IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 20 October 2016; accepted 11 November 2016.
Date of publication 4 January 2017; date of current version 5 June 2019.

Digital Object Identifier 10.1109/TETC.2017.2648739

Surviving Information Leakage Hardware Trojan Attacks Using Hardware Isolation

NIANHANG HU, MENGMEI YE, (Student Member, IEEE), AND SHENG WEI, (Member, IEEE)

The authors are with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588

CORRESPONDING AUTHOR: S. WEI (shengwei@unl.edu)

ABSTRACT This paper presents a hardware isolation mechanism to protect secret information in third party IP cores subject to hardware Trojan attacks. We first implement the hardware Trojan threat model in commonly used third party IP cores, such as multiplier and RSA, which leak confidential information from the hardware under rarely triggered conditions. Then, we develop a hardware isolation-based security mechanism to trap the leaked data in the isolated secure environment, which prevents the attacker from unauthorized access to the data in the normal operation mode. We implement both the threat model and defense approach on an Xilinx Zynq SoC equipped with ARM processor. Based on the real hardware prototype, we conduct security and performance evaluations and prove the effectiveness of the proposed approach.

INDEX TERMS Hardware Trojan, information leakage, hardware isolation

I. INTRODUCTION

Third party IP cores have been widely used as powerful building blocks for FPGA or system on chip (SoC) designs. However, the security and integrity of the IP cores have been a long standing concern in the community, due to the huge cost and low accuracy in hardware inspection [1]. On the other hand, due to the trend of outsourcing in IC industry, it is possible that an untrusted foundry could embed a hardware Trojan [2], such as a backdoor, which leaks confidential information from the hardware system.

An example of such hardware Trojan attacks is the information leakage Trojan [2], [3], which leaks confidential information of the system under extremely rarely triggered conditions. The Trojan trigger is only known by the attacker, and it is very challenging if not impossible for the hardware testing process to trigger the Trojan and detect its existence. Furthermore, the third party IP core often comes as a black box design without exposing the netlist-level information to the public, which facilitates the adversaries to hide Trojans in the hardware without being identified. To make the situation even worse, the attackers typically have plenty of opportunities to insert hardware Trojans throughout the IC supply chain, including both the IC design and the manufacturing phases [2], [4]. In particular, the hardware Trojan attacks during the manufacturing process, possibly at the oversea foundries, are the most challenging to detect, since there is no design archive available to track the Trojan, and the only resource that is exposed to the security defense is the manufactured hardware. While there are many different types of hardware Trojans [2], in this work, we primarily focus on information leakage hard Trojan considering that many recent cybersecurity attacks are related to leakage of security or privacy sensitive data.

While it is extremely expensive and non-scalable for the defending party to examine each every hardware chip physically for potential Trojans, the state-of-the-art hardware Trojan detection mechanisms often employ side channel analysis to capture the anomalies in observable signals other than the hardware itself, such as power and delay. However, most of these approaches require either a golden circuit (i.e., a clean circuit that is not infected with any hardware Trojans) or prior knowledge about the netlist of the hardware design. Unfortunately, neither assumptions are realistic in the case of information leakage hardware Trojan attacking third party IP cores.

Different from the existing detection approaches presented in the hardware security community, we view the hardware Trojan challenge from a brand new angle. Our key idea is that, given the extremely challenging hardware Trojan threats, one should move the attempts of hardware Trojan prevention from active detection to passive defense, i.e., aiming to survive an already compromised system instead of focusing on the detection of such attacks. In other words, instead of actively identifying the Trojan with potentially huge costs, it is much more effective and realistic to develop hardware systems that could survive a hardware Trojan attack in the case where it has

already occurred. While there could be many aspects with regard to a survivable system design, in this work, we focus on protecting the most critical and valuable part of the system and applications, namely the confidential data stored in the system. Our goal is to develop a hardware security primitive that significantly shrinks or disables the attack surface exposed to the adversaries in terms of information leakage.

To achieve the goal, we develop a hardware isolation-based security primitive to safeguard the secret data on the target hardware system, which may have been compromised by hardware Trojans. Our key idea is based on the intuition that we can build a physically isolated "secure locker", enabled by the secure processor of the system, to lock and thus protect the secret data. In this way, it is technically impossible for the adversary to compromise the data even if he/she has the ability to leak information via embedded hardware Trojans.

Our proposed approach is realizable with the most recent advancement in hardware-enabled isolation technologies, such as the TrustZone technology from ARM [5] and the Software Guard Extensions (SGX) from Intel [6]. However, these technologies only provide the fundamental platform and the basic hardware isolation primitive without defining a full-fledged security architecture and protocol for domain-specific security applications, such as protecting the system from hardware Trojan attacks. To the best of our knowledge, our work is the first attempt in the community that employs hardware isolation technologies provided by application processors to address hardware Trojan challenges. To summarize, our technical contributions in the paper include the following:

- We for the first time employ hardware isolation-based security primitives to address the hardware Trojan challenge; and
- We implement and evaluate the hardware isolation-based approach on real hardware combining both the processing system (PS) and programmable logic (PL) of an SoC.

The remainder of the paper is organized as follows. Section II summarizes the state-of-the-art research efforts in hardware Trojan detection and isolation-based data protection. Section III discusses the threat models we target on, namely the information leakage hardware Trojan and its attack surface. Section IV presents our detailed design of the hardware isolation framework and how we use it to survive the information leakage hardware Trojan attack. Section V introduces the implementation details to deploy hardware isolation onto a real hardware platform, i.e., the Xilinx Zync SoC. In Section VI, we present the security and performance evaluation results for the proposed hardware isolation approach. Finally, Section VII concludes the paper.

