Fingerprint-Based Detection and Diagnosis of Malicious Programs in Hardware

Bao Liu, *Senior Member, IEEE*, and Ravi Sandhu, *Fellow, IEEE*

***Abstract*—In today's Integrated Circuit industry, a foundry, an Intellectual Property provider, a design house, or a Computer Aided Design vendor may install a hardware Trojan on a chip which executes a malicious program such as one providing an information leaking back door. In this paper, we propose a fingerprint-based method to detect any malicious program in hardware. We propose a tamper-evident architecture (TEA) which samples runtime signals in a hardware system during the performance of a computation, and generates a cryptographic hash-based fingerprint that uniquely identifies a sequence of sampled signals. A hardware Trojan cannot tamper with any sampled signal without leaving tamper evidence such as a missing or incorrect fingerprint. We further verify fingerprints off-chip such that a hardware Trojan cannot tamper with the verification process. As a case study, we detect hardware-based code injection attacks in a SPARC V8 architecture LEON2 processor. Based on a lightweight block cipher called PRESENT, a TEA requires only a 4.5% area increase, while avoiding being detected by the TEA increases the area of a code injection hardware Trojan with a 1 KB ROM from 2.5% to 36.1% of a LEON2 processor. Such a low cost further enables more advanced tamper diagnosis techniques based on a concurrent generation of multiple fingerprints.**

***Index Terms*—Security, integrated circuits, built-in self-test.**

Acronyms and Abbreviations

AES	advanced encryption standard
ASIC	application-specific integrated circuit
BIST	built-in self-test
CAD	computer-aided design
DMR	dual-module redundancy
EDC	error-detecting code
EDCC	error-detecting and correcting code
FPGA	field-programmable gate array
FSM	finite-state machine
IC	integrated circuit
IP	intellectual property
LFSR	linear feedback shift register
NBTI	negative biased temperature instability
PMOS	p-type metal-oxide semiconductor
RMT	redundant multi-threading
ROM	read-only memory
RTL	register-transfer level
SRAM	static random access memory
SIS	structure integrity checking
TEA	tamper-evident architecture
XOR	exclusive OR

Notations

$M = (m_1, m_2, \ldots, m_n)$	n-bit message
$f(M)$	Rabin fingerprint of M
$\mathrm{concat}(A, B)$	concatenation of A and B

I. Introduction

HARDWARE is the foundation of any security system providing the root of security and trust. In recent years, there has been a growing trend of migrating security solutions down to the hardware level [37], [38], [44], [62], [63], [66]. However, hardware is not free of security threats. An adversary may extract confidential data, a cryptographic key, or intellectual property from a hardware device through testing [2], [73], side channel analysis [26], [33], [34], [39], [71], or reverse engineering [64].

Further, an adversary involved in an Integrated Circuit (IC) design and manufacturing process such as a designer, an Intellectual Property (IP) provider, a Computer-Aided Design (CAD) tool vendor, or a foundry may tamper with an IC chip by installing a Trojan horse component such as a logic bomb that compromises computation integrity, or an information leaking back door that compromises data confidentiality [23].

Testing [2], [73], side channel analysis [26], [33], [34], [39], [71], and online monitoring [68] techniques have been proposed to detect such hardware Trojans. However, hardware Trojans can be very difficult to detect. First, a hardware Trojan can be very difficult to activate. For example, a hardware Trojan that is only triggered by an IC aging sensor may not be activated in manufacturing tests without destroying the chip. Second, a hardware Trojan may leave little trace to detect. For example, a back door may leak information by a side channel without affecting authentic computation results. Further, hardware Trojans are unknowns. Without knowledge about the attack schemes, it is difficult to defend against all possible attacks. Lastly, a supply chain adversary may gain knowledge of an IC design, including any

Manuscript received November 03, 2013; revised August 26, 2014; accepted November 17, 2014. Associate Editor: S. Shieh.

