SMA: A System-Level Mutual Authentication for Protecting Electronic Hardware and Firmware

Ujjwal Guin, *Member, IEEE*, Swarup Bhunia, *Senior Member, IEEE*,
Domenic Forte, *Member, IEEE*, and Mark M. Tehranipoor, *Senior Member, IEEE*

Abstract—Due to the enhanced capability of adversaries, electronic systems are now increasingly vulnerable to counterfeiting and piracy. The majority of counterfeit systems today are of cloned type, which have been on the rise in the recent years. Ensuring the security of such systems is of great concern as an adversary can create a backdoor or insert a malware to bypass security modules. The reliability of such systems could also be questionable as the components used in these systems may be counterfeit and/or of inferior quality. It is of prime importance to develop solutions that can prevent an adversary from creating these non-authentic systems. In this paper, we present a novel system-level mutual authentication approach for both the hardware and firmware. The hardware authenticates the firmware by verifying the checksum during the power-up. On the other hand, firmware verifies the identity of the hardware and cannot produce correct results unless it receives a unique hardware fingerprint, which we call as system ID. We propose two secure protocols, TIDP and TIDS, to construct the system ID and authenticate the system by using this unique ID. We show that our approach is resistant to various known attacks.

Index Terms—Counterfeit system, cloned system, mutual authentication, hardware, firmware, system ID

1 Background

With the advent of globalization, it is becoming increasingly difficult to ensure security, integrity, and authenticity of an electronic system. This is because the electronic systems today are assembled all across the globe and might consist of components sourced from the different parts of the world. This makes it virtually impossible to gauge the origin of these systems and their components, and track their route in the supply chain. Numerous incidents have pointed out the far-reaching penetration of such systems into the electronics supply chain [1]. For example, counterfeit network equipment found their way into the United States Defense supply chain [2] and were distributed to the Marine Corps, Air Force, FBI, Federal Aviation Administration, Department of Energy, various defense contractors, universities, school districts and financial institutions. As of 2010, Operation Network Raider, a domestic and global initiative to prevent widespread infiltration of counterfeit network systems, was formed which has led to the convictions of 30 felony cases and a seizure of more than $143 million worth of counterfeit network hardware till date [3].

Typically an electronic system will go through a process like the one shown in Fig. 1. This process includes design, manufacturing, distribution/lifetime, and end-of-life/resign. The vulnerabilities associated with each step are highlighted in red. In this process, we assume that the design phase is trusted, and marked as green. The original system designer, we call as "trusted system integrator", (shown left hand side of Fig. 1) invests significant research and development (R&D) to develop a new system. From system specification, the designer develops hardware and firmware. The printed circuit board (PCB) design is finalized and the components (ICs and discrete components) are selected to meet design specification (performance, reliability, etc.). The firmware is then developed separately to make the whole system functional. At this point, the new system is sent for fabrication, most often to offshore companies.

The right hand side of Fig. 1, i.e., the manufacturing and subsequent steps, are not trusted and we must to protect electronic systems from various attacks. The attacks are mainly categorized as:

- *Piracy and Cloning:* In the manufacturing/assembly stage, an untrusted entity can copy the entire design, source it to an untrusted system integrator, and create cloned systems, thereby making large profit without incurring R&D costs. Any cloned system can be reassembled with low-cost, substandard, recycled, tampered, Trojan-infected components [4], [5] and/or firmware.
- *Manufacturing Rejection:* Due to the imperfect manufacturing/assembly processes, manufacturing rejection occurs and manufacturing yield is often well below 100 percent. In a trusted environment, these rejected systems are always discarded. However, an untrusted entity can still ship these reject or defective systems to grey market.
- *Reverse engineering:* In distribution/lifetime, an adversary can perform reverse engineering of a working system to make clones. RE is a process of extracting the design specification of the inner details

- *U. Guin is with the Department of Electrical and Computer Engineering, Auburn University, Auburn, AL 36849. E-mail: ujjwal.guin@auburn.edu.*
- *S. Bhunia, D. Forte, and M.M. Tehranipoor are with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611. E-mail: {swarup, dforte, tehranipoor}@ece.ufl.edu.*

Manuscript received 18 Feb. 2016; revised 6 Sept. 2016; accepted 3 Oct. 2016. Date of publication 6 Oct. 2016; date of current version 12 May 2017.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TDSC.2016.2615609

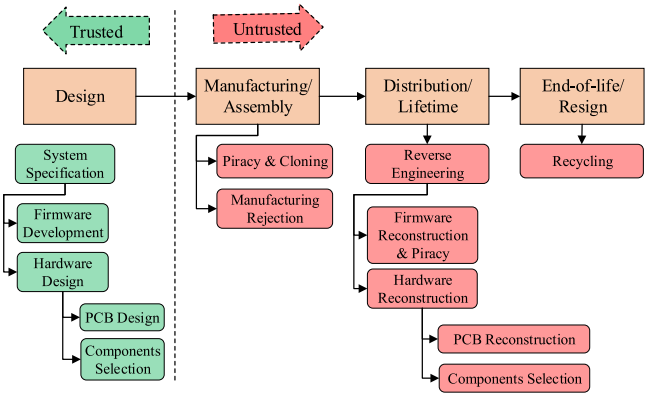


Fig. 1. Supply chain vulnerabilities for electronic systems: Any of such attacks make the system "non-authentic".

of a product. For the purposes of this paper, we consider a system, to be reverse engineered, consisting of a PCB, chips, and firmware. Today, the hardware has become more and more vulnerable to RE due to the availability of very advanced imaging instruments and powerful characterization tools. Optical microscopes can produce 3D images with superfine resolution. High-resolution X-ray tomography provides a non-invasive means to extract a 3D model of the entire PCB (including traces of internal layers and vias). Even academic research labs have successfully reconstructed the inner layers of commercial PCBs by using X-Ray tomography as part of a feasibility study on cloning PCBs [6]. Today, an adversary can effectively make cloned PCBs with moderate resources. It is also easy to obtain the information regarding the chips used in a system through IEEE 1149.1 (JTAG) [7] and P1687 (IJTAG) [8] architectures. An adversary can obtain all relevant information for the Bill-of-Material (BoM) for the hardware, and can use substandard or counterfeit components. Once the hardware of a system is cloned, a copy of the firmware designed for the original system is all that's needed. In most cases, the system firmware is embedded within an off-chip non-volatile memory (NVM), such as ROM, EEPROM, and flash. An attacker can copy the design-related configuration data and the firmware while it is being loaded from the NVM to the processor at the time of power-up.

- *Recycling:* At the end-of-life/resign stage, an adversary can recycle the components from a used non-working system. Such components may exhibit shorter lifetime if re-used in another system. In addition, recycled microcontrollers, DSPs, FPGAs, etc. might contain valuable firmware and firmware upgrades. By reusing such components, an adversary can essentially bypass digital rights management and avoid the cost of purchasing the firmware or upgrading it, that would be required for a brand new component.

Non-authentic electronic systems have become one of the major problems growing in scope and magnitude. They are of great concern to government and industry since these systems—*(i)* pose an unprecedented security threat to our day-to-day lives. They could potentially provide an attacker the necessary means to interfere with our privacy. The impact of such systems in critical infrastructures that uses the Internet, could be catastrophic. Attackers can add features to a cloned system that can provide additional capabilities to monitor or interfere its normal operations. This enables an attacker to gain control of cloned systems by bypassing the security modules. It is easy for an attacker to insert a malware (malicious programs such as computer viruses, Trojan horses, worms, etc.) of his or her choice to launch various attacks. It can spoof the IP address of a networked device and launch man-in-the-middle attacks that bypass the encryption of data passing through the networked devices. An attacker can also transmit secret information through a backdoor created in these cloned systems. Additionally, an attacker could launch a denial-of-service (DoS) attack to make a system or network resource unavailable temporarily or indefinitely by interrupting or suspending the services when it is essentially needed; *(ii)* may have substandard or counterfeit components. The reliability of such components are questionable and the system may fail before the specified end-of-life which cause serious safety concerns; and finally *(iii)* pose negative impact to the R&D and causing significant financial loss due to the loss of business from the trade of these non-authentic products.

1.1 Prior Work

To the best of our knowledge, this is the first work to comprehensively address supply chain vulnerabilities for electronic systems. Conventional security analysis and countermeasures primarily target the hardware and firmware separately. In this section, we will discuss the existing related work in literature.

1.1.1 Protection of Hardware

A non-authentic hardware can be verified by testing a unique ID and checking whether or not the hardware is registered. We call this as system ID (*SID*), which is introduced in this paper to authenticate the hardware. However, a significant research has been carried out to create chip IDs (*CIDs*) to uniquely identify the ICs. It is relevant to discuss the prior works on *CIDs* as our proposed solution uses these *CIDs* to create a unique *SID*.

In recent years, physically unclonable functions (PUFs) [9], [10], [11] have received much attention in the hardware security community, as they can generate unique and unclonable bits for the identification and authentication of ICs. PUFs use inherent uncontrollable and unpredictable variations from a manufacturing process to produce random and unclonable bits. Several PUF architectures have been proposed over the years which include arbiter PUF [9], ring oscillator PUF [10], SRAM PUF [11], etc. As the PUF generates random and unclonable bits, it can be used to generate *CID*. However, it is challenging to generate a *SID* by using PUFs as they reflect the quality of ICs rather than a whole system.

At present, Deoxyribonucleic acid (DNA) markings [12] are commercially available to provide traceability for electronic components and detect counterfeit ICs [13]. Detection can be accomplished via fast or detailed authentication. In fast authentication, the marks fluoresce under UV light, and depending on the color, a decision is taken regarding the

authenticity of an IC. Detailed authentication can be performed in a specific laboratory, which makes it extremely time-consuming and costly [14]. Further, if counterfeiters add a different mechanism to the chip that can produce the same kind of fluorescence, DNA marking will be ineffective for fast authentication.