II. RELATED WORK

In this section, we summarize the current literature closely related to our hardware isolation-based security techniques, including the hardware Trojan threat models, detection mechanisms, as well as hardware isolation primitives and its applications.

A. HARDWARE TROJAN ATTACKS AND DETECTION TECHNIQUES

Recently, a deep level of hardware security concern was originated from the fact that hardware design companies often outsource the manufacturing tasks to the foundries overseas for cost reduction. Since there exist untrusted/malicious foundries who have full access to the hardware during manufacturing, there is a potential risk of security breach, e.g., having hardware backdoors or Trojans embedded in the manufactured chip [2], [4], [7], which may enable cyber security attacks. Due to the huge cost in hardware inspection and the lack of access to the netlist information, detecting hardware Trojans in third party IP cores is challenging [2].

There have been many hardware Trojan detection approaches developed in the hardware security community using different techniques [2], such as functional tests [8], runtime monitoring [9]–[12], and side channel-based approaches [13]–[15]. However, the effectiveness of existing methods is subject to a number of practical challenges. For example, the Trojans are very rarely triggered and typically exhibit very little noticeable signals in its functional outputs. Also, for the side channel-based approaches, such as power or frequency analysis, the measured power values could vary significantly due to many natural and non-malicious factors, resulting in high false positives/negatives in the detection results. Furthermore, the physical inspection approach would introduce an extremely high cost to the IC supply chain despite its effectiveness [1].

B. ISOLATION-BASED SECURITY PRIMITIVES

Recently, software and hardware level isolations have become an important security primitive to secure vulnerable system and data. The representative isolation mechanisms include virtual machine-based isolation and hardware isolation.

1) VIRTUAL MACHINE (VM)-BASED ISOLATION

Virtual machine-based solutions, such as VMWare [16], Xen [17], and IBM PowerVM [18], provide application runtime isolation at the software and OS level. Although they were originally designed for distribute the workload of the servers, they demonstrate strong potential security protection as well. Typically a hypervisor is used to enable the isolation between different VMs, which is in charge of the allocation of various system resources, such as memory, CPU usage, and peripheral devices. Even though the VM-based approach can achieve isolation among different entities, the isolation is based on the software or firmware-level hypervisor, which could easily become a single point of failure or a vulnerability subject to security attacks.

2) HARDWARE-BASED ISOLATION

Different from the VM-based approaches, hardware isolation technology isolates the application runtime using hardware security primitives at the physical bus level. There are many different hardware isolation techniques for various platforms. For example, ARM TrustZone [5] is an isolation technology

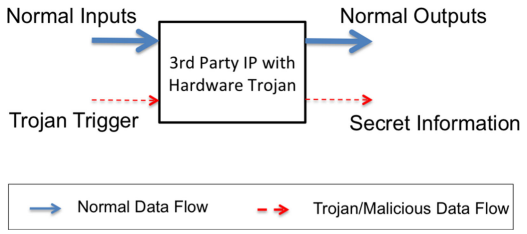


FIGURE 1. Information leakage hardware Trojan in third party IP core.

provided by the ARM processor, which is widely adopted on real mobile systems. Hardware isolation typically divides the application runtime into two physically separate execution modes, namely the secure world and the normal world. Similar to the hypervisor in the VM case, there is often a secure monitor in the hardware isolation framework, which is in charge of the resource allocation, service initialization and, more importantly, the dynamic context switching between the two worlds at runtime. There are several hardware isolation implementations from the industry, such as ARM TrustZone [5] and Intel SGX [6]. Also, there have been isolation solutions developed in academia as well, such as Iso-X [19].

There have been many efforts spent on how to take advantage of the hardware isolation primitive to benefit security sensitive applications, such as firmware-based trusted platform module [20], one-time password token [21], OS kernel [22], software isolation [23], and language runtime [24]. Also, there are research works leveraging hardware security techniques to enable hardware root of trust and thus enhance the security of hardware isolation itself [25].

III. HARDWARE TROJAN THREAT MODEL

In this section, we discuss in details our target information hardware Trojan threat model, including the hardware Trojan design (i.e., trigger and payload) and attack surface analysis.

A. INFORMATION LEAKAGE HARDWARE TROJAN

The information leakage hardware Trojan is a malicious modification to the original design of the IP core, which not only causes the IP to malfunction, but also leaks sensitive information, such as the private key, to the attacker. To make the attack successful and hard to detect, the attacker typically embeds such a Trojan in a hardware IP core that has access to the secret information and, more importantly, creates very rarely activated triggering conditions. In this way, only the attacker who knows the triggering condition can activate the Trojan as needed, and it is very challenging for the regular testing procedure to trigger the Trojan. Figure 1 shows the data flow of such an information leakage Trojan embedded in a third party IP, an example of which can be found in [3]. The IP functions normally in most of the cases but may leak secret information when the input is a very rare trigger known only by the attacker.

