = 1.064\,\mu$m. These correspond to common near-IR commercial objectives capable of imaging over $0.1 - 1$ mm fields of view (several thousand to half a million gates simultaneously). An image collected in this manner (i.e. at low NA, without a solid immersion lens) would, rather than resolve the detailed substructure of individual gate, produce a slowly varying image that tracks the average reflectance of each gate over its area. Although the individual gates are comprised of unique layouts of metal lines, Figure 2·2 shows that with both 15 nm and 45 nm technology, the responses of three different gates have distinctive signatures across the spectrum.

Figure 2·2 shows that fill cells which are engineered to contain maximum amount of metal, while meeting the metal density design rule constraints. These fill cells achieve distinguishable stronger response compared to common functional cells. We can leverage these stronger responses to strengthen the contrast of the response images in order to achieve higher HT detection rates.

Therefore, backside imaging of IC chips can result in clear patterns depending

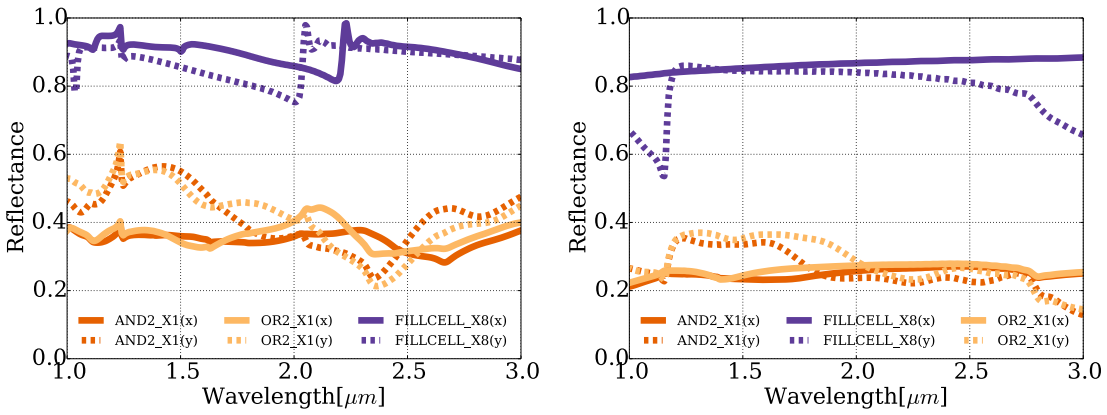


Figure 2·2: (a) The reflectance spectrum of functional gates and fill cells designed in 45 nm technology (computed via FDTD simulations). The response is computed for both X and Y polarizations of the illuminating field (solid and dashed lines, respectively). For X polarization, the incident electric field is polarized along the VDD and VSS rails. For Y polarization, the polarization is perpendicular. (b) The reflectance spectrum of functional gates and fill cells designed using 15nm technology. Fill cells still show much higher reflectance compared to functional cells.

on the standard cell layout. These patterns can serve as a robust, easily recordable optical watermark of IC chip. Any modifications through movement of cells or insertion of unwanted cells will result in a change in the watermark that can be measured with high fidelity. Imaging large fields of views provides the potential to perform these measurements on a large number of gates simultaneously. It also has the added benefit of considerably simplifying the required optical setup in comparison with commercial failure analysis tools. Our approach is a simple, rapid test to check for tampering of the IC chip at the fabrication stage. It utilize off-the-shelf tools to embed our engineered signature and it can be seamlessly integrated into typical IC fabrications.

2.3.2 Technology Scalability

While we demonstrate our approach at the 45 *nm* technology node level, we expect it to scale well to smaller technology nodes because our technique *does not* require high resolution imaging. We can estimate the efficacy of our proposed technique as gate sizes are reduced by considering the size of the impulse response of our optical system as described by Equation (2.2). The gate sizes reduction will limit our ability to detect a shift in the position or a change in the magnitudes of response of cells, and therefore the performance of our technique.

We can show that the reflectance difference between different gates in optical imaging can be clearly distinguished. Case in point, in a 45 nm technology node (See Figure 2·2(a)), the reflectance difference between AND gate and OR gate can be as much as 0.2, which is almost 50% of the total reflectance from AND2. Replacement of AND gate with an OR gate results in 50% change in reflectance variation. In contrast to the AND gate and OR gate, the reflectance of fill cells is as high as 90%. The distinguishable difference between various gates scales in a 15nm technology node (See Figure 2·2(b)). The gates in 15nm technology node have smaller sizes, and also the reflectance of all the gate is weaker compared to gates in 45 nm technology node. The difference between reflectance of different gates also scales with the technology nodes. However, the reflectance ratio between AND gate and OR gate still remains the same. Note that the AND gate is still around 50% of the reflectance of OR gate. By comparing the relative reflectance, we can detect the changes in the gate type. Moreover, in our method, the uniqueness of the image of a design is determined using the responses from individual gates and the neighboring effects. The pixel size of our optical imaging method is $0.1\mu m^2$ while the smallest gate size of 45 nm technology node is $0.4 \times 1\mu m^2$. This means that each gate can have multiple comparison points in HT detection. In our case, the minimum gate size of 45 nm technology is $0.26\mu m^2$,

which can contain as much as 30 data points. Even with 15nm technology, each gate can still have up to ~ 15 data points, which is sufficient for HT detection.

By applying interpolations on sampled points, our technique also scales well in modeling areas much larger than single gate. This enables measurements over a large field of view and therefore a large number of simultaneously imaged gates. Considering an IC with $\sim 1\,\mathrm{B}$ transistors, an average of 4 transistors per gate corresponds to approximately $250\,\mathrm{M}$ gates. For a gate size of $1.3\,\mu\mathrm{m}^2$ corresponding to the 45 nm technology node, using a resolution of ≈ 400 nm and MHz acquisition rates would mean that the whole chip would be imaged in a few minutes. This is a significant improvement over the high NA imaging methods, where the sample is typically scanned with a scan size of $10 \sim 100nm$ as opposed to 400 nm that we consider in this work.

2.3.3 Optical Simulation Methodology

FDTD computation for one gate area requires roughly half an hour computation time. The computational time grows exponentially with areas to simulate, i.e. a larger gate with area of $5 \sim 6$ times of single gate takes $1 \sim 2$ days to finish. Simulating the response for the whole layout of a typical $3 \sim 4$ cm^2 IC chip is computationally infeasible. Thus, in this work, we simulate each individual gate to construct a library of responses from near-IR excitement. The constructed response library determines the response of a given layout through simulations. Our method does not require the need for in-field measurements of "golden reference". The FDTD method is widely used in modeling different microscopes, and it has been shown to work both analytically and experimentally (Török et al., 2008; Çapoğlu et al., 2011).

In the previous work, we considered the FDTD simulations for each gate with periodic boundary conditions by simulating illumination on infinite tiling of the same gate (Zhou et al., 2015). With these boundary conditions, we simulate the response of each gate separately and obtain their optical response by calculating the reflected

power of each gate, normalized by the source power. Since the size of the impulse response is of the order of the size of the gates, the normalized power is then convolved with the impulse response of the optical system to simulate the image of a chip. The periodic boundary conditions allow us to obtain the optical response at each wavelength simultaneously, at a reduced computation time. We consider only the 6 basic gate types (AND, OR, NAND, NOR, XOR, XNOR) in the evaluation of our optical watermarking technique. In this work, we consider the complete gate library, which includes gates that are larger than the impulse response. We extract the internal metal structures and contacts from back-end GDS files and rectlinearly decompose polygon structures into rectangular structures for FDTD calculations. In order to calculate the optical response of those gates, we develop a rigorous simulation method for gates whose sizes are larger than the illumination spots. For this purpose, instead of using plane wave illumination with the same gate as the boundaries, we use other gates as boundaries of the illuminated gate and use a focused beam with a predetermined NA and perfectly matched layer (absorbing) boundary conditions. The optical response of each gate is obtained by normalizing the power of the reflected light confined within the predetermined NA by the source power. In this set-up, we can obtain the optical response of each gate for one wavelength and one polarization. In addition, since we are using a focused light illumination, the focused spot needs to be scanned at multiple locations as opposed to the case with plane wave illumination. Selecting the scan size to be the same as the size of the impulse response allows us to obtain all information within a gate, while ensuring a rapid acquisition time.

Modeling active regions and polysilicon transistor gates are beyond our scope for simulations, as their contribution is expected to be significantly less than the metal structures. Though in modern CMOS process, the polysilicon and diffusion layers are metalized through a silicide process, these metalized layers are 5-10 nm thick,

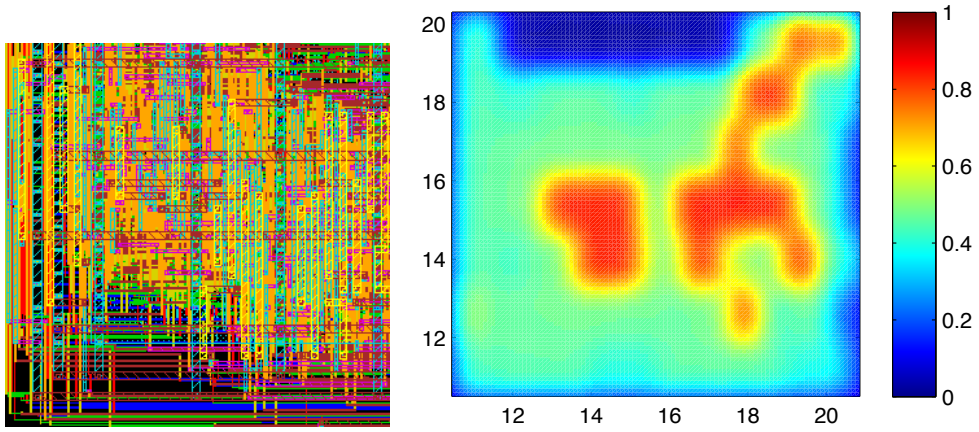
therefore their contributions towards reflection is negligible. As some of the gates are larger than the size of the impulse response, the focused spot is scanned starting from the center of the gate and moving to the sides with the specified step size until the majority of the illuminated area does not overlap with the gate. Note that given a layout, placing the responses from the constructed library for each gate at its location in the layout results in a set of response values on an irregular grid. As the experimental measurements are taken on a regular grid with predetermined scan size on the chip, we interpolate the simulated data to a regular grid using bicubic interpolation (Keys, 1981). To construct the overall layout on a regular grid, we set the horizontal step size to be equal to the size of the impulse response, and vertical step size to be equal to the height of the gates.

2.4 HT Detection Process

In this section, we use the AES-T100 circuit from Trust-HUB (Tru, 2014) to explain our proposed method of HT Detection using near-IR imaging. To detect the HT in the example circuitry, we need to (i) generate "golden reference" from simulations in Section 2.4.1 and (ii) automate the cross-comparisons between the measured response and the "golden reference" to detection HTs in the chip in Section 2.4.2. We present the results in Section 2.4.3.

2.4.1 Methodology to Generate "Golden Reference"

To generate the "golden reference", we synthesize, floor plan and place&route a digital block, using *Cadence* RC and Encounter tools. From the final fabrication ready Geometrical database for information exchange (GDSII) file generated from place&route the designs, we can extract the locations and orientations of each gate and export the data to a DEF file (Gate Level Descriptive Files). We simulate the near-IR response of each gate in the standard cell library and create a library of gate responses. Ac-



(a) (b)

Figure 2·3: (a) Physical layout of a $10\mu m \times 10\mu m$ region of the AEST100 hardware block. (b) Backside image (reflectance value) of the $10\mu m \times 10\mu m$ region. The fill cells have the highest reflectance.

cording to the dimensions, locations and orientations of all the gates from the GDSII file, we map the FDTD calculated responses of individual gates into the locations of corresponding gates in full-circuitry near-IR image. Figure 2·4(a) and Figure 2·4(b) presents the layout of original GDSII design and near-IR imaged color map from the FDTD calculation, respectively. To form the image in Figure 2·3(b), we convolve the reflectance map of AES circuitry with an impulse response from near-IR imaging.

In our attack model, an attacker needs to replace, remove or shift fill cells to accommodate room for HTs (Tru, 2014). Any replacements or shift will result in a different optical image compared to the one we generated from the original design, because fill cells are significantly different from functional cells. In other situations, when attacker only replaces the functional cells with functional cells of the HTs, changes in the responses of the cells and the neighboring cells result in changes of overall optical responses. Since most of the gates have responses of 0.5, the fill cells enhance the contrast of overall signature to strengthen the resistance against noise. In both cases, engineered fill cells are critical in improving the accuracies of HT detection.

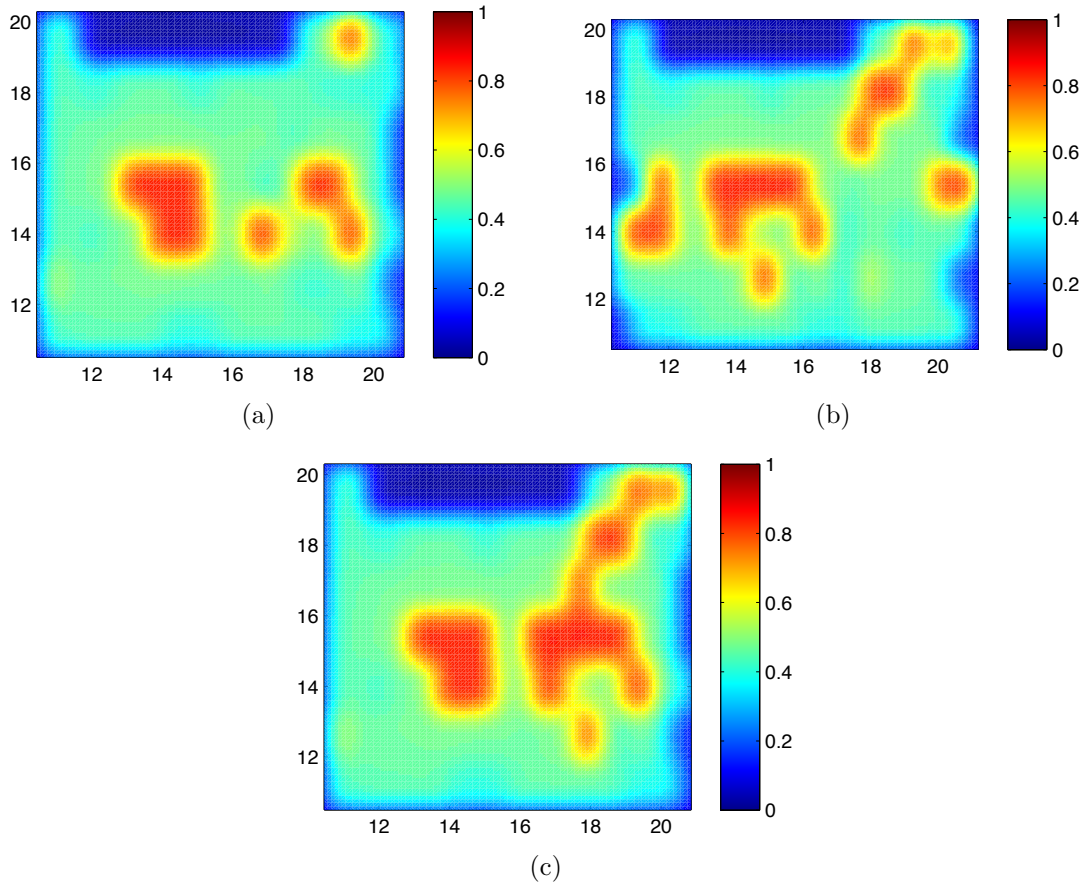


Figure 2·4: (a) Backside image (reflectance value) of the $10\mu m \times 10\mu m$ region where the fill cells are replaced with functional gates that constitute the HTs. (b) Backside image (reflectance value) of the $10\mu m \times 10\mu m$ region where the bottom 3 rows are shifted by 5 μm to the left to make room for cells that constitute the HT. (c) Backside image (reflectance value) of the $10\mu m \times 10\mu m$ region when the functional cells are replaced by a different set of cells that constitute the HT.

In one example, Figure 2·4(a) and Figure 2·4(b) shows that an attacker replaces/shifts fill cells, and the response of optical imaging can result in changes compared to the responses of the design without HTs in Figure 2·3(b). Any modifications or shifts of these gates (red areas in Figure 2·4) are more prominently observed in the near-IR image than non-engineered gates (blue and green areas in Figure 2·4). Figure 2·4(c) shows an example of replacing functional cells with functional cells of HTs. The responses from engineered fill cells significantly increase the contrast of the overall

image. Although in Figure 2·4(c), modifications are not easily detectable to the naked eye, we can calculate the differences between measured results and use the "golden reference" to discriminate the HT inserted ICs from non-tampered ICs.

2.4.2 Correlation Method

To reliably and accurately detect HTs in large quantities requires an automated comparison method to cross-validate between "golden reference" and measured results. In our previous work (Zhou et al., 2015), we use 2D-correlation coefficients as the threshold metric for in HT detection. In this work, we further improve the results by using noise based detection method.

In image processing, the correlation coefficient describes the similarity between two images. If the correlation coefficient is larger than the threshold, we consider the circuitry as non-tampered. If the correlation coefficient is smaller than the threshold, we consider it to be tampered. Our 2-D correlation metric can be expressed in Equation 2.3.

$$\rho_{M_X M_Y} = \frac{cov(M_X, M_Y)}{\sigma_{M_X}\sigma_{M_Y}} \tag{2.3}$$

where *cov* is the co-variance of the two matrices $M_x \& M_y$ and σ is the standard deviation. Here, we use M_X to denote the near-IR imaged matrix and M_Y to represent the imaged matrix with noise. In our previous work (Zhou et al., 2015), in order to minimize false positives and false negatives in HT Detection, we determine the threshold on correlation value between the measured results and "golden reference". We consider the design tampered, whose correlation results are lower than our predetermined threshold. We perform Monte-Carlo simulations of false positive rate and false negative rate against a variety of detection thresholds on AES-T100 testbench (AES with HTs from (Tru, 2014) in Figure 2·5). From Figure 2·5, we conclude that

assuming that the HT inserted in our test chip has an HT of a similar size as HT in AES-T100, we can optimize the correlation threshold to minimize the error rates at an SNR at 10.

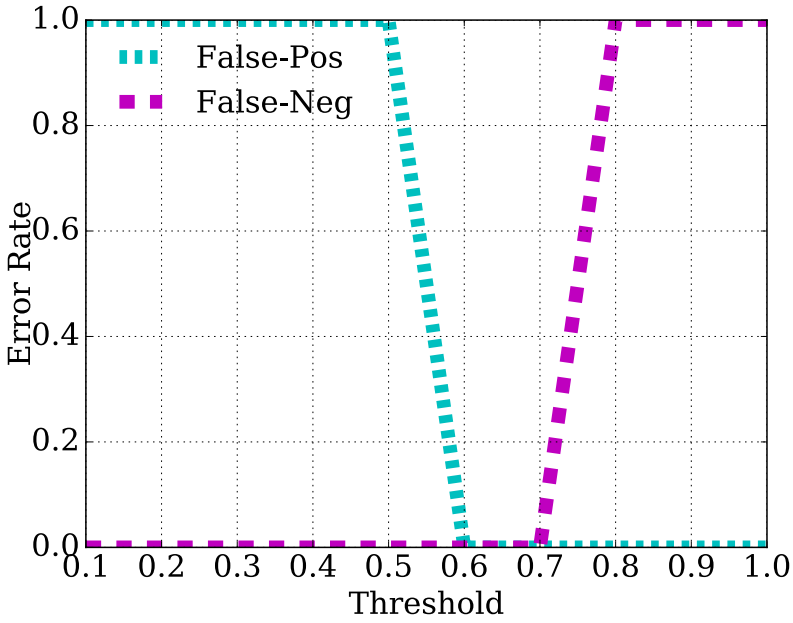


Figure 2·5: By having Monte-Carlo simulation of detection error rate against detection threshold, we calculate false positive and false negative error rates versus threshold. The optimized detection threshold should be 0.65 for a fixed SNR of 10.

2.4.3 Results

We used testbenches from Trust-HUB (Tru, 2014) and (Wei et al., 2012a) to evaluate our proposed approach. Using 45 nm Nangate library, we generated the GDS files for testbenches. The area of all testbenches are listed in Table 2.1. Figure 2·6(a) and Figure 2·6(b) show the error rates versus SNR for the testbenches from (Wei et al., 2012a) and (Tru, 2014), respectively. The sizes of these HTs range from 0.06% to 9.83% of the total area in the testbenches. Figure 2·6(c) presents the evaluations of our method in all these testbenches. Each dot in the Figure 2·6(c) represents the error rate analysis of SNR varying from 10^{-8} to 0.1. If all error rates in this SNR spectrum

Testbench	Area without Trojans $[\mu m^2]$	Trojan Area $[\mu m^2]$	Trojan Area Percentage (%)
c1355	266	4.522	1.7
c1908	251.6	0.798	0.317
c2670	333	0.798	0.240
c499	257	4.522	1.760
c880a	197.6	0.798	0.403
PIC100	4215.0	351	8.33
PIC200	4215.0	89.6	2.13
PIC300	4215.0	253.2	6.01
PIC400	2969.9	292.1	9.83
AES100	274177.6	253.2	0.0923
AES200	274177.6	169.5	0.0618
AES700	322705	297.4	0.0922
AES900	318359	267.3	0.0840
AES1000	274177.6	251.1	0.0915
AES1200	323348	450.3	0.1392
AES1700	320670	1388.3	0.4329

Table 2.1: Area (in μm^2) In this table, we show power consumption of all the testbenches that we implemented in this paper.

fall within the error rate range, we use corresponding color dot to represent it. For example, all the error rates of c880a testbench are 0% in the SNR ranging from 10^{-8} to 0.1. We use a blue dot to denote it in the table. We use other 3 kinds of color dots to represent if all error rates are in $0 \sim 1\%$, $1 \sim 3\%$, and $> 3\%$, respectively. From this we can see that our correlation detection method presents high performance in c880a, and c1355. Nevertheless, it does not have the same performance in PIC testbenches, because the threshold is optimized for small testbenches but not universally all the testbenches.

2.5 Noise-Based Detection Method

In Section 2.4.2, we presented the correlation coefficient as an evaluation metric for HT detection. Correlation enables a straight forward comparison between the reference and the imaged data. However, it suffers three major drawbacks. First, it is hard for a tester to determine the threshold value for detecting HTs in Device Under Test

(DUT). Testers do not have prior knowledge of HT sizes. Various HT sizes and different SNRs require different threshold values in order to maintain high fidelity. In practice, multiple thresholds hinder the flexibility and applicability of this approach. Second, in correlation detection method, we average out all our information of the response from the ICs to one metric in comparison, which limits the information for differentiating tampered chip from benign ones. One metric is suitable in concluding the overall group behavior; yet it is not ideal for differentiating individuals from the group behavior. In our case, we need to identify the individual gates in the layout. The information that we can get from data sampling contains means and variations of each gate response. Although we cannot use all of the sampled information, using as much information as possible would yield better HT detection results. Hence, we propose a noise based detection method to enhance robustness of our method. Noise based detections evaluate the fitness of the noise introduced to the response image in our pre-determined model. Different noise models require corresponding modeling of the noise in the detection method. Since measurement noise and process variation is the main source of the noise, we assume that the noise is Additive Gaussian white noise(AWGN) (Popović and Taflove, 2004; Zjajo, 2014). The discussion of other modeling of the noise sources is beyond the scope of this work.

2.5.1 Methodology Explanation

To explain our use of the noise-based detection method, we use an example of replacing one NAND gate with an AND gate in the c1355 testbench. In Figure 2·7, the four figures represent results from four phases of this detection method. After we extract gate locations and orientations from the GDSII files, we combine the data we calculated from the FDTD simulation of each individual gate, and map each gate reflectance to a gate location. If we assign values from the FDTD simulation to each pixel in single gates, we can get Figure 2·7(a). As we discussed in Section 2.3, we

need to interpolate the image in Figure 2·7(a) in order to generate realistic images. We apply bicubic interpolations to the image in Figure 2·7(a).

We use M_X to denote post interpolated results shown, in Figure 2·7(b). M_X matrix is the "golden reference". Taking noise into consideration, we use M_Y to denote the response image with noise. If $M_Y - M_X$, is AWGN (Figure 2·7(c)), we consider that the imaged circuit is not tampered, since the difference between the "golden reference" and measured response consists of noise only. We use M_R to denote $M_Y - M_X$. If M_R is not purely AWGN, then the measured matrix may have other components in its mean or variation which can indicate a HT. Figure 2·7(d) presents the imaged figure after the subtraction from the "golden reference". In the comparison between Figure 2·7(c) and Figure 2·7(d), we can see the difference in the top left corner, which indicates an HT. We use the data matrix M_R to test if the noise is normally distributed. We show both false positive and false negative tests of gathered data with different resolutions. In false negative tests, we expect the M_X matrix to only have normally distributed noise. We use *D'Agostino's X-squared test* to identify the shape of the expected probability density function (PDF). *D'Agostino's X-squared test* is a metric to evaluate how well one statistic set of data fits normal distribution's PDF. *D'Agostino's X-squared test* has a suggested p-value to evaluate the skewness of a data set compared to normal distribution, which is independent from mean or variation of the data set. In experiment tests here, we use 0.05% as the p-value threshold as it is commonly adopted in estimation of Gaussian Distributions. This threshold value describes the skewness of subtracted image, which is independent from testbenches. For the false positive tests, we consider several examples with HTs embedded in the layout. A similar testing method is applied to test whether the M_R matrix can be identified as non-normally distributed signals.

Most side channel HT detection techniques that use a physical property as the

"side channel" consist of two parts, data gathering and data analysis. In the first part, under a given excitation on the side channel, a benign chip will have a unique response which will not match the response of a chip with a HT. For example, when using thermal imaging (Hu et al., 2013) in HT detection, the thermal map is the physical property of a chip under test. The physical excitements are the input electrical signals of the chip, and the output response is the infrared image of the chip. Our proposed technique utilize near-infrared illumination as the physical property, near-IR laser signals as excitement and near-IR images as responses. In the second part, by comparing the simulated results to backside image of DUT image, changes due to HTs can be detected. Here, our proposed noise based comparison method does not require any prior assumptions about the HTs before testing and is independent from any statistical information from any data gathering method. Therefore, a noise based detection method can also be a generalized method applied in all kinds of data analysis in HT side channel detection.