The authors in [5] proposed an authentication approach to detect counterfeit PCBs. This approach creates a unique and robust signature from the process induced variations present in the trace impedance of the PCBs. However, this approach is only limited to the hardware of the PCBs and not applicable to the firmware of a system. In addition, this scheme fails to detect counterfeit components present in a system. The authors in [15] proposed an encrypted chip ID technique to authenticate a system. This scheme can be circumvented by careful data analysis as the key bits are transmitted during the chip ID request. As the locations of the key bits are fixed in the data stream, an attacker can find those locations in linear time.

1.1.2 Protection of Firmware

Firmware has been extensively targeted over the years to exploit embedded systems. A majority of these attacks include modification of the firmware designed for various devices including network systems [16], [17], [18], [19], hard drives [20], electronic game console [21], etc. The attack can be carried out either on the target devices directly or a peripheral device first which can be used to eventually infect the target device. The authors in [22] have shown that malware can be injected into printers during remote firmware update. The attack on the control system of a car by uploading a firmware have been presented in [23]. Similar attacks have also been reported in [24], [25], [26], [27].

A variety of solutions have been proposed over the years to circumvent the attacks. The authors in [20] proposed the integrity verification of the peripherals' firmware by using remote software-based attestation. Cumulative Attestation Kernel (CAK) was proposed to verify the integrity of the firmware over an interval of time [17]. The authors in [18] proposed a solution to protect the persistent state of a trusted module in external memory by maintaining an authenticated channel between the trusted module and the memory, to protect the firmware against non-invasive attacks. The authors in [19] proposed a signature verification scheme to prevent the installation of malicious firmware on a mouse. The authors in [21] proposed integrity verification at different levels of the boot-up process to ensure the loading of proper firmware into the memory. The authors in [28] proposed a key-based control flow obfuscation to prevent piracy and malicious modification.

Recently, Trusted Platform Module (TPM) has gained popularity because it allows users the ability to perform functions securely by ensuring trust in the computing platform [29]. The TPM module stores pre-run time configuration parameters to run an application to its desired state. However, TPM cannot prevent cloning as it cannot control the software that is running on a system, rather than verifying its integrity.

1.2 Contribution

Separate protection of hardware or firmware cannot provide protection against all vulnerabilities listed in Fig. 1. In this paper, we propose the System-level Mutual Authentication (SMA) scheme where the hardware authenticates firmware and vice versa. More specifically, if the legitimate firmware runs in a different board, it produces incorrect results and if a legitimate board runs a different firmware, it again produces incorrect responses or raises a flag. SMA requires a unique system ID that certifies the authenticity of the hardware and an obfuscated firmware that becomes operational once it runs on an authentic hardware. Thus, we will first describe a robust approach for constructing a unique system ID (see Section 2), and then present a novel firmware obfuscation methodology (see Section 3). Finally, we will integrate these two solutions and present our proposed system-level mutual authentication scheme (see Section 4). Our contributions in this paper include—(i) creation of a unique system ID and verification of the ICs in a electronic system, (ii) development and security analysis of an obfuscated firmware, and (iii) development of a mutual hardware-firmware authentication scheme.

- *System ID:* We propose a robust approach for creating a unique system ID that can serve as a unique signature for an electronic system. This proposed *SID* is created from the IDs of different chips (*CIDs*) present in a system and is the XOR summation of these chip IDs. This ID is unique and resistant to cloning as it is never exposed to an outside world. Once a system is assembled, each *SID* for a system is created and stored in a secure database at the trusted system integrator's site for future authentication. In the current manufacturing flow, there is no protection of a system from being non-authentic. In our proposed approach, we obfuscate the firmware (see Section 3 for details) in such a way that it can only work upon receiving a correct system ID. Once the systems are manufactured and assembled, they must be shipped to the original system designer to compensate the obfuscated firmware.

When a system is powered up in the field, it is necessary to construct the *SID* for proper operation. The in-system processing unit (e.g., microprocessor, digital signal processor, or microcontroller) is responsible for creating the *SID*. Hereafter, we will represent in-system processing unit as processor. We propose a secure protocol, TIDP (see Section 2.2), which helps the processor to collect all the encrypted *CIDs* from the chips. Like *SID*, the chip IDs never leave the chips without encryption. We also propose a novel communication protocol, TIDS (see Section 2.3), to transfer the *SID* securely to an authenticating server located on the system integrator's site. We show that TIDP and TIDS are resistant to various known attacks (see Sections 5.2 and 5.3 for details).

This unique *SID* provides excellent protection for the hardware. If one of the ICs (including the processor) is replaced with a new one (recycled or low grade counterpart having a different *CID*), it will be reflected in the *SID*. For the defective systems, the *SID* is never registered in the system integrator's database. In summary, any mismatch of the *SID* will raise a flag of being a system as non-authentic.

- *Obfuscated Firmware:* We propose a methodology to obfuscate the firmware targeted for a system. The firmware is obfuscated in such a way that it can produce the correct result when it is loaded to a system uniquely designed for that firmware copy. In our proposed approach, we remove a selected number of instructions from the firmware. During the run

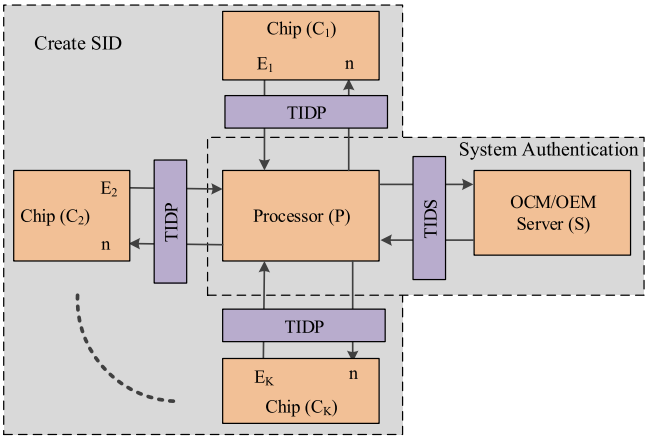


Fig. 2. Proposed authentication process to construct SID and verify IDs.

time, these instructions are recovered from the SID and inserted to the respective locations. We propose a novel architecture to construct a set of machine code in binary format from the SID. We show that the attacker's effort to predict correct instructions belongs to the class of exponential complexity with the program size (see Section 3.4).

• *System-Level Mutual Authentication:* The system ID provides an easy way of detecting a non-authentic hardware. However, it cannot prevent an adversary creating this non-authentic hardware. On the other hand, an adversary cannot reconstruct an original firmware from the obfuscated one. Incorporating these two can prevent an adversary from creating a non-authentic system. Thus, we propose a system-level mutual authentication scheme where the hardware authenticates firmware first and then gives control to the firmware. The firmware verifies the authenticity of the hardware by verifying the system ID and works properly when it runs on an authentic hardware. We believe that a comprehensive solution which includes both the hardware and firmware, is a robust way of preventing non-authentic systems. SMA needs to be performed every time when the system powers up from the off state. Note that these systems need to be connected to the Internet during the authentication as it is necessary to send the $SIDs$ to the system integrator for verification.

The rest of the paper is organized as follows. In Section 2, we describe our proposed approach to protect the hardware of an electronic system from vulnerabilities listed in Fig. 1. We present our proposed system ID (SID), TIDP protocol to transfer the $CIDs$ to the processor to construct SID and TIDS communication protocol to transfer the SID to the server for the verification of a system. In Section 3, we propose our obfuscation scheme to protect the firmware from copying. In Section 4, we integrate the approaches for protecting hardware and firmware to our proposed SMA scheme. We present the area overhead and attack analysis in Section 5. We conclude the paper in Section 6.

2 Protection of Hardware

To prevent a system being non-authentic, it is absolutely necessary to protect the hardware from unwanted modification. The modification may be the replacement of different chips or the PCBs with low grades or counterfeit ones. We propose to create a unique and unclonable system ID to verify the authenticity of an electronic system. Any modification of the hardware will reflect in the system ID and can be detected by a simple ID matching scheme during authentication. In addition, the obfuscated firmware will work properly when it receives this SID. In this section, we propose an approach to create a unique SID that needs to be kept secret to prevent copying and cloning by an adversary.

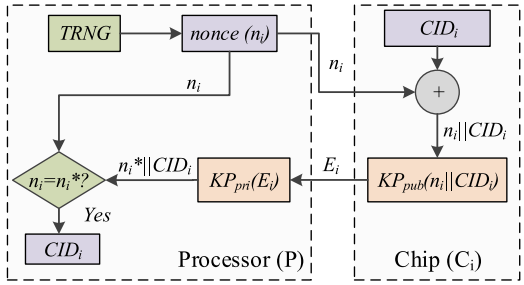


Fig. 3. Proposed TIDP protocol to securely receive the chip ID from an IC.

The authentication of the hardware is performed in two phases. In Phase-1, a unique system ID is created and then in Phase-2, all this SID validated in the system integrator's (OCM/OEM) server. Here, OCM and OEM stand for original component manufacturer and original equipment manufacturer, respectively. Fig. 2 shows the system level overview of our proposed authentication process. At Phase-1, the processor communicates with all the chips by using our proposed communication protocol, TIDP (see Section 2.2) to receive all the encrypted chip IDs. At Phase-2, the processor uses our proposed communication protocol, TIDS (see Section 2.3) to transfer the SID to the server located at the trusted system integrator's site. The server validates SID to verify the authenticity of the system. It is important to note that the $CIDs$ and SID never leave the system without encryption.

2.1 System ID

As a system contains ICs as its core components, it is imperative that its signature can be formed from the biometrics of the ICs. Thus, we propose a simple but efficient method of forming a system ID (SID) from the $CIDs$. The SID can be expressed as the exclusive-OR summation of all the $CIDs$ and represented below:

$$SID = CID_p \oplus CID_1 \oplus CID_2 \oplus \ldots \oplus CID_K, \quad (1)$$

where,

K is the number of chips on an electronic system;
CID_p is the chip ID of the processor; and
CID_i is the chip ID of the ith chip.