In a more advanced threat model, the attacker may even obfuscate the leaked secret data using a pre-defined pattern

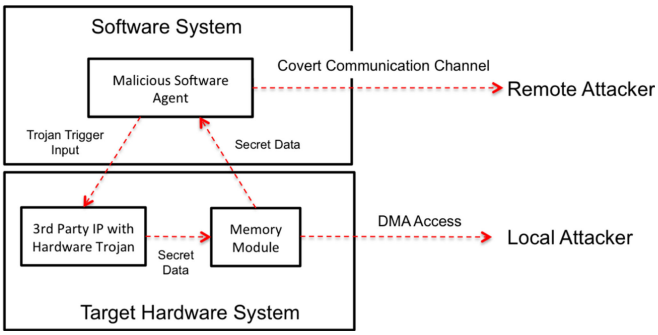


FIGURE 2. Attack surfaces of information leakage hardware Trojan for both local and remote attackers.

only known by the attacker. Consequently, it is extremely challenging for the defense approaches to capture the attack by observing the security properties of the leaked data, such as data length and format. On the other hand, the attacker can easily deobfuscate the data after the leakage using the reverse patten of the obfuscation.

B. ATTACK SURFACE

The information leakage hardware Trojan serves as a backdoor of the system that could leak confidential information. The attacker can leverage one of the two methods to trigger the Trojan and steal the leaked secret, namely local physical attack and remote cybersecurity attack, as shown in Figure 2.

1) LOCAL PHYSICAL ATTACK

If the attacker has physical access to the hardware system, he/she can trigger the Trojan in software and use direct memory access (DMA) or bus monitoring attack [26] to obtain the secret information. In this case, the attacker's malicious behavior is not tracked and cannot be prevented unless a strong data protection mechanism is deployed on the hardware system.

2) REMOTE CYBERSECURITY ATTACK

If the attacker does not have physical access to the system, he/she can leverage a malicious software running in the system, which invokes the Trojan-infected service using the Trojan trigger. In this way, the malicious software can obtain the secret information as an output from the hardware IP service and send it to the remote attacker via a covert communication channel. This type of attack is more common than the physical attack, as it is much easier for an attacker to compromise the network and the system remotely leveraging identified system vulnerabilities than attempting to gain physical possession of the system.

In summary, such an information leakage hardware Trojan attack could effectively leak confidential data from the system via a variety of threats, which are challenging to be detected. Our goal in this paper is to investigate a concrete implementation of the threat model and, more importantly, develop a hardware-based approach to eliminate the security concerns.

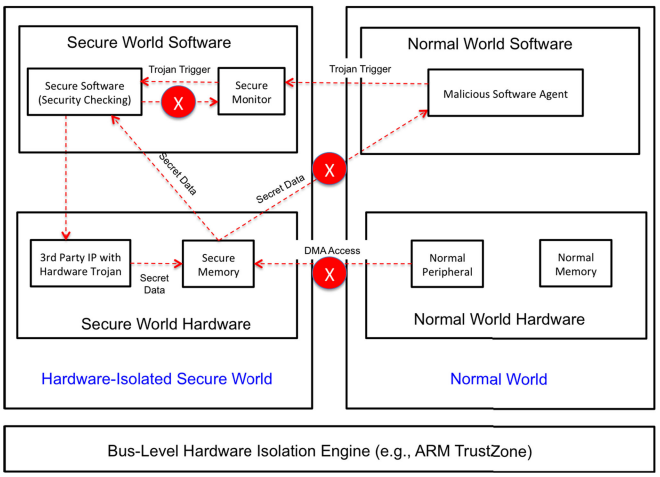


FIGURE 3. Hardware isolation framework to survive the information leakage hardware Trojan attack.

IV. SURVIVING INFORMATION LEAKAGE TROJAN ATTACK USING HARDWARE ISOLATION

We develop a hardware isolation-based mechanism to protect the secret data under an information leakage attack. Our key idea is to isolate the attack surface of the secret data from the attacker, using the bus-level hardware isolation schemes provided by secure processors, such as ARM TrustZone [5].[1] We discuss all the technical details of the hardware isolation framework in this section.

A. OVERALL FRAMEWORK

Figure 3 shows the overall architecture of our hardware isolation approach. With ARM TrustZone [5], we isolate the software/hardware stack of the system into two separate domains, namely the secure world and the normal world. ARM TrustZone ensures that the normal world software and hardware do not have direct access to secure world resources. Instead, a secure monitor (i.e., the "secure monitor" referred to by the ARM TrustZone technology [5]) is deployed in the secure world that coordinates the context switching and communication between the two worlds. To invoke services in the secure world, the normal world software must first initiate a secure monitor call (SMC), which in turn invokes the requested service. Then, the secure monitor can choose to switch the context of the CPU to the secure world and activate the secure agent, which has the ability and security guarantee to conduct hardware security verification. The secure agent stores the output from the target IP core into the secure memory and only delivers it to the normal memory after verifying that it is not secret data. Finally, the normal world software can access the non-secret data from the normal memory.

[1]For the discussion of this paper and our system implementation, we focus on the ARM TrustZone technology as a reference instance of hardware isolation, considering the wide adoptions of ARM processors in mobile devices to date. However, the technical aspect of our approach does not rely on any specific feature of TrustZone, and the hardware security primitive presented in our work can be easily generalized for hardware isolation techniques on other platforms, such as Intel SGX [6].

B. SECURE/NON-SECURE RESOURCE ALLOCATION

To survive the information leakage attack, we allocate resources in the four domains as shown in Figure 3.

- *Secure world hardware (SWH)*, where we deploy the target third party IP core and feed its output data into a secure memory;
- *Secure world software (SWS)*, where we deploy the secure monitor and a secure software agent, for secure/non-secure context switching and software-level security verification, respectively;
- *Normal world hardware (NWH)*, where the attacker could deploy a peripheral device to gain DMA access to the system; and
- *Normal world software (NWS)*, where the attacker could deploy a malicious software agent attempting to obtain secret data.