The authors are with the Institute for Cyber Security, the University of Texas at San Antonio, San Antonio, TX 78249 USA (e-mail: bao.liu@utsa.edu; ravi.sandhu@utsa.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2015.2430471

tamper detection scheme implemented on the chip, and invalidate it.

We propose to overcome these difficulties and detect any malicious program launched by a hardware Trojan as follows. We leverage the existing online monitor or concurrent checking techniques [18], [45]. Alternative to testing, such techniques do not require activation of a hardware Trojan. They check the internal states of a program besides the final results. There is no need to have knowledge about the possible hardware Trojans. However, the traditional online monitor or concurrent checking techniques target detection of runtime errors caused by the physical world such as due to radiation or aging, while an adversary may tamper with the system and avoid being detected by 1) generating correct check bits, or 2) invalidating the checking mechanism. We fix such vulnerabilities as follows. We propose a tamper-evident architecture (TEA) which samples runtime signals in a hardware system during the performance of a computation, and generates a cryptographic hash-based fingerprint which uniquely identifies a sequence of sampled signals. A hardware Trojan cannot tamper with any sampled signal that contributes to a cryptographic hash-based fingerprint without leaving evidence such as a missing or incorrect fingerprint. Further, we verify such a fingerprint off-chip, for example, by comparing with a pre-computed fingerprint, or re-computing the fingerprint by simulation or emulation. A hardware Trojan cannot tamper with such an off-chip verification process.

As a case study, we present an application of this technique which detects hardware-based code injection attacks in a SPARC V8 architecture LEON2 processor [17]. Our logic synthesis results based on the 45 nm Nangate open cell library show that, based on a lightweight block cipher PRESENT, a TEA requires only a 4.5% layout area increase for a LEON2 processor; while avoiding being detected by the TEA increases the area of a code injection hardware Trojan with 1 KB ROM from 2.5% to 36.1% of a LEON2 processor.

The rest of the paper is organized as follows. We give an overview on the hardware security problem and related techniques in Section II, before presenting the proposed fingerprint-based tamper detection technique in Section III. We present our case study on detecting hardware-based code injection attacks in a LEON2 processor in Section IV. We further extend the proposed technique for tamper diagnosis in Section V, and conclude in Section VI.

II. Background

A. Hardware Data Confidentiality

With physical access to a hardware device, an adversary may apply a number of techniques to extract confidential data and even cryptographic keys. Testing is a powerful tool to break cryptography algorithms [2], [73], and to extract sensitive information, e.g., from memory. Side-channel analysis techniques extract critical information by differential power analysis [33], [39], timing analysis [26], [34], [71], or fault injection [6], [8].

A number of techniques are available to prevent such information leaks. Testing is protected by encoding, lock and key [35], or checking the signature of test vectors to guarantee the test vectors are authentic [21]. Including additional circuitry prevents power analysis attacks (by inducing noise [33] or hiding supply variation [55]), timing analysis attacks (by reducing the performance difference or increasing performance uncertainty [34]), and fault injection attacks (by concurrent checking [31], [32]).

B. Hardware Design Integrity

Other than data confidentiality, security-providing hardware further needs to ensure hardware design integrity. Due to lack of security mechanisms, in today's global IC industry, an IP provider, an IC design house, a CAD company, or a foundry can easily tamper with a hardware device, for example, by installing a Trojan horse component that corrupts the authentic computation, bypasses security checks, or creates a back door for information leaks [23]. Preventing this tampering from happening is the supply chain risk management problem, which has been identified as a national priority in the recently released Comprehensive National Cyber Security Initiative [48].

Many of the existing hardware integrity-ensuring techniques are based on ensuring data integrity. For example, a FPGA design can be protected by encrypting and hashing its configuration bit stream [3]. In computer architecture, static code integrity verification protects instructions and data in memory, e.g., by encrypting and hashing in writing, and decrypting and hash matching in reading [14], [37], [38], [62], [63]. Encrypting and hashing register file contents further prevents leakage of decrypted instructions and data at system interruptions [37].