2.2 TIDP: Communication Protocol for Transferring IDs to the Processor

The success of creating an unclonable SID depends on the secrecy of the $CIDs$. If an attacker finds the $CIDs$ from a working system, he can easily form the SID. In this section, we propose TIDP protocol to provide the adequate security with minimal area overhead.

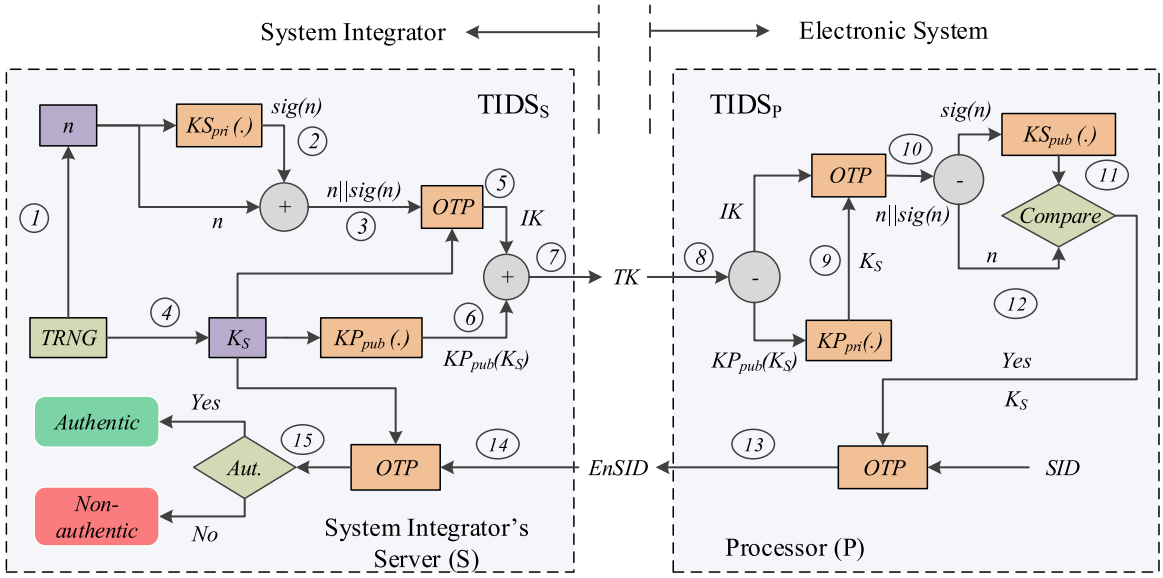


Fig. 4. Proposed TIDS communication protocol to transfer SID to the system integrator.

Fig. 3 shows the TIDP protocol to receive the chip ID from an IC. This protocol is described below:

1) The processor generates a nonce (n_i) and sends it to chip i (C_i).
2) Upon receiving n_i, the Chip C_i concatenates it with the chip ID.

$$n_i||CID_i = \{n_i, CID_i\},$$

3) The chip, encrypts $n_i||CID_i$ with the public key of the processor (KP_{pub}) and sends it to the processor. One can implement widely used Rivest-Shamir-Adleman (RSA) algorithm [30] to encrypt $n_i||CID_i$. To reduce area overhead, we can implement a minimum size datapath of the RSA block using only 861 gates [31].

$$E_i = KP_{pub}(n_i||CID_i),$$

4) The processor decrypts E_i by using its private key (KP_{pri}) to extract $n_i^*||CID_i$ and performs a comparison of the stored nonce (n_i) and recovered nonce (n_i^*). If these two nonces are equal, then the comparison passes. Finally, the processor uses the CID_i to construct the system ID by using Equation (1).

2.3 TIDS: Communication Protocol for Transferring SID to the Server

In our proposed architecture, the processor is responsible for transferring the *SID* to the server. It is of utmost importance to validate the identity of the server before sending the *SID*. We use digital signatures to authenticate the server. In addition, key establishment is necessary between the processor and the server to encrypt the IDs. TIDS uses key transport from the server to the processor by using asymmetric key encryption. RSA algorithm [30] can be used for this purpose. It is worthwhile to note that one can use discrete logarithm or elliptic curve algorithms based on performance [32].

Fig. 4 shows our proposed protocol for the secure transfer of K_S from the server to the processor. To achieve this, the public key of the server (KS_{pub}) and the private key of the processor (KP_{pri}) needs to be embedded in the processor. The server has two other counterparts, i.e., private key of the server (KS_{pri}), and public key of the processor (KP_{pub}). The system integrator generates these two key pairs and store $\{KS_{pub}, KP_{pri}\}$ in the processor after the assembly of the systems and they are same for all the systems. The steps for this communication protocol, TIDS are listed below:

1) The server generates a random nonce (n) using a true random number generator (TRNG) which is unique every time when it initiates a communication.
2) The nonce n is encrypted with its private key KS_{pri} to form a signature such that the processor can uniquely identify the origin of this nonce. This step ensures the end-point authentication

$$sig(n) = KS_{pri}(n).$$

3) The nonce n and its signature $sig(n)$ are concatenated

$$n||sig(n) = \{n, sig(n)\}.$$

4) The server generates a unique and random session key (K_S) for every communication and stores it for decrypting encrypted IDs from the processor.
5) A one-time-pad (OTP) encrypts $n||sig(n)$ with K_S to form IK

$$IK = K_S(n||sig(n)) = K_S \oplus (n||sig(n)).$$

6) The session key, K_S, is encrypted with the public key (KP_{pub}) of the processor such that only the processor can retrieve K_S.
7) The transmission key (TK) is formed by concatenating encrypted K_S and IK and sent to the processor

$$TK = \{KP_{pub}(K_S), IK\} = KP_{pub}(K_S)||IK.$$

8) Upon receiving the TK from the server, the processor separates encrypted K_S and IK.
9) Session key K_S is retrieved by decrypting $KP_{pub}(K_S)$ with KP_{pri}

$$KP_{pri}(KP_{pub}(K_S)) = K_S,$$

10) A one-time-pad is used to decrypt IK to retrieve $n||sig(n)$

$$IK \oplus K_S = K_S \oplus (K_S \oplus (n||sig(n))) = n||sig(n).$$

11) The processor computes n from the signature by using the public key of the server, KS_{pub}

$$KS_{pub}(sig(n)) = KS_{pub}(KS_{pri}(n)) = n,$$

12) A comparison is performed to match n and the decrypted signature $sig(n)$. This step ensures that the nonce was indeed originated from the server if and only if n equals to the $KS_{pub}(sig(n))$. If there is a mismatch, the processor stops the communication and log this attempt to a non-volatile memory by storing a counter value.
13) After verifying the authenticity, the processors now encrypts the SID by using a OTP with the session key K_S and sends it to the server

$$EnSIDs = K_s(SID) = K_s \oplus SID,$$

14) The server decrypts $EnSID$ with the previously generated session key K_S

$$K_s \oplus EnSID = K_s \oplus K_s \oplus SID = SID,$$

15) The server then searches for the SID in its database. If a match is found, the system is now authenticated and marked as genuine.

The verification of SID provides an effective way of determining the authenticity of a system. An authentic system is registered by the system integrator. An adversary cannot register a non-authentic system to the trusted system integrator's database, unless it is compromised, which is beyond the scope of this paper. In addition, our proposed SID is "unclonable" as it is constructed from the different $CIDs$, which exploit uncontrollable randomness of intrinsic manufacturing process variations.

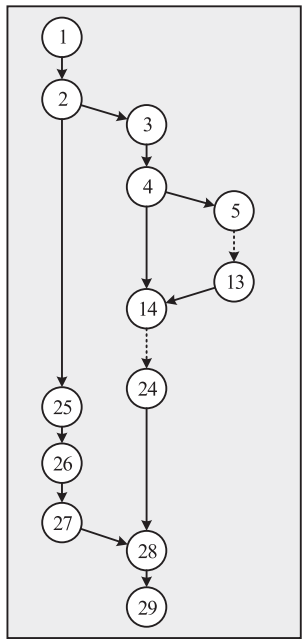
3 Protection of Firmware

Protection of electronic systems requires the complete protection of system firmware while maintaining a legitimate hardware. The system firmware is a special type of software that resides in a processor and performs the configuration and controls the operation of a system. The firmware generally can be loaded into a non-volatile memory (NVM), such as ROM, EEPROM, and flash. During the power-up of a system, the firmware is loaded into the internal SRAM from the NVM. An attacker has an opportunity to gain access to this firmware during system power-up.

The encryption of the entire firmware may not be a cost effective approach to prevent piracy. Encryption will certainly prevent an attacker to reconstruct the original firmware extracted from a non-volatile memory. However, it will increase the complexity of the system and may not be applicable to many low-cost embedded systems.

```
...
.text
.global main
#begin_text
1  main: lw  $v0  0x10000030
2  blt  $v0  $zero  error_display
3  li  $t1  2
4  blt  $v0  $t1  done
5  move  $a0  $v0
6  li  $t0  0
7  li  $v0  1
8  li  $a1  1
9  loop: add  $t1  $t0  $v0
10 move  $t0  $v0
11 move  $v0  $t1
12 addi  $a0  -1
13 bgt  $a0  $a1  loop
14 done: move  $a1  $v0
15 la  $a0  result
16 li  $v0  4
17 syscall
18 move  $a0  $a1
19 li  $v0  1
20 syscall
21 la  $a0  eol
22 li  $v0  4
23 syscall
24 j  end
25 error_display: la  $a0  error_msg
26 li  $v0  4
27 syscall
28 end: li  $v0  17
29 syscall
#end_text
```

a) MIPS program



b) DAG of the MIPS program

Fig. 5. An example MIPS program to calculate the value of nth Fibonacci numbers and its directed acyclic graph.