2.5.2 Results

In Section 2.4, we discuss the difference between a correlation and noise based detection. Here, we use the noise based detection method for data analysis. We use false positive and false negative error rate versus SNR(Signal to Noise Ratio) to evaluate the performance of our proposed HT detection method. We use small circuits from (Wei et al., 2012a), PIC circuits and AES large circuits from (Tru, 2014) as our testbenches. In small circuits from (Wei et al., 2012a), we have different designs for each HT testbenches. In PIC and AES circuits from (Tru, 2014), we have two basic designs, AES and PIC, while the HTs are different in each testbench.

Similar to the correlation method, for each test case, we synthesize, floor-plan and place&route each circuit with *Nangate45nm* technology using *Cadence* RC and Encounter tool flow to generate GDSII files. We generate and use those files along

DEF files, with both location and orientation information of each gate to generate the optical response data. We interpolate the reflective response image to reference image for later comparisons. Depending on the resolution used for experiments, we adjust the image interpolation density accordingly.

We then use interpolated data in the noise-based detection method to determine whether the circuit has a HT or not. For a given circuit, we divide the imaged matrices into batches of smaller areas in the layouts. Each batch has the same area size and represents the reflectance data from a certain area of the image. We denote one area a detection window frame. If we detect HTs in one of the window frames, we fail the test of that batch. Considering that the simulation time for area of $1mm^2$ is on the order of weeks, we choose a window frame with a size of 10×10 μm^2 for small test benches, and 250×250 μm^2 for large testbenches. In Section 3.5, we are going to analyze the impact of the window frame size on detection accuracy.

Figure 2·8(a) shows false positive and false negative rates in c499 and c1908, two testbenches from (Wei et al., 2012a) at SNR from 10^{-8} to 10^{8}. Since triggers in these testbenches are designed for low probability triggering rates, they are extremely hard to detect using functional testing (Wei et al., 2012a). In this work, we illustrate HT detection in the testbenches of triggers instead of the payload, given that the area of payload circuits are much larger than the areas of triggers. Figure 2·8(b) shows the results of evaluations in the AES testbenches from the Trusthub website using noise detection method. For c1908, c499 from (Wei et al., 2012a) and AES-T1000, AES-T1200, AES-T1700 from (Tru, 2014), we can see that error rates drop to zero with SNR starting from 10^{1} in the false negative error rates.

The testbenches in Figure 2·8(c) are AES encryption engines with HTs, and PIC testbenches are tampered PIC16x64 circuit. In the AES testbenches, we apply a window frame of $250\mu m \times 250\mu m$ in the simulation setup. Figure 2·8(c), similar to

Figure 2·6(c), summaries the all the testbenches used in this evaluation. Each dot in the Figure 2·8(c) represents the error rate analysis of SNR varying from 10^{-8} to 0.1. If all error rates in this SNR spectrum fall into the error rate range, we use corresponding color dot to represent it. As we can see from Figure 2·8(c), most of the false positive and false negative results from various testbenches have low error rates in our HT detection method.

2.6 Optimizations of HT Detection

To analyze the robustness and applicability of our approach, we consider the impact of: process variations (Section 2.6.1), different resolutions and window frame sizes (Section 2.6.2), and different pattern insertions (Section 2.6.3) on our proposed method. Process variations have become the major concern for all HT side channel detection methods. The power and area overhead of HTs can often be smaller than the power and area variations due to process variations. Since process variations are inevitable in IC production, detection methods should be robust enough to overcome that and still have high HT detection rates. In our data analysis method, the detection window frame size determines the number of *D'Agostino's X-squared tests* in one chip test and resolution of sampled data decides the quantity of sampling data in single test. Both of these two factors affect the detection accuracy in the Monte-Carlo Simulations. Here, we evaluate the impacts of both of these factors on the detection accuracy of our method. In addition to engineering fill cells to increase the detection accuracy, we propose to strategically place the fill cells in the design in order to improve our method.

2.6.1 Process Variation

In order to model the impacts of process variations on the near-IR reflectance, we simulate the process variations on the functional cells by stretching or shrinking the

metal structures in the horizontal and/or vertical direction. In Figure 2·10, we show the XOR2_X1 as an example gate to explain our methodology to analyze the effect of process variations. We use ±10% process variations in the horizontal and/or vertical dimension and perform FDTD analysis on all these structures, except on the power rails. In Figure 2·10, we show examples of XOR2_X1 cell with no process variation and with ±10% in the horizontal dimension for the M1 metal layer and M1 contacts. In this modeling, we only consider the process variations happening on the inner structures of M1 and contacts. We simulate the responses of 8 kinds of basic cells with/without process variations on the spectrum from $1\mu m$ to $3\mu m$ on X and Y polarizations. The results show that the determinism of gate response does not change due to process variations, which means that the difference in optical response caused by process variation is smaller than the difference between two different gates. In order to minimize the impacts by process variations, we pick the wavelength and polarization that is the least affected by process variations, which is $1.2\mu m$ and Y polarization for our evaluation of the noised-based detection method in both Section 2.4 and Section 2.6.4.

2.6.2 Resolution and Window Size

In noise-based data analysis, resolution determine the quantities of data to be processed and detection window size decides the number of individual *D'Agostino's X-squared tests* in HT detection of one chip. Intuitively, higher resolution imaging can better differentiate the details of the designs. Yet the physical property of the side channel in data gathering constraints the resolution of imaged results. In order to quantitatively analyze impacts of resolution on accuracy, we use the down-sampled near-IR imaging matrices to conduct comparisons between different resolutions, as we do not have the data from other side-channel techniques. In near-IR imaging, we use $0.1\mu m$ as the resolution for interpolation. Here, we apply $0.2\mu m$ and $0.4\mu m$ as

interpolation resolutions in simulations to evaluate the potential lower resolution to use in HT detection. These resolution comparisons show that the decisive factor in side-channel HT detection is resolution.

In Section 2.5, for a given testbench, we divide the imaged matrices into batches of the same size. We denote the area of a batch as the detection window frame size. If we detect HTs in one of the window frames from the tested circuitry, we assume that all the batches fail the test. In this way, we utilize much more data than one value for evaluation. Since our test is based on the distribution of the noise, if one window frame covers a much smaller area, the data set of white noise is more likely to fail the noise based test. If one window frame covers much larger area, the HTs might not be detected since the noise will average out the small modifications in the data set.

2.6.3 Pattern Insertion

To further improve the detection accuracy of our method, we propose to strategically place fill cells into the floor plan before *place&route* the design. We engineer the fill cells to have higher reflectance than other cells. Compared to other cells, which have reflectance around 50%, engineered fill cells have reflectance of almost 100%. As we discussed in the Section 2.4, HTs often replace fill cells with functional cells, or move fill cells to make room for inserting fill cells corresponding to functional cells. Extra pre-placed fill cells secure locations on the floor plans, since fill cells are the most distinguishable gate in backside imaging. Any changes to these cells are more recognizable than other functional cells. In the case when HTs replace functional cells with other set of functional cells, the pre-placed fill cells can also contribute to improvements of the detection rates. The reflectance of cells is not only determined by its own reflectance but also the reflectance of neighboring gates. Not only these fill cells can have a more distinguishable signature than functional cells, but they can

contribute their response to the neighboring cells. Changes of reflectance near fill cells can lead to more changes on the eventual reflectance image, compared to functional cells.

In designing the pattern, we ensure the engineered fill cells can cover as much area as possible. In many of the cases, the pre-placed fill cells do not increase the overall area of the design, since there are extra spaces in the original design for fill cells. To design the pattern inserted, we need to use the minimum amount of cells to maximize the coverage of fill cells. We experimented using multiple designs with different sizes of fill cells and different distances between cells. The optimal solution without increasing the area of the design, which yielded highest detection rates among various SNR, gave the pattern in Figure 2·11.

2.6.4 Results

Process Variations

As discussed in Section 2.6, process variations can be minimized by imaging with the wavelength which is most resistant to process variations. We simulate the responses of 8 kinds of standard cells with/without process variations on the spectrum ranging from $1\mu m$ to $3\mu m$ on X and Y polarizations (Figure 2·9). In Figure 2·9, the thick red line represents the reflectance of functional cells that are free of process variation. The other lines represent the reflectance with process variations. At the wavelength of $1.2\mu m$, the changes to the reflectance for the XOR2_X1 cell is less than $\pm 5\%$ in all cases. In most of the cells, process variations cause less than $\pm 5\%$ variations in the reflectance in X polarization imaging, and at most $\pm 10\%$ in Y polarization imaging at the wavelength of $1.2\mu m$. Only in FILLCELL_X1, the reflectance of the fill cell is unusually low, due to narrow, thin cell shape with much less metal fillings in the horizontal direction.

	1	2	3	4
0.19	0.266	1.064	2.394	4.256
0.38	0.532	2.128	4.788	8.512
0.57	0.798	3.192	7.182	12.768

Table 2.2: Area of Inserted fill cells[μm^2], 0.19, 0.38, 0.57 represents the width of each fill cell in the array in [μm], 1, 2, 3, 4 represents number of rows and columns of the array.

Resolution and Window Size

Here, we use 5 different testbenches, c1355, c1908, c2670, c499, and c880a to evaluate the effects of resolution and window frame sizes on the detection rate. Figure 2·12(a) and Figure 2·12(b) shows error rate versus signal to noise ratio for imaging in three testbenches from (Wei et al., 2012a). Figure 2·12(a) uses $0.1\mu m$ interpolation resolution for imaging, while Figure 2·12(b) uses $0.2\mu m$. In the comparison between Figure 2·12(a) and Figure 2·12(b), the results clearly show that higher resolution provides performance improvements at lower SNR.

Here, we conclude that in smaller testbenches, improving resolution from $0.1\mu m$ to $0.2\mu m$ does not affect detection rate much. Figure 2·12(c) summaries the relationship between resolutions and window frame sizes between different testbenches. If all error rates in this SNR spectrum fall into the error rate range, we use corresponding color dot to represent it. Since we are using 45 nm technology, if we decrease the resolution to $0.4\mu m$, many of the metal structures with minimum designed sizes become blurry in imaging. In Figure 2·12(c), error rates in the $0.4\mu m$ column are much lower than the previous two resolutions. In the detection window size analysis, error rates increase as the detection window size increases, due to the imaged response of the HT can be averaged out in large detection windows. In Figure 2·12(c) shows that high resolution imaging can increase the HT detection accuracy.

Pattern Insertions

By strategically placing the engineered fill cells before placing the designs, we can further improve detection accuracies. As shown in Figure 2·11, our fill cells starts from bottom left corner. The distance between left and bottom edges are $1.14\mu m$ and $1.4\mu m$, respectively. We fixed the distance between two fill cells to be $4\times$ the width of the fill cell in the horizontal direction and $1\times$ the height of the fill cell in the vertical direction. We vary the size of fill cells and number of the fill cells inserted in the design to find out the optimal amount of fill cells that we need to place in the floor plan. In Figure 2·13, the table shows that more fill cells placed in the design do not necessarily improve HT detection accuracy. Table 2.2 shows all the area increase for placing fill cells. The extra area cost of all the designs is less than 5% of the design area. Since our experiments require the aspect ration of the design to be $1:1$, there are no extra gates needed to be inserted in the design.

2.7 Nanoantenna Implementations

To further differentiate between various gates, we leverage the uniqueness of farfield imaging response for designed metal structures embedded in individual gates. We refer to these metal structures as nano-antennas. We engineer the nano-antennas as asymmetrical plasmonic structures (metal rectangles), and place them between two neighboring gates. These structures can reach high reflectance that is unique to the illumination polarizations, wavelengths and angles. Since our design of nano-antennas are asymmetrical, the nano-antennas have sharp contrast in polarizations, with designed illumination wavelengths and angles. We compute the reflectance under the signature illumination condition (polarizations, wavelengths and angles) and form a dictionary of gate signatures. After the fabrications of the ICs, we measure the farfield responses of these individual gates, and form the map of each gate pair on

the layout. By comparing the locations of pre-fabricated design and the measured map, we can easily identify the modified, shifted, or replaced gates. Our nano-antenna designs are resistant to process variations, since the signature of the gate is constrained by the aspect ratio of the nano-antennas.

In this work, we modified Nangate 45nm libray (nan, 2019) to illustrate the HT detection using backside imaging with nano-antennas. We engineered nano-antennas for the following gates in the library as examples of modified gates with nano-antennas: AND, NAND, OR, NOR, XOR, XNOR. We engineer the placement of the nano-antenna into the empty spaces between the gates, instead of placing the nano-antenna inside the gate. We achieve a 300% improvements of the gate farfield reflection response in the reflectance signature. In addition to the implantation of nano-antennas, we enhance the optical reflectance of our design using the periodic structures in the standard gate layout, resembling a grating around the nano-antenna. Figure 2·14(a) displays the metal one layer layout of AND, OR and XOR logic gates as an example, with the periodic structures shown in the black frame. Our proposed optical structures consist of the plasmonic nanoantenna, surrounded by the periodic structures, shown schematically in Figure 2·14(b).

2.8 Conclusion

In this work, we developed a new technique for HT detection. Our technique uses near-Infrared light to image the metal structures from the backside of the IC. We use the FDTD simulations to generate the imaged results from the metal structures in the library. After the design, we can extract the locations of each gate from our design. Combined with the near-Infrared image results, we can generate the reflection results of the design as “golden reference”.

In order to achieve the high contrast in backside imaging to detect HTs, we mod-

ified the fill cells in the library to fill with metal. Any modifications, shift or replacements of these fill cells can be detected by our technique. We evaluate our techniques with different noise levels and different observation windows. Our analysis shows that we are able to detect HTs that occupy less than 0.1% of the total chip area (which makes it very difficult to detect these HTs using power and delay analysis).

We also engineer the nano-antennas inside the gate pairs to increase the differences between the signatures of the gates. The nano-antenna enhances the reflectance of the different gates under unique different illumination frequency and angles. The results have shown that with the multi-illumination method, we are able to differentiate six different gate pairs using Nangate 45nm technology.

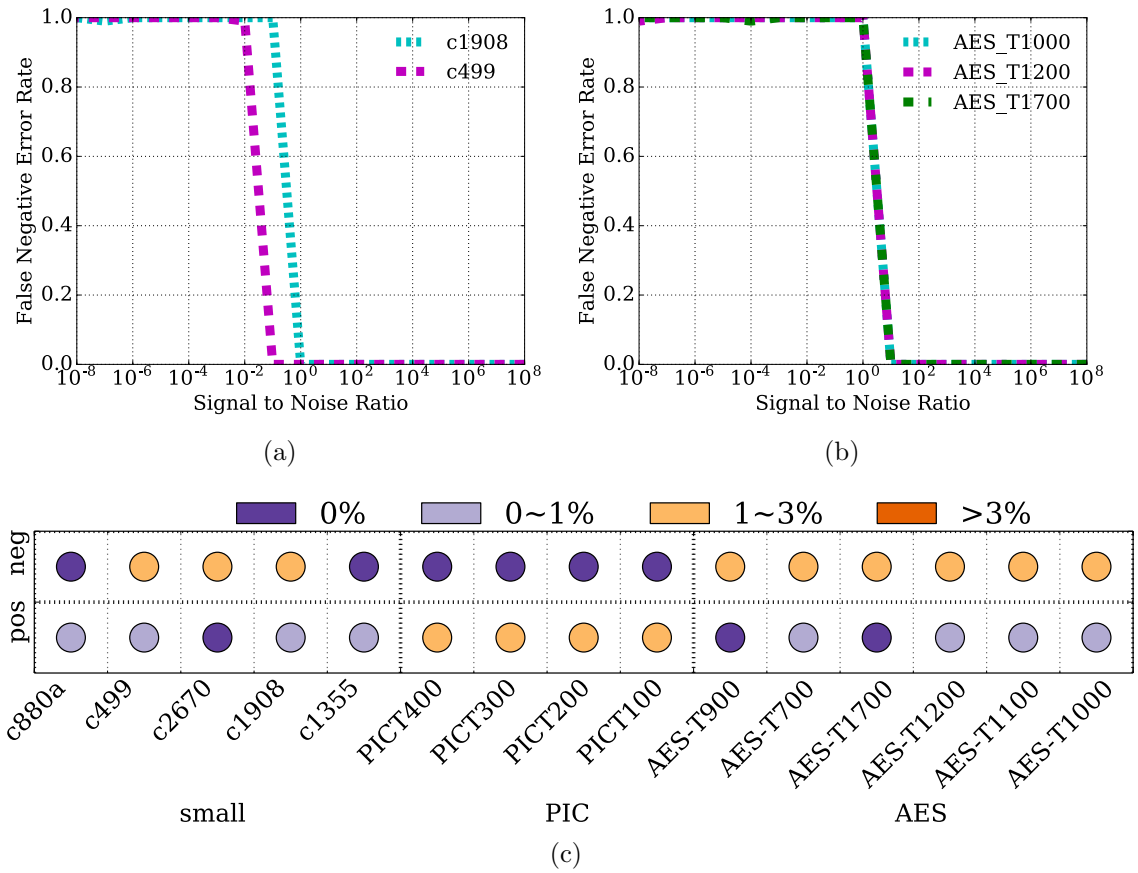


Figure 2·6: (a) shows false negative error rates versus SNR of c1908 and c499 testbenches from (Wei et al., 2012a) using correlation comparison method. (b) shows AES-T1000, AES-T1200, and AES-T1700 testbenches from (Tru, 2014) using correlation comparison method. (c) is a summary of various testbenches from both (Wei et al., 2012a) and (Tru, 2014). We use colors in the legend to denote the error rates at SNR from 10^{-8} to 0.1 belonging to the corresponding range.

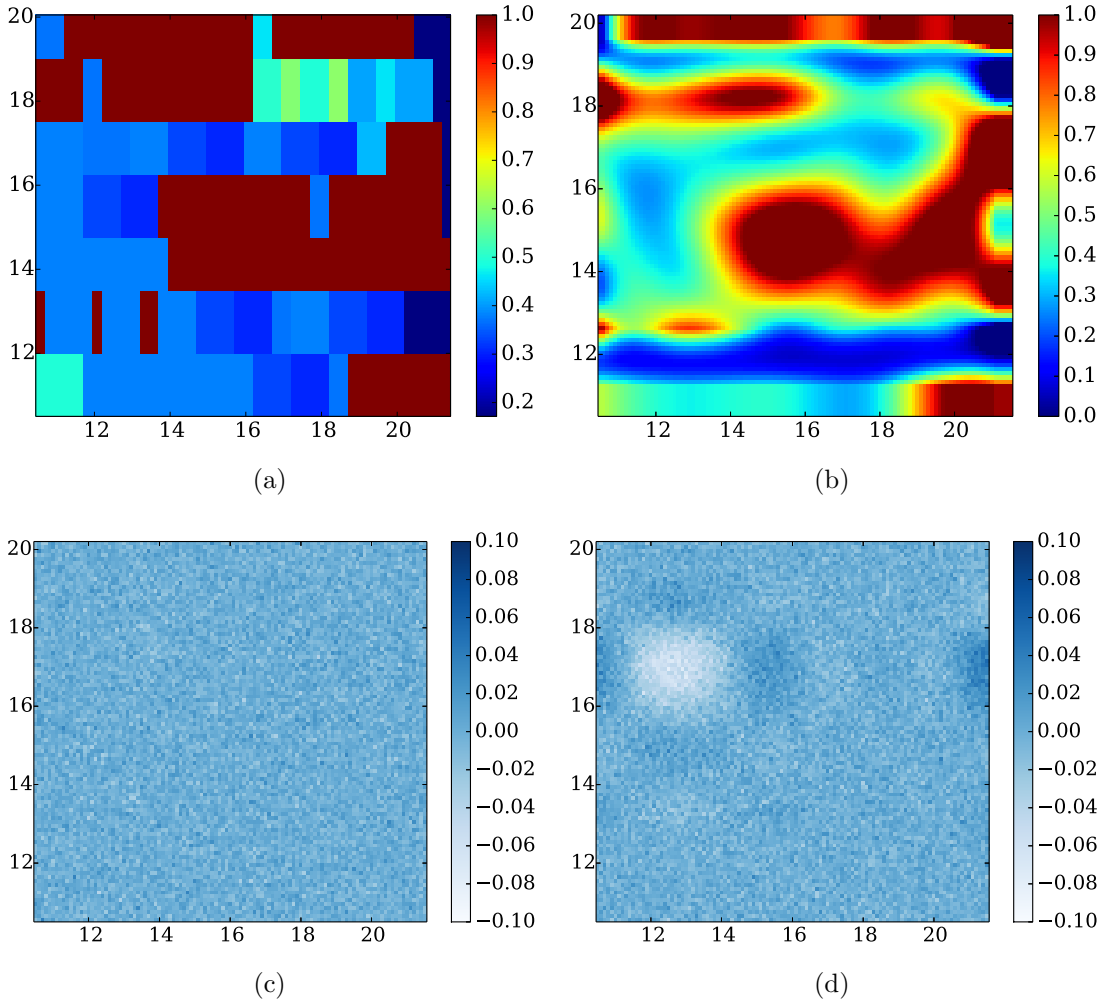


Figure 2·7: (a) shows part of reflectance layout in testbench c1355. We use single gate FDTD results of the optical responses to represent every pixels in the gate location. (b) shows the cubic interpolated results from (a) as the reference of non-tampered circuit, which we denote it M_Y. (c) is AWGN with a variant 0.01 of the same area. (d) is one tampered circuit example, in which we replace one NAND gate with an AND gate. We use the measure response, M_Y, subtracted by the "golden reference", M_X, to get the image in (d).

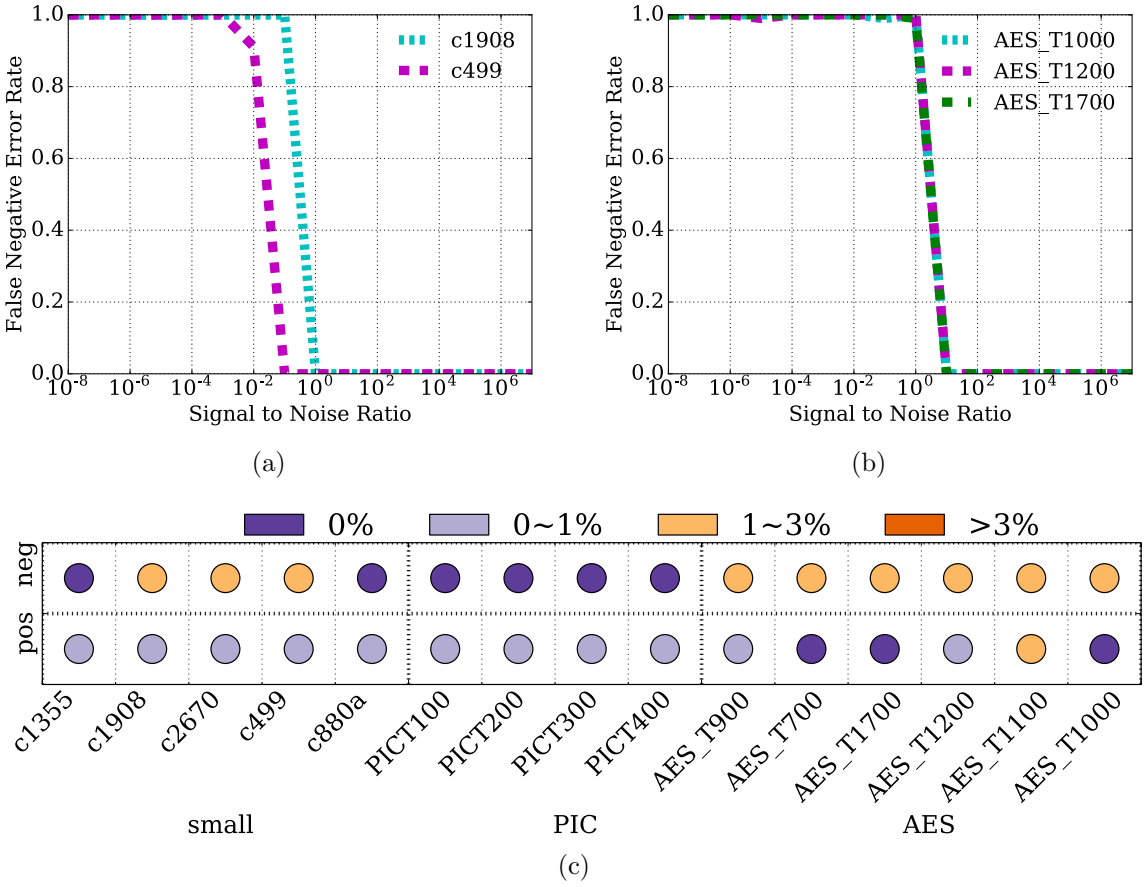


Figure 2·8: (a) shows false negative error rates versus SNR of two testbenches from (Wei et al., 2012a) using noise detection method. (b) shows three testbenches from (Tru, 2014) using noise detection method. (c) is a summary of various testbenches from both (Wei et al., 2012a) and (Tru, 2014). We use colors in the legend to denote the error rates at SNR from 10^{-8} to 0.1 belonging to the corresponding range.