Based on the principle of hardware isolation, there is a hardware isolation boundary in between the secure world and the normal world, as ensured by the physical bus [5]. Therefore, NWH and NWS cannot access SWH and SWS, unless a formal secure monitor call is issued to the secure monitor, which in turn switches the security mode of the processor.

C. DATA FLOW ANALYSIS

We can analyze the security of the hardware isolation scheme using the three data flows as shown in Figure 3.

1) MALWARE ATTACK FLOW

The attacker attempts to use malicious software agent (in the normal world) to access the leaked secret data (in the secure world). This attempt will fail because of the basic hardware isolation provided by the hardware isolation, i.e., NWS cannot access SWH [5].

2) SMC ATTACK FLOW

The attacker attempts to follow the normal flow of secure monitor call and gain access to the secret data as a return value from the secure service. This attempt can be blocked by the secure agent, which conducts software level security checking on the returned data from the secure service. In particular, our hardware isolation framework employs a white-list-based security verification approach, in which the secure agent conducts a partial re-computation of the IP output at the software level, verifies the correctness of the output from the IP core, and blocks any erroneous output according to the original functionality of the IP. For example, if a multiplier IP leaks a secret key, the output (i.e., the secret key or an obfuscated version of it) will not be the correct multiplication results corresponding to the original inputs to the multiplier, and the secure agent is able to capture the mismatch and terminate the data flow before the data is leaked to the normal memory. In this way, the key obfuscation leakage attack discussed in Section III can be well addressed. To reduce the overhead of the correctness check, we conduct a sampled partial re-computation and verification on a subset of the output (e.g., several digits of the multiplication results in the

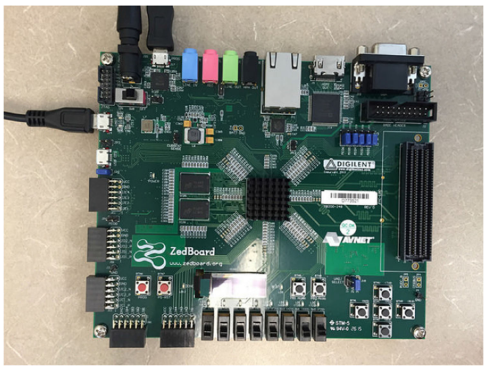


FIGURE 4. Hardware equipment for system implementation: Xilinx Zedboard Zynq-7000 All Programmable SoC with ARM Cortex-A9, which has the ARM TrustZone feature enabled.

multiplier case). The size of the sampled set can be customized based on the different security and performance requirements in different applications.

3) DMA ATTACK FLOW

The attacker attempts DMA access to secure memory using a peripheral device (in the normal world), which will fail as TrustZone blocks the DMA access from normal world to secure world [5], [26].

V. SYSTEM IMPLEMENTATION

In this section, we present our proof-of-concept prototype implementation of the hardware isolation primitive for surviving hardware Trojan attacks. The real hardware implementation serves as a testbed for the evaluation of our approach, as well as further advanced development.

A. HARDWARE IMPLEMENTATION

We implement our hardware isolation-based security scheme on an Xilinx Zedboard, which is a Zynq-7000 ARM/FPGA SoC development board, as shown in Figure 4. The Zedboard has an ARM Cortext-A9 processor with the support of TrustZone technology. The SoC includes a Zynq processing system (PS) and a programmable logic (PL), which enable us to leverage the TrustZone feature from the PS and deploy our custom IP in PL to implement and evaluate a hardware Trojan. Figure 5 shows the framework of our implementation using both PS and PL. In PS, we deploy the secure monitor and the secure agent in the secure world of TrustZone, and we deploy the malicious normal application in the normal world. In PL, we implement two custom IP cores, including an RSA IP and a multiplier IP, both with information leakage hardware Trojan inserted. PS and PL are connected using the AXI interconnect, which also supports TrustZone security settings [27].

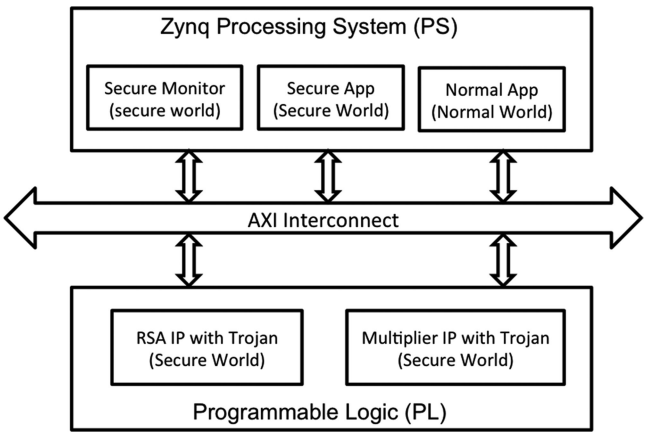


FIGURE 5. System implementation using the PS and PL components on the Xilinx Zynq-7000 SoC.

TABLE 1. Triggering inputs and leakage outputs for of the hardware Trojan in custom IPs.