3.1 Motivation

The objective is to prevent an attacker from copying the entire firmware rather than to prevent an attacker from modifying it during run time to exploit it. There are numerous solutions already in practice or in the literature to detect/prevent run time exploitation (see Section 1.1.2). Here, we will only focus on developing a comprehensive solution to prevent an attack that could copy the firmware. The main idea is to obfuscate the firmware in such a way that it produces incorrect results unless it is correctly compensated at run time by its associated hardware system. Note that an attacker can copy the contents of a NVM to get the firmware, however, he cannot make it operational.

Fig. 5a shows a MIPS program to calculate the value of nth Fibonacci number for a non-negative number, n and its directed acyclic graph (DAG). The program consists of 29 instructions. Fig. 5b shows the directed acyclic graph of that MIPS program. There are three paths exists in this DAG. Path 1 consists of a set of 7 instructions, {1,2,25,26,27,28,29}. Path 2 consists of a set of 14 instructions, {1,2,3,4,14,...,14,28,29} and finally Path 3 consists of the set of 26 nodes, {1,2,3,4,5,13,14,...,14,28,29}. We will then calculate an attacker's effort to find a removed instruction from this program.

Let us start with removing instruction 11 $(move\ \$v0\ \$t1)$ to analyze the attacker's effort. Here the attacker can construct the DAG with missing node 11. The attacker first need to find out the path that produces the incorrect result. In this case, Path 3 of this obfuscated program will produce incorrect result. After the path is selected, the attacker must

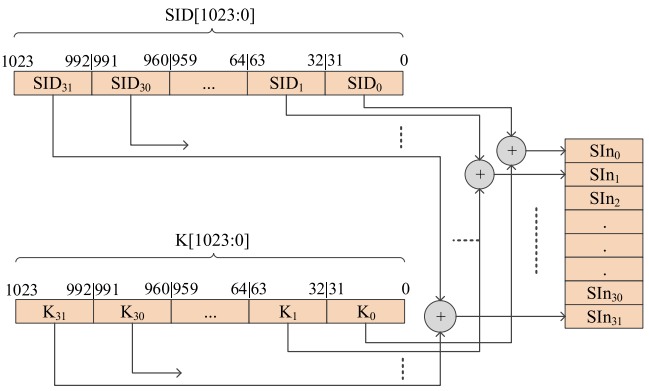


Fig. 6. Proposed architecture to generate machine instructions for firmware obfuscation.

find the correct instruction and inserts it into the correct location. Finding the correct instruction, an attacker need to simulate $C \times M$ times, where, C is the average number of combinations for the operands of an instruction, and M is the number of instructions for an instruction set architecture. Two scenarios may occur: Case 1, where the attacker arbitrarily inserts the instruction to make the program work; Case 2, where the attacker analyzes the code and predicts the location. In Case 1, the attacker's effort will simply be $\binom{26}{1}CM$. In Case 2, the attacker isolates the nodes from Path 1 and Path 2. These nodes are {5,6,...,13} and the attacker's effort can reduce to $\binom{13}{1}$. Thus the attacker's effort will be $min\{\binom{26}{1}, \binom{13}{1}\}CM = 13CM$. Here one can argue that if the system designer removes instruction 12, an attacker can detect this instruction with a careful code analysis, as there may be a decrement inside the loop. For this reason, we recommend to remove the instructions related to the operations rather than the formats or structures of a function. In addition, the analysis codes based on functionality may be challenging for a large program.

Let us now consider another example, where the system integrator removes instruction 2 ($blt\ \$v0\ \$zero\ error\_display$) or (instruction 4 ($blt\ \$v0\ \$t1\ done$)). The attacker cannot form a complete DAG from the given instructions and need to find the correct locations for these two instructions. In this case, the attacker's effort can be $\binom{29}{1}CM$. One can argue that an attacker can easily figure out the missing instructions by comparing with a standard function already available. This problem can be addressed by targeting instructions from the custom design functions, only developed by the trusted system integrators.

In the following, we propose a novel firmware obfuscation technique based on careful removal of few instructions. This obfuscation technique does not require encryption and can be implemented in a computationally efficient manner. It is also capable of operating with minimal hardware support. The fundamental idea of obfuscating the firmware is to remove a few instructions and then store this modified firmware into a non-volatile memory. The instructions are chosen in such a way that the flow of the program gets interrupted and produces incorrect results. The correct execution of the program can only be achieved once these instructions (reconstructed from SID) are inserted in the correct location of the firmware code.

3.2 Generate Instructions from the System ID

Developing a firmware that runs only on authentic hardware is the key to prevent an adversary constructing a non-authentic system. One way of achieving this is to add a unique hardware signature to the firmware such that it runs correctly on an authentic hardware. In this section, we propose an architecture to generate machine codes (instructions) from the hardware signature, which is the unique system ID developed in Section 2.

Fig. 6 shows the architecture behind the construction of machine codes. Each instruction is mapped to a binary code targeted for either reduced instruction set computers (RISC) [33] (e.g., MIPS ISA [34]) or complex instruction set computers (CISC) (e.g., X86 ISA [35]). The number of generated instructions depends on the size of the SID and the instruction set architecture (ISA). For example, a 1024 bit SID can generate 32 different MIPS instructions as each MIPS instruction is 32 bit long. As the bits in the SID are uniform and different for each system, it is hard to map a SID to a fixed set of instructions. To solve this challenge, we introduce a key (K), which needs to be the same length of SID.

The system integrator is responsible for generating K. After the assembly of a system, the system integrator obtains the SID from the system. As he/she has developed the firmware, he/she knows the instructions ($SIns$) and can easily compute $K(\{K_0||K_1||\ldots||K_{31}\})$ by simply performing an XOR operation with the $SID = \{SID_0||SID_1||\ldots||SID_{31}\}$ and $\{SIn_0||SIn_1||\ldots||SIn_{31}\}$

$$K_i = SID_i \oplus SIn_i,\ i = 0, 1, \ldots, 31, \tag{2}$$

Once the key value has been calculated (using Equation (2), the system integrator stores this key into an in-system one-time programmable memory [36]. The $SIns$ will now be generated during the system runtime (see Fig. 6) and are given by,

$$SIn_i = K_i \oplus SID_i,\ i = 0, 1, \ldots, 31, \tag{3}$$

3.3 Firmware Obfuscation Methodology

Fig. 7 shows our proposed flow to create an obfuscated firmware. During the development of the firmware, the system integrator removes I number of instructions from the firmware body such that it produces incorrect results. These removed instructions are recovered from the system ID during the run time to make the firmware complete.

The obfuscation process starts by finding a set of feasible control-flow paths in the program based on the dependencies of the input parameters. The algorithm greedily selects k paths from which I instructions are removed. We will show in Section 3.4 the security of the obfuscated firmware can be $O(N^I)$ and the designer can select any I paths.

Algorithm-1 shows the pseudo-code for the algorithm to find out all control flow paths of a DAG using a Depth-first Search (DFS) algorithm [28]. In this approach, an assembly program is modeled as a DAG. In this paper, we focus only on the MIPS instruction set architecture (ISA) for simplicity. This algorithm can be implemented for other ISAs as well. Each instruction of the program represents a node of the graph and each node may have one child for the non-branch instructions or two children for the branch instructions. We do not consider loops as we need to maintain the acyclic nature of the graph. Note that any child for a particular level

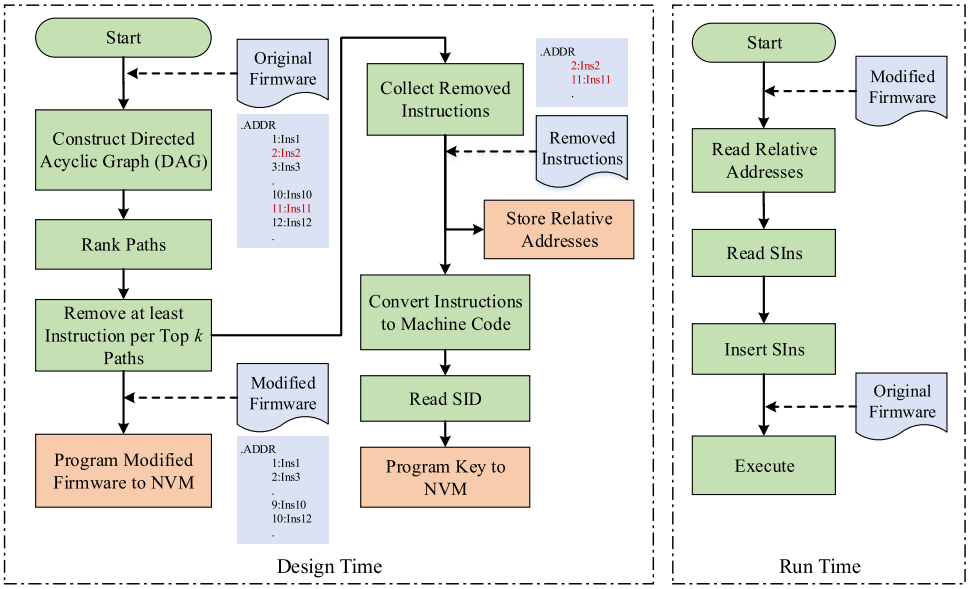


Fig. 7. Proposed flow for firmware obfuscation.

always represent the next instruction. The detailed description of this algorithm can be found in [28].