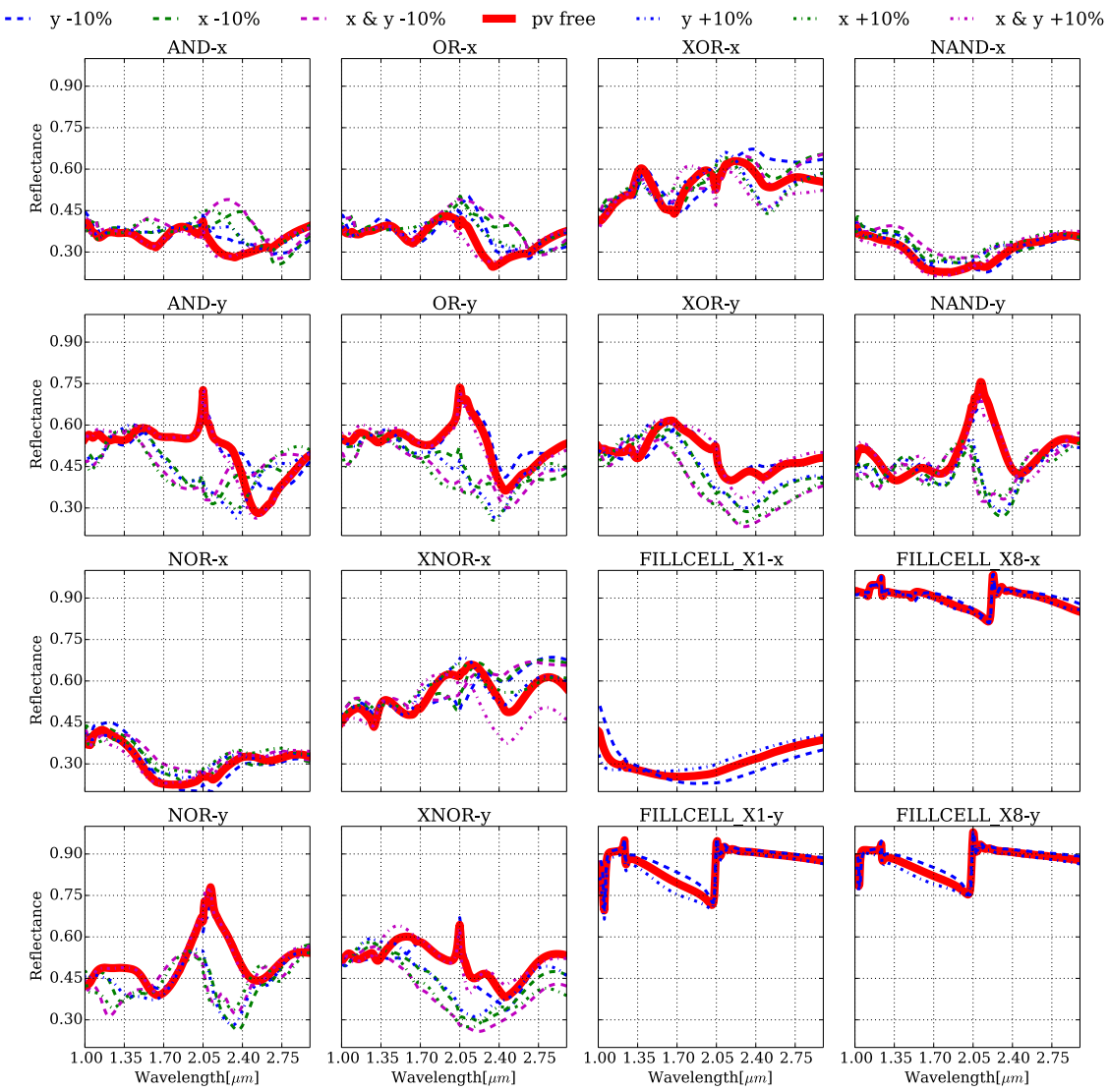


Figure 2·9: Reflectance under ±10% variation in dimensions of metal1 structures through near-IR laser of wavelength from 1μm to 3μm. X polarization laser reflectance is on the left and Y polarization laser reflectance is on the right.The thick red line represents the process variation free reflectance of XOR2_X1 cell. The other lines are the reflectance with process variation.

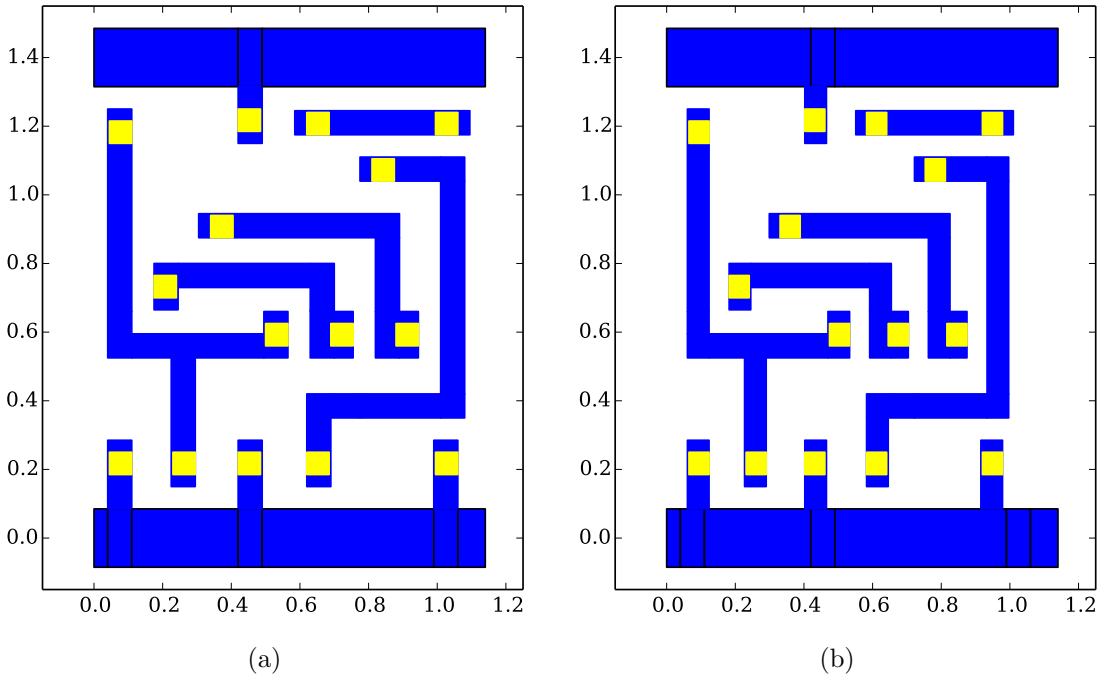


Figure 2·10: (a) Process variation free XOR2_X1 gate. (b) XOR2_X1 with 10% process variation in the X dimension. Here, all the metal structures inside of the gate have been compressed by 10% on the X dimension to model process variations.

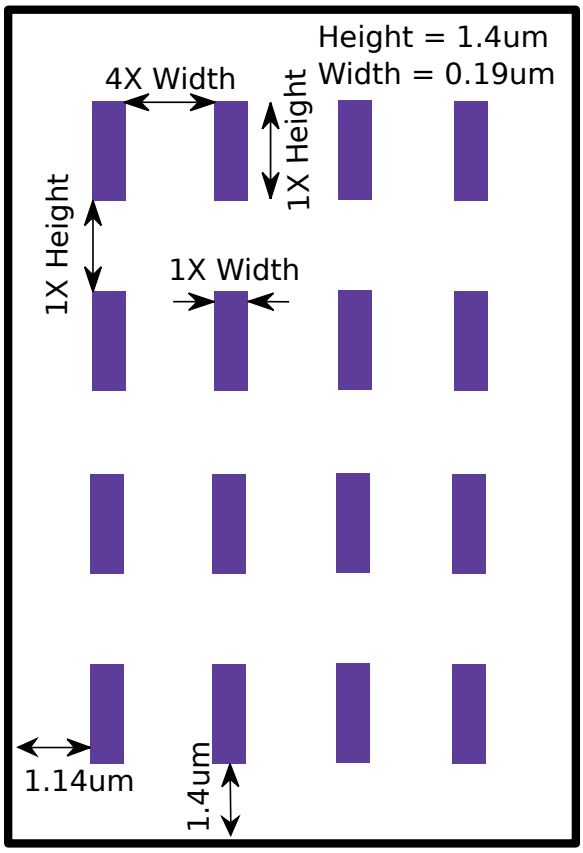


Figure 2·11: Fill cell pattern that we inserted before *place&route* of the design. We put this array of fill cells to secure these regions from shifts or replacements of fill cells and replacement of functional cells with other set of functional cells.

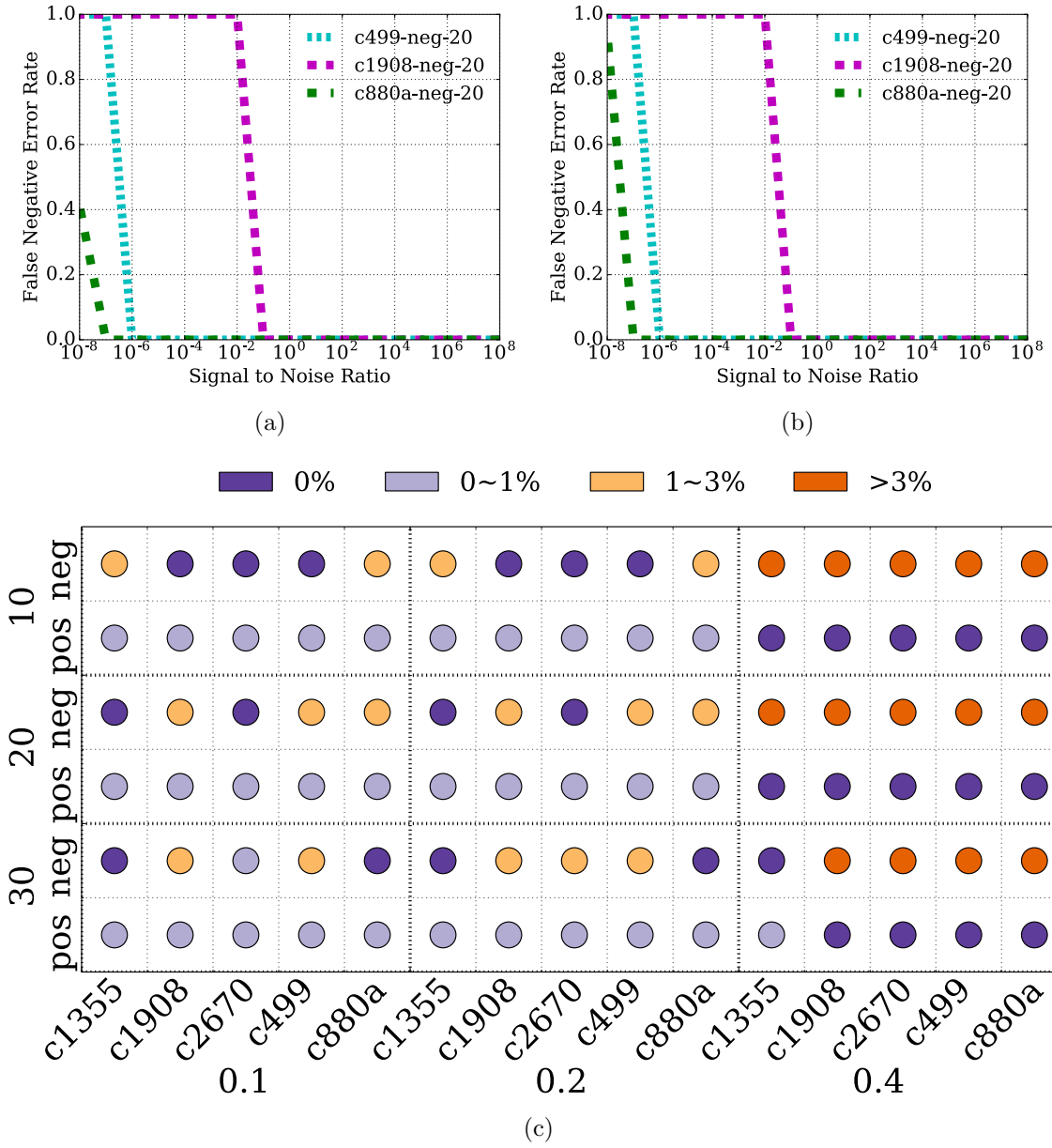


Figure 2·12: (a) shows false negative error rates versus SNR of three testbenches from (Tru, 2014) with an imaging resolution of $0.1\mu m$. (b) shows three of testbenches from (Tru, 2014) using noise detection method with an imaging resolution of $0.2\mu m$. (c) is a summary of various testbenches from both (Wei et al., 2012a). We use colors in the legend to denote the error rates at SNR from 10^{-8} to 0.1 belonging to the corresponding range. Here, 10, 20, and 30 refer to detection window sizes $10\mu m$, $20\mu m$, $30\mu m$, On the X axis, 0.1, 0.2, and 0.4 refer to resolutions of the image as $0.1\mu m$, $0.2\mu m$, and $0.4\mu m$, respectively.

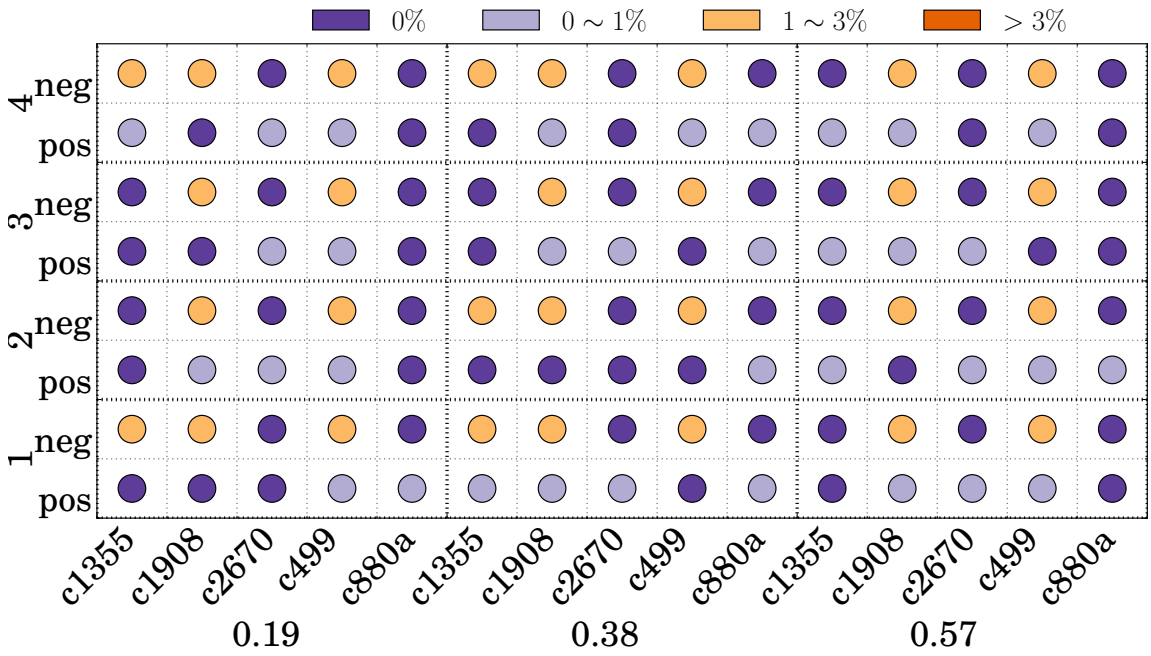
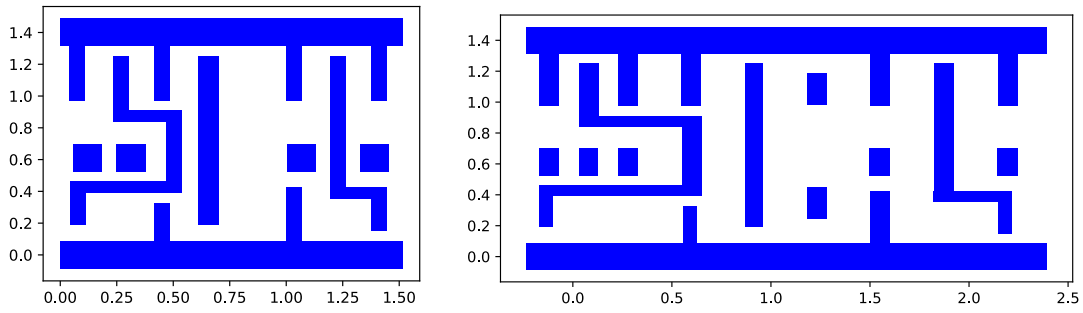


Figure 2·13: This figure shows a table of detection rate with different designs of patterns. In all the patterns, we use same number of rows and columns of fill cell arrays. On the y axis, the number represents number of rows and columns that we used in fill cell array. On the x axis, the number represents the size of fill cells in the array.



(a) (b)

Figure 2·14: (a) shows the metal 1 layer of the AND and NAND gate pair. (b) shows the metal 1 layer of the AND and NAND with 322 nm distances between metal structures and 200 nm nano-antenna between the two gates

Chapter 3

Cryptographic Algorithms on a Programmable SoC for IoT Devices

3.1 Introduction

IoT devices are low-energy devices that are connected through networks for communication, performing sensing and remote control. They are deployed in our daily lives as well as municipal and military facilities, which requires secure system operations. Secure system operation relies on both the computing and storage hardware of the system and also the communication channels used by the systems to communicate with the external world. Most of the IoTs leverage strong primitives to protect the confidentiality and integrity of the messages sent among themselves. It is extremely important for the Internet of Things (IoTs) to protect the communication channels between devices. IoT devices share some properties with embedded systems, but have unique characteristics and requirements when it comes to their constrained energy budget and extensive lifetime after deployment. Moreover, IoT devices frequently handle sensitive information, such as command and control signals for smart homes and factory floors, or alerts from smoke and fire detectors. Thus, to keep this information safe from prying hands of adversaries, our society would be best served by leveraging strong cryptographic primitives that can guarantee the integrity and confidentiality of IoT data. The primitives that provide these features are commonly classified into symmetric and asymmetric cryptography and secure hash functions.

While strong cryptographic primitives could have avoided some of the most recent security breaches in this area (e.g., (Miller et al., 2012; Miller and Valasek, 2014; Illera et al., 2014; Hernandez et al., 2014; Zonouz et al., 2014)), the thorough use of cryptography to protect IoT data has not materialized yet. Furthermore, the growth of data-volume in the IoT space is projected to be unprecedented. For example, Cisco predicts that the amount of data produced and consumed by $4 \sim 5$ billion IoT devices will grow to 15 exabytes (a billion billion bytes) by 2020 (Cis, 2016). Protecting the data at that scale will become increasingly challenging (Mayer, 2009).

The increasing demand for data has led to the following conflict. On the one hand, capabilities of IoT devices are constrained by a low power budget, can be as low as 6.45 μ watts (Klinefelter et al., 2015) in energy harvesting applications. On the other hand, cryptographic operations are required to protect all the data produced and consumed by the IoT devices. Unfortunately, vendors of IoT devices frequently trade-off the increasing demand of power hungry cryptographic operations by omitting prudent security practices. For example, instead of generating keys for different AES block cipher, the vendors use the same key for different blocks (ECB block cipher mode), which results in low entropies for the ciphered texts.

The prospect of growth, the device longevity of the IoT devices, and the observation that cryptographic operations are relatively costly when compared to regular system functionality, call for novel solutions for IoT devices to provide cryptographic primitives that are scalable, flexible, and energy efficient. Historically, energy efficient and scalable implementations of cryptographic algorithms have been introduced by chip manufacturers as dedicated crypto-engines. For example ARM, Intel, and AVR, all provide hardware accelerated Advanced Encryption Standard (AES) crypto-engines in their ARMv8 (ARM, 2016), x86 (Int, 2016a), and Atmel (AVR, 2016) platforms, respectively. These solutions come in the form of dedicated ASIC

units embedded with the processor. ASIC implementations provide performance and energy efficiency advantages but they are not flexible. The longevity of IoT devices necessitates flexible cryptographic functionality. For example, cryptographic algorithms that are found insecure (e.g. DES) should be replaced with more secure algorithms (e.g. AES). Cryptographic algorithms with shorter key length but higher security levels (e.g., Ecliptic Curve Cryptography) can replace outdated algorithms (e.g. RSA) in order to improve efficiency.

While the lifespan of a modern cryptographic algorithm hopefully extends beyond the lifetime of any IoT device, less severe changes and improvements also benefit from increased flexibility. For example, individual implementations of cryptographic algorithms frequently fall victim to side-channel attacks. Case and point, is a differential power analysis side channel attack (Kocher et al., 1999), which was countered with a more robust design (Oswald et al., 2005). However, a side-channel vulnerabilities in an ASICs cannot be fixed, short of replacing the device. Thus, to achieve high performance, energy efficiency, and flexibility, we propose to implement the cryptographic primitives in a Field Programmable Gate Array (FPGA). This choice is motivated by the observation that the CPU architecture of choice for IoT deployments is ARM. Moreover, many ARM platforms designed for the IoT easily combine general purpose processors (GPPs) with an FPGA substrate. A similar move is expected to occur in the commodity PC realm as a consequence of Intel's recent acquisition of Altera (Int, 2016b). In this work, we design and thoroughly evaluate the performance, energy efficiency, and flexibility of various cryptographic algorithms. To this end, we evaluate each algorithm in a realistic setting where the cryptographic functionality is implemented in an FPGA and exposed to Linux user-space programs through a simple modification of the OpenSSL (Ope, 2016) cryptographic library. Specifically, we perform our evaluations on a Digilent Zedboard, a System on Chip (SoC) platform that

features an ARM core connected to an FPGA. This setup further illustrates that our solution can seamlessly integrate with existing software that depends on the hugely popular OpenSSL library. For example, the bare Ubuntu Linux operating system distribution contains as many as 113 packages depending on the OpenSSL library. Our implementation seamlessly benefits all these applications without further modifications to these packages. We have completed the following work towards providing high-performance low-energy implementations of cryptographic operations on SoCs.

- To provide IoT devices with high performance and energy efficient cryptographic primitives, we propose a flexible hardware solution based on commodity off-the-shelf FPGAs (Section 3.3.3).

- To demonstrate feasibility, we implement cryptographic engines for symmetric and asymmetric cryptographic algorithms, as well as cryptographic hash functions on the Zedboard platform in a ready-to-use system. We develop our functions in application program for evaluations (Section 3.4).

- We achieve flexibility in online re-configuration of FPGA hardware and block cipher modes in software. As our FPGA-based symmetric encryption engine is integrated into the OpenSSL library, we inherit support for the various block cipher modes implemented by OpenSSL (Section 3.4).

- We synthesize implementations of AES, Rivest & Shamir & Adleman (RSA), Data Encryption Standard (DES) and Secure Hash Algorithm (SHA) hardware as obtained from Opencores (ope, 2016) (Section 3.5).

- We thoroughly evaluate these implementations along the dimensions of performance, energy, and power. To assess improvements along these dimensions, we also compare the FPGA-based implementations with their respective software-only counterparts (Section 3.5).

3.2 Threat Model

We trust the secret sharing system but not the sensor or controller communication channels. We also trust the physical devices that run the systems. The attacker can perform eavesdropping, man-in-the-middle, or any other attacks towards the communication channel.

3.3 Background

3.3.1 Cryptographic Algorithms

The basic classifications of the common cryptographic algorithms are symmetric encryption, asymmetric encryption and hash functions. Cryptographic algorithms are often found to be flawed and then updated (Curtin, 2005).

Symmetric encryption or secret key cryptography encompasses algorithms that use the same key to encrypt and decrypt. DES was used as the standard symmetric encryption algorithm for almost 35 years. In 1999, DES was broken in DES I, DES II and DES III contests (Curtin, 2005). As a result, the National Institute of Standards and Technology (NIST) officially abandoned DES in 2005. In its place, in 2001, NIST introduced AES as a mathematically efficient and elegant cryptographic algorithm (why, 2016b). AES became the standard encryption algorithm after DES is abandoned in 2005. Since then, it has become the standard symmetric encryption algorithm, and is used to encrypt government documents and daily web browsing sessions.

Asymmetric cryptography uses different keys for encryption and decryption, one public key and one private key. In asymmetric encryption, RSA was developed in 1977. Since RSA has never been broken, it is still being widely used. In 2000, ecliptic curve cryptography (ECC) was developed, which could provide equivalent security with much shorter key length than RSA. The NIST reports that the ECC needs

224 bit key length to achieve the same level of security as RSA with 2048-bit key length (Barker et al., 2007).

Many versions of hash functions have been widely implemented and replaced in the past three decades. In 1990, MD4 was invented, which was later developed into MD5. In 1993, the National Security Agency (NSA) moved from MD5 to SHA. After two years, a weakness was found in the algorithm and it was updated to SHA1. SHA1 was a popular hash algorithm for nearly ten years. Hashing collision means that hash function map to the same value even with different input values. Hashing collisions are not acceptable in cryptographic hash functions, since collisions in cryptographic hashing can be exploited for identity stealing. In 2006, SHA1 was found to have collisions. In order to achieve collision free, today, SHA256, which is a version of SHA2, has been widely accepted as secure hash function (why, 2016a). Considering the constant breaks in hash function, NSA worried that SHA256 may get compromised some day due to previous breaks in secure hashing functions. Thus, NSA prepared a back up secure hash function to replace SHA256 later. A SHA3 competition was held by NIST to find the algorithm that is far different from SHA2 and AES to reduce the risk that all these cryptographic algorithms are found broken simultaneously. In 2015, SHA3 competition had a winner, Keccak. SHA3 was officially released in (NIS, 2016), while SHA2 is still in use.

3.3.2 Cryptographic Accelerators

In the last decade, crypto-engines have been implemented in hardware in various ways. Their implementations can be classified into four basic categories according to their designs in architecture (Bossuet et al., 2013): custom processor, crypto coprocessor, crypto processor, and crypto array.

Custom processor is a General Purpose Processor (GPP) that has a custom crypto-unit embedded inside it. Intel, ARM and AVR have adopted this design in their

processors (ARM, 2016; Int, 2016a; AVR, 2016). The crypto-engines operations have been included in the ISAs and compilers have been adopted to emit opcodes for the instructions of cryptographic operations (Bossuet et al., 2013; Sakiyama et al., 2007; Hämäläinen et al., 2007).

Crypto coprocessors contain a main GPP and several other coprocessors with each coprocessor containing a custom hardware implementation of a unique crypto algorithm. In order to achieve a balance between the need for flexibility and data transfer efficiency, the crypto coprocessors are often implemented with FPGAs instead of ASICs (Chaves et al., 2006; Wang et al., 2010; Vaslin et al., 2007; Saarinen, 2014). None of the coprocessors listed (Chaves et al., 2006; Wang et al., 2010; Vaslin et al., 2007; Saarinen, 2014) in are shown to be re-configurable.

Crypto processors (Martin et al., 2008; Grand et al., 2009; Theodoropoulos et al., 2008) are dedicated cryptographic multiprocessor system that include GPP and full processors with crypto-engine instructions. As a result, crypto processors are not designed for general purpose computing, but they are dedicated to crypto-related computing. Many proposed architectures are Very Long Instruction Word (VLIW) architecture crypto-processors.