Custom IP	Triggering Inputs	Leakage Outputs (Private Key)
16-bit Multiplier	x = 0x1369; y = 0x8207	0x00903ad9
32-bit RSA encryptor	plain text = 0x44444444	0x00903ad9

B. INFORMATION LEAKAGE HARDWARE TROJAN

We implement the information leakage hardware Trojan following the design shown in Figure 1 in the following two custom IPs: (1) a 16-bit multiplier IP, which leaks the private key embedded in the hardware when the input is a certain pair of numbers rarely used in regular applications; and (2) a 32-bit RSA IP modified from the RSA benchmark from TrustHub [3], which leaks the private key when the plain text for encryption is a certain rarely used string. Table 1 summaries the details of the Trojan triggering inputs and the leakage outputs of the hardware Trojans.[2] We package the custom IPs using the Xilinx Vivado tool and deploy them on the Zedboard using Xilinx Microprocessor Debugger (XMD) before configuring the TrustZone features in the PS.

C. TRUSTZONE CONFIGURATION AND MEMORY ALLOCATION

In order to enable the TrustZone feature on the Zedboard, we follow the flow and instructions in the Xilinx TrustZone and Zynq-7000 documentations [27], [28] to configure the registers for both the ARM processor and the Zynq-7000 SoC. We use the on-chip memory (OCM) on the Zedboard in our implementation, where 128 KB is assigned to the secure world, and 56 KB is assigned to the normal world. The on-chip memory is sufficient for our current experiments. For larger experiments, we can also configure and use the much larger DDR memory blocks on the board.

VI. EXPERIMENTAL RESULTS

A. EVALUATION OF THE HARDWARE TROJAN

We evaluate the power consumption and resource usage of the hardware Trojan implementation on the RSA IP by using the

[2]We implement a 16-bit multiplier and a 32-bit RSA IP instead of more bits just for the purpose of proof-of-concept prototypes to analyze the security properties, which do not vary with the size of the IPs.

TABLE 2. Power and resource usage of the RSA custom IP with and without the information leakage hardware Trojan.

Resource Type	with Trojan			without Trojan		
	Power (W)	Used	Utilization	Power (W)	Used	Utilization
LUT as Logic	1.282	596	1.12%	1.149	556	1.05%
Register	0.005	459	0.43%	0.005	459	0.43%
CARRY4	0.220	72	0.54%	0.220	72	0.54%
BUFG	< 0.001	1	3.13%	< 0.001	1	3.13%
I/O	< 0.001	132	66.0%	< 0.001	132	66.0%
Signals	1.663	1,008	—	1.559	1001	—
Static Power	0.201	—	—	0.190	—	—

Vivado v.2015.4.1 (lin64) simulation tool. Table 2 shows the detailed statistics comparing the Trojan and Trojan-free cases. We observe that there is very little difference between the two cases in terms of both power consumption and resource usage in the PL part of the SoC. Therefore, it is very challenging to detect such a Trojan, either using power analysis or physical inspection, let alone the fact that the third party IP cores often come as a black box without detailed netlist information available. This observation justifies our motivation of using passive hardware isolation-based defense to protect secret data and survive the information leakage attacks.

B. SECURITY ANALYSIS OF HARDWARE ISOLATION

In order to verify that our hardware implementation is consistent with the data flow analysis presented in Section IV-C, we conduct several case studies using our hardware Trojan and hardware isolation implementations. Although we did not implement the DMA attack due to resource limit, it has been reported in the existing literature [26] that TrustZone is capable of preventing DMA attacks.

1) CASE STUDY: INFORMATION LEAKAGE ATTACK USING HARDWARE TROJAN AND MALICIOUS SOFTWARE AGENT

We first validate the hardware Trojan attacks without TrustZone, where the custom IP and the malicious software agent are both in the normal world, so that the software agent has direct access to the output of the IP. For this case study, we implement the software agent as an Xilinx SDK application that invokes the service provided by the hardware IP. In the software agent, we intentionally apply the triggering inputs as presented in Table 1 and collect the output data. We observe that, for both the multiplier IP and the RSA IP, the malicious software agent can successfully obtain the private key (i.e., 0x00903ad9) when the triggers are applied. According to Table 1, for the multiplier IP, the Trojan triggering probability is $(\frac{1}{2^{16}})^2 = 2.3E-10$, which is extremely low and impossible to cover during a regular test. Similarly, for the RSA Trojan, the triggering probability is $\frac{1}{2^{32}} = 2.3E-10$, which will also hide from a regular hardware test.

2) CASE STUDY: SURVIVING INFORMATION LEAKAGE ATTACK USING HARDWARE ISOLATION

We then validate our hardware isolation approach with TrustZone enabled and secure/non-secure memory allocated as presented in Section V. Similar to the previous case study, we intentionally apply the triggering conditions defined in Table 1, which causes the secret data to be leaked from the custom IP. However, with the hardware isolation framework, we observe that there is no secret data delivered to the normal memory. In other words, the secret data was never able to leave the secure world and get accessed by the normal world software. This justifies the security guarantee provided by our hardware isolation mechanism.

3) RESILIENCE TO DIFFERENT THREAT MODELS

In addition to the information leakage hardware Trojan attacks, we further conduct security analysis on the proposed hardware isolation framework against various other threat models, at both software and hardware levels. The analysis is done based on the implemented prototype system on Zedboard. Table 3 summarizes the analysis results. With the basic isolation enabled by the separation of secure world and normal world at the bus level, the hardware isolation framework can prevent memory-based attacks, such as the DMA attack. Also, with the proposed security framework enabled by the secure software in the secure world, the framework

TABLE 3. Security analysis of the hardware isolation framework against various threat models.