Algorithm 1. Procedure Enum_Paths_DFS Enumerate all Possible Control Flow Paths ($\mathbb{P} = \{P_1, P_2, \ldots, P_m\}$ of a DAG Representing an Assembly Language Program

input: Directed Acyclic Graph G, $instr\_stack$, $curr\_node$, and $last\_node$
output: Current Path
if $curr\_node \neq \Phi$ **then**
  $push\_on\_stack(instr\_stack, curr\_node)$;
  if $curr\_node == last\_node$ **then**
    $e.pathcount \leftarrow (e.pathcount + 1)\ \forall\ e$ on current path;
  $Enum\_Paths\_DFS(G, instr\_stack, curr\_node \rightarrow left\_child, last\_node)$;
  $Enum\_Paths\_DFS(G, instr\_stack, curr\_node \rightarrow right\_child, last\_node)$;
  $pop\_from\_stack(instr\_stack)$;
else
  return ;
end

The obfuscated firmware is then stored in a non-volatile memory. An attacker can have the access to the instructions. However, to make it operational he/she needs to insert the removed instruction to the respective locations. We will perform an attack analysis to find out the attacker's effort to break the system and make the program working in Section 3.4. Once the instructions are removed from the program, the system integrator collects those instructions (Ins_2, Ins_{11}, ...) and store their relative addresses (2, 11, ...) to an on-chip NVM such that an attacker cannot have access to it. The system integrator reads the system ID from the working system and generates the key (see Section 3.2). Finally, the key is stored in an NVM of the system.

During the run time, a stitcher module inserts the previously removed instructions to the respective locations of the program. This stitcher module first reads the relative address of the instructions and then read the $SIns$ (shown in Fig. 6). Here the $SIns$ are Ins_2, Ins_{11}, etc. Thus, the original program is recovered and ready to execute.

Note that the update process of firmware requires to store the relative addresses corresponding to the removed instructions of the new firmware. If those instructions are same as the previous ones, we do not need to update the key (see Fig. 6), otherwise we need to update the key to correctly generate these removed instructions at run time.

3.4 Security Analysis

The security analysis is based on the attacker's effort to break the obfuscated firmware, which requires an attacker to identify the exact I instructions with their relative addresses. We denote the attacker's efforts as E. Let us assume that there are N instructions of a program corresponding to a set $S_N = \{In_0, In_1, \ldots, In_N\}$ which forms a directed acyclic graph shown in Fig. 8. There are m paths denoted by P_1, $P_2, \ldots, P_m$. Path P_1 contains the instruction set $S_{P_1} = \{In_0, In_1, In_4, \ldots, In_{N-1}, In_N\}$. Obviously, $S_{P_1} \subset S_N$ and $S_{P_1} \cup S_{P_2} \cup \ldots \cup S_{P_m} = S_N$. We also assume that the length of these paths are $L_1, L_2, \ldots, L_m$.

Now, there are k paths from the total number of m paths that have been modified and among them, one or more instructions have been removed. Specifically, these are the paths that produce incorrect results. To make the complete program working, an attacker first need to find these paths. Thus, the attacker's effort to find these k paths from the total m paths can be $\binom{m}{k}$.

Let us assume that there are r average instructions per path that have been removed. An attacker will first insert one instruction in a path and try to make it work. Then he inserts 2 instructions and keeps adding instructions until r. Thus, the attacker's effort for a path will be the summation of all the trials. For path P_1, the attacker's effort would be,

$$E_{P_1} = \sum_{i=1}^{r} \binom{L_1}{i} * C * M,$$

where,

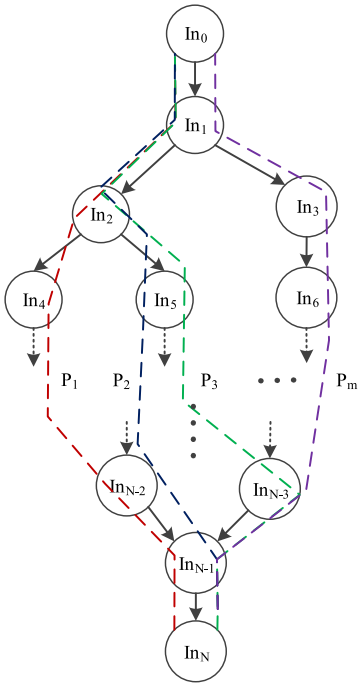


Fig. 8. Directed Acyclic Graph of a program.

M: Number of instructions for an instruction set architecture.

C: Average number of combinations for the operands of an instruction.

The total attacker's effort,

$$\begin{aligned} E_P &= \binom{m}{k}\left\{E_{P_1} + E_{P_2} + \cdots + E_{P_k}\right\} \\ &= \binom{m}{k}\left\{\sum_{i=1}^{r}\binom{L_1}{i}CM + \sum_{i=1}^{r}\binom{L_2}{i}CM \right. \\ &\quad \left. + \cdots + \sum_{i=1}^{r}\binom{L_k}{i}CM\right\} \\ &= \binom{m}{k}\left\{\sum_{j=1}^{k}\sum_{i=1}^{r}\binom{L_j}{i}CM\right\}, \end{aligned} \tag{4}$$

Here $k * r = I$ where, I is the total number of instructions constructed from the SID.

Now, an attacker can arrange I instructions from the total number of N instructions for a program such that one arrangement works. The attacker's effort now becomes $\binom{N}{I} * C * M$ as one of the combination will certainly work. Thus the upper limit of the attacker's effort becomes

$$E_P = \binom{N}{I}CM, \tag{5}$$

Thus, the attacker effort will be

$$min\left[\max_{k,r}\left(\binom{m}{k}\left\{\sum_{j=1}^{k}\sum_{i=1}^{r}\binom{L_j}{i}CM\right\}\right), \binom{N}{I}CM\right], \tag{6}$$

For a small program, the number of paths (m) may be less than the number of instructions (N). However for a reasonable size program, we expect that the number of paths (m) are greater than number of vertices (N) (see Table 2). In this case, our objective is to find r and k such that $E_P \geq \binom{N}{I}CM$ and we can certainly state that $E_P = O\left(\binom{N}{I}CM\right)$. Considering $m = N$, we can expressed E_P as

$$E_P = \binom{N}{k}\left\{\sum_{j=1}^{k}\sum_{i=1}^{r}\binom{L_j}{i}CM\right\}, \tag{7}$$

Now consider a case where $k = k_{max} = I$. Here, I is the total number of instructions need to be removed from a program. In this case, the value of r becomes 1.

$$E_P = \binom{N}{I}\left\{\sum_{j=1}^{I}(L_j)CM\right\} \geq \binom{N}{I}CM, \tag{8}$$

Thus, we will always achieve the upper limit of attacker's effort, when we consider to remove one instruction per path from any I paths.

Note that the occurrence of an instruction with specific operands cannot simply be the probability of $1/(C * M)$ due to the existence of different types of instructions in an ISA and their interdependencies in a program. For the sake of simplicity, we multiply $1/C$ and $1/M$ to find the probability of occurrence assuming all instructions are independent and equally likely. However, the value of CM cannot be less than 1 for any circumstances as an attacker must try at least one guess to predict the correct instruction.

By further simplifying Equations (5) and (8), we get

$$\begin{aligned} O(E_P) &= O\left(\frac{N(N-1)\ldots(N-I)}{I!}CM\right) \\ &= O\left(\frac{min(N^I, N^{(N-I)})}{I!}CM\right) \\ &= O(N^I), \quad \text{as } N \gg I, \end{aligned}$$

From the above analysis, it can be concluded that an attacker need to simulate the obfuscated firmware, $O(N^I)$ times to make it perfectly operative.

There are three key points worth noting with respect to the firmware security. First, the instructions can be related inside a function and one can potentially find them out when he/she knows the complete functional behavior. However, it will be a much harder work to predict the correlation when an adversary does not know the complete functionality of a firmware. Second, an adversary need to find the location of the missing instruction inside that complex function. This can also be a very difficult task when he/she does not know the complete control flow details even though he/she may know the complete or part of the overall functionality of the system. Finally, an adversary needs to find the right instruction for that location. The difficulty depends on the instruction set architecture (ISA) and the topology of the firmware. A judicious entropy-guided selection of both location and type of instruction is needed to make the guessing process for firmware reconstruction practically unfeasible. As the system integrator develops the firmware, he can select a set of uncorrelated instructions (e.g., the difficulty of predicting a branch

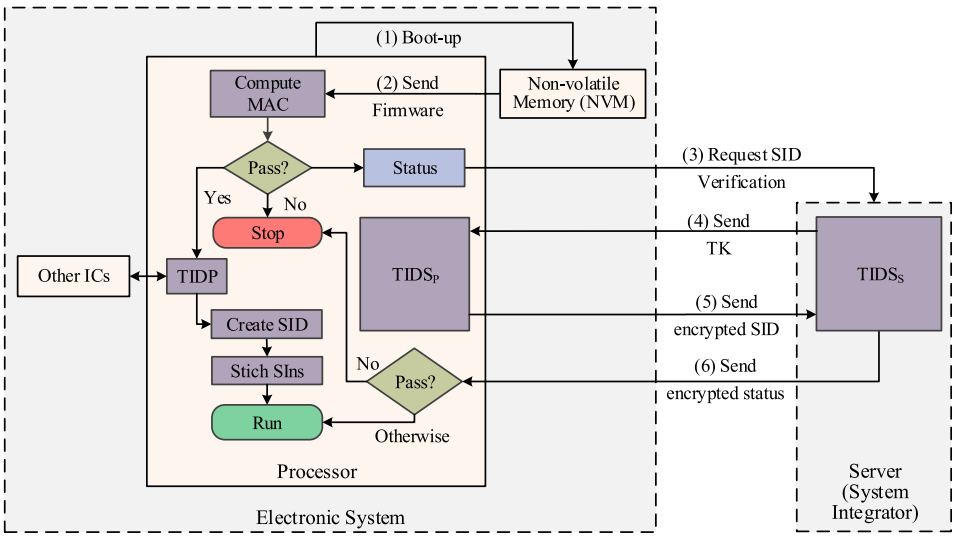


Fig. 9. System-level mutual authentication of the hardware and firmware.

instruction is much more than a load instruction) for removal based on an efficient algorithm. Such a selection algorithm will be the subject of our future research.

4 System-Level Mutual Authentication

The individual solutions for securing hardware and firmware cannot prevent a system being non-authentic. Protection of both system hardware and firmware is necessary. In this section, we present a system-level mutual authentication (SMA) approach where the system hardware can authenticate the firmware running on it and the firmware can authenticate the hardware. This authentication needs to be performed every time when the system powers up from the off state.