Crypto arrays (Theodoropoulos et al., 2008; Elbirt and Paar, 2005) are groups of GPPs coupled with cryptographic logic units. They are coarse-grained reconfigurable architectures for large systems. They are designed to have large number of function units interconnected by the network. The system is not designed for tuning each single function unit, but for maximizing the system performance.

Most of IoT devices do not need high performance crypto processors or even large system that has crypto arrays. IoT devices, however, do require GPP as the main processing unit, and so we propose that the GPP should be implemented with hardware core designed for general purpose. The encryption engine for the IoT devices

should however be implemented as an accelerator on FPGA. Unlike custom hardware designs, the proposed design has reconfigurability properties whereby one can update the system if the algorithms are upgraded.

3.3.3 A case for FPGA based Crypto

Performance, energy consumption, and security are three major aspects for IoT devices. Using software based cryptographic operations on IoT devices enable us to upgrade the IoT device security. Nevertheless, these software based cryptographic operations are expensive in terms of energy for IoT devices. Hence, custom hardware is used in cryptographic implementations as it has less energy consumption and also better performance. Custom hardware implementation of the algorithm is a viable option, as it has performance boost and lower power consumption compared to its software implementation. Yet cryptography implementations are frequently changing compared to the lifetime of IoT devices. The IoT devices can be employed for 5-10 years (iot, 2016); on the contrast, the OpenSSL library is updated once a month on average (Ope, 2016). These custom hardware implementations cannot be reconfigured with upgrades in the implementations of cryptographic algorithms.

The FPGA clock speed is much slower than the clock in a general purpose processor. However, the FPGA can complete the computation intensive functions much faster than the software implementation. For example, software implementation of AES can take up to 600,000 cycles on an ARM machine, while a hardware implementation only needs 20 cycles (ope, 2016). Even though FPGA clock speed can be 10× slower than GPP, the FPGA implementation can still be up to 3,000× faster.

In addition to the performance boost, FPGA implementation also consumes much lower power to complete the same amount of computation. Compared to GPP, FPGAs do not have extra operations such as instruction fetching or instruction decoding. Once data are loaded in FPGA, all the operations in the FPGAs contribute to the

computations of the function. Without redundant operations, FPGAs have much higher energy efficiency in computational-intensive computations.

FPGA provides a lot of benefits in performance and power, but it also has some drawbacks. Since the manufacturing technologies are different for FPGAs and GPPs, manufacturers only provide GPP and FPGA on different dies, but connected in the same package. In some platforms, they are even in different packages. One of the key drawbacks is that in order to access discrete FPGA, GPPs are required to use off-die or off-chip channels, which have higher latency compared to on-chip channels. System on chip solutions like Zedboard, where GPP and FPGA have been integrated on the same chip, can communicate through the Advanced eXtensible Interface (AXI) bus. AXI ports are used for off-die, high speed communications, such as CPU communicating with main memory. These Zedboard platforms have lower communication cost and are viable. Thus we have used Zedboard as our target system.

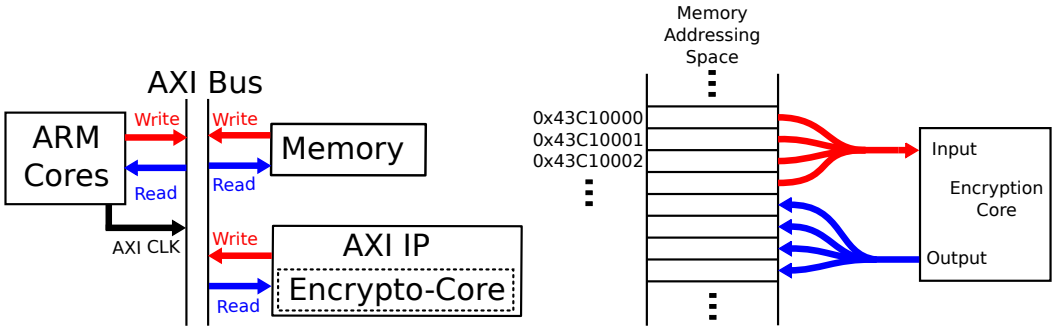


Figure 3·1: System Architecture for AES Encryption Engine

3.4 Experimental Setup

To evaluate the use of FPGAs as an energy-efficient re-configurable substrate for crypto-engines, we use AES, RSA, and SHA as examples of symmetric crypto, asymmetric crypto and hash functions, respectively. We demonstrate our approach on Zedboard platform by *Digilent*. The Zedboard platform uses the *Zynq 7000* SoC

chip containing two *ARM9* cores and one *Zynq 7000* FPGA. The FPGA substrate interfaces directly with the system main memory through AXI ports. The persistent storage on the Zedboard has two partitions, one is the boot partition for loading FPGA bitstreams and starting the Linux Kernel. The other partition contains an installation of the Linaro Linux distribution (Lin, 2016). The boot partition contains three files: one file for mapping bitstreams to FPGA, one file for Linux Kernel, and the device tree file defines all the devices that are present in the system. To minimize unwanted energy consumption, we select minimum device tree, which boots up the Linux system and configures the *Zynq 7000* FPGA. We build a system with the device tree we created, which consists of no other functionalities but Linux running on the ARM core, AXI peripheral blocks and encryption IP cores as it is shown in Fig 3·1. *Vivado* provides custom AXI IP core wrapper, which integrates well with CPU communication channels on the one hand, and wraps the custom design inside the IP wrapper on the other hand. The ARM core provides 512 32-bit channels as communication channels for communications at maximum. The interfaces of these channels on the FPGA side can be synthesized into registers on the programmable logic. On the ARM core side, these channels are memory mapped I/Os, which can be accessed by memory operations.

We downloaded the hardware implementation of AES, DES, RSA and SHA for evaluations, from Opencores (ope, 2016), synthesized them in the custom AXI wrapper and mapped them on the Zedboard. The input, output and configuration bits are connected to communication channel interface registers, and the AXI wrapper along with our custom designs are mapped to the FPGA. These registers can be addressed with memory offset provided by *Vivado* after the synthesis of the wrapper. With the calculated memory locations of these registers, offloading encryption operations consists of the writing phase and reading phase. During the writing phase, plain texts

and keys are written to the corresponding memory locations. In the reading phase, encrypted data are copied out to the output of encryption function. With FPGA acceleration, the cryptographic functions only consists of mapping memory location pointers to the memory locations of corresponding registers and memory copying.

To explain FPGA algorithm design, we use AES encryption as an example. In AES encryption, two rounds of SubBytes operations have a "full diffusion", which means that every state bit depends on all the state bits two rounds ago. In practice, there has never been found a computation faster than brutal force attack within 6 rounds of SubBytes operations, and another 4 rounds are added as security margin. That makes the AES encryption 10 rounds of SubBytes operations. The more rounds AES has, the higher security level the algorithm has (Leith, 2010). GPP can only do permutations or substitutions once per cycle, and as a result, it takes hundreds of thousands of clock cycles to complete one 128-bit AES encryption/decryption. Compared to GPP, the dedicated hardware implementation has a much higher efficiency. In order to maximize the throughput of AES encryption, the encryption block can be implemented as fast as 20 cycles per 128-bit block encryption. As more operations can be done in one cycle, more resources are needed to implement all the computational operations. We implemented the AES hardware block in 20 cycles as a trade-off between performance and resource utilization. The AES implementation on hardware can be achieved with identical modules, where each of them includes one round of SubBytes operations.

In the experiments for evaluation, we use our own application programs to load data into memory and to encrypt/decrypt/hash for repeated measurements. We measure AES operations in Electronic Code Book mode (ECB) and block cipher modes. We use the same crypto-engine core for both modes in cryptography. In block cipher modes, current block operations depend on previous block results, indicating

a read-after-write dependency between two operations. As a result, the AES crypto-engine can not be implemented in a streaming fashion, where plain text data are continuously fed into crypto-engines and encrypted data are produced at the same time. We integrate our implementation of the AES crypto function into OpenSSL to enable other block cipher modes. In order to achieve system re-configuration, we synthesize all the algorithms and write them to bitstreams on the memory storage. Our hardware implementation embeds well with existing libraries so that users can easily utilize our implementations for their designs. All that is needed to upgrade the cryptographic libraries is to replace the bitstream file with the new bitstream file through secure channel, such as secure shell (SSH). Whenever a re-configuration is requested, re-configuration of the FPGA can be achieved by sending the bitstreams to the configuration device port. After generated crypto-engine bitstreams are copied to FPGA, the Zedboard system takes less than a second to reconfigure. In this way, the system can get updated bitstreams and map them to FPGA accordingly.

For the power and energy analysis, we first measure the static power consumption of the entire Zedboard platform and then record the power consumption for each individual algorithm. We make sure that fluctuations have been averaged out in enough number of experiments, meanwhile the measurements can be completed in a short time. In each algorithm measurements, we record the power consumption of 16^6 256-bit blocks for SHA, 16^6 64-bit blocks for DES, 10^5 1K-bit blocks for RSA and 10^7 128-bit blocks for AES for 10 different runs at a sample rate of 1 sample per second. After the sampling, we average each data point among 10 runs of experiments to get the average power for each sampling interval. We deduct the static system power consumption of both ARM, FPGA and peripheral devices to get the dynamic power consumption for the measurements with FPGA acceleration. For the pure software implementation measurements, in addition to the deduction of overall static

power consumption, we also deduct the static power consumption of FPGA. Since the software implementation takes much longer time to finish, we use percentages of processed data to normalize the total time for energy comparisons in the figures.

3.5 Evaluation

In this section, we show the performance, energy and Energy Delay Product (EDP) based comparison between software implementation and hardware FPGA implementation of AES, RSA, DES, and SHA. We perform the analysis for both encryption and decryption for all the symmetric and asymmetric cryptographic algorithm. We present measurements results of AES Cipher Block Chaining (CBC) and AES Galois/Counter Mode (GCM) modes as the examples for block cipher modes. AES CBC use the bit-wise exclusive or between previous encrypted block and current plain text block as the input for the next block in order to increase pseudo randomness in the cipher text. AES GCM, in addition, has the authentication of data along with encryption in block cipher modes.

Look-Up Tables (LUTs) are gate level units that encode any boolean expression by modeling such functions in truth tables. The number of LUTs decides the resources that can be used for logic on FPGA. Table 3.1, shows a summary of LUTs utilizations after crypto-engines are mapped to the FPGA.

	AES	RSA	SHA	DES
Number of Gates	41,427	18,687	29,650	1,275
LUTs Utilization	77.87%	35.11%	55.71%	2.4%

Table 3.1: This table shows the LUTs utilizations in FPGA of different crypto-engines.

In the measurement tests, we process data ranging from 16 to $268,435,456(16^7)$ bytes for SHA and DES blocks, from 1 to $1,600,000$ bytes for RSA testbenches, and from 1 to $160,000,000$ bytes for AES. The Zedboard with the Linaro OS running the

system consumes 4 watts on average (power consumption of the system with running no application program but operating system). The programmable components consumes less than 35 m watts (Hosseinabady and Nunez-Yanez, 2014) static power. In the following subsections, we discuss how much the FPGA implementations improve on performance, power and EDP in symmetric cryptography (§ 3.5.1), asymmetric cryptography (§ 3.5.2) and hash functions (§ 3.5.3), compared to software.

3.5.1 Symmetric Cryptography

In DES implementation, the hardware implementation is 1.9× faster in encryption and 1.6× faster in decryption than software (see Figure 3·2). Though the hardware implementation does not show a significant performance improvement, the FPGA consumes much less power. The energy consumption of the hardware implementation, is 3.9× lower in encryption and 1.9× lower in decryption (see Figure 3·2). The EDP of FPGA for encryption is 7.6× lower in encryption and 3.0× lower in decryption than the software implementation (see Figure 3·2).

In the AES implementation, the VHDL code breaks the AES into 10 rounds of permutation and substitution of SubBytes and inverse SubBytes operations. We expand rounds of operations to achieve maximum throughput. As we can see from Figure 3·3, the AES encryption has 18.8× better performance and AES decryption has 116.3× better performance than the software implementation. Power consumption in both cases is almost the same (see Figure 3·3). Since FPGA implementation is much faster than the software implementation, the energy consumption of FPGA is 13.9× lower in encryption and 6.0× lower in decryption than the software implementation. Therefore, we can see 261.9× EDP improvements in encryption and 704.6× EDP improvements in decryption improvement in EDP (see Figure 3·3). We also evaluate the block cipher modes in a similar fashion (see Figure 3·4). Block cipher modes are all implemented in software, while the crypto-engines are implemented in hardware.

AES CBC mode shows 6.2× better performance for encryption and 38.6× better performance for decryption. It also shows a 33.1× energy reduction for encryption and 154.1× performance for decryption. GCM mode exhibit 1.1× performance improvement and 1.2× energy savings. Compared to CBC mode, the additional computation in software makes GCM have less performance boost and energy savings.

3.5.2 Asymmetric Cryptography

In the case of RSA, the software implementation is much slower compared to the other encryption algorithms, since RSA requires large amount of multiplications that can be performed in parallel in hardware. With a key length of 1KB, the FPGA shows a performance improvement of 71.2× in encryption and 2983.1× in decryption compared to the software implementation (see Figure 3·5). Figure 3·5 shows that the power consumption for RSA is less than the software implementation. Hence, the energy cost in FPGA is 6.5× lower in encryption and 4033× lower in decryption than software implementation. As a result, the EDP of FPGA shows 462.7× and $12,000,000+\times$ improvement in encryption and decryption over software implementation, respectively.

3.5.3 Hash Function

In case of SHA, the FPGA has a higher power consumption (see Figure 3·6), however the performance is better by a factor of 6.6× (see Figure 3·6). As a result, the energy consumption is 4.6× less in the FPGA implementation. Therefore the EDP of the FPGA implementation is 30.3× lower than in the software implementation of SHA.

3.6 Conclusion

The current IoT devices leverage strong cryptographic algorithms (AES, RSA, and SHA) to secure the communication channel. These cryptographic operations have

high-performance overheads and energy costs that are not acceptable for low-power budget IoT devices. Modern processors embed the ASIC implementations of these cryptographic algorithms to improve the performance and reduce power consumption. However, the longevity of some IoT devices can exceed the use of algorithms. In order to have flexibility, high-performance, and low-power implementations, we propose the use of FPGA based SoC in these IoT devices. The reconfigurability provides the IoT devices the capabilities of upgrading the algorithms. In our experiments, we showed that the performance boost, energy savings and EDP reductions in FPGA implementations compared to software implementations ranges from 1.5× to 2983×, from 1.8× to 4033× and from 3.0× to $12,000,000\times$, respectively across a variety of cryptographic algorithms.

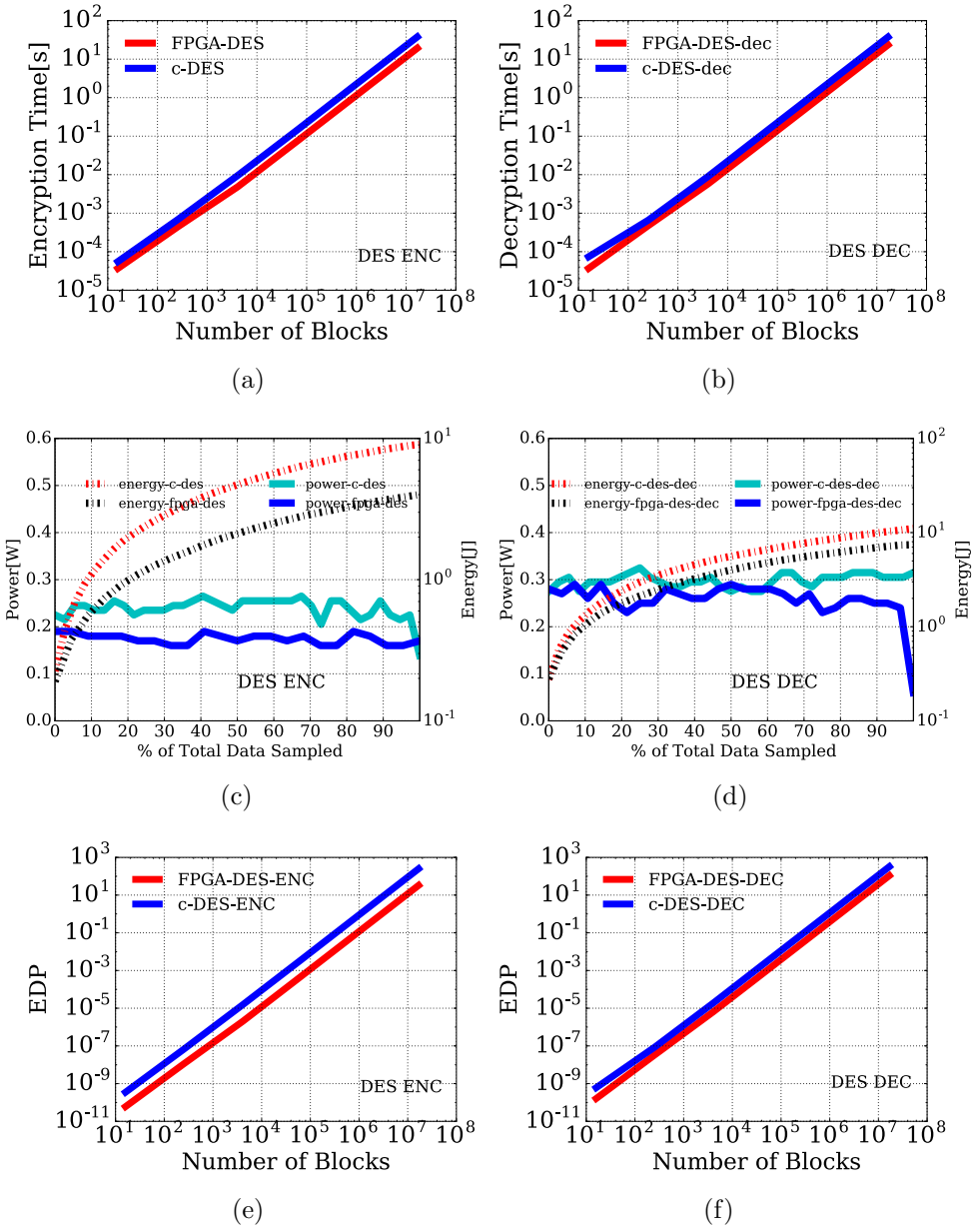


Figure 3·2: Figures show the comparisons between C implementation (software), and FPGA implementation (hardware) of DES in encryption (ENC) and decryption (DEC). We encrypt and decrypt ranging from 16 blocks to 16^6 blocks. (a) and (b) show time comparisons. Encryption shows 1.9× faster and decryption is 1.6× faster. (c) and (d) show power consumption and energy comparisons. Encryption has 3.9× energy reduction and decryption has 1.9× energy reduction. (e) and (f) show EDP comparisons. Encryption has 7.6× savings and decryption has 3.0× savings. We can see performance boost, energy savings and EDP reductions of hardware implementation in DES.

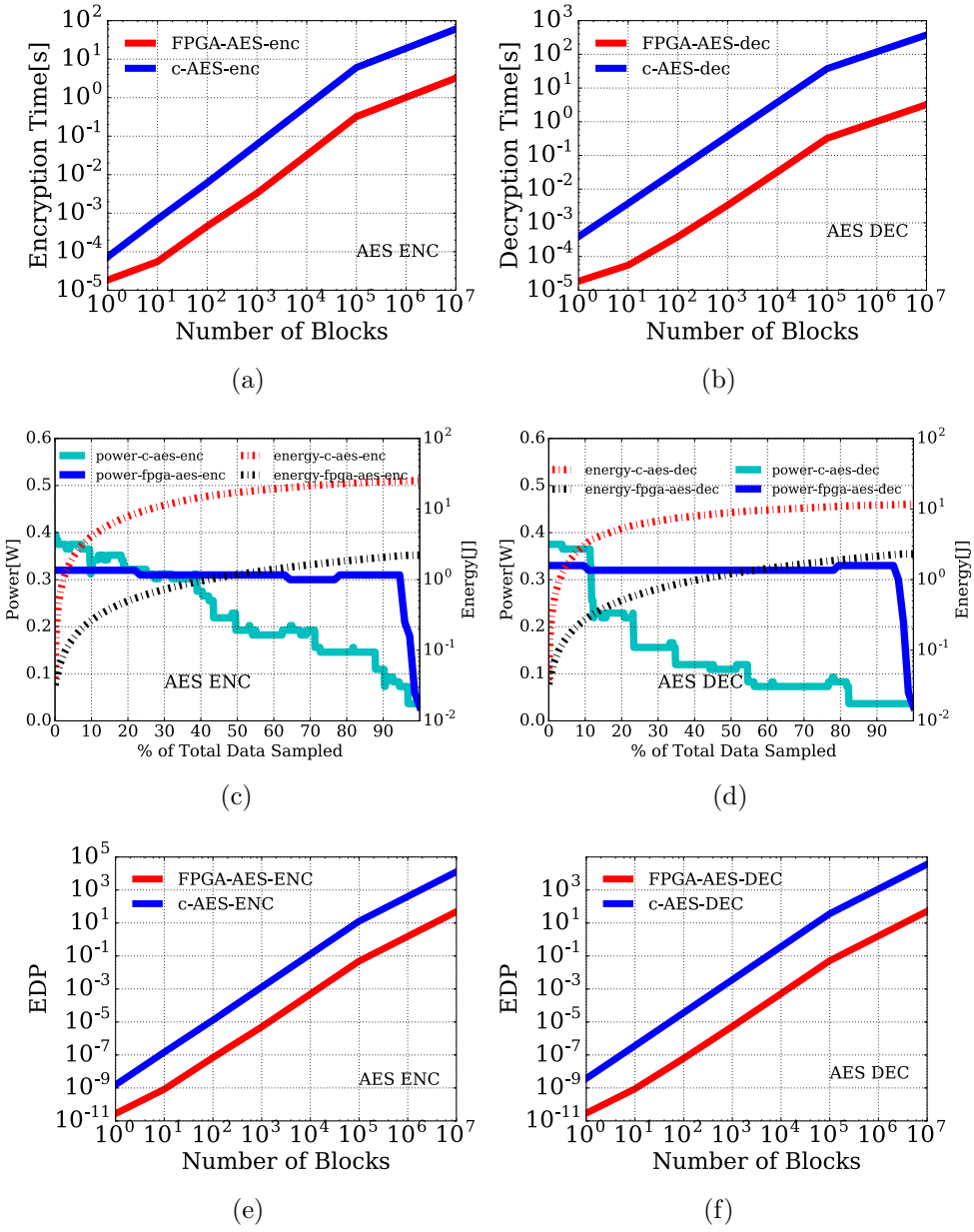


Figure 3·3: Figures show the comparisons between C implementation (software), and FPGA implementation (hardware) of AES ECB mode in encryption (ENC) and decryption (DEC). We encrypt and decrypt ranging from 1 block to 10^7 blocks. (a) and (b) show time comparisons. Encryption is 18.8× faster and decryption is 116.6× faster. (c) and (d) show power consumption and energy comparisons. Encryption has 13.9× energy reduction and decryption has 6.0× energy reduction. (e) and (f) show EDP comparisons. Encryption has 261.9× savings and decryption has 704.6× savings. We can see orders of magnitude performance boost, energy savings and EDP reductions of hardware implementation in AES ECB mode.

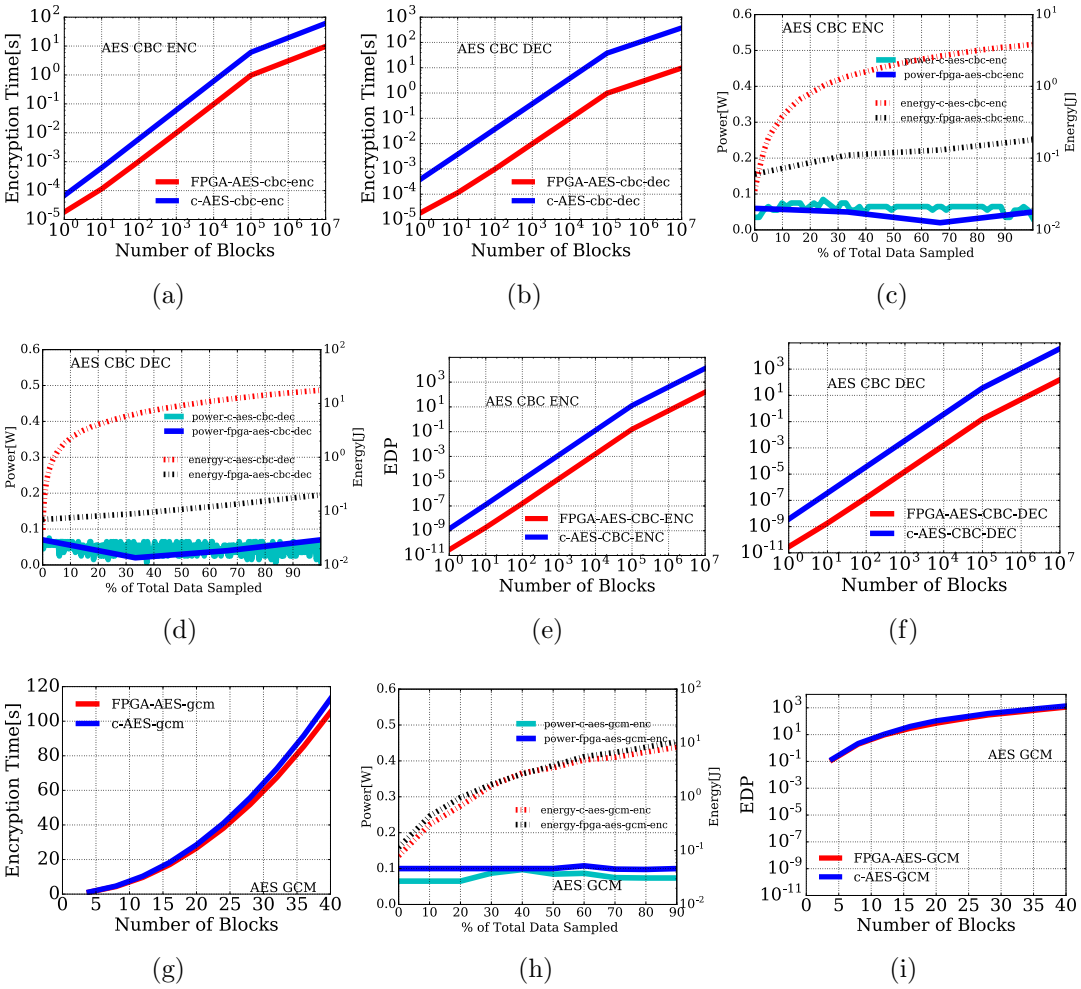


Figure 3·4: Figures show the comparisons between C implementation (software), and FPGA implementation (hardware) of AES block cipher modes, for both CBC and GCM. In all the hardware implementations, AES core engine operations use FPGA, while block cipher modes use c implementation. Thus, we still can see significant performance boost and energy savings in CBC modes, while much less benefits in GCM modes. In CBC mode, we encrypt and decrypt data ranging from 1 block to 10^7 blocks. In GCM mode, we encrypt data ranging from 4 to 40 blocks. (a), (b) and (g) show time comparisons. CBC encryption is 6.3× faster, CBC decryption is 38.6× and GCM is 1.1× faster. (c), (d) and (h) show power consumption and energy comparisons. CBC encryption has 33.2× energy reduction, CBC decryption has 154.8× energy reduction and GCM has 1.2× energy reduction. (e), (f) and (i) show EDP comparisons. CBC encryption has 82.4× savings, CBC decryption has 233.4× savings, and GCM has 1.3× savings.