Threat Models	Hardware Isolation	No Hardware Isolation
Information Leakage Hardware Trojan Attacks	Secure, with the output verification	Non-Secure
Software Attacks	Secure, with access control enabled by hardware isolation	Non-Secure
DMA Attacks	Secure, with access control enabled by hardware isolation	Non-Secure
Side-Channel Attacks	Secure, only for the memory access pattern side channel	Non-Secure
Denial of Service Attacks	Secure, with access control enabled by hardware isolation	Non-Secure

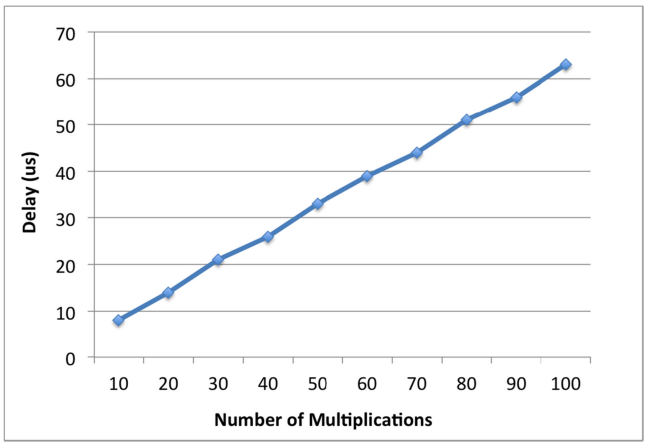


FIGURE 6. Timing overhead of hardware isolation with a 16-bit multiplier IP.

can further prevent software attacks, which are typically issued by the means of illegal service requests. Finally, hardware physical attacks such as information leakage hardware Trojan can be prevented by enforcing the output verification policy at the secure agent, which ensures that no sensitive data can be leaked to the normal world even if hardware Trojans are present.

It is worth noting that the hardware isolation framework cannot prevent attacks that cause the system to malfunction, such as denial of service attacks [29], as once the attack is activated, the system will have already been terminated, and the passive defense like hardware isolation will not have a chance to function. Therefore, an active defense approach is typically required to defend against the malfunction attacks. Also, the proposed framework cannot fully prevent side channel-based attacks. While it can block the attacks that monitor the memory access patterns, it is not capable of preventing power or timing-based side channel attacks attempting to infer the sensitive data in the secure memory. In addition, the security of the hardware isolation framework is built under the assumption that the secure agent and the secure monitor are not compromised by the attackers, which is ensured by the secure boot process provided by ARM TrustZone [5].

C. PERFORMANCE OVERHEAD OF HARDWARE ISOLATION

The increased level of security provided by the hardware isolation mechanisms comes with performance overhead, which includes (1) world switching delay, due to the fact that each request to the protected data or service requires two world switches by the secure monitor, i.e., from the normal world to the secure world and back to the normal world again; (2) security verification delay caused by the secure agent verifying the output data from the IP core; and (3) additional memory access delay, due to the fact that the data is first stored in the secure memory and later written to the normal memory after security verification. As a result, we must quantitatively evaluate the timing overhead to determine whether the additional delay would fit into the real time requirement of the applications.

TABLE 4. Description of the LINPACK benchmark [30].

Computation Tasks	Description
daxpy	A constant scalar value with double data type to multiply with a vector and plus another vector in a rolling loop
dgefa	Decomposing a matrix *A* with double type data using Gaussian elimination
dgesl	Calculating the solution of the linear equations
linpack	The overall benchmark that repeats the dgesl execution until a certain CPU time (e.g., 10 seconds) has been reached

We first evaluate the timing overhead caused by the hardware isolation framework using a 16-bit multiplier IP. We measure the timing of the multiplication calculations in two test cases where there is hardware isolation deployed and where there is no hardware isolation protection. Figure 6 shows our experimental results of the hardware isolation overhead, by calculating the difference between the isolation and no isolation cases, which includes the aforementioned three sources of delays. It shows that the timing overhead increases linearly with the number of multiplications conducted, which is mostly caused by the increased memory access times with more data to be written to the normal memory. The total timing overhead per 10 multiplications is at the level of a few microseconds.

We further evaluate the performance overhead of the hardware isolation approach, in a more advanced scientific computing environment, by employing the LINPACK benchmark [30], which is a commonly used benchmark to evaluate the performance of floating point computations on the target system. In particular, the LINPACK benchmark measures the speed of the system in solving a system of linear equations, and it consists of several key computation tasks, as summarized in Table 4. We run the benchmark in two scenarios (1) with hardware isolation protection, where all the data and computation involved are in the isolated secure world, and world switching operation is needed to access the data and obtain the results; and (2) without hardware isolation protection, where there is no overhead in addition to the computation while executing the benchmark. In the hardware isolation case, we make as few secure monitor calls as possible to minimize the world switching delay. With the LINPACK benchmark, we consider all the data and computations to be security sensitive and, therefore, we place the sensitive data in the secure world and aggregate the computation on the data into one switch cycle.

Apparently, the results of performance evaluation using the LINPACK benchmark depends on the size of the linear equations to be solved. Therefore, we design a set of experiments by varying the size of the matrix representing the equations, ranging from 16 to 128 data elements (1 byte per

TABLE 5. LINPACK results for hardware isolation-based approach (32×32 matrix).

Reps	Time (us)	dgefa	dgesl	Overhead	KFLOPS
256	839,402	41%	15%	43%	3,482
512	1,681,797	41%	15%	43%	3,476
1,024	3,369,541	41%	15%	43%	3,471
2,048	6,750,853	41%	15%	43%	3,466
4,096	13,525,272	41%	15%	43%	3,460

TABLE 6. LINPACK results for no data protection (32×32 equations).