Fig. 9 shows our proposed SMA approach. The process starts by authenticating the firmware first and then the firmware certifies the hardware. The detailed steps are listed below:

Step 1: When a system is powered up, it goes through a multi-phase (preloader and boot loader) boot-up process. The instructions controlling the basic operating characteristics of the system are executed from a read only memory (ROM) to initialize the system and these instructions are responsible for loading the firmware into the random access memory (RAM). In this work, we use the existing boot-up process to initialize the system.

Step 2: Before the firmware is loaded into the RAM, the boot loader computes a message authentication code (MAC) from the firmware and the result is compared to an expected MAC maintained in it. This step verifies the integrity of the firmware and is already employed in practice [21]. This enforces to use the original modified firmware. Upon successful verification, the processor uses TIDP protocol to collect all the $CIDs$ from other ICs (see Fig. 3) and then construct SID. The stitcher module then constructs the $SIns$ from the key (K) and SID. It then adds $SIns$ to the firmware (see Fig. 7) and loads it to the RAM for the execution.

Step 3: Upon the successful boot-up, the firmware takes control of the system operation and it then authenticates the system hardware. The firmware sends the verification status message to the system integrator and requests to initiate the hardware authentication process.

Step 4: The server generates a random nonce (n_i), forms a signature ($sig(n_i)$) and then concatenates them (Steps 1-3 of Fig. 4). TRNG generates a random number, which is the concatenation of K_S and K_{status}, where K_{status} will be used later in Step 6 to encrypt the status of the SID verification. Then, the server generates the transmission key (TK) using Steps 4 to 7 (see $TIDS_S$ block of Fig. 4) and sends it to the processor.

Step 5: The processor performs Step 8 through Step 12 of TIDS (see $TIDS_P$ block of Fig. 4) to obtain K_S and K_{status}. It then verify that the request was originated from the system integrator. Upon successful verification, the processor sends the encrypted system ID ($EnSID = SID \oplus K_S$) to the system integrator.

Step 6: The server decrypts $EnSID$ to get back the original SID. It then checks the SID with previously stored ID by the system integrator. It then sends an encrypted verification status to the processor. Note that the status is encrypted using an OTP with the key K_{status}.

Step 7: The Firmware decrypts the status using the key K_{status} and issues a force reset signal upon failed hardware authentication to the processor to stop all the processes.

In the SMA approach, different levels of protection are already implemented in our proposed flow. First, the conventional MAC comparison (see Step 2 in Section 4) stops a tampered or modified firmware being loaded into the memory. Second, the stitcher module inserts $SIns$ to make the firmware complete. Finally, the system ID is verified over the Internet. The first and second level of authentication do not require the Internet.

For IoT devices that are connected to the Internet, SID verification can be performed using system designer's secure database as discussed in the paper. A recent report from Cisco shows that the connected devices will be increased exponentially in coming years and there will be 50 Billion connected devices by 2020 that will have access to internet [37]. Even though the requirement for our protocol seems stringent, it will be very common in near future.

For systems, which are not connected to Internet, an alternative verification approach can be designed using Trusted Platform Module (TPM) on board, which can securely store

TABLE 1
Area Overhead Analysis

SoCs	ASIC Gates	Memory	Logic Overhead (%)	Memory Overhead (%)
Z-7010	430 K	240 KB	0.74	0.260
Z-7015	1.1 M	380 KB	0.29	0.164
Z-7020	1.3 M	560 KB	0.24	0.112
Z-7030	1.9 M	1060 KB	0.17	0.059
Z-7035	4.1 M	2000 KB	0.08	0.031
Z-7045	5.2 M	2180 KB	0.06	0.029
Z-7100	6.6 M	3020 KB	0.05	0.021

the signatures ($TIDS_S$) and compare. Such an approach is quite viable for systems which already have TPM, but would incur additional cost for systems which do not have TPMs. The revised manuscript is updated with this information.

5 Analysis

In this section, we analyze the area overhead for our proposed TIDS and TIDP protocols for SMA approach. We also evaluate the security of these protocols. Finally, we calculate the attacker's effort to break our obfuscated firmware.

5.1 Area Overhead Analysis

The area overhead of our proposed solution is due to the communication protocol, TIDS. Many of the modules for TIDS already exists in the modern processors. However, we still estimate an approximate gate count for our proposed TIDS. They are summarized below:

(i) RSA module: The majority of the area overhead comes from the RSA module, which consists of datapath and key storage elements. The area from the datapath can be reduced significantly by selecting a slower RSA module as the speed of operation is not a major concern. It is reported that a minimum size RSA datapath can be implemented by using only 861 gates [31]. For a key storage element, we need to allocate 2048 bits of memory per RSA key.

(ii) OTP module: The size of the one-time-pad depends on the size of the SID. For a 128 bit SID, we need 128 XOR gates. For a larger size SID, we can perform XOR operation in stages rather than all at once. For example, we need 8 stages, which use the same 128 XOR gates for a 1024 bit long SID.

(v) TRNG: A single TRNG is used for generating the nonce (n) and session keys (K_Ss). We propose the use of an area efficient cryptographically secure pseudorandom number generator [38] or [39] depending on the implementation choice.

(vi) Non-volatile memory: A non-volatile memory is required to store the RSA keys and the session keys (K_Ss). We need 4 K bits to store RSA public and private keys. In addition, we need to store 1 K bit key (K) for generating machine code.

Considering all these modules, we estimate approximately a total gate count of slightly more than 3 K and a memory of 5 K bits. Table 1 summarizes the overhead analysis in the context of the Xilinx 7 series FPGAs [40]. We choose Xilinx FPGAs as their sizes closely match with different processors. The logic overhead varies from 0.74 to 0.05 percent and the memory overhead varies from 0.260 to 0.021 percent. The data indicates that both the logic and memory overhead lies well below 1 percent and can be ignored for larger SoCs, for example Xilinx Z-7035, Z-7045, and Z-7100 programmable devices. These overheads are also significantly low for smaller SoCs. Note that the actual area overhead can be even less as most of the modules are already in modern processors.

The area overhead for TIDP is not significant. Each participating chip requires only one key storage element and RSA datapath, which can be implemented by using only 861 gates [31]. The key can be stored in a anti-fuse based one-time-programmable memory due to its low size (only two transistors per bit) [36].

5.2 Attack Analysis on TIDP

The security of the TIDP protocol lies on the security of the RSA algorithm. The 2048-bit RSA key provides an equivalent security of 112 bit [41], which is common in practice. We assume that RSA cannot be broken by brute force attacks, which means that it is computationally unfeasible to estimate $n_i||CID_i$ after observing n_i and E_i, unless he knows KP_{pri}. In the following, we provide the security analysis for two common attacks on this protocol.

(i) Replay attack: In this attack scenario, an attacker monitors several previous communications and then replays with a previously used value. He first observes $\{n_1, n_2, \ldots, n_k\}$ and corresponding $\{E_1, E_2, \ldots, E_k\}$. He now monitor the nonce n_{k+1} and then replaces it with one of the previously observed n_i. The chip will return E_i to the processor. At the processor, $n_i||CID_i$ will resulted after decryption. At this point the comparison will fail as $n_{k+1} \neq n_i$. We assume that the nonces are completely random as they are generated from a TRNG. In addition, the attacker needs to perform this attack every time when the system powers up, which makes it unpractical.

(ii) Man-in-the-middle attack: The RSA keys can be programmed in an anti-fuse based one-time-programmable memory [36] after the manufacturing of a system by the trusted system integrator. It is thus safe to assume that man-in-the-middle attack is not possible due to the excellent tamper resistivity of the anti-fuse memory.

5.3 Attack Analysis on TIDS

This protocol is similar to Pretty Good Privacy (PGP) by Phil Zimmermann, which is commonly used today for email delivery system. PGP has demonstrated excellent secrecy over the years [42]. However, our protocol can provide better secrecy compared to conventional PGP due to two reasons. First, we use one-time-pad (OTP) instead of AES, or some other block ciphers. There are several vulnerability reported when AES works with Cipher Block Chaining (CBC) mode [43]. There are also several countermeasures (e.g., AES with Counter mode) developed to alleviate such vulnerabilities. We can eliminate such vulnerabilities by incorporating OTP, which is possible due to small fixed size of $\{n, sig(n)\}$ and K_S. Second, in normal implementation, man-in-the-middle attacks may be possible due wrong implementations of the certificates. Today, numerous CAs are available, which issue certificates. In our proposed protocol, the trusted system integrator is responsible for programming the public and private keys to the system, which will remove the possibility of replacing these keys. Thus, it

TABLE 2
Attacker's Effort to Find the Missing Instructions

Program	# Instructions (N)	# Paths (m)	$\mathbf{E_{P_8}}$ $(CM=1)$	$\mathbf{E_{P_{16}}}$ $(CM=1)$	$\mathbf{E_{P_{32}}}$ $(CM=1)$
hanoi.mips	132	169	1.8e + 12	1.6e + 20	4.5e + 30
MD5.mips	250	114	3.8e + 14	2.0e + 22	6.2e + 31
connect4.mips	270	4,146	6.3e + 14	2.4e + 25	3.6e + 41
DES.mips	372	5,241	8.4e + 15	4.6e + 27	1.7e + 46
sudoku.mips	436	111,113	3.0e + 16	6.2e + 28	3.4e + 48
ID3Editor.mips	878	98,724	8.5e + 18	5.2e + 33	3.3e + 58
cipher.mips	1,956	150,129	5.2e + 21	2.1e + 39	6.2e + 69
decoder.mips	21,024	$\gg 21{,}024$	9.4e + 29	6.9e + 55	7.9e + 102

is fair to say that TIDS is as good as PGP. In following, we will investigate various attacks on TIDS.