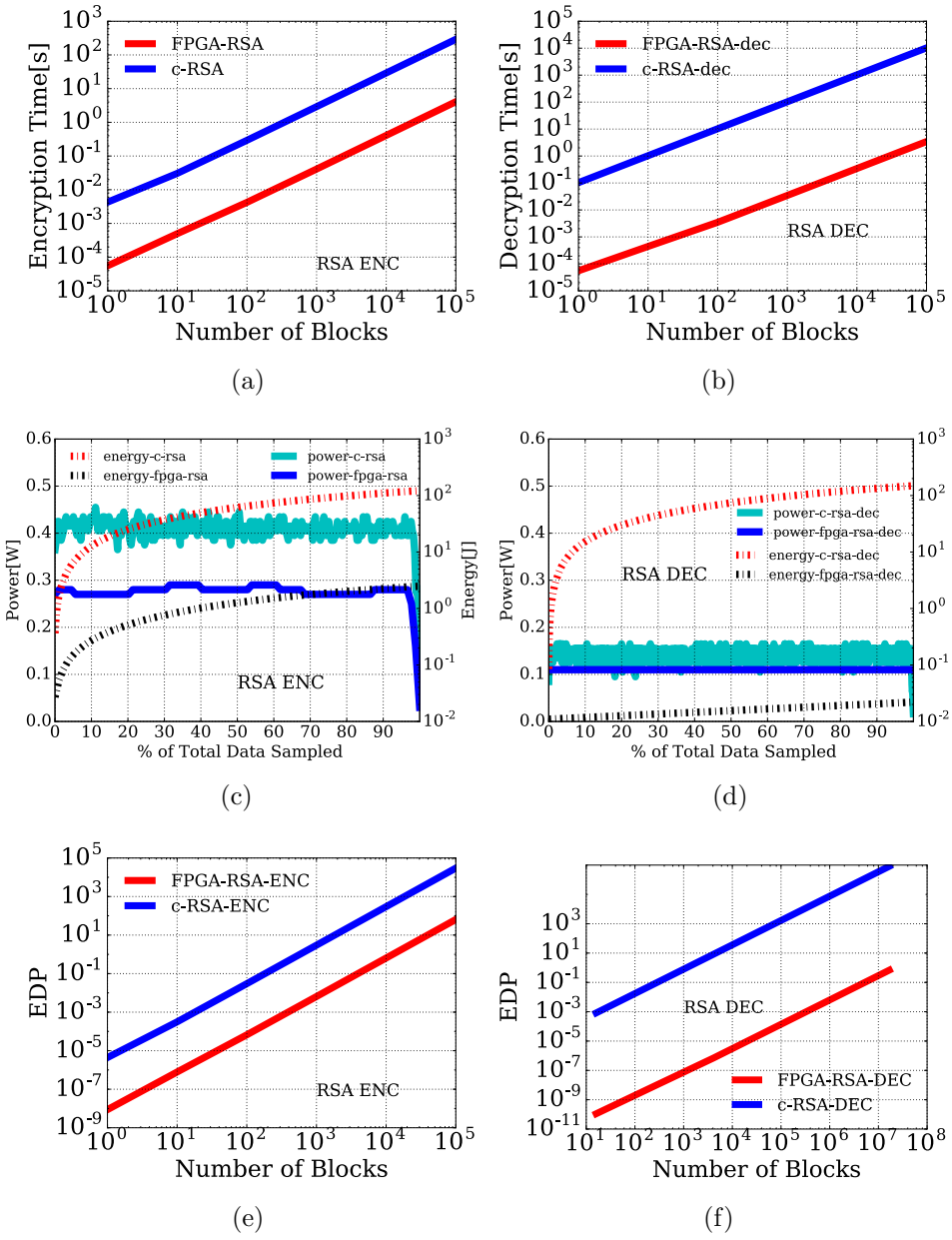


Figure 3·5: Figures show the comparisons between C implementation (software), and FPGA implementation (hardware) of RSA in encryption (ENC) and decryption (DEC). We encrypt and decrypt ranging from 1 block to 10^5 blocks. (a) and (b) show time comparisons. Encryption is 71.2× faster and decryption is 2983.1× faster. (c) and (d) show power consumption and energy comparisons. Encryption has 6.5× energy reduction and decryption has 4033.0× energy reduction. (e) and (f) show EDP comparisons. Encryption has 462.7× savings and decryption has $12,000,000+\times$ savings. We can great performance boost, energy savings and EDP reductions of hardware implementation in RSA decryption, since RSA private key length is much longer than public key length.

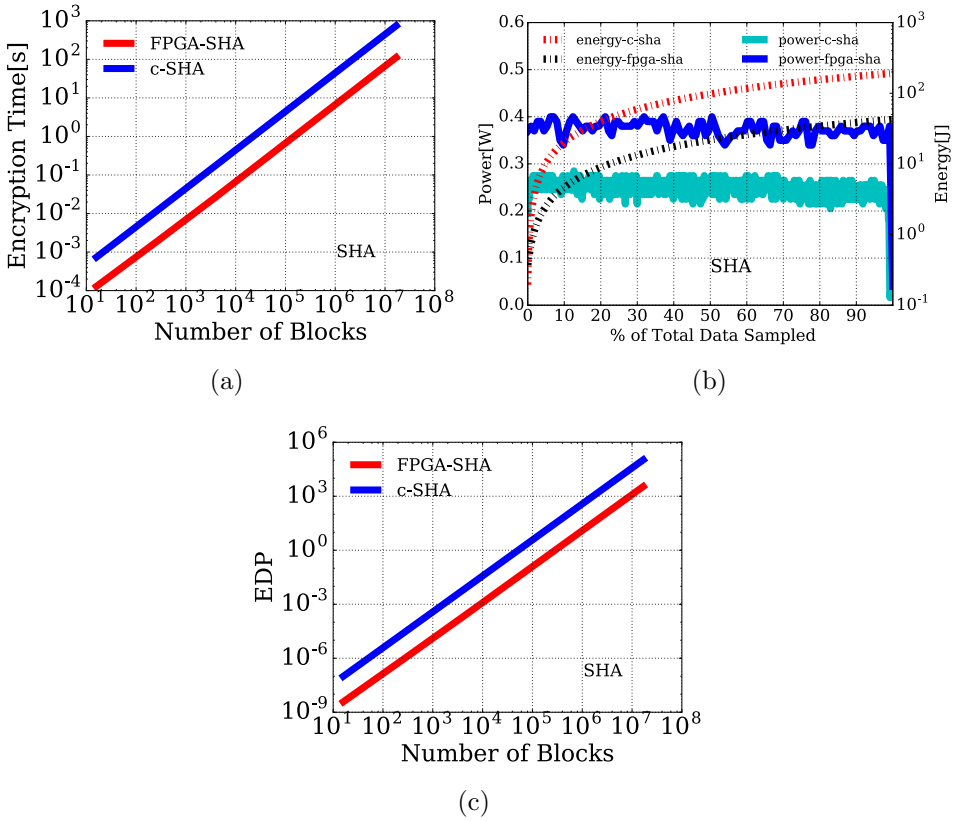


Figure 3·6: Figures show the comparisons between C implementation (software), and FPGA implementation (hardware) of SHA. We do hash ranging from 16 blocks to 16^6 blocks. (a) show time comparisons. (b) show power consumption and energy comparisons. (c) show EDP comparisons. We can see performance boost, energy savings and EDP reductions of hardware implementation in SHA. It shows 6.6× faster in time, 4.6× reduction in energy, and 30.3× savings in EDP.

Chapter 4

Malware Detection using Hardware Performance Counters with Machine Learning

4.1 Introduction

Distinguishing between malicious and benign software has been and will continued to be one of the biggest challenges facing computer security. As signature-based anti-virus scanners are easily thwarted by polymorphic malware, most commercial and academic anti-malware solutions rely on behavioral analysis. Behavioral analysis monitors programs as they execute, collects information on the process, and upon a violation of a behavioral profile, classifies the program as malware. To this end, software-based behavioral analysis can draw from a wealth of semantically rich information sources, such as file names, registry keys, or network endpoints, which characterize the program's behavior. As software-level behavioral analysis performs malware detection at the cost of performance overhead, recent research proposes to reduce this performance overhead by leveraging Hardware Performance Counters (HPCs) to classify programs as benignware or malware.

HPCs are hardware units that count the occurrences of micro-architectural events such as instruction counts, hits/misses in various cache levels and branch (mis-)predictions during runtime. Modern processors can capture more than 100 micro-architectural events, but a design-imposed strict limit of 4 (on Intel (Int, 2010)) and

6 (on AMD (AMD, 2015)) counter registers dictates that HPCs can only monitor a small subset of these events at one time.

Under these constraints, previous works (Demme et al., 2013; Patil et al., 2004; Kazdagli et al., 2016; Wang et al., 2016) have proposed to leverage the measured HPC values to classify an unknown program as either benign or malicious. To this end, measured HPC values are sampled at a fixed frequency and the resulting data is aggregated into a time-series. Previous works record data of labeled programs in time-series, and use the HPC values in time series to train various supervised machine learning models. The measured HPC values yield classifiers that can subsequently distinguish unknown programs as either benign or malicious.

The underlying assumption for previous HPC-based malware detectors is that *malicious behavior affects measured HPC values differently* than benign behavior. However, it is questionable, and in fact counter-intuitive, why the semantically high-level distinction between benign and malicious behavior would manifest itself in the micro-architectural events that are measured by HPCs. As a concrete example, consider that malware as well as benignware make use of the cryptographic APIs. While ransomware might maliciously encrypt the user data, the user might rely on encryption to safeguard privacy and data confidentiality. In both cases, ransomware and benignware, the program performs cryptographic operations. One cannot discriminate between malicious and benign usage based on the measured HPC values. The semantic difference of whether the encryption was performed maliciously or not exclusively depends on who holds the decryption keys, i.e., the attacker or the user. There is no indication that any HPC event would correlate with the ownership of the keys.

Given the substantial semantic difference between the high-level malicious behavior and the low-level micro-architectural events, it is expected from previous works

that assert the utility of HPCs for malware detection to provide a rigorous analysis, interpretation, and justification of *why* the extracted features from measured HPC values identify the maliciousness of programs. This includes, for example, an analysis of the events found to be the most predictive of malicious behavior and a discussion of why these features capture behavioral information at all. Unfortunately, existing works elide any such discussions, and instead commit the logical fallacy of *"cum hoc ergo propter hoc"*[1] — or concluding causation from correlation. Moreover, the correlations and resulting detection capabilities reported by previous works frequently result from small sample sets and experimental setups that put the detection mechanism at an unrealistic advantage.

To shine a light on the feasibility of using HPCs for detecting malicious behavior, we survey the existing literature in this field, and identify common traits that exhibit impractical setups and mis-interpretation of data analysis. Subsequently, we design, implement, and evaluate an experimental setup that allows us to reproduce previous works in this area, and compare these previous results with results obtained under more realistic scenarios.

In this work, we build an experimental setup close to the user environment, and evaluate fidelity of machine learning models. We run all experiments in a bare-metal environment instead of relying on virtualization techniques. This choice is motivated by two observations. First, our experiments indicate that measured HPC values collected for the same program running inside a virtual machine substantially differ from those collected on a bare metal system (comparisons in §1.2). Second, regular users likely execute programs directly on their systems outside of virtual machines. Further contributing to the realism of our experiments is the selection of training data for the machine learning models. Previous works (Demme et al., 2013; Kazdagli et al., 2016; Wang et al., 2016) test their machine learning models using

[1] "with this, therefore because of this"

measured HPC values from the same programs used during training (In §4.4.3, we refer to this approach as **TTA1**). In a real-world deployment, this scenario would reflect a situation where all programs (benign and malicious) are known and labeled for training. In such situations, machine learning is unnecessary, as each program could be perfectly identified based on its hash. As Anti-Virus (AV) vendors report thousands of new malware samples every day, this scenario is highly unlikely to ever occur in reality. Thus, we test our models with measured HPC values from programs that have not been observed during training. This reflects a realistic scenario where, during training machine learning models, malware samples from the same category or family are available, but not the exact same malware that a user may encounter.

We train 6 different machine learning classifiers and compare the results obtained with both realistic and unrealistic approaches. Unsurprisingly, we observe that classifiers trained in the realistic scenario perform worse than those trained in an unrealistic scenario. To rigorously evaluate the performance of our classifiers, we perform 1,000 iterations of 10-fold cross-validations and consistently observe False Discovery Rate[2] of larger than 20%. Such high False Discovery Rates would disqualify HPC-based malware detectors from real-world deployments, as it would flag 264 programs in a default Windows 7 installation as malicious. Finally, we illustrate how fragile the resulting classifiers are by *simply* composing a benign program (Notepad++) with malicious functionality (ransomware). This straight-forward composition evades all our classifiers, even when they are trained with the benign and malicious components individually. In summary, this work makes the following contributions:

- We identify the prevalent unrealistic assumptions and the insufficient analysis used in prior works that leverage HPCs for malware detection (§1.2).

[2] $F_+/(F_+ + T_+)$, where F_+ is number of benignware classified as malware and T_+ is number of malware classified as malware

- We perform thorough experiments with a program count that exceeds prior works (Demme et al., 2013; Kazdagli et al., 2016; Wang et al., 2016; Tang et al., 2014; Singh et al., 2017) by a factor of $2\times \sim 3\times$, and the number of experiments in cross-validations that is 3 orders of magnitude more than previous works.

- We train and test dataset similar to what prior works have done, as well as, in a realistic setting where testing programs are not in the training programs. We compare the effects of this choice on the quality of the machine learning models (§ 4.5).

- Finally, to facilitate reproducibility, and enable future researchers to easily compare their experiments with ours, we make all code, data, and results of our project publicly available under an open-source license: `https://bit.ly/2F3YRDd`

4.2 Threat Model

In this work, we trust the entire system booting up until our detection system starts. We trust the entire booting process and any hardware running beneath the system. We do not limit the attackers while the detection system is running. The proposed HPC system should detect the malicious programs.

4.3 Experimental Setup

In this section, we explain how we set up the experiments to gather values of HPCs from benignware and malware. We ran our experiments on a cluster with 15 machines as worker nodes, and a master node to distribute jobs to measure and to collect data from worker nodes. We dispatched our jobs to the worker nodes using the Rabbitmq message system (Rab, 2017). We collected the data back from the worker nodes using

a Samba (Sam, 2017) server on the master node. We used Bindfs (Bin, 2017) to fuse the permission bits of Samba server storage folder to be writable, not modifiable, not readable, and not executable. Note that the Portable Operating System Interface (POSIX) permission structure cannot provide the above-mentioned permission bits. These permission bits allowed the worker nodes to record the measured HPC values, while these permission settings prevented malware from overwriting or deleting the measured HPC values. On the worker nodes, we ran our experiments in Windows 7 32-bit operating system to be compatible with malware experiments in other works (Ozsoy et al., 2015; Khasawneh et al., 2015; Khasawneh et al., 2017). Previous works applied time-based HPC sampling, i.e., they gathered values at a fixed sampling frequency (Demme et al., 2013). We used AMD CodeAnalyst APIs to build a time-based HPC monitoring tool, *Savitor*, since AMD CodeAnalyst itself cannot provide time-based measured HPC values (Drongowski, 2008).

4.3.1 *Savitor* (HPC measuring tool)

We designed *Savitor*, a tool that monitors a target process and gathers HPC values related to the process. *Savitor* runs the target process, pins the process to one core, reads the HPC values from that core and then writes the measured HPCs values to files on another core, in order to reduce the noise during the sampling. *Savitor* records 6 HPCs at a time, which is the maximum number of HPCs that can be recorded on the AMD Bulldozer micro-architecture without time-multiplexing. *Savitor* performs time-based sampling and kills the target process at the end of each experiment. Following the frequency limits in CodeAnalyst, we used the maximum possible sampling frequency of 1 KHz for *Savitor*. Considering our limited resources (time and hardware), we only ran each experiment of both malware and benignware for 1 minute.

4.3.2 Malware and Benignware

For forming the set of malware, we downloaded 1,000 malware from Virustotal (Virustotal, 2017), and performed a test run of those 1,000 malware on worker nodes. After the test run, we identified 962 malware which could run for more than 1 minute and used them in our malware experiments. According to *AVClass tool* (Sebastián et al., 2016), our dataset consisted of 35 distinct malware families.

In order to collect benignware programs, we first installed all the packages and software from Futuremark (Fut, 2017), python performance module (per, 2017), ninite.com (Nin, 2017), and Npackd (Npa, 2017) on the worker nodes. After installation, we traversed all the files in "Startup Menu" and "C:\Program Files" folder to include all the unique executable programs in our benignware dataset. We avoided the complication of re-installation by excluding all the executable program files with "uninstall" in their names. We performed a test run of all these programs, and selected 1,382 benignware that could run for 1 minute.

To avoid the classification bias, we matched the number of malware and benignware used in our experiments. Classification bias exists in classification problems if the number of items in each class is different. For example, in a classification problem with two classes, A and B, if class A makes up 80% of the data set and class B makes up 20% of the dataset, the baseline of precision in classifying A is 80%. Any designed machine learning models whose precision is lower than 80% are worse than the precision estimated with prior probability. In our work, we matched the number of benignware and malware; at the same time, we reported precision, recall and F1-score to eliminate any bias.

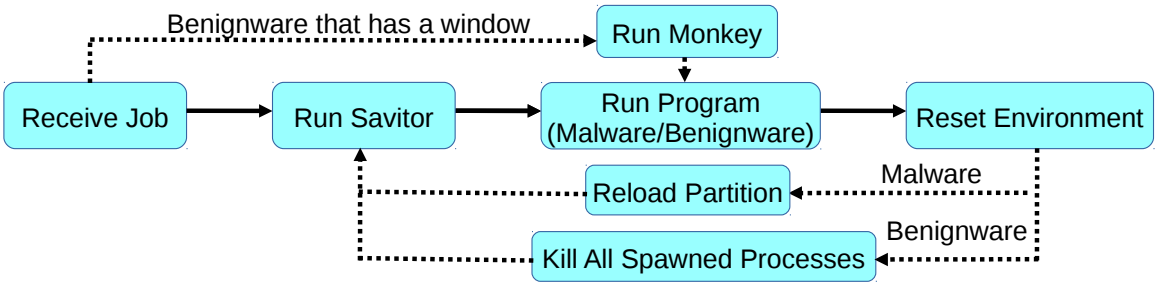


Figure 4·1: Our workflow of benignware/malware experiments: The worker node receives the dispatched jobs of experiments from the master node. The worker node spawns a *Savitor* process, and then *Savitor* runs the target process (benignware/malware). The dotted arrow (⇢) means that the action does not always happen. If the application has a window for interaction, we attach a monkey tester to the window. The solid arrow (→) shows that actions always happen. We reset the environment after each experiment. the worker node kills any other processes spawned by the target process after each benignware experiment. At the end of each malware experiment, we reboot the machine into the Debian partition to reload a clean Windows image.

4.3.3 Method for Running Experiments

We ran our benignware and malware experiments on identical hardware and operating system. However, there are a few differences between malware and benignware experiments. We explain the workflow of malware and benignware experiments using one dispatched job in Figure 4·1. The boxes are the steps that we follow, and the solid arrow means that the next step always happens. The dotted arrow means that the action happens under the conditions of the labels.

Malware Experiment

We follow the steps in Figure 4·1 to run the experiments. Before any malware experiments, we dropped all the requests to any network outside the master node to ensure that malware does not affect other machines. At the beginning of each experiment, the worker node runs a clean copy of Windows and waits for a new job. Once the worker node receives the job from the master node, *Savitor* runs the malware

and records the measured HPC values. After running each malware experiment, we provide an identical, malware-free environment for the next malware experiment by reloading the Windows partition. In order to reload Windows image, we installed Debian 8 in the other partition of the hard drive on each worker node. Whenever a worker node boots into the Debian partition, the worker node copies a clean Windows image to the other partition. We modified the GNU GRand Unified Bootloader (GRUB) to make the machine boot into an alternate partition every time it reboots. After reloading the image, the system reboots into Windows again and runs the next job dispatched from the master node.

Benignware Experiment

Similar to the malware experiments, benignware experiments also follow the workflow in Figure 4·1. We connected the worker nodes to the outside network to ensure the benignware receives network responses. Programs, such as browsers, require network responses to perform similarly as in a user environment. When the worker node receives a job from the master node, *Savitor* starts the target process (benignware program), and a monkey tester is attached to the target process if the target process has an interactive window. The Monkey tester works similar to Android's Monkey tester (And, 2017), as it interacts with the target process by periodically sending random keystroke, mouse clicks, and scrolling operations to the window of the target process. The behavior of the monkey tester simulates the interaction between a user and the programs. After *Savitor* samples the measured HPC values, the system resets by killing any processes spawned during the experiments. Since the benignware does not try to infect the Windows partition and perform malicious operations, we do not reload the Windows partition. After killing the spawned processes, the worker node receives the next job from the master node and starts the next experiment.

4.4 Machine Learning Models

In this section, we present how we apply machine learning models on measured HPC values. One of the common problems in machine learning is the *Curse of Dimensionality*. *Curse of Dimensionality* means that machine learning models in a high-dimensional space have lower detection rates compared to models in lower-dimensional spaces (Goodfellow et al., 2016). The redundant dimensions in high dimensions contribute to the measurement of noise in the training dataset, which result in a decrease in the detection rates of testing. *Curse of Dimensionality* motivates the reduction of dimension; however, reducing dimensions may cause underfitting due to the lack of representation during training. In order to overcome both overfitting and underfitting, the design of machine learning models requires the minimum number of features that represent most of the measured HPC values. To this end, we perform a quantitative analysis to extract features from the measured HPC values of our selected micro-architectural events.

4.4.1 Reduction of Dimensions

In this work, we use Principal Component Analysis (PCA) to reduce the dimensions. By reducing the dimensions, the machine learning models can use the linearly independent components to easily classify the examples into different classes. Here, we show one *synthetic* dataset (a subset from our experiments) separated from overlapping measured HPC values by applying PCA results. In the next subsection (§4.4.2), we explain how we choose the sizes of examples and features.

PCA applies eigen-decomposition to decompose the training standard matrix (A), where columns are features and rows are examples, into the multiplication of eigenvectors (V) and eigenvalues (λ) in Equation 4.1. The standard matrix (A) is transformed into lower-dimensional data space by multiplying the eigenvector matrix V, which can

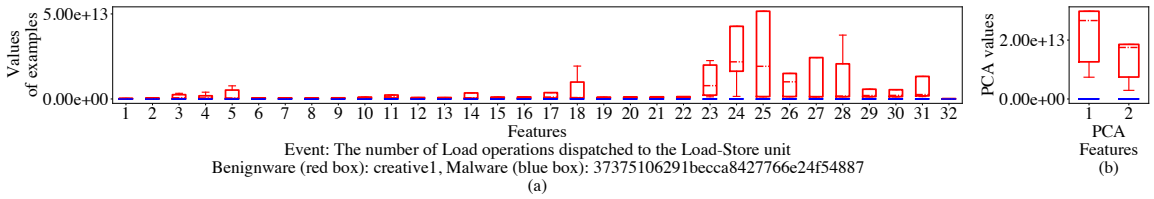


Figure 4·2: X axis is the feature number and Y axis is the values of each example. Red box corresponds to the malware and blue box corresponds to the benignware. The dashed line is the mean of each distribution. The boxes represent $25\% \sim 75\%$ of the distributions. The whiskers (the short, horizontal lines outside the boxes) represent the confidence interval equivalent to $\mu \pm 3\sigma$ of Gaussian Distribution ($0.3\% \sim 99.7\%$). We measure *The number of Load operations dispatched to the Load-Store unit* event 5 times in one benignware (creative2 from Futuremark) and one malware. The distributions of the two subplots represent 5 examples in the experiments. (a) Distributions of sampled values before the reduction of dimensions: We cannot distinguish between the 5 malware examples and the 5 benignware examples. (b) Distributions of sampled values after the reduction of dimensions: We apply the reduction of dimensions to examples in (a) to get examples in (b). We can separate all the examples in (b) due to the gaps between values of malware and benignware in both features.

also be approximated with the major eigenvector matrix (V').

$$A = V\lambda V^{-1} \approx V'\lambda V'^{-1} \tag{4.1}$$

We present the distributions of examples before and after the reduction of dimensions, $A_{5\times32}$ in Figure 4·2(a) and $A_{5\times32}V'_{32\times2}$ in Figure 4·2(b). We measure *the number of Load operations dispatched to the Load-Store unit* (Table 4.1) event 5 times in one benignware (creative2 from Futuremark) and one malware [3]. The input matrices (A) of both benignware and malware have 32 features and 5 examples. In Figure 4·2, X axis shows the feature number and Y axis shows the values of each example. Red box refers to the malware and blue box refers to the benignware. The dashed line is the mean of each distribution. The boxes represent $25\% \sim 75\%$ of the distributions. The whiskers (the short, horizontal lines outside the boxes) represent the confidence

[3]SHA256 hash value: 3737 5106 291b ecca 8427 766e 24f5 4887

interval equivalent to $\mu \pm 3\sigma$ of Gaussian Distribution (0.3% $\sim$ 99.7% of the total distributions).