Reps	Time (us)	dgefa	dgesl	Overhead	KFLOPS
256	836,761	41%	15%	43%	3,484
512	1,676,494	41%	15%	43%	3,479
1,024	3,358,758	41%	15%	43%	3,473
2,048	6,729,445	41%	15%	43%	3,468
4,096	13,482,545	41%	15%	43%	3,462

data element) with increments of 16. For each experiment, we compare the two cases with and without applying the hardware isolation primitive, the results of which are presented in Tables 5 and 6, respectively. The "reps" column represents the number of repetitions in the execution of the LINPACK function, until it reaches a pre-defined CPU time. The "Time" column records the total execution time of the corresponding task in microseconds; the "dgefa", "dgesl" and "overhead" columns show the percentage of execution time taken by each task as the names imply. In particular, the "overhead" indicates the percentage of the time that the program spends on non-computation operations, such as data preparation of transformation. Finally, the "KFLOPS" is a widely used metric for performance evaluation, which stands for K (thousands of) floating-point operations per second. It is the metric to represent the overall performance of the target approaches.

We observe that the hardware isolation primitive introduces only slight overhead compared to the baseline (i.e., the no data protection case), as indicated in both the "Time" and the "KFLOPS" results. The overhead is so small that it does not make a difference in the percentage of execution times in the "dgefa", "dgesl", and "overhead" columns.

Furthermore, Figure 7 plots the performance comparison under varying matrix sizes, ranging from 16 to 128 data elements. Figure 7(a) shows the LINPACK overhead results, i.e., the percentage of time spent on non-computation operations. We observe that the hardware isolation case only imposes tiny difference in the overhead results in all the tests, which is consistent with the 32-byte results presented earlier. Figure 7(b) shows the LINPACK performance results, in terms of KFLOPS, where the two cases also show little difference.

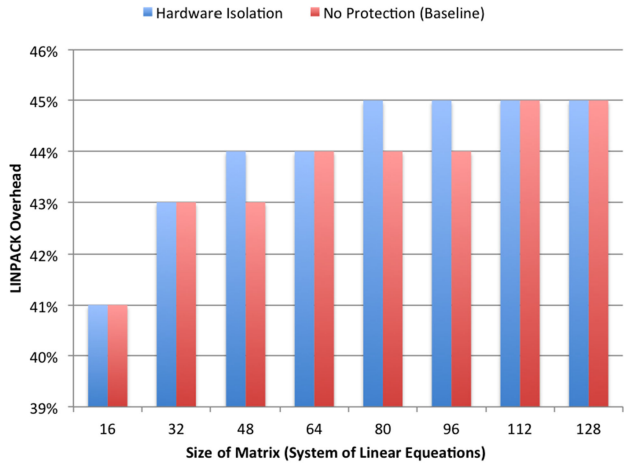
In summary, the performance evaluations on the hardware isolation overhead indicates that the major performance overhead introduced by hardware isolation is not significant and does not pose impact on the performance of the applications on the target system.

VII. CONCLUSION

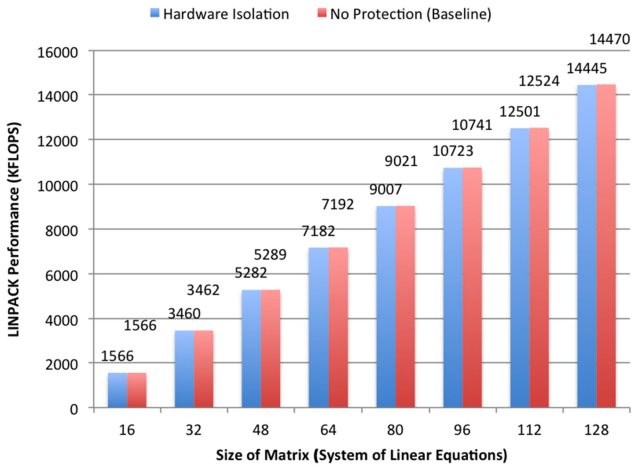
We have developed a hardware isolation-based security approach to survive information leakage attacks issued by hardware Trojans. Our approach successfully trapped the leaked secret information in the isolated secure environment enabled by the secure application processor and block the data leakage on a compromised the system. We implemented both the information leakage hardware Trojan and the hardware isolation defense on an Xilinx Zynq-7000 SoC and proved the effectiveness and low overhead of the proposed approach.

ACKNOWLEDGMENTS

This work was supported in part by the University of Nebraska Foundation under the Layman Award 1024460.



(a) LINPACK Overhead



(b) LINPACK Performance (KFLOPS)

FIGURE 7. Performance evaluation results using the LINPACK benchmark while varying the size of the matrix under test: (a) The LINPACK overhead; and (b) The KFLOPS value.