(i) Encryption: In our proposed communication protocol (see Fig. 4), we use RSA to generate the signature (Step 2) and to encrypt the session key (Step 6). An equivalent security of 112 bit (exhaustive search of 2^{112}) can be achieved, when we choose a key length of 2048 bits [41]. However, a higher level of security can be obtained with the cost of area necessary for additional key storage. A perfect secrecy can be achieved when we use one-time-pad to encrypt $n||sig(n)$. Thus, our protocol provides an overall RSA equivalent security.

(ii) Man-in-the-middle attack: As the RSA key-pairs are programmed by the system integrator during the assembly of the board, it is safe to assume that man-in-the-middle attack is not possible.

(iii) Replay attack: In this attack scenario, an attacker monitors previous communications and then apply a previously used value to break the system. He first observes {TK_1, TK_2, ..., TK_{k-1}} and corresponding {$EnSID_1$, $EnSID_2$, ..., $EnSID_{k-1}$}. He now monitor the TK_k and then replaces with one of the TK_i, $i < k$. This will pass the verification stage (Stage 12 in Fig. 4), as this TK_i was generated from the server. At this point, the processor will return encrypted SID corresponding to that session with session key ($K_{S,i}$) where $EnSID_i = K_{S,i} \oplus SID$. At the server, the output of the OTP will be $EnSID_i \oplus K_{S,k} = K_{S,i} \oplus SID \oplus K_{S,k} \neq SID \; \forall \; K_{S,i} \neq K_{S,k}$. Thus, the attack will fail as these K_Ss are completely random and generated from a TRNG.

One may ask why the processor will transmit $EnSID$ for a stale session. This situation can also be addressed by a simple modification. The processor needs to send a $nonce$ before the communication begins. The server forms the signature $sig(n||nonce)$ rather than only $sig(n)$ (see Step 2 of TIDS). The processor compares $n||nonce$ with decrypted $sig(n||nonce)$ (see Step 11 of TIDS). If an attacker try to replay TK, the comparison will be failed (Steps 11 and 12 of Fig. 4) and the attack will be detected. In the original TIDS protocol, we have not added this step as the replay attacks are well-addressed.

(iv) Reverse engineering: In this attack scenario, an attacker reverse engineers the processor to retrieve the public key of the server (KS_{pub}) and private key of the processor (KP_{pri}). However, it is not possible to construct the private key of the server (KS_{pri}). The only way this protocol can be broken is when an attacker replaces KS_{pub} with his public key inside the processor. This seems infeasible when using an antifuse-based one-time programmable memory [36] to store the keys since antifuse cannot be modified afterwards.

5.4 Attack Analysis on Obfuscated Firmware

In this section, our objective is to estimate the attacker's effort such that we can evaluate the effectiveness of our firmware obfuscation technique. We have simulated different MIPS benchmark programs for such purpose. We have simulated a C/C++ program to construct the DAG from a MIPS program and find all possible paths. Our obfuscation approach can be applied to other RISC/CISC programs.

Table 2 shows the number of simulations an attacker needs to perform to make a firmware completely working. The attacker's efforts are represented by E_{P_8}, $E_{P_{16}}$, and $E_{P_{32}}$ and shown in Column 4, 5, and 6. We have estimated E_P considering 256, 512, and 1024 bit $SIDs$. The missing instructions from a firmware will be 256/32=8, 512/32=16, and 1024/32=32 respectively for the above SIDs. The benchmark programs are arranged in the table from small to large which is shown in Column 2. The detailed descriptions for these benchmarks can be found in [28].

We now analyze E_P for *MD5.mips* where the number of paths are less than the program size. Here, the number of paths (m) is 114 and maximum length of the paths (L_{max}) is 103. The maximum E_P can be achieved when $k = I$ and $r = 1$. Thus E_{P_8} will be $min\left[\binom{114}{8}\{\sum_{j=1}^{8} L_j CM\}, \binom{250}{8} CM\right] = min\left[4.34e{+}14, 3.8e{+}14\right] CM{=}3.8e{+}14$ (considering $CM{=}1$, and hereafter we will ignore CM). Now, $E_{P_{16}}$ will be $min\left[\binom{114}{16}\{\sum_{j=1}^{16} L_j\}, \binom{250}{16}\right]{=}min\left[2.0e{+}22, 6.8e{+}24\right] = 2.0e{+}22$. Using similar analysis, we have achieved $E_{P_{32}}$ of $6.2e + 31$.

For other programs where $m > N$, we can always guarantee that the upper limit of the attacker's effort will be $\binom{N}{I}$ when we select I different paths. An attacker needs to simulate $(4.5e + 30)$ times to find the missing instructions at their respective places in the original firmware when we consider a SID of 1024 bits for *hanoi.mips*. For very small program, it is wise to construct a wide SID. For a medium size program like *DES.mips*, an attacker needs to simulate $4.6e + 27$ times for a SID of 512 bits and it is sufficient to provide an adequate security. One can also choose a larger SID of 1024 bits which is theoretically impossible to simulate all $1.7e + 46$ ($\approx 2^{153}$) combinations using current computing resources. For any large benchmarks like *decoder.mips*, it is sufficient to use a SID of only 256 bits to protect a firmware. One can also select 512 or 1024 bit SID to provide a security which can practically be impossible to break by the brute force.

Note that our approach is heuristically secure as we have used provably secure modules (OTP, and RSA), and communication protocol (PGP). Furthermore, the our proposed SID is "unclonable" as it uses $CIDs$, which exploit uncontrollable

randomness of intrinsic manufacturing process variations. The *CIDs* are demonstrated to be unclonable in existing literature. We believe that it is not required to develop a formal security model as our approach builds on existing provably secure modules.

6 Conclusion

In this paper, we have presented mutual authentication, where the hardware verifies the firmware and the vice versa to protect electronic systems from various attacks. We have proposed an approach to create a unique system ID to verify the authenticity of the hardware. This *SID* is constructed from the IDs of various on-board ICs. Our proposed TIDP protocol protects a chip ID from being exposed to the outside world which makes our proposed *SID* unique and robust. The *SID* is securely transferred to the system integrator by using our proposed communication protocol, TIDS. For systems, which are not connected to the Internet, an alternative verification approach can be designed using TPM on board, which can securely store the signatures ($TIDS_S$) and compare them. Such an approach is quite viable for systems which already have TPM, but would incur additional cost for systems which do not have TPMs. Future work will include development of an efficient algorithm for judicious entropy-guided selection of both location and type of instruction that can make firmware reconstruction based on guessing missing instructions practically infeasible.

Acknowledgments

The authors would like to acknowledge Dr. Rajat Subhra Chakraborty, IIT Kharagpur, India, for his help on simulating the attacker's effort.

References

[1] M. M. Tehranipoor, U. Guin, and D. Forte, *Counterfeit Integrated Circuits: Detection and Avoidance*. Berlin, Germany: Springer, 2015.
[2] Texas brothers plead guilty to selling counterfeit 'Cisco' computer products, News Releases, U.S. Immigration and Customs Enforcement, Aug. 2009.
[3] The Federal Bureau of Investigation, "Departments of justice and homeland security announce 30 convictions, more than $143 million in seizures from initiative targeting traffickers in counterfeit network hardware," May 2010.
[4] U. Guin, D. DiMase, and M. Tehranipoor, "Counterfeit integrated circuits: Detection, avoidance, and the challenges ahead," *J. Electron. Testing*, vol. 30, no. 1, pp. 9–23, 2014.
[5] F. Zhang, A. Hennessy, and S. Bhunia, "Robust counterfeit PCB detection exploiting intrinsic trace impedance variations," in *Proc. IEEE VLSI Test Symp.*, Apr. 2015, pp. 1–6.
[6] N. Asadizanjani, S. Shahbazmohamadi, M. Tehranipoor, and D. Forte, "Analyzing the impact of X-ray tomography for non-destructive counterfeit detection," in *Proc. Int. Symp. Testing Failure Anal.*, Nov. 2015, pp. 1–10.
[7] IEEE Standards Association and others, 1149.1–2001—IEEE Standard Test Access Port and Boundary Scan Architecture, IEEE Standards no. 1149.1–2001, 2001.
[8] IJTAG - Internal Joint Test Action Group, 1687-2014—IEEE Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device, IEEE Computer Society Standard no. 1687-2014.
[9] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Silicon physical random functions," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2002, pp. 148–160.
[10] G. Suh and S. Devadas, "Physical Unclonable Functions for device authentication and secret key generation," in *Proc. ACM/IEEE Des. Autom. Conf.*, 2007, pp. 9–14.
[11] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, *FPGA Intrinsic PUFs and Their use for IP Protection*. Berlin, Germany: Springer, 2007.
[12] M. Miller, J. Meraglia, and J. Hayward, "Traceability in the age of globalization: A proposal for a marking protocol to assure authenticity of electronic parts," in *Proc. SAE Aerosp. Electron. Avionics Syst. Conf.*, Oct. 2012, [Online]. Available: https://saemobilus.sae.org/content/2012-01-2104
[13] U. Guin, K. Huang, D. DiMase, J. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain," *Proc. IEEE*, vol. 102, no. 8, pp. 1207–1228, Aug. 2014.
[14] Semiconductor Industry Association (SIA), "Public comments - DNA authentication marking on items in FSC5962," Nov. 2012.
[15] J. Dworak, Z. Conroy, A. Crouch, and J. Potter, "Board security enhancement using new locking SIB-based architectures," in *Proc. IEEE Int. Test Conf.*, Oct. 2014, pp. 1–10.
[16] F. Zhang, H. Wang, K. Leach, and A. Stavrou, "A framework to secure peripherals at runtime," in *Computer Security-ESORICS*. Berlin, Germany: Springer, 2014, pp. 219–238.
[17] M. LeMay and C. Gunter, "Cumulative attestation kernels for embedded systems," *IEEE Trans. Smart Grid*, vol. 3, no. 2, pp. 744–760, Jun. 2012.
[18] D. Schellekens, P. Tuyls, and B. Preneel, "Embedded trusted computing with authenticated non-volatile memory," in *Trusted Computing-Challenges and Applications*. Berlin, Germany: Springer, 2008, pp. 60–74.
[19] J. Maskiewicz, B. Ellis, J. Mouradian, and H. Shacham, "Mouse trap: Exploiting firmware updates in USB peripherals," in *Proc. 8th USENIX Conf. Offensive Technol.*, 2014, pp. 12–12.
[20] Y. Li, J. M. McCune, and A. Perrig, "Viper: Verifying the integrity of peripherals' firmware," in *Proc. 18th ACM Conf. Comput. Commun. Secur.*, 2011, pp. 3–16.
[21] D. Morais, J. Lange, D. R. Simon, L. T. Chen, and J. D. Benaloh, "Use of hashing in a secure boot loader," US Patent 6,907,522, Jun. 14, 2005.
[22] D. Peck and D. Peterson, "Leveraging ethernet card vulnerabilities in field devices," in *Proc. SCADA Secur. Scientific Symp.*, 2009, pp. 1–19.
[23] A. M. Garcia Jr, "Firmware modification analysis in programmable logic controllers," DTIC Document, Fort Belvoir, VA, USA, Tech. Rep. AFIT-ENG-14-M-32, 2014.
[24] K. Chen, "Reversing and exploiting an apple firmware update," presented at the *Black Hat*, Las Vegas, NV, USA, 2009.
[25] S. Hanna, R. Rolles, A. Molina-Markham, P. Poosankam, K. Fu, and D. Song, "Take two software updates and see me in the morning: The case for software security evaluations of medical devices," in *Proc. 2nd USENIX Workshop Health Secur. Privacy*, 2011, pp. 1–5.
[26] C. Miller, "Battery firmware hacking," presented at the *Black Hatins*, Las Vegas, NV, USA, pp. 3–4, 2011.
[27] B. Jack, "Jackpotting automated teller machines redux," presented at the *Black Hat*, Las Vegas, NV, USA, 2010.
[28] R. S. Chakraborty, S. Narasimhan, and S. Bhunia, "Embedded software security through key-based control flow obfuscation," in *Security Aspects in Information Technology*. Berlin, Germany: Springer, 2011, pp. 30–44.
[29] T. Morris, "Trusted platform module," in *Encyclopedia of Cryptography and Security*. Berlin, Germany: Springer, 2011, pp. 1332–1335.
[30] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.
[31] A. Miyamoto, N. Homma, T. Aoki, and A. Satoh, "Systematic design of RSA processors based on high-radix montgomery multipliers," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 19, no. 7, pp. 1136–1146, Jul. 2011.
[32] C. Paar and J. Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners*. Berlin, Germany: Springer, 2009.
[33] D. A. Patterson, "Reduced instruction set computers," *Commun. ACM*, vol. 28, no. 1, pp. 8–21, 1985.
[34] C. Price, "MIPS IV instruction set," 1995, [Online]. Available: http://www.weblearn.hs-bremen.de/risse/RST/docs/MIPS/mips-isa.pdf
[35] Intel, "Intel® 64 and ia-32 architectures software developer's manual," *Volume 1: Basic Architecture*, Intel, Santa Clara, CA, USA, 2010.
[36] B. Stamme, "Anti-fuse memory provides robust, secure NVM option," *EE Times*, Jul. 2012, [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1279746
[37] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," 2011. [Online]. Available: http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/iot_ibsg_0411final.pdf