From Figure 4·2(a), we can see overlapping of boxes and whiskers in all the columns. Figure 4·2(b) shows the results of the data matrix (A) multiplied with the eigenvector matrix (V'). We can clearly classify the malware or benignware, since there are gaps between the distributions of malware and benignware in both features. By multiplying the eigenvector matrix, different features contribute to classification with weights according to their abilities to discriminate data. Hence, we can achieve higher classification rates with lower dimensional data.

4.4.2 Selection of Events

As discussed in §4.4.1, we reduce the dimensions to extract features from the measured HPC values to form the machine learning models. At the same time, the hardware limitation on number of HPCs without time-multiplexing requires the selection of events from more than 100 available micro-architectural events. Hence, we designed a method to select our micro-architectural events, while reducing the dimensions of examples at the same time.

In our method, our selection of events is based on minimizing 3 sources of losses These 3 main sources of losses in the measured HPC values are:

- Jitter: the timing variations between identical measurements of the measured HPC values.
- Noise: the amplitude variations between identical measurements at the same time-stamp of the measured HPC values.
- Approximation error: the loss of the minor eigenvectors.

Jitter and noise are introduced due to the limitations in the measurements. As we will discuss in §4.6, noise and jitter cannot be eradicated. To minimize the impact

from jitter, we divide the measured results into 32 equal time intervals, and sum the gathered values in each time interval to form 32 histogram bins (each bin corresponds to one feature). This is the same design choice as the one used by Demme et al. (Demme et al., 2013). Histogram bins preserve the sampled information, while reducing the effects of jitter in the values of HPCs. In addition to jitter, we observe noise in the measured HPC values, as Weaver et al. do in their work (Weaver et al., 2013; Weaver and McKee, 2008). To minimize the noise for our selection of events, we repeat the measurements on the same program and the same events 32 times, and then we calculate the cumulative sum in each bin, in order to increase the Signal-to-Noise Ratio (SNR). Assuming the noise introduced during the measurement is Additive White Gaussian Noise (AWGN) (Haykin, 1983), this approach increases the SNR by a factor of 32.

Approximation error is introduced by the elimination of minor eigenvectors in V when we transform V to V'. For each example, we multiply the measured HPC values to the major eigenvector matrix V' instead of V. In our method, by trading off the number of eigenvectors in the major eigenvector matrix, we reduce the dimensionality and increase the approximation error in Equation 4.2. We use the product of the standard matrix A and the eigenvector matrix V' as our input matrix in machine learning model as we described in Equation 4.1.

$$AV = \sum_{i=1}^{m} v^{(i)}\lambda^{(i)} + \sum_{i=m+1}^{n} v^{(i)}\lambda^{(i)} \tag{4.2}$$

$$= \sum_{i=1}^{m} v^{(i)}\lambda^{(i)} + \epsilon(\alpha v\lambda) \tag{4.3}$$

In equation 4.2, $\lambda^{(i)}$ denotes the i^{th} largest eigenvalue with n eigenvalues (λ). $v^{(i)}$ is the corresponding eigenvector of $\lambda^{(i)}$, and m is the number of reduced dimensions.

Equation 4.2 represents the separation of m major and $n - m$ minor eigenvectors. The first term in Equation 4.2 is AV'. The approximation error is the difference between AV and AV', which is the second term in Equation 4.2. In Equation 4.3, ϵ denotes the upper bound function. α denotes error coefficient, with the error term $(AV - AV')$ divided by the original input data (AV). Equation 4.3 expresses that with a given m value, we can estimate the approximation error using α. By having more eigenvectors in the eigenvector matrix V' (larger m), we can reduce α, which corresponds to a lower approximation error. As we observe from Equation 4.3, the approximation error depends on the choice of eigenvectors. We cannot determine the eigenvectors before we train and test our dataset. However, we can use a subset of programs to compute the eigenvectors and choose the parameters in Equation 4.3.

As in real-life, it is impossible to use the entire dataset for the selection of events. Here, we chose a subset of programs, **7** programs from the Futuremark (Fut, 2017) benchmark for the selection of events. The choice of programs from Futuremark benchmark suite is driven by the fact that Futuremark has analyzed user behavior and automated this behavior in the benchmarks. All the programs of Futuremark benchmark are real-world applications commonly used in office. We measured the programs at at the frequency of 1 kHz for 1 minute, as we described in §4.3.1. Our experimental hardware (AMD Bulldozer micro-architecture) enables us to monitor **130** events (6 at a time). We accumulated the measured HPC values into **32** bins, with each measurement summed into 32-dimension vector. Thus we ran each of the 7 programs from Futuremark Benchmarks on 130 micro-architectural events 32 times (**130**×**32**×**7**).

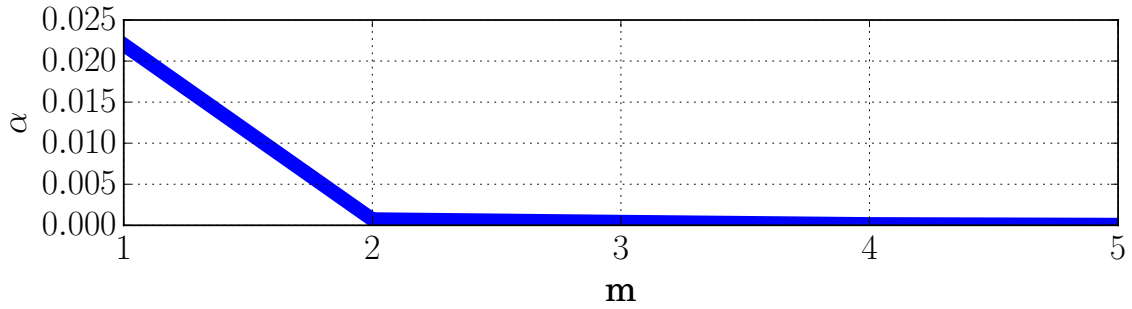


Figure 4·3: Error Bound vs the Number of Eigenvetors Plot: when choosing different number of eigenvectors for reduction in dimensions, the error bound α changes according to m eigenvectors.

$$A_{e_k} V_{e_k} = \sum_{i=1}^{m} v_{e_k}^{(i)} \lambda_{e_k}^{(i)} + \epsilon(\alpha v_{e_k} \lambda e_k) \tag{4.4}$$

$$\alpha(m) = \min_{e_j} \frac{\sum_{i=m+1}^{n} v_{e_j}^{(i)} \lambda_{e_j}^{(i)}}{v_{e_j} \lambda e_j} \tag{4.5}$$

With the results (**130**×**32**×**7**) from the experiments, we denote the k^{th} event as e_k, its i^{th} eigenvalue as $\lambda_{e_k}^{(i)}$, and the corresponding eigenvector as $v_{e_k}^{(i)}$ for $k = 1, 2, \ldots 130$ in Equation 4.3, in order to re-write Equation 4.3 into Equation 4.4. In Equation 4.5, e_j corresponds to the 6 events with the minimum α when $j = 1, 2, \ldots 6$, excluding the events whose measured HPC values are all zeros. We apply Equation 4.1 to compute v_{e_k} and λ_{e_k}. We calculate the eigenvalues for 130 events and find out that there is no event among 130 events with more than 10 eigenvectors ($n \leq 10$). We exclude all the events that only have zero values in the measured HPC values, since these events provide no signal in the measured HPC values. By changing the number of eigenvectors (m), we can calculate the error coefficient (α) in Equation 4.5. We plotted the error coefficients for $m = 1, 2 \ldots 5$ in Figure 4·3. The gradient of α decreases when m is more than 2. Subsequently, we consider the optimal trade-off between m and α when $m = 2$ and $\alpha(2) = 0.072\%$, which corresponds to the upper

Table 4.1: Description of the Selected Events (AMD, 2015)

Events	Definition
0x04000	The number of accesses to the data cache for load and store references
0x03000	The number of CLFLUSH instructions executed
0x02B00	The number of System Management Interrupts (SMIs) received
0x02904	The number of Load operations dispatched to the Load-Store unit
0x02902	The number of Store operations dispatched to the Load-Store unit
0x02700	The number of CPUID instructions retired

bound of error as 0.072% AV, with the linear combination of 2 components from v_{e_j}.

The 6 events, which we selected in our experiments, are listed in Table 4.1. We assemble the eigenvectors of 6 events, 2 for each event, and we get the v matrix in Equation 4.6.

$$v_{192\times 12} = \begin{bmatrix} v_{e_{k_1}}^{(1)}, v_{e_{k_1}}^{(2)}, & & & \\ & v_{e_{k_2}}^{(1)}, v_{e_{k_2}}^{(2)}, & & \\ & & \ddots & \\ & & & v_{e_{k_6}}^{(1)}, v_{e_{k_6}}^{(2)} \end{bmatrix} \tag{4.6}$$

In Equation 4.6, $v_{e_{k_j}}^{(i)}$ represents the i^{th} largest eigenvector in j^{th} event. Each $v_{e_{k_j}}^{(i)}$ is a 32×1 eigenvector. By multiplying each example with the v eigenvector, we reduce the dimensions from 192 (6 *events* $\times$ 32 *bins*) to 12 (6 *events* $\times$ 2 *components*).

In summary, we list following steps to select the events:

- Run 7 programs 32 times and measure 130 micro-architectural events.
- Divide the total run time of each program into 32 intervals and sum the measured HPC values in each interval into a separate bin.
- Sum the bins across different runs of the identical measurements.
- Apply PCA on 130 events with 7 programs.
- Compute the approximation errors for 7 programs.
- Find 6 events with the least approximation errors.

Four of the selected events in our experiments align with other works that do not provide any analysis of their selection of events (Demme et al., 2013; Kazdagli et al., 2016; Ozsoy et al., 2015; Khasawneh et al., 2015; Khasawneh et al., 2017). We observe that one event is related to data cache load and store references. Two other micro-architectural events are related to load and store operations, which have also been used in other works. It is not clear how load and store operations deterministically contain the information of malicious behavior. Any statistics of memory behavior should be legitimate in program execution, since the memory accesses inheretly exist in every program. In our selection of events, we include the events in kernel mode to capture complete program behavior. The remaining 3 selected hardware events related to cache flush behavior, system management interrupts and CPUID instructions. However, we cannot infer any reasons why these instructions/operations by the kernel can be mapped to any malicious user-level behavior.

4.4.3 Classification Models

In §4.4.2, we selected the 6 events to monitor and formulate the eigenvector matrix in Equation 4.6. With this method, we can extract features from the measured HPC values to get examples for machine learning models, i.e. traces in our datasets of benignware and malware.

To have the same number of measurements on the same program samples as in §4.4.2, we run each benignware program and each malware program 32 times, and collect 61,568 measured HPC values($2\times962\times32$), 30,784 for benignware and 30,784 for malware (1,026 CPU hours). We sum the measured HPC values into 32 histogram bins (as described in §4.4.2) for each of 6 events. Each example of histogram binned HPC values has 192 (6 $events \times 32$ $bins$) features. By multiplying each example with the v eigenvector in Equation 4.6, we reduce the dimensions from 192 (6 $events \times 32$ $bins$) to 12 (6 $events \times 2$ $components$). To this end, we convert the **measured HPC**

values into histogram bins, and then transform them into **traces**.

Using the reduction of dimensions (§4.4.2), the input matrix $A_{30,784\times192}$ (30,784 examples and 192 features) of benignware or malware is transformed to lower-dimensional space as $A'_{30,784\times12}$ (30,784 examples and 12 features). For training and testing of the machine learning models, we are going to separate the examples in matrix A' into training and testing datasets (training-and-testing split). In our experiments, we consider 2 Training-and-Testing Approaches (TTA) to divide our dataset into training set and testing set. The two approaches are as follows:

TTA1 Dividing 30,784 traces with a split of 90:10 ratio, resulting in 27,704 traces (90% of 30,784 traces) as training dataset and 3,078 traces (10% of 30,784 traces) as testing dataset both in benignware and malware experiments.

TTA2 Dividing 962 programs with a split of 90:10 ratio, resulting in traces of 866 programs (90% of programs) as training dataset and traces of 96 programs (10% of programs) as testing dataset both in benignware and malware experiments.

In the first training-and-testing approach (TTA1), we randomly choose 27,704 traces as training dataset and 3,078 traces as testing dataset both in benignware and malware experiments. In this approach, the traces resulting from the same program sample can appear in both training and testing datasets. As a result, such an approach corresponds to a highly optimistic and unrealistic scenario where the testing programs (benignware or malware) are available during training. Given that thousands of new malware appearing everyday, it is impossible to include all the malware that user may encounter. Hence, TTA1 should not be applied in training machine learning models for malware detection.

In the second training-and-testing approach (TTA2), we randomly choose traces of 866 programs as training dataset and traces of 96 programs as testing dataset

Table 4.2: Detection Rates with TTA1 and TTA2: Red means the value is less than 50% and **bold** means that the value is more than 90%

	TTA1				TTA2			
Models	Precision[%]	Recall[%]	F1-Score[%]	AUC[%]	Precision[%]	Recall[%]	F1-Score[%]	AUC[%]
Decision Tree	83.04	83.75	83.39	89.65	83.21	77.44	80.22	87.36
Naive Bayes	70.36	7.97	14.32	58.11	56.72	5.425	9.903	58.38
Neural Net	82.41	75.4	78.75	84.41	**91.34**	22.16	35.66	66.43
AdaBoost	78.61	71.73	75.01	80.57	75.78	65.6	70.32	77.96
Random Forest	86.4	83.34	84.84	**91.84**	84.36	78.44	81.29	89.94
Nearest Neighbors	84.84	82.37	83.59	89.26	82.7	77.88	80.22	86.98

both in benignware and malware experiments. TTA2 corresponds to a realistic case where during training model, we do not have access to the exact programs, benign or malicious, that users run in the real life. To validate across our models, we perform 10-fold cross-validations 1,000 times. For each 10-fold cross-validation, we randomly shuffle the dataset to ensure difference across 1,000 rounds. In each 10-fold cross-validation, each example in the dataset is used in training 9 times and testing once. This ensures the identical times of training and testing for every single example, compared to randomly shuffling the data and validating the machine learning models. With 1,000 10-fold cross-validations, we can ensure that the standard deviations of detection rates increase no more with more rounds of validations.

In our experiments, we perform training and testing with both TTA1 and TTA2. We compare the detection results in terms of precision, recall, F1-score, and Area Under Curve (AUC) in both approaches. We use the implementations of machine learning models in scikit-learn package (Pedregosa et al., 2011): DT, RF, NN, KNN, AdaBoost, and Naive Bayes. The seed for randomness in machine learning initialization and division of data comes from the random number generator "`/dev/urandom`". During training, we set the parameters of the machine learning models as described below to prevent the machine learning models from underfitting due to default limitations in computational resources set by scikit-learn. We used default values for the remaining parameters in scikit-learn.

- DT: We set the maximum depth as 100 to classify the malware and benignware.

The number of levels is sufficient to avoid any underfitting of the model.

- RF: We set the maximum depth to be the same as in the DT. We enable a maximum 200 estimators in the RF. The number of estimators is sufficient to avoid any underfitting of the model.

- NN: The network we use here is a Multilayer Perceptron (MLP) neural network having 4 layers with 100 neurons in each layer. We apply "tanh" function as the activation function. We use $L2$ regularization on the parameters in the NN.

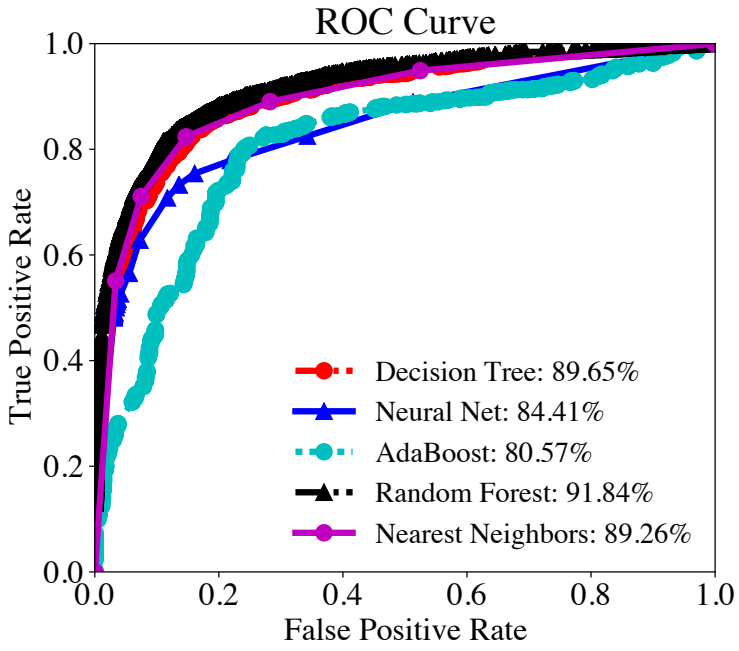
- KNN: We choose 5 as the number of nearest neighbors, and perform experiments with K value varying from 1 to 20. When K equals to 5, the F1-score reaches the highest detection rates.

- AdaBoost: Adaboost is an Ensemble classifier, which utilizes a collection of estimators. Adaboost fits a sequence of classifiers on the training data. The predictions are decided based on a majority vote (Freund and Schapire, 1995). The default value of the number of estimators is 50. We use 200 estimators instead of 50, since our test experiments show that 200 estimators have a higher detection rate for AdaBoost.

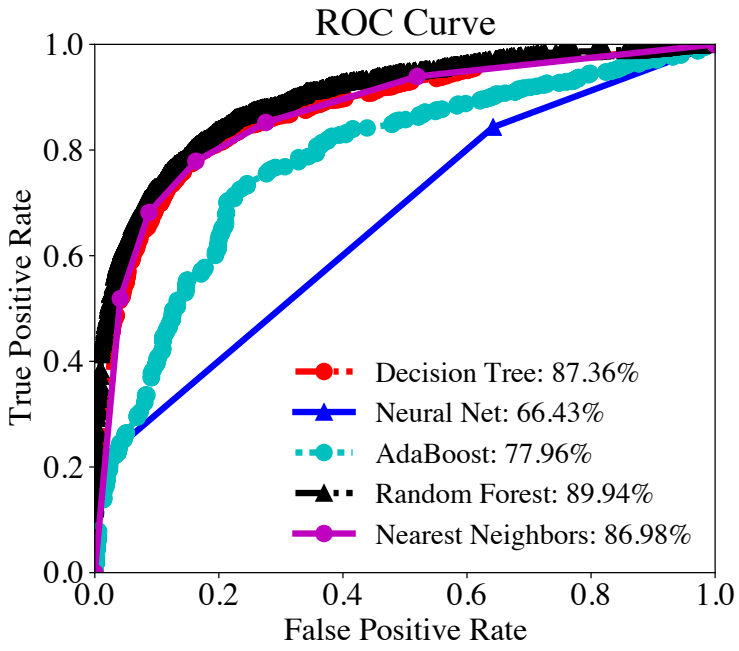
- Naive Bayes: We use the same number of malware and benignware traces in the model. The prior probability is 50%.

4.5 Experimental Results

In this section, we show our results with the experiments to detect malware using HPCs and contrast the ones obtained in previous works. We report malware detection rates in terms of precision, recall, F1-score, and Area Under Curve (AUC) in Receiver Operating Characteristic (ROC) plots. We use the positive label to denote malware



(a)



(b)

Figure 4·4: Receiver Operating Characteristic (ROC) curve of 5 models. (a) The AUC of DT, NN, AdaBoost, RF, and KNN using (TTA1) is 89.65%, 84.41%, 80.57%, 91.84%, and 89.26%, respectively. (b) The AUC of DT, NN, AdaBoost, RF, and KNN using (TTA2) is 87.36%, 66.43%, 77.96%, 89.94%, and 86.98%, respectively.

and the negative label to denote benignware. True positive samples (T_+) are malware programs that are classified as malware. False positive samples (F_+) are benign programs that are classified as malware. False negative samples (F_-) are malware programs that are classified as benignware. Precision is defined as the number of true positive samples (T_+) divided by the number of all the positive samples, ($T_+ + F_+$) in Equation 4.7. Recall is defined as the number of true positive samples (T_+) divided by the sum of the number of true positive (T_+) and the number of false negative (F_-) samples in Equation 4.7. The F1-score is the harmonic mean of precision and recall in Equation 4.8.

$$Precision = \frac{T_+}{T_+ + F_+} \qquad Recall = \frac{T_+}{T_+ + F_-} \tag{4.7}$$

$$F1 - score = \frac{2 \times \ Recall \ \times \ Precision}{Precision \ + \ Recall} = \frac{2T_+}{2T_+ + T_- + F_-} \tag{4.8}$$

The ROC curve represents how the true positive rate varies with different thresholds for the false positive rate. We can reach 100% true positive rate only if we accept a false positive rate of 100%. Conversely, if we want to achieve a 0% false positive rate, then that leads to 0% true positive rate. By changing the false positive rate threshold, we can trade-off the false positive rate with the true positive rate. Thus we use AUC of ROC curve to measure how effective classifiers are at various false positive rate thresholds.

4.5.1 Malware Detection

In this section, we report the detection rates (precision, recall, and F1-score) with 2 different data divisions, **TTA1** and **TTA2**. **TTA1** is the division of data according to the *traces*; while **TTA2** is the division of data according to the *programs*, as defined in §4.4.3.

Results from TTA1 Experiments

We train and test traces using various machine learning models and determine the detection rates (precision, recall, and F1-score) with **TTA1**. Then we plot the ROC curves and compute the AUCs. Table 4.2 shows the precision, recall, F1-score, and the AUCs of ROC curves. Any results with a value larger than 90% and smaller than 50% are set in **bold** and red, respectively. Figure 4·4 shows the ROC curves and the AUCs of ROC for different machine learning models.

DT uses the different features to classify examples at different tree branches. RF uses a collection of DTs to perform classifications. KNN determines the classes of each examples by comparing the number of examples within predefined distances. DT, RF, and KNN models target classifying outliers in the dataset (John, 1995), which fit our malware detection problem. According to our results, the detection rates of precision, recall, and F1-score are higher in DT, RF, and KNN models than any other models. The F1-scores in DT, RF, and KNN models are 83.39%, 84.84%, and 83.59%, respectively. Figure 4·4 shows the higher true positive rates of RF and DT with different thresholds, compared to other models. Accordingly, the AUCs in DT, RF, and KNN are 89.65%, 91.84%, and 89.26%, respectively. Figure 4·4(a) shows that the AUCs of ROC curves in DT and RF are the highest in various thresholds of false positive rates.

AdaBoost model leverages a collection of machine learning models. AdaBoost and NN model are designed to classifying clusters of examples. They perform worse in terms of detection rates compared to DT, RF, and KNN, as these models are designed to classify outliers. The F1-scores in AdaBoost and NN are 75.01% and 78.75%, respectively. In Figure 4·4(a), AdaBoost and NN models are also worse than DT and RF models. The AUC values for AdaBoost and NN are 80.57% and 84.41%, respectively.

The classification of Naive Bayes is only based on the probabilities of the occurrences of malware and benignware, which is a poor assumption to design classifiers (Rennie et al., 2003). In our design, we use the prior probability (50%) to design the Naive Bayes classifier. Naive Bayes model has many false negatives, which causes the F1-score value to be as low as 14.32%. The AUC of Naive Bayes is 58.11%. As a result, Naive Bayes classifies examples between malware and benignware with low detection rates.

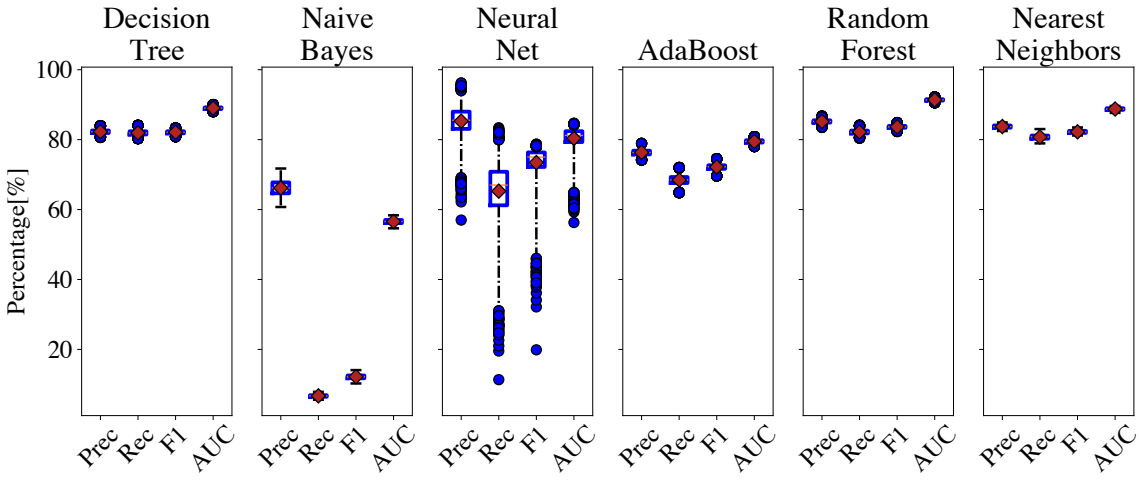
Results from TTA2 Experiments

We perform another experiment of training and testing using various machine learning models to show the detection rates (precision, recall, and F1-score) with **TTA2**. The F1-scores of DT, RF, KNN, Naive Bayes, AdaBoost, and NN models are 80.22%, 81.29%, 80.22%, 9.903%, 70.32%, and 35.66%, respectively, using **TTA2**, compared to 83.39%, 84.84%, 83.59%, 14.32%, 75.01%, and 78.75%, respectively, using **TTA1** in Table 4.2. By using **TTA2**, the detection rates are lower compared to the scenario using **TTA1**.