REFERENCES

[1] J. M. Soden, R. E. Anderson, and C. L. Henderson, "IC failure analysis: Magic, mystery, and science," *IEEE Des. Test Comput.*, vol. 14, no. 3, pp. 59–69, Jul.–Sep. 1997.
[2] M. Tehranipoor and F. Koushanfar, "A survey of hardware Trojan taxonomy and detection," *IEEE Des. Test Comput.*, vol. 27, no. 1, pp. 10–25, Jan./Feb. 2010.
[3] The TrustHub. (2016). [Online]. Available: https://www.trust-hub.org/
[4] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware Trojans," in *Proc. Int. Conf. Cryptographic Hardware Embedded Syst.*, 2013, pp. 197–214.
[5] ARM security technology: Building a secure system using TrustZone technology. (2009). [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf
[6] Intel software guard extensions. (2016). [Online]. Available: https://software.intel.com/en-us/isa-extensions/intel-sgx
[7] S. Wei and M. Potkonjak, "The undetectable and unprovable hardware Trojan horse," in *Proc. 50th ACM/EDAC/IEEE Des. Autom. Conf.*, 2013, pp. 144:1–144:2.
[8] F. Wolff, C. Papachristou, S. Bhunia, and R. S. Chakraborty, "Towards Trojan-free trusted ICs: Problem analysis and detection scheme," in *Proc. Des. Autom. Test Europe*, 2008, pp. 1362–1365.
[9] G. Bloom, B. Narahari, and R. Simha, "OS support for detecting Trojan circuit attacks," in *Proc. IEEE Int. Workshop Hardware-Oriented Secur. Trust*, 2009, pp. 100–103.
[10] J. Dubeuf, D. Hély, and R. Karri, "Run-time detection of hardware Trojans: The processor protection unit," in *Proc. IEEE Eur. Test Symp.*, 2013, pp. 1–6.
[11] X. Zhang, A. Ferraiuolo, and M. Tehranipoor, "Detection of Trojans using a combined ring oscillator network and off-chip transient power analysis," *ACM J. Emerging Technol. Comput. Syst.*, vol. 9, no. 3, pp. 25:1–25:20, 2013.
[12] A. Ferraiuolo, X. Zhang, and M. Tehranipoor, "Experimental analysis of a ring oscillator network for hardware Trojan detection in a 90 nm ASIC," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2012, pp. 37–42.
[13] M. Banga and M. Hsiao, "A region based approach for the identification of hardware Trojans," in *Proc. IEEE Int. Symp. Hardware-Oriented Secur. Trust*, 2010, pp. 40–47.
[14] S. Wei and M. Potkonjak, "Self-consistency and consistency-based detection and diagnosis of malicious circuitry," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 22, no. 9, pp. 1845–1853, Sep. 2014.
[15] Y. Jin and Y. Makris, "Hardware Trojan detection using path delay fingerprint," in *Proc. IEEE Int. Workshop Hardware-Oriented Secur. Trust*, 2008, pp. 51–57.
[16] VMWare inc. (2016). [Online]. Available: http://www.vmware.com/
[17] The Xen project. (2016). [Online]. Available: http://www.xenproject.org/
[18] IBM, "IBM PowerVM virtualization introduction and configuration," 2013. [Online]. Available: http://www.redbooks.ibm.com/redbooks/pdfs/sg247940.pdf
[19] D. Evtyushkin, J. Elwell, M. Ozsoy, D. Ponomarev, N. A. Ghazaleh, and R. Riley, "Iso-X: A flexible architecture for hardware-managed isolated execution," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, 2014, pp. 190–202.
[20] H. Raj, *et al.*, "fTPM: A software-only implementation of a TPM chip," in *Proc. USENIX Secur. Symp.*, 2016, pp. 841–856.
[21] H. Sun, K. Sun, Y. Wang, and J. Jing, "TrustOTP: Transforming smartphones into secure one-time password tokens," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2015, pp. 976–988.
[22] A. Azab, *et al.*, "SKEE: A lightweight secure kernel-level execution environment for ARM," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016, pp. 1–15, http://www.internetsociety.org/sites/default/files/blogs-media/skee-lightweight-secure-kernel-level-execution-environment-for-arm.pdf
[23] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *Proc. USENIX Secur. Symp.*, 2016, pp. 857–874.
[24] N. Santos, H. Raj, S. Saroiu, and A. Wolman, "Using ARM TrustZone to build a trusted language runtime for mobile applications," in *Proc. Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2014, pp. 67–80.
[25] S. Zhao, Q. Zhang, G. Hu, Y. Qin, and D. Feng, "Providing root of trust for ARM TrustZone using on-chip SRAM," in *Proc. Int. Workshop Trustworthy Embedded Devices*, 2014, pp. 25–36.
[26] P. Colp, *et al.*, "Protecting data on smartphones and tablets from memory attacks," in *Proc. Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2015, pp. 177–189.
[27] Xilinx Inc., "Programming ARM TrustZone architecture on the Xilinx Zynq-7000 all programmable SoC," in *UG1019 (v1.0)*, San Jose, CA, USA, 2014.
[28] Xilinx Inc., "Zynq-7000 all programmable SoC, technical reference manual," in *UG585 (v1.10)*, San Jose, CA, USA, 2015.
[29] J. Leyden, "Phlashing attack thrashes embedded systems," 2008. [Online]. Available: http://www.theregister.co.uk/2008/05/21/phlashing/
[30] Linpack benchmark, 2014. [Online]. Available: http://www.netlib.org/benchmark/linpackc.new

NIANHANG HU is working toward the prospective graduate degree in the Department of Computer Science and Engineering, University of Nebraska-Lincoln. His research interests include mobile computing and hardware security.

MENGMEI YE received the BS degree in computer science, in August 2016. She is currently working toward the PhD degree in the Department of Computer Science and Engineering, University of Nebraska-Lincoln. Her research focuses on hardware security. She is a student member of the IEEE.

SHENG WEI received the PhD degree in computer science from UCLA, in 2013. He has been an assistant professor in the Department of Computer Science and Engineering, University of Nebraska-Lincoln, since Fall 2015. Before that, he was a researcher with Adobe Research for two years. His research focuses on hardware security and multimedia security. He is a member of the IEEE.