The dominant hardware IP protection technique is watermarking [1]. IP watermarking secretly conveys the information on content ownership and IP rights. Compared with steganography, IP watermarking further requires the property of robustness, i.e., being infeasible to remove or make useless without destroying the object at the same time. Hardware IP watermarking techniques can be categorized as static, and dynamic [1], [11], [24]. In static hardware IP watermarking, the watermark is detected without running the IP. The dominant technique is to include ownership-indicating constraints in a design optimization process [53], such as logic optimization [28], or place and route [29]. In dynamic hardware IP watermarking, the watermark can only be detected by running the IP. For example, watermarks can be embedded in dont-care logic values, e.g., under logic inputs that are never applied in operation [74]. A watermarked FSM gives the encrypted ownership information if the correct key sequence is applied [65], or exhibits a unique property if an encrypted ownership message is applied [49].

These hardware IP watermarking techniques do not lead to hardware IP tamper-proofing. Compared with watermarking, tamper-proofing further requires the watermarks to be verified effectively in runtime [51]. Static hardware IP watermarks are difficult to verify, e.g., they require reverse engineering to retrieve logic or physical design properties. Dynamic hardware IP watermarks are verified by applying special inputs. They do not verify system runtime behavior under all possible inputs.

C. Related Techniques for Computation Integrity

1) Concurrent Checking: To verify system runtime behavior, a concurrent checking system generates information bits, and

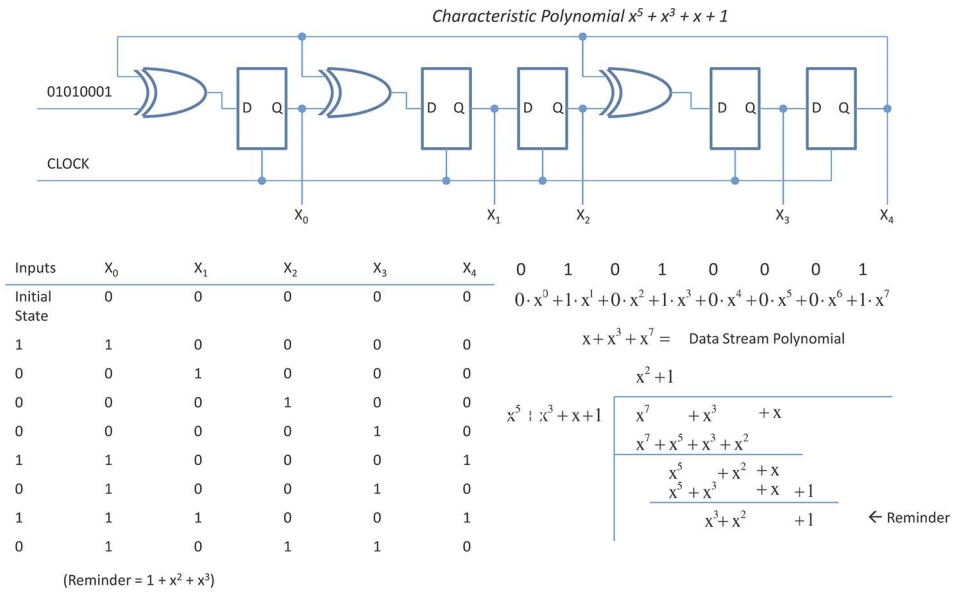

Fig. 1. Modular linear feedback shift register (LFSR) as a response compactor in built-in self test (BIST) [10].

check bits, for example, in an error-detecting code [18], [45]. The simplest check bits can be parity bits, or duplicates of the information bits in a dual-module redundancy (DMR) scheme. Checking the consistency between the information bits and the check bits can detect runtime errors such as soft errors or an adversary tampering (e.g., triggered by a timer), which cannot be detected by testing.