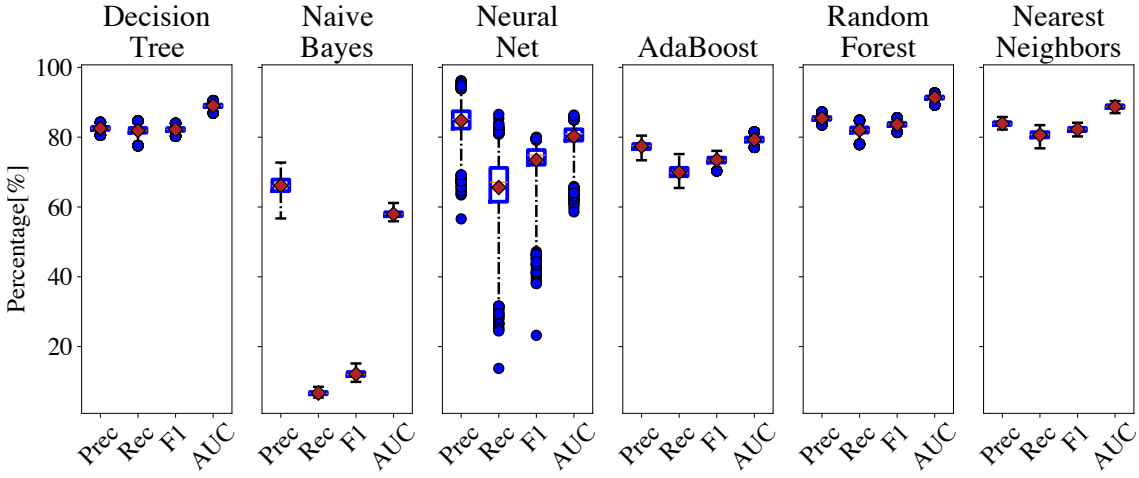
Figure 4·4(b) shows the ROC curves and the AUCs of ROC for different machine learning models using . The AUCs of ROC of DT, RF, KNN, Naive Bayes, AdaBoost, and NN models are 87.36%, 89.94%, 86.98%, 58.38% 77.96%, and 66.43%, respectively, using **TTA2** in Figure 4·4(b), compared to 89.65%, 91.84%, 89.26%, 58.11%, 80.57%, and 84.41%, respectively, using **TTA1** in Figure 4·4(a).

Demme et al. showed precision varying from 25% $\sim$ 100% (Demme et al., 2013) among different families of malware, without any recall values reported using **TTA1**. The median precision among all the families of malware is around 80%, with **TTA1**. Precision value of 80% corresponds to the False Discovery Rate[4] of 20%. Consider that a default Windows 7 installation has 1,323 executable files, an AV system with a

[4]False Discovery Rate $(F_+/(F_+ + T_+)$



(a)



(b)

Figure 4·5: Box plots of distributions of 10-fold cross-validation experiments using (a) TTA1 and (b) TTA2. Red diamonds are means, and blue box corresponds to cross-validation experiment results that lie between 25 and 75 percentiles. The whiskers (the short, horizontal lines outside the blue box) represent confidence interval equivalent to $\mu \pm 3\sigma$ of a Gaussian Distribution. The blue dots are outliers that are outside the $\mu \pm 3\sigma$ regime. On the X-axis, Prec is precision, Rec is recall, and F1 is F1 score. AUC is area under curve in ROC. These 10-fold cross-validation experiments show that we cannot achieve 100% malware detection accuracy.

20% False Discovery Rate would flag 264 of these files incorrectly as malware – clearly such a detection system would not be practical. As a result, such a malware detec-

tion method is not usable in real-life systems. With thousands of malware reported everyday, the offline training of malware detection cannot capture the same malware program that a user may encounter. In real-life cases, the malware detection rates of HPC-based malware detection would be those in columns of **TTA2** of Table 4.2 and Figure 4·4(b). These results show that high detection rates and robustness in detection are over-estimated due to division of data during training. Our comparison using **TTA1** and **TTA2** shows that using **TTA2** can cause the precisions to be even lower. Thus, prior works could have even worse precisions by using **TTA2**. In the next subsection, we will show that the results presented in this subsection are not an exception.

4.5.2 Cross-Validation

Cross-validation is a common practice in machine learning for avoiding the overfitting of machine learning models. Cross-validation is used to validate whether the detection rates are consistent with repeated training and testing (Kohavi et al., 1995). If the detection rates fluctuate during cross-validation, we can infer that the machine learning models are not trained properly. We observe that previous works either have no cross-validation or report no results from cross-validations. The lack of proper cross-validation motivates us to further evaluate the machine learning models using cross-validation. We use 3 times standard deviation (3σ) to quantify the fluctuations in detection rates. 3σ refers to $0.3\% \sim 99.7\%$ of random instances distributed within the range of 3σ. In the context of malware detection, a high value of 3σ in detection rates means that the performance of the model is not stable across different datasets.

Cross-validations for TTA1 Experiments

A common practice of cross-validation is using 10-fold cross-validation (Kohavi et al., 1995). 10-fold cross-validation divides the dataset into 10 subsets with equal number

of examples. It then performs training on 9 subsets and testing on the remaining one, with each subset as a testing subset. The standard deviations of detection rates in 10 experiments show whether the detection rates of the model are stable across 10 experiments. We consider that either a split of $60 - 20 - 20$ training-testing-validation or 10-fold cross-validation is not sufficient cross-validation, since the standard deviations of the detection rates increase with more examples in the dataset. We repeated the 10-fold cross-validations until the standard deviations of detection rates do not increase with more cross-validations. In this work, we perform cross-validation 1,000 times (randomly shuffling the examples before each training-and-testing split), which is 3 orders of magnitude more than previous works.

Figure 4·5 shows the distributions of detection rates (precision, recall, and F1-scores) with both **TTA1** and **TTA2** for various machine learning models. In Figure 4·5, the red diamonds are the means, and the blue boxes correspond to distributions of detection rates (Precisions, Recalls, and F1-scores) lying between 25 and 75 percentiles. The whiskers (the short, horizontal lines outside the blue box) represent the distributions of detection rates lying between 0.3% and 99.7%, which is equivalent to $\mu \pm 3\sigma$ of a Gaussian Distribution. The blue dots are outliers that are outside the $\mu \pm 3\sigma$ regime. A wide spread of distributions in detection rates means that the detection rates fluctuate across different training datasets. Conversely, a narrow spread of distributions means that the detection rates are stable across different training datasets. In DT, RF, KNN, NN, AdaBoost, and Naive Bayes models, the mean of distributions of F1-scores are 82.17%, 83.75%, 82.28%, 74%, 72.27%, 12.15%, respectively, with 3 standard deviations (3σ) of 1.416%, 1.326%, 1.388%, 13.2%, 2.365%, 2.392%, respectively.

Cross-validations for TTA2 Experiments

In DT, RF, KNN, NN, AdaBoost, Naive Bayes models, the mean of distributions of F1-scores using **TTA2** are 82.13%, 83.61%, 82.2%, 73.69%, 73.43%, 12.21%, compared to 82.17%, 83.75%, 82.28%, 74%, 72.27%, 12.15% using **TTA1**, respectively. In DT, RF, KNN, NN, AdaBoost, Naive Bayes models, the mean of distributions of F1-scores using **TTA2** are 2.145%, 2.336%, 2.248%, 14.88%, 3.29%, 2.611%, compared to 1.416%, 1.326%, 1.388%, 13.2%, 2.365%, 2.392% using **TTA1**, respectively. Comparing the results using **TTA1** and **TTA2**, the standard deviations of DT, RF, KNN, NN, AdaBoost, Naive Bayes models increased by 1.515×, 1.762×, 1.62×, 1.127×, 1.391×, 1.092×, respectively. The overall detection rates using **TTA2** have much higher variations compared to ones using **TTA1**.

As previous works did not report standard deviations of their cross-validations, we cannot compare these results. From the Figure 4·5, we can conclude that reporting results of one training-and-testing experiment does not provide sufficient information in performance of machine learning models. We can only evaluate the performance of these models by providing a distribution of detection rates.

The difference between standard deviations in Figure 4·5(a) and Figure 4·5(b) is due to the unrealistic assumption that the programs in the training set appear in the testing dataset. Figure 4·5(b) presents the results where the malicious program is not included in the training dataset. In conclusion, the mean of the distribution using **TTA2** is lower than that using **TTA1**, while the standard deviation of distribution using **TTA2** is higher than that using **TTA1**. In order to have a full evaluation on the machine learning models, it is imperative to use **TTA2** and exhibit a distribution of precision, recall, F1-score, and AUC of ROC curves.

4.5.3 Ransomware

In previous sections, the machine learning models are trained over the traces of HPCs to discriminate malware from benignware. We build a malware embedded in benignware and then show that this malware can evade HPC-based malware detection.

We craft the malware *simply* by infusing Notepad++ with a ransomware. Ransomware is malware that maliciously encrypts files and extorts users in exchange for the decryption keys (Young and Yung, 1996). Since 2016, ransomware has become one of the most popular malware, as Kaspersky Security Bulletin 2016 has shown that at least one business is attacked by ransomware every 40 seconds (Kas, 2016). We implement our ransomware to encrypt files when Notepad++ launches. The embedded ransomware traverses all the files in the "Pictures" folder and encrypts each file every 5 seconds with Microsoft Cryptography APIs (Cry, 2017). We measure the values of HPCs for modified Notepad++ in our experimental setup (§ 4.3). We randomly select 90% of the benignware and malware samples as the training set, while we test on Notepad++ and modified Notepad++. The precision of DT, Naive Bayes, NN, AdaBoost, RF and KNN is 0%, 0%, 0%, 50.85%, 0%, and 0%, respectively.

These results are not surprising, as machine learning models tolerate the noise and jitters during training on sampled HPCs, in order to extract the malicious behavior in the programs. These tolerance necessitates the machine learning algorithms to have errors even with the training datasets. In our malware example, the changes of HPC values caused by ransomware is overshadowed in the sampled values of HPCs from running Notepad++. The variation tolerance results in classifying the modified Notepad++ as benignware.

4.6 Discussion

We run Windows 7 32-bit operating system on AMD 15h family Bulldozer micro-architecture machine. The malware used in our experiments have an average age of 4.85 years (311 without age records). Weaver et al. performed extensive studies investigating the determinism of the measured HPC values in various micro-architectures (Weaver et al., 2013). By comparing the HPC values across different micro-architectures, Weaver et al. show that the HPCs in various architectures have similar levels of variations during sampling. Hence, our conclusions from Bulldozer micro-architecture are applicable to other micro-architectures. In our benignware and malware experiments, we chose to allow the access to the network for benignware and prevent malware from accessing network. This design choice does not affect the results of HPC measurements, since benignware and malware both function properly during experiments. For the reduction of dimensions, many other approaches can serve the same purpose as PCA. We use PCA in our designs as PCA is one of the most popular methods for reduction of dimensions.

Among all the algorithms that we applied to our detection systems, we chose the depths of DT the same as the depths of RF. The RF with lower depths can provide higher detection rate, since the RF performs majority votes among all the trees, which does not require the same depths as the DT. We decided to use the same depths of depths in RF as DT for simplicity. This decision should not affect the conclusion of this work.

4.7 Conclusion

Previous works have shown the use of Hardware Performance Counters in malware detections. The positive results in the prior works are due to optimistic assumptions and unrealistic experimental setups. In this work, we rigorously evaluate the malware

detection using HPCs and machine learning. We perform our experiments with a program count that exceeds $2 \sim 3$ times the previous works with 1000 times more cross-validations compared to the prior works. Our best result shows an F1-score of 80.78%. The corresponding False Discovery Rate ($F_+/(F_+ + T_+)$ is 15%. We believe that there is no cause-effect relationship between low-level micro-architectural events and high-level software behaviors. To show how fragile the detection system is, we infused one of the ransomeware into one benignware. Our system cannot detect the ransomeware with both benignware and ransomeware trained in the systems.

Chapter 5

Conclusions and Future Work

The attacks on computing systems at various layers of the computing stack have driven security researches to develop new techniques for protection at each layer. In this work, we have presented three different techniques deployed on different computing layers to secure the systems: HT detection using backside imaging, mapping cryptographic engines on the programmable SoC platform, and evaluating malware detection using HPCs and machine learning. We have addressed the threat models in each system; however, these areas still have open research problems. In this chapter, we are going to discuss completed research and the future work.

5.1 Hardware Trojan Detection using near-Infrared Backside Imaging

In this work, we have proposed a new technique that uses backside imaging for detecting HTs. The challenge of the backside imaging CMOS chips is to image the logic gates, which are smaller than the imaging wavelengths, $1 \sim 3\mu m$ for the near-Infrared light. To address this problem, we must engineer nano-scale metal structures in each gate to enhance its uniquenesses of the optical responses. We have developed our technique in two phases: fill cell design phase, and generic functional gate design phase.

We embed the maximum amount of metal into the fill cells in order to provide high optical reflectance. By doing so, we can detect any HTs inserted by replacement,

modification or shifting of the fill cells during the fabrication stage. However, if the functional gates are replaced with another set of the functional gates, the backside imaging cannot reliably detect the existence of any HTs. In order to increase the differences in the reflectance signatures between various gates, we engineer nano-antenna structures between the logic gate pairs. This nano-antenna structure in each gate pair can only be illuminated with designed polarizations, frequencies, and angles. Hence, we can detect any replacements of the function cells by comparing the locations of the nano-antennas with our simulated "golden references".

The nano-antenna designs in the generic gate design phase are initial demonstrations of watermarking the digital logic. There are existing open challenges in embedding the nano-antennas. In this work, we *manually* embed each nano-antenna into the gate pair designs, which present challenges for the gate designers to incorporate the nano-antennas. One potential improvement is to automate the nano-antenna embedding process using gate design tools. Another challenge of this work is to minimize the area overhead caused by the embedded nano-antennas. In our work, inserting nano-antennas results in as large as 20% increase in the gate area. Moving forward, novel antenna designs need to be developed to place the nano-antennas into individual gates in order to reduce the area overhead. For example, designing antennas that can be embedded in individual gates can be one potential research thrust. Moreover, by strategic placements of the fill cells and functional cells, we can improve the silicon area utilization during the chip level place-and-route phase.

5.2 High-Performance Low Energy Implementation of Cryptographic Algorithms on a Programmable SoC for IoT Devices

To secure the communication channels of the IoT devices, we must use cryptographic operations. Unfortunately, cryptographic operations are among the most power-hungry operations in today's computing systems. At the same time, IoT devices have low power budget. Moreover, the longevity of the IoT devices exceeds the cryptographic algorithms. We propose the use of FPGA to provide flexible hardware solutions for cryptographic operations, including AES-ECB, AES-CBC, AES-GCM, RSA, SHA, and DES. Our paper evaluates the mapping of crypto-engines to the FPGA of Zedboard as the FPGA can enable energy-efficient crypto operations than in software as well as provide the option to upgrade the cryptographic algorithms through re-configurations.

There are many other cryptographic algorithms that should be implemented in the IoT devices. The next step is to implement other cryptographic algorithms on the FPGA SoC platforms, for example, Elliptic-curve cryptography (ECC). ECC is the next generation asymmetric cryptographic algorithm that has been proposed as the replacement for the current asymmetric algorithm, RSA. ECC utilizes shorter key (256 bits versus 3072 bits in RSA) compared to RSA, at the same time, it provides the same security level for the computing system as RSA does. The short key size saves the memory space for low-power devices, such as IoT devices. We can measure the performance boost, energy savings and EDP reductions of these algorithms on the Zedboard platform.

Many of the malware in the current systems can modify the boot sequence to disable the system security in order to launch future attacks. Such malware becomes threats to the critical systems, such as the command and control system in factories,

hospitals, and power plants. Once the booting sequences are compromised, any protection systems launched after booting-up are vulnerable to other attacks. As the next steps for our security implementations, we can design the secure boot system on the SoC with FPGA. We can program the FPGA as the first level of the boot loader for the ARM CPU. This requires the modifications to the Operating System Kernel to start the booting logics on the FPGA. By embedding the unique signatures inside the FPGA, ARM processor cannot boot up the system unless the signatures are verified. The distributions of the signatures require secure channels for such deployments, which present challenges in developing these systems. These booting sequences can be deployed into the FPGA substrate, which provides the reconfigurations in the booting logics. By implementing the secure boot system on the FPGA, we can protect the boot sequence.

5.3 Malware Detection using Hardware Performance Counters with Machine Learning

HPCs are hardware units that are designed to count low-level, micro-architectural events. Many works have investigated malware detection using HPC profiles. However, we believe that there is no causation between low-level micro-architectural events and high-level software behavior. The strong positive results in the previous works are due to a series of optimistic assumptions and unrealistic experimental setups. In this work, we rigorously evaluate the idea of malware detection using HPCs through realistic assumptions and experimental setups. We observe a low fidelity in HPC-based malware detection when we increase the number of programs by a factor of $2\times \sim 3\times$ and increase the experiment numbers in cross-validation to 3 orders of magnitude higher than previous works. Our best result shows an F1-score of 80.78%. The corresponding False Discovery Rate ($F_+/(F_+ + T_+)$ is 15%. This means that among

1,323 executable files in the Windows operating system, 198 files will be flagged as malware. We also demonstrate the infeasibility in HPC-based malware detection with Notepad++ infused with a ransomware, which cannot be detected in our HPC-based malware detection system.

We have evaluated the HPC malware detection on the AMD machines. We can also apply similar techniques to the Intel, ARM, and other processor architectures. There are Other processor architectures are different from AMD processor architecture and have different micro-architectural events. This will result in different choices of the micro-architectural events for detecting malware. A thorough evaluation of malware detection using HPC on these architectures will help determine if our claims are applicable to other architectures.

To perform application-level malicious behaviors that are more accurate and can perform more complicated operations than counting, such as observing architectural, micro-architectural events and perform basic computations. We can embed these hardware monitors into processor to monitor hardware behaviors. We can use these monitors to design defense systems against the low-level attacks, such as "row-hammer". "Row-hammer" attacks repeatedly rewrite the memory rows in order to change the stored values in the neighboring rows, which can be used for overwriting sensitive information, such as passwords. These malicious behaviors should be captured by low-level monitors. By monitoring the behavior on the memory micro-controllers, we can identify the statistical increments in the "row-hammer" attacks on the memory controller side. These monitors should be designed such that they can perform simple arithmetic operations, such as comparison, addition, and bit shifts, in order to identify the abnormal memory accesses, at the same time, not trigger any false alarms.

References

(2010). *Intel Itanium Architecture Software Developer's Manual*. Intel Corporation.

(2014). Trust-hub website. https://bit.ly/2TCHRwd. Accessed: 2014-11-30.

(2015). *BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 10h-1Fh Processors*. Advanced Micro Devices, Inc.

(2016). Armv8 technology preview. https://bit.ly/2KzT7WW. Accessed: 2016-02-09.

(2016). Avr1318: Using the xmega built-in aes accelerator. https://bit.ly/2UHPxhf. Accessed: 2016-02-09.

(2016). Cisco visual networking index: Global mobile data traffic forecast update, 20152020 white paper - cisco. https://bit.ly/1W26UQo. (Accessed on 07/24/2016).

(2016a). Essays:others- schneier on security. https://bit.ly/2Icnu3C. (Visited on 02/18/2016).

(2016a). Intel advanced encryption standard instructions (aes-ni). https://intel.ly/2UtXdEy. Accessed: 2016-02-09.

(2016b). Intel completes acquisition of altera — intel newsroom. https://intel.ly/2n46Oh4. (Accessed on 07/28/2016).

(2016). Internet of everything. https://stanford.io/2U6b9PW. Accessed: 2016-02-10.

(2016). Kaspersky security bulletin 2016. review of the year. overall statistics for 2016. https://bit.ly/2GhrPPJ. (Accessed on 12/10/2017).

(2016). Linaro. http://www.linaro.org/. (Accessed on 08/04/2016).

(2016). Nist releases sha-3 cryptographic hash standard. https://bit.ly/1IOb1M8. (Accessed on 07/24/2016).

(2016). opencores. http://opencores.org/projects. Accessed: 2016-02-10.

(2016). Openssl. https://www.openssl.org/. (Accessed on 07/26/2016).

(2016b). What aothers? https://bit.ly/2UcUdHm. (Visited on 02/18/2016).

(2017). Android debug bridge. https://bit.ly/2jOqfuZ. (Accessed on 4/22/2019).

(2017). bindfs. https://bindfs.org/. (Accessed on 12/05/2017).

(2017). Cryptography reference (windows). https://bit.ly/2v01wYN. (Accessed on 11/18/2017).

(2017). Dynamorio dynamic instrumentation tool platform. https://bit.ly/2D7ucnp. (Accessed on 12/02/2017).

(2017). Futuremark. https://www.futuremark.com/. (Accessed on 11/15/2017).

(2017). Ninite. https://ninite.com/. (Accessed on 11/15/2017).

(2017). Npackd. https://npackd.appspot.com/. (Accessed on 11/15/2017).

(2017). Performance: Python package index. https://bit.ly/2v0cfm2. (Accessed on 11/30/2017).

(2017). Rabbitmq. http://www.rabbitmq.com/. (Accessed on 11/12/2017).

(2017). Samba - opening windows to a wider world. https://www.samba.org/. (Accessed on 12/05/2017).

(2019). As chip design costs skyrocket, 3nm process node is in jeopardy - extremetech. https://bit.ly/2IrGYAC. (Accessed on 02/06/2019).

(2019). Equifax breach exposed millions of drivers licenses, phone numbers, emails — ars technica. https://bit.ly/2IkINzH. (Accessed on 01/30/2019).

(2019). Facebook and cambridge analytica: What you need to know as fallout widens - the new york times. https://nyti.ms/2HP4Dr3. (Accessed on 01/30/2019).

(2019). How to reverse engineer software (windows) in a right way. https://bit.ly/2u0GRU7. (Accessed on 03/04/2019).

(2019). iphone 5 a6 soc reverse engineered, reveals rare hand-made custom cpu, and tri-core gpu - extremetech. https://bit.ly/2O16AVO. (Accessed on 03/04/2019).

(2019). Library creator platform. https://bit.ly/2UPv4aR. (Accessed on 04/22/2019).

(2019). Meltdown and spectre. https://meltdownattack.com/. (Accessed on 01/30/2019).

(2019). Reverse engineering integrated circuits with degate - home. https://bit.ly/2VSRZym. (Accessed on 03/04/2019).

(2019). The semiconductor industry and the power of globalisation. https://econ.st/2KRcwiB. (Accessed on 03/01/2019).

(2019). Trusted integrated circuits (trust). https://www.darpa.mil/program/trusted-integrated-circuits. (Accessed on 04/22/2019).

Alkabani, Y. and Koushanfar, F. (2009). Consistency-based characterization for ic trojan detection. In *Proceedings of the International Conference On Computer Aided Design (ICCAD)*, pages 123–127.

Banga, M. and Hsiao, M. S. (2009a). A novel sustained vector technique for the detection of hardware trojans. In *Proceedings of the International Conference on VLSI Design (VLSID)*, pages 327–332. IEEE.

Banga, M. and Hsiao, M. S. (2009b). Vitamin: Voltage inversion technique to ascertain malicious insertions in ics. In *Proceedings of the Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 104–107. IEEE.

Banga, M. and Hsiao, M. S. (2011). Odette: A non-scan design-for-test methodology for trojan detection in ics. In *Proceedings of the Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 18–23. IEEE.

Bao, C., Xie, Y., and Srivastava, A. (2015). A security-aware design scheme for better hardware trojan detection sensitivity. In *Proceedings of the Symposium on Hardware Oriented Security and Trust (HOST)*, pages 52–55. IEEE.

Barker, E., Barker, W., Burr, W., Polk, W., and Smid, M. (2007). Recommendation for key management. part 1: General. nist special publication 800-57 part 1 revisions 4. *NIST Special publication, Gaithersburg, MD: National Institute of Standards and Technology http://dx.doi.org/10.6028/NIST.SP.800-57pt1r4*, 800(57):1–142.

Baumgarten, A., Tyagi, A., and Zambreno, J. (2010). Preventing ic piracy using reconfigurable logic barriers. *Proceedings of the Design & Test of Computers (DTC)*, (1):66–75.

Bellard, F. (2005). Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46.

Bossuet, L. et al. (2013). Architectures of flexible symmetric key crypto engines-a survey: From hardware coprocessor to multi-crypto-processor system on chip. *ACM Computing Surveys, 45(4): Article 41. doi:10.1145/2501654.2501655.*

Bulens, P., Standaert, F.-X., Quisquater, J.-J., Pellegrin, P., and Rouvroy, G. (2008). Implementation of the aes-128 on virtex-5 fpgas. In *International Conference on Cryptology in Africa*.

Çapoğlu, İ. R., White, C. A., Rogers, J. D., Subramanian, H., Taflove, A., and Backman, V. (2011). Numerical simulation of partially coherent broadband optical imaging using the finite-difference time-domain method. *Optics letters (Opt. Lett.)*, 36(9):1596–1598.

Chakraborty, R. S. and Bhunia, S. (2009). Harpoon: an obfuscation-based soc design methodology for hardware protection. *Transaction on Computer-Aided Design of Integrated Circuits and Systems*, 28(10):1493–1502.

Chakraborty, R. S., Narasimhan, S., and Bhunia, S. (2009). Hardware trojan: Threats and emerging solutions. In *Proceedings of the International High Level Design Validation and Test Workshop (HLDVT)*, pages 166–171. IEEE.

Chaves, R. et al. (2006). Reconfigurable cryptographic processor. In *Workshop on Circuits, Systems and Signal Processing–CSSP*.

Curtin, M. (2005). Data encryption standard. *Brute Force: Cracking the Data Encryption Standard, New York: Copernicus Books. (Pp. 1122)*.

DARPA (2019). Integrity and reliability of integrated circuits (iris). `https://bit.ly/2NZtigF`. (Accessed on 03/11/2019).

Demme, J., Maycock, M., Schmitz, J., Tang, A., Waksman, A., Sethumadhavan, S., and Stolfo, S. (2013). On the feasibility of online malware detection with performance counters. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, page 559.

Drongowski, P. J. (2008). An introduction to analysis and optimization with amd codeanalyst performance analyzer. `https://bit.ly/2GsOl8k`.

Elbirt, A. J. and Paar, C. (2005). An instruction-level distributed processor for symmetric-key cryptography. *IEEE Transactions on Parallel and distributed Systems*, 16(5):468–480.

electroiq.com (2019). Intels 22-nm trigate transistors exposed. `https://bit.ly/2EW4Cll`. (Accessed on 03/04/2019).

Exurville, I., Zussa, L., Rigaud, J.-B., and Robisson, B. (2015). Resilient hardware trojans detection based on path delay measurements. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 151–156. IEEE.