[38] D. E. Holcomb, W. P. Burleson, and K. Fu, "Initial SRAM state as a fingerprint and source of true random numbers for RFID tags," in *Proc. Conf. RFID Secur.*, 2007, [Online]. Available: https://pdfs.semanticscholar.org/987b/3119f356477ee49834098201745ff2666fcf.pdf
[39] B. Sunar, W. Martin, and D. Stinson, "A provably secure true random number generator with built-in tolerance to active attacks," *IEEE Trans. Comput.*, vol. 56, no. 1, pp. 109–119, Jan. 2007.
[40] [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf
[41] B. Kaliski, "Twirl and RSA key size," 2003. [Online]. Available: http://www.emc.com/emc-plus/rsa-labs/historical/twirl-and-rsa-key-size.htm
[42] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach (6th Edition)*, 6th ed. Upper Saddle River, NJ, USA: Pearson, 2012.
[43] [Online]. Available: https://www.us-cert.gov/ncas/alerts

Ujjwal Guin (S'10–M'16) received the BE degree from the Department of Electronics and Telecommunication Engineering, Bengal Engineering and Science University, Howrah, India in 2004, and the MSc degree from the Department of Electrical and Computer Engineering, Temple University, Philadelphia, PA, in 2010. He received the PhD degree from the Electrical and Computer Engineering Department, University of Connecticut, in 2016. He is currently an assistant professor in the Electrical and Computer Engineering Department of Auburn University, Auburn, AL. He has developed several on-chip structures and techniques to improve the security, trustworthiness, and reliability of integrated circuits. His current research interests include hardware security & trust, supply chain security, cybersecurity, and VLSI design & test. He is a co-author of the book *Counterfeit Integrated Circuits- Detection and Avoidance*. He has authored several journal articles and refereed conference papers. He received the Best Student Paper Award from the IEEE North Atlantic Test Workshop NATW in 2013. He is an active participant in the SAE International's G-19A Test Laboratory Standards Development Committee.

Swarup Bhunia (S'01–M'05–SM'08) received the BE (Hons.) degree from Jadavpur University, Kolkata, and the MTech degree from the Indian Institute of Technology (IIT), Kharagpur. He received the PhD degree from Purdue University, in 2005. Currently he is a professor of electrical and computer engineering, University of Florida. Earlier he was the T. and A. Schroeder associate professor of electrical engineering and computer science with Case Western Reserve University, Cleveland, Ohio. He has more than fifteen years of research and development experience with more than 200 publications in peer-reviewed journals and premier conferences in the area of integrated circuit and system design, computer-aided design tools and test techniques. His research interests include hardware security and trust, adaptive nanocomputing with emerging technologies, computing at extreme, and implantable/wearable microsystems. He has worked in the semiconductor industry on synthesis, verification, and low power design for about three years. He received the IBM Faculty Award, National Science Foundation (NSF) career development award, Semiconductor Research Corporation (SRC) technical excellence award as a team member, several best paper awards and best paper nominations, and SRC Inventor Recognition Award. He has been serving as an associate editor of the *IEEE Transactions on CAD*, the *IEEE Transactions on Multi-Scale Computing Systems*, the *ACM Journal of Emerging Technologies*, and the *Journal of Low Power Electronics*; served as guest editor of the *IEEE Design & Test of Computers* (2010, 2013) and the *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2014). He has served as co-program chair of IEEE IMS3TW 2011, IEEE NANOARCH 2013, IEEE VDAT 2014, and IEEE HOST 2015, and in the program committee of many IEEE/ACM conferences. He is a senior member of IEEE.

Domenic Forte (S'09–M'13) received the BS degree in electrical engineering from Manhattan College, Riverdale, New York, in 2006, and the MS and PhD degrees in electrical engineering from the University of Maryland, College Park, Maryland, in 2010 and 2013, respectively. From 2013 to 2015, he was an assistant professor in the Department of Electrical and Computer Engineering, University of Connecticut, in Storrs, CT. He is currently an assistant professor in the Electrical and Computer Engineering Department, University of Florida, Gainesville, Florida, where he has been since July 2015. His research is primarily focused on the domain of hardware security and includes investigation of hardware security primitives, hardware Trojan detection and prevention, security of the electronics supply chain, and anti-reverse engineering. He has served on the program committees of several workshops and conferences in addition to serving as session chair in many technical events. He is a co-author of the book *Counterfeit Integrated Circuits- Detection and Avoidance*. He is a Guest Editor of *the IEEE Computer* 2016 Special Issue on "Supply Chain Security for Cyber-Infrastructure." He received the Young Investigator Program (YIP) award from Army Research Office (ARO) in 2016, the Northrop Grumman Fellowship in 2012, and the George Corcoran Memorial Outstanding Teaching Award by the Electrical and Computer Engineering Department at University of Maryland in 2008. His work has been recognized through several best paper awards and nominations, including AHS 2011, DAC 2012, HOST 2015, and HOST 2016. He is a member of the IEEE.

Mark M. Tehranipoor (S'02–M'04–SM'07) received the PhD degree from the University of Texas, Dallas, in 2004. He is currently the Intel Charles E. Young Preeminence Endowed professor in Cybersecurity with the University of Florida. His current research projects include: hardware security and trust, supply chain security, VLSI design, test and reliability. He has published More than 300 journal articles and refereed conference papers and has given more than 150 invited talks and keynote addresses. He has published six books and eleven book chapters. He received the several best paper awards as well as the 2008 IEEE Computer Society (CS) Meritorious Service Award, the 2012 IEEE CS Outstanding Contribution, the 2009 NSF CAREER Award, and the 2014 MURI award. He serves on the program committee of more than a dozen of leading conferences and workshops. He served as program chair of the 2007 IEEE Defect-Based Testing (DBT) workshop, program chair of the 2008 IEEE Defect and Data Driven Testing (D3T) workshop, co-program chair of the 2008 International Symposium on Defect and Fault Tolerance in VLSI Systems (DFTS), general chair for D3T-2009 and DFTS-2009, and vice-general chair for NATW-2011. He co-founded the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST) and served as HOST-2008 and HOST-2009 general chair. He is currently serving as an associate editor for the *Journal of Electronic Testing: Theory and Applications*, the *Journal of Low Power Electronics*, the *IEEE Transactions on VLSI Systems,* and the *ACM Transactions on Design Automation of Electronic Systems*. Prior to joining UF, He served as the founding director for CHASE and CSI centers at the University of Connecticut. He is currently serving as co-director for Florida Institute for Cybersecurity Research (FICS). He is a senior member of the IEEE, a Golden Core Member of the IEEE, and Member of the ACM and the ACM SIGDA.