At the architecture level, fault tolerant processor design includes a variety of system-level redundant execution and concurrent checking techniques [45]. Lock-stepping schemes compare internal states (e.g., program control flow [42], [46], [58], hardware control signals [13], memory access [47], and reasonableness of results [43], [59]) in each cycle with duplicated program runs in a watchdog co-processor. Non-lock-stepping schemes such as Redundant Multi-Threading (RMT) compare only the outputs of committed instructions [4], [5], [19], [56], [67]. Error-detecting and correcting code (EDCC)-based hardware assertion techniques lead to more hardware-efficient fault tolerant processors compared with lock-stepping or RMT [60], [61], [72].

A specific category of concurrent checking techniques is the control flow checking techniques which verify dynamic code integrity. In structure integrity checking (SIS), each branch-free code sequence is assigned a random number as its signature, while a watchdog co-processor runs a simplified code, which keeps the control structure, but includes only receive signature and check signature instructions [42]. In basic path signature analysis, each branch-free code sequence (node) has a signature which comes from, e.g., a parity, checksum, or linear feedback shift register (LFSR) of the instruction words. The watchdog co-processor computes a signature dynamically at runtime, and compares with the signature computed statically at assembly time [46]. In generalized path signature analysis, each set of paths (including branches, sharing the same starting node and the same ending node) have a common signature with included justifying signatures. The watchdog co-processor compares such signatures computed dynamically and statically [46]. In branch address hashing, a branch target address is computed at runtime based on a signature computed at runtime. An incorrect control flow results in an incorrect signature, and an incorrect branch target address, such that the subsequent node has an incorrect signature, which will be detected [58].

2) Built-In Self Test: A similar category of techniques is the built-in self test (BIST) category [10], [16]. A typical BIST scheme includes a pattern generator, a response compactor, and a comparator. BIST leads to significant test cost reduction. However, it may not be effective in detecting adversary tampering. For example, a Trojan may not alter the computation result, or a Trojan may only be triggered by some input patterns that are not included in the BIST scheme. Nevertheless, we observe that the response compaction techniques in BIST are very efficient in reducing the size of data to verify while monitoring a hardware system over a long period of time.

Linear feedback shift registers (LFSRs) provide efficient implementation for pseudo-random number generation and response compaction in BIST. A LFSR consists of D flip-flops, and linear exclusive-OR (XOR) gates. For pseudo-random number generation, a LFSR of n D flip-flops will cycle through $2^n - 1$ finite states. For response compaction, a LFSR takes as input a bit stream of output data from the circuit-under-test. Taking this bit stream as a descending order coefficient polynomial, a LFSR performs polynomial division of this bit stream polynomial by the characteristic polynomial of the LFSR. The final state of a modular LFSR is the remainder of the polynomial division (Fig. 1) [10].

We observe that the response compaction techniques in BIST are fingerprinting techniques. Specifically, modular LFSR-based response compaction produces Rabin fingerprints [54].

3) Fingerprinting: Fingerprints are short tags for larger objects. They have the property that if two fingerprints are different, then the corresponding objects are certainly different,

and there is only a small probability that two different objects have the same fingerprints. The latter event is called a collision. Such a model and requirements are similar to those for universal hashing. However, the emphasis and the relation between the number n of objects and the fingerprint length k are different. For hashing, we are interested in bounding the number of collisions, and typically n is much larger than 2^k; for fingerprinting, we want to avoid collisions altogether, and we must take $n \leq 2^k$ [9].

For a given n-bit message $M = (m_1, m_2, \ldots, m_n)$, we represent it by a polynomial $M(x)$ of degree $n-1$ over finite field $GF(2^n)$.

$$M(x) = m_1 x^{n-1} + m_2 x^{n-2} + \ldots + m_n \quad (1)$$

A Rabin fingerprint $f(M)$ is the remainder after polynomial division of $M(x)$ by an irreducible polynomial $P(x)$.

$$f(M) = M(x