Fan, X., Gong, G., Lauffenburger, K., and Hicks, T. (2010). Fpga implementations of the hummingbird cryptographic algorithm. In *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 48–51. IEEE.

Forte, D. et al. (2013). Temperature tracking: An innovative run-time approach for hardware trojan detection. In *Proceedings of the International Conference On Computer Aided Design (ICCAD)*, pages 532–539.

Freund, Y. and Schapire, R. E. (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer.

Genkin, D., Shamir, A., and Tromer, E. (2014). Rsa key extraction via low-bandwidth acoustic cryptanalysis. In *Proceedings of the International Cryptology Conference (CRYPTO)*, pages 444–461. Springer.

Good, T. and Benaissa, M. (2005). Aes on fpga from the fastest to the smallest. In *International Workshop on Cryptographic Hardware and Embedded Systems*.

Good, T. and Benaissa, M. (2006). Very small fpga application-specific instruction processor for aes. *IEEE Transactions on Circuits and Systems I: Regular Papers*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Granado-Criado, J. M., Vega-Rodríguez, M. A., Sánchez-Pérez, J. M., and Gómez-Pulido, J. A. (2010). A new methodology to implement the aes algorithm using partial and dynamic reconfiguration. *INTEGRATION, the VLSI journal*.

Grand, M. et al. (2009). A reconfigurable crypto sub system for the software communication architecture. In *Military Communications Conference, 2009. MILCOM 2009. IEEE*.

Hamalainen, P., Alho, T., Hannikainen, M., and Hamalainen, T. D. (2006). Design and implementation of low-area and low-power aes encryption hardware core. In *9th EUROMICRO Conference on Digital System Design (DSD'06)*.

Hämäläinen, P., Hännikäinen, M., and Hämäläinen, T. D. (2007). Review of hardware architectures for advanced encryption standard implementations considering wireless sensor networks. In *International Workshop on Embedded Computer Systems*, pages 443–453. Springer.

Haykin, S. (1983). *Communication System*. Wiley Series in Management Series. John Wiley & Sons.

Hernandez, G., Arias, O., Buentello, D., and Jin, Y. (2014). Smart nest thermostat: A smart spy in your home. *Black Hat USA*, pages 1–8.

Hicks, M., Finnicum, M., King, S. T., Martin, M. M., and Smith, J. M. (2010). Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically. In *Proceedings of the Symposium on Security and Privac (SP)*, pages 159–172. IEEE.

Hoang, T. et al. (2012). An efficient fpga implementation of the advanced encryption standard algorithm. In *2012 IEEE RIVF International Conference on Computing & Communication Technologies, Research, Innovation, and Vision for the Future*, pages 1–4. IEEE.

Hosseinabady, M. and Nunez-Yanez, J. L. (2014). Run-time power gating in hybrid arm-fpga devices. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL) doi:10.1109/FPL.2014.6927503*, pages 1–6. IEEE.

Hu, K., Nowroz, A. N., Reda, S., and Koushanfar, F. (2013). High-sensitivity hardware trojan detection using multimodal characterization. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1271–1276. IEEE.

Illera, A. et al. (2014). Lights off! the darkness of the smart meters. *BlackHat Europe https://ubm.io/2IEB002*.

Ippolito, S. B. et al. (2004). High spatial resolution subsurface thermal emission microscopy. *Applied Physics Letters (APL)*, 84(22):4529.

Ippolito, S. B., Goldberg, B. B., and Ünlü, M. S. (2005). Theoretical analysis of numerical aperture increasing lens microscopy. *Proceedings of the Applied Physics (AIP)*, 97(5):053105.

Jin, Y. and Makris, Y. (2008). Hardware trojan detection using path delay fingerprint. In *Proceedings of the International Workshop on Hardware-Oriented Security and Trust (HOST)*, pages 51–57.

John, G. H. (1995). Robust decision trees: Removing outliers from databases. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pages 174–179. AAAI Press.

Karri, R., Rajendran, J., Rosenfeld, K., and Tehranipoor, M. (2010). Trustworthy hardware: Identifying and classifying hardware trojans. *Computer Journal*, 43(10):39–46.

Kazdagli, M., Reddi, V. J., and Tiwari, M. (2016). Quantifying and improving the efficiency of hardware-based mobile malware detectors. In *Proceedings of the 49th International Symposium on Microarchitecture (MICRO)*, pages 1–13.

Keys, R. G. (1981). Cubic convolution interpolation for digital image processing. *the Transactions on Acoustics, Speech and Signal Processing (TASSP)*, 29(6):1153–1160.

Khasawneh, K. N., Abu-Ghazaleh, N., Ponomarev, D., and Yu, L. (2017). Rhmd: evasion-resilient hardware malware detectors. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 315–327.

Khasawneh, K. N., Ozsoy, M., Donovick, C., Abu-Ghazaleh, N., and Ponomarev, D. (2015). Ensemble learning for low-level hardware-supported malware detection. In *International Workshop on Recent Advances in Intrusion Detection (RAID)*, pages 3–25.

Kindereit, U. et al. (2007). Quantitative Investigation of Laser Beam Modulation in Electrically Active Devices as Used in Laser Voltage Probing. *Transactions on Device and Materials Reliability (TDMR)*, 7(1):19–30.

Kirat, D., Vigna, G., and Kruegel, C. (2014). Barecloud: Bare-metal analysis-based evasive malware detection. In *USENIX Security Symposium (SP)*, pages 287–301.

Klinefelter, A., Roberts, N. E., Shakhsheer, Y., Gonzalez, P., Shrivastava, A., Roy, A., Craig, K., Faisal, M., Boley, J., Oh, S., et al. (2015). 21.3 a 6.45μw self-powered iot soc with integrated energy-harvesting power management and ulp asymmetric radios. In *2015 IEEE International Solid-State Circuits Conference-(ISSCC) Digest of Technical Papers*.

Kocher, P., Jaffe, J., and Jun, B. (1999). Differential power analysis. In *Proceedings of the Annual International Cryptology Conference (CRYPTO)*, pages 388–397. Springer.

Koeune, F. and Standaert, F.-X. (2005). A tutorial on physical security and side-channel attacks. In *Foundations of Security Analysis and Design III*, pages 78–108. Springer.

Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, volume 14, pages 1137–1145. Morgan Kaufmann Publishers, Inc.

Köklü, F. H. and Unlü, M. S. (2010). Subsurface microscopy of interconnect layers of an integrated circuit. *Optics Letters (Opt. Lett.)*, 35(2):184–6.

Leith, V. (2010). The rijndael block cipher.

Li, J. and Lach, J. (2008). At-speed delay characterization for ic authentication and trojan horse detection. In *Proceedings of the International Workshop on Hardware-Oriented Security and Trust (HOST)*, pages 8–14. IEEE.

Love, E., Jin, Y., and Makris, Y. (2012). Proof-carrying hardware intellectual property: A pathway to trusted module acquisition. *Transactions on Information Forensics and Security*, 7(1):25–40.

Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V. J., and Hazelwood, K. (2005). Pin: building customized program analysis tools with dynamic instrumentation. In *Acm sigplan notices*, volume 40, pages 190–200. ACM. extras:luk05:pin.

Martin, A. et al. (2008). Development approaches for an international tactical radio cryptographic api. In *Proceedings of the Software Design Radio Technical Conference–SDR*.

Mayer, C. P. (2009). Security and privacy challenges in the internet of things. *Electronic Communications of the EASST*.

Miller, B. et al. (2012). A survey scada of and critical infrastructure incidents. In *Proceedings of the 1st Annual conference on Research in information technology*.

Miller, C. and Valasek, C. (2014). A survey of remote automotive attack surfaces. *black hat USA https://bit.ly/2ZjXtEG*, 2014:94.

Nethercote, N. and Seward, J. (2007). Valgrind: a framework for heavyweight dynamic binary instrumentation. In *ACM Sigplan notices*, volume 42, pages 89–100. ACM.

Ngo, X. T., Bhasin, S., Danger, J.-L., Guilley, S., and Najm, Z. (2015). Linear complementary dual code improvement to strengthen encoded circuit against hardware trojan horses. In *Proceedings of the Symposium on Hardware Oriented Security and Trust (HOST)*, pages 82–87. IEEE.

Novotny, L. and Hecht, B. (2006). *Principles of Nano-Optics*. Cambridge University Press, Cambridge, UK.

Nowroz, A. N., Hu, K., Koushanfar, F., and Reda, S. (2014). Novel techniques for high-sensitivity hardware trojan detection using thermal and power maps. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(12):1792–1805.

Oswald, E., Mangard, S., Pramstaller, N., and Rijmen, V. (2005). A side-channel analysis resistant description of the aes s-box. In *International Workshop on Fast Software Encryption*.

Ozsoy, M., Donovick, C., Gorelik, I., Abu-Ghazaleh, N., and Ponomarev, D. (2015). Malware-aware processors: A framework for efficient online malware detection. In *Proceedings of the 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 651–661.

Pappu, R., Recht, B., Taylor, J., and Gershenfeld, N. (2002). Physical one-way functions. *Science*, 297(5589):2026–2030.

Patil, H., Cohn, R., Charney, M., Kapoor, R., Sun, A., and Karunanidhi, A. (2004). Pinpointing representative portions of large intel itanium programs with dynamic instrumentation. In *Proceedings of 37th International Symposium on Microarchitecture (MICRO)*, pages 81–92.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Popović, M. and Taflove, A. (2004). Two-dimensional fdtd inverse-scattering scheme for determination of near-surface material properties at microwave frequencies. *Proceedings of the Transactions on Antennas and Propagation (TAP)*, 52(9):2366–2373.

Potkonjak, M., Nahapetian, A., Nelson, M., and Massey, T. (2009a). Hardware trojan horse detection using gate-level characterization. In *Proceedings of the 46th Annual Design Automation Conference*, pages 688–693. ACM.

Potkonjak, M., Nahapetian, A., Nelson, M., and Massey, T. (2009b). Hardware trojan horse detection using gate-level characterization. In *Proceedings of the Design Automation Conference (DAC)*, pages 688–693. IEEE.

Pramstaller, N., Rechberger, C., and Rijmen, V. (2006). A compact fpga implementation of the hash function whirlpool. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*.

Rad, R. et al. (2008). Power supply signal calibration techniques for improving detection resolution to hardware trojans. In *Proceedings of the International Conference On Computer Aided Design (ICCAD)*, pages 632–639.

Rennie, J., Shih, L., Teevan, J., and Karger, D. (2003). Tackling the poor assumptions of naive bayes classifiers (pdf). ICML.

Rohatgi, P. (2009). Electromagnetic attacks and countermeasures. In *Cryptographic Engineering*, pages 407–430. Springer.

Rostami, M. et al. (2014). A primer on hardware security: Models, methods, and metrics. *Proceedings of the IEEE*, 102(8):1283–1295.

Rouvroy, G., Standaert, F.-X., Quisquater, J.-J., and Legat, J.-D. (2004). Compact and efficient encryption/decryption module for fpga implementation of the aes rijndael very well suited for small embedded applications. In *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*.

Roy, J. A., Koushanfar, F., and Markov, I. L. (2008). Epic: Ending piracy of integrated circuits. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 1069–1074. ACM.

Saarinen, M.-J. O. (2014). Simple aead hardware interface (sæhi) in a soc: Implementing an on-chip keyak/whirlbob coprocessor. In *Proceedings of the 4th International Workshop on Trustworthy Embedded Devices*.

Saarinen, M.-J. O. and Brumley, B. B. (2014). Lighter, faster, and constant-time: Whirlbob, the whirlpool variant of stribob. *IACR Cryptology ePrint Archive*.

Sakiyama, K., Batina, L., Preneel, B., and Verbauwhede, I. (2007). Hw/sw co-design for public-key cryptosystems on the 8051 micro-controller. *Computers & Electrical Engineering*, 33(5-6):324–332.

Salmani, H., Tehranipoor, M., and Plusquellic, J. (2012). A novel technique for improving hardware trojan detection and reducing trojan activation time. *Transactions on Very Large Scale Integration Systems*, 20(1):112–125.

Schlösser, A., Nedospasov, D., Krämer, J., Orlic, S., and Seifert, J.-P. (2012). Simple photonic emission analysis of aes. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 41–57. Springer.

Sebastián, M., Rivera, R., Kotzias, P., and Caballero, J. (2016). Avclass: A tool for massive malware labeling. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 230–253. Springer.

Serebrin, B. and Hecht, D. (2011). Virtualizing performance counters. In *Proceedings of the European Conference on Parallel Processing*, pages 223–233, Bordeaux, France.

Shah, S., Velegalati, R., Kaps, J.-P., and Hwang, D. (2010). Investigation of dpa resistance of block rams in cryptographic implementations on fpgas. In *2010 International Conference on Reconfigurable Computing and FPGAs*.

Singh, B., Evtyushkin, D., Elwell, J., Riley, R., and Cervesato, I. (2017). On the detection of kernel-level rootkits using hardware performance counters. In *Proceedings of the 17th Asia Conference on Computer and Communications Security (AsiaCCS)*, pages 483–493. ACM.

Song, P., Stellari, F., Pfeiffer, D., Culp, J., Weger, A., Bonnoit, A., Wisnieff, B., and Taubenblatt, M. (2011). Marvelmalicious alteration recognition and verification by emission of light. In *Proceedings of the International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 117–121. IEEE.

Stellari, F., Song, P., and Ainspan, H. A. (2014). Functional block extraction for hardware security detection using time-integrated and time-resolved emission measurements. In *Proceedings of the VLSI Test Symposium (VTS)*, pages 1–6. IEEE.

Sullivan, D., Biggers, J., Zhu, G., Zhang, S., and Jin, Y. (2014). Fight-metric: Functional identification of gate-level hardware trustworthiness. In *Proceedings of the Design Automation Conference (DAC)*, pages 1–4. IEEE.

Tang, A., Sethumadhavan, S., and Stolfo, S. J. (2014). Unsupervised anomaly-based malware detection using hardware features. In *International Workshop on Recent Advances in Intrusion Detection (RAID)*, pages 109–129.

Tehranipoor, M. and Koushanfar, F. (2010). A survey of hardware trojan taxonomy and detection. *IEEE design & test of computers*, 27(1):10–25.

Theodoropoulos, D. et al. (2008). Cproc: An efficient cryptographic coprocessor. In *16th IFIP/IEEE International Conference on Very Large Scale Integration*.

Török, P., Munro, P., and Kriezis, E. E. (2008). High numerical aperture vectorial imaging in coherent optical microscopes. *Optics Express (Opt. Express)*, 16(2):507–523.

Torrance, R. and James, D. (2011). The state-of-the-art in semiconductor reverse engineering. In *Proceedings of the Design Automation Conference (DAC)*, pages 333–338. IEEE.

Tsoutsos, N. G. and Maniatakos, M. (2014). Fabrication attacks: Zero-overhead malicious modifications enabling modern microprocessor privilege escalation. *Transactions on Emerging Topics in Computing*, 2(1):81–93.

Vaidyanathan, K., Das, B. P., and Pileggi, L. (2014a). Detecting reliability attacks during split fabrication using test-only beol stack. In *Proceedings of the Design Automation Conference (DAC)*, pages 1–6. ACM.

Vaidyanathan, K., Liu, R., Sumbul, E., Zhu, Q., Franchetti, F., and Pileggi, L. (2014b). Efficient and secure intellectual property (ip) design with split fabrication. In *Proceedings of the Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 13–18. IEEE.

Valamehr, J., Sherwood, T., Kastner, R., Marangoni-Simonsen, D., Huffmire, T., Irvine, C., and Levin, T. (2013). A 3-d split manufacturing approach to trustworthy system development. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(4):611–615.

Vaslin, R. et al. (2007). Low latency solution for confidentiality and integrity checking in embedded systems with off-chip memory. In *Reconfigurable communication-centric Socs 2007*.

Virustotal (2017). Virustotal. https://bit.ly/1hr5HS1. (Accessed on 07/12/2017).

Waksman, A. and Sethumadhavan, S. (2011). Silencing hardware backdoors. In *Proceedings of the Symposium on Security and Privacy (SP)*, pages 49–63. IEEE.

Waksman, A., Suozzo, M., and Sethumadhavan, S. (2013). Fanci: identification of stealthy malicious logic using boolean functional analysis. In *Proceedings of the conference on Computer & Communications Security (CCS)*, pages 697–708. ACM.

Wang, M.-Y. et al. (2010). Single-and multi-core configurable aes architectures for flexible security. *Transactions on Very Large Scale Integration (VLSI) Systems*.

Wang, X., Chai, S., Isnardi, M., Lim, S., and Karri, R. (2016). Hardware performance counter-based malware identification and detection with adaptive compressive sensing. *ACM Transactions on Architecture and Code Optimization (TACO)*, 13(1):3.

Weaver, V. M. and McKee, S. A. (2008). Can hardware performance counters be trusted? In *Proceedings of International Symposium on Workload Characterization (IISWC)*, pages 141–150. IEEE.

Weaver, V. M., Terpstra, D., and Moore, S. (2013). Non-determinism and overcount on modern hardware performance counter implementations. In *Proceedings of International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 215–224. IEEE.

Wei, S., Li, K., Koushanfar, F., and Potkonjak, M. (2012a). Hardware trojan horse benchmark via optimal creation and placement of malicious circuitry. In *Proceedings of the Design Automation Conference (DAC)*, pages 90–95. ACM.

Wei, S., Meguerdichian, S., and Potkonjak, M. (2010). Gate-level characterization: foundations and hardware security applications. In *Proceedings of the Design Automation Conference (DAC)*, pages 222–227. IEEE.

Wei, S., Nahapetian, A., Nelson, M., Koushanfar, F., and Potkonjak, M. (2012b). Gate characterization using singular value decomposition: Foundations and applications. *Transactions on Information Forensics and Security*, 7(2):765–773.

Wei, S. and Potkonjak, M. (2012). Scalable hardware trojan diagnosis. *Transactions on Very Large Scale Integration Systems*, 20(6):1049–1057.

Wilcox, I., Saqib, F., and Plusquellic, J. (2015). Gds-ii trojan detection using multiple supply pad v dd and gnd i ddq s in asic functional units. In *Proceedings of the Symposium on Hardware Oriented Security and Trust (HOST)*, pages 144–150. IEEE.

Xiao, K., Forte, D., and Tehranipoor, M. (2014). A novel built-in self-authentication technique to prevent inserting hardware trojans. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(12):1778–1791.

Xiao, K., Forte, D., and Tehranipoor, M. M. (2015). Efficient and secure split manufacturing via obfuscated built-in self-authentication. In *Proceedings of the Symposium on Hardware Oriented Security and Trust (HOST)*, pages 14–19. IEEE.

Yang, K., Hicks, M., Dong, Q., Austin, T., and Sylvester, D. (2016). A2: Analog malicious hardware. In *Proceedings of the Symposium on the Security and Privacy (SP)*, pages 18–37. IEEE.

Young, A. and Yung, M. (1996). Cryptovirology: Extortion-based security threats and countermeasures. In *Proceedings of Security and Privacy*, pages 129–140. IEEE.

Zhou, B., Adato, R., Zangeneh, M., Yang, T., Uyar, A., Goldberg, B., Unlu, S., and Joshi, A. (2015). Detecting hardware trojans using backside optical imaging of embedded watermarks. In *Proceedings of the Design Automation Conference (DAC)*, pages 1–6. IEEE.

Zjajo, A. (2014). Random process variation in deep-submicron cmos. In *Stochastic Process Variation in Deep-Submicron CMOS*, pages 17–54. Springer.

Zonouz, S., Rrushi, J., and McLaughlin, S. (2014). Detecting industrial control malware using automated plc code analytics. *Proceedings of the Symposium on Security & Privacy (SP)*, 12(6):40–47.

CURRICULUM VITAE

Boyou Zhou

Department of Electrical and Computer Engineering, Boston University
8 St Mary's St, Photonics Center, Room 340, Boston, MA 02215
Email: bobzhou@bu.edu, Phone: 617-678-8480, `https://cv.boyouz.com`

Education

- Boston University, Department of ECE, Ph.D. candidate, 2013 - present

- Southeast University, Department of ECE, B.S., 2013

Summary

I am a Ph.D Student in Integrated Circuits and System Group of Electrical and Computer Engineering in Boston University. My research focus is Computer Hardware Design. During my graduate school, I have completed on 4 projects, and am currently on the 5th project. I have worked 3 internships with 2 companies. I am looking for a full-time job in the hardware area and I am expecting to graduate in spring 2019.

Research Projects

- BlackParrot Open-source RISC-V core Design

  I have worked on a RISC-V multi-core in-order processor open-source design of RISC-V processor. The project is a part of Posh Open Source Hardware program (POSH).

  - My main role is to implement the Front-End of the RISC-V design, except Instruction-Cache and Instruction-TLB.
  - I have worked on the testing infrastructures for RISC-V tests.

- Evaluation of Malware Detection using Hardware Performance Counter

  By literature survey, I found prevalent unrealistic assumptions and incorrect evaluations in Hardware Performance Counters (HPCs) based malware detection using machine learning.

  - I re-evaluated the HPC-based malware detection by measuring HPC traces on 1,000 malware and 1,000 benignware.

  - I performed data analysis on these HPC values using K Nearest Neighbors (KNN), Decision Tree (DT), Multilayer Perceptrons (MLP), Naive Bayes, AdaBoost and Random Forest (RF) with 1,000 10-fold cross-validations.
  - In summary, I found out that HPCs cannot indicate the maliciousness of the programs.

- Hardware Trojan Detection Using Near-IR Imaging

  Hardware Trojans (HT) are maliciously-inserted hardware blocks in Integrated Circuits (IC). Traditional testing/validation techniques, for example, power and timing validation methods, are not able to detect HTs. Our group collaborated with Optical Characterization and Nanophotonics Laboratory to develop near-Infrared backside imaging.

  - I engineered optical structures in standard cells, and designed optical watermarks in the ICs.
  - I generated clean circuits and HT inserted circuits for evaluations.
  - I used correlation, noise analysis, and machine learning to enhance the accuracy of detecting HTs.

- FPGA based Cryptographic Accelerators

  To decrease the power consumption and accelerate computations on the Internet of Things (IoT) devices, I worked on the FPGA-assisted cryptographic accelerations using Zedboard as the reconfigurable substrate evaluation platform.

  - I implemented cryptographic algorithms including symmetric, asymmetric cryptography, and secure hash functions.
  - I integrated our cryptographic engines into OpenSSL library to inherit the library's support for block cipher modes.

- On-going Project: Control Flow Integrity Enforcements

  Control Flow Integrity (CFI) prevents the computer system from diverting control flow. Current commercially available CFI enforcements use software-based techniques. All the hardware-assisted CFIs under research have low target precisions. In order to provide fine-grained CFI enforcements, the hardware needs to support high target precisions.

  - I am working on developing the fine-grained CFI assisted by hardware and will perform evaluations on RISCV.

Work Experience

- Qualcomm Inc. Santa Clara, CA, 2016
  - I implemented the Digital Signal Processing (DSP) module in the touch-screen chip design. I modeled my signal processing in Python with the builder design pattern, to validate my digital design.
  - The builder pattern provided the interfaces for modifying and remodeling in DSP for product improvements.
  - I designed an automated connection of individual blocks in Verilog with Python, emacs, and bash.
  - I hosted the Sphinx (website-based-documention) for my works during the internship.
- Analog Device Inc. Wilmington, MA, 2015
  - Real Number Modeling (RNM) is the technique to model analog blocks on the modular level, instead of continuous signal simulation. The benefits of RNM is to provide fast simulation, and direct integration with digital blocks.
  - I mainly explored the Real Number Modeling in modeling ICs and peripheral structures in System Verilog.
  - I verified between my RNM model and continuous simulation model using bash and python.
- Analog Device Inc. San Jose, CA, 2014
  - I mainly explored the Real Number Modeling in modeling ICs and peripheral structures.
  - I used Verilog and Verilog-ams to model the voltage behaviors.
  - I automated the verification between my RNM model and continuous simulation model using bash and perl.

Patent

- Fill Cell Watermark for Circuit Layout Validation, patented application submitted

Relevant Courses

- Cyber Security, Embedded System, Computer Architecture, Operating System, Hardware Security, Data Structure.

Skills

- Softwares: Linux core-utils, Debuggers (gdb, adb), Network Tools (nmap, iptables, netcat, tcpdump)
- Programming Languages: Python, C/C++, Bash, Java, HTML, CSS, Javascript

Teaching Experience

- Graduate Teaching Fellow for Operating System (Raspberry pi working platform) at Boston University (Spring 2014).
- Graduate Teaching Fellow for Introduction to Software Engineering (C++) (Fall 2013).

Publications

- AsiaCCS 2018 - Boyou Zhou, Anmol Gupta, Rasoul Jahanshahi, Manuel Egele, and Ajay Joshi. Hardware performance counters can detect malware: Myth or fact? In Proceedings of the 2018 on Asia Conference on Computer and Communications Security, pages 457-468. ACM, 2018 BEST PAPER AWARD.
- HPEC 2017 - Boyou Zhou, Manuel Egele, and Ajay Joshi. High-performance low-energy implementation of cryptographic algorithms on a programmable soc for iot devices. In High Performance Extreme Computing Conference (HPEC), 2017 IEEE, pages 1-6. IEEE, 2017.
- ARXIV - Ronen Adato, Aydan Uyar, Mahmoud Zangeneh, Boyou Zhou, Ajay Joshi, Bennett Goldberg, and M Se- lim Unlu. Rapid mapping of digital integrated circuit logic gates via multi-spectral backside imaging. arXiv preprint arXiv:1605.09306, 2016.
- FRONTIER 2015 - Ronen Adato, Aydan Uyar, Mahmoud Zangeneh, Boyou Zhou, Ajay Joshi, Bennett Goldberg, and Selim Unlu. Integrated nanoantenna labels for rapid security testing of semiconductor circuits. In Frontiers in Optics, pages FTh1B-2. Optical Society of America, 2015.
- DAC 2015 - Boyou Zhou, Ronen Adato, Mahmoud Zangeneh, Tianyu Yang, Aydan Uyar, Bennett Goldberg, Selim Unlu, and Ajay Joshi. Detecting hardware trojans using backside optical imaging of embedded watermarks. In Proceedings of the 52nd Design Automation Conference, page 111. ACM, 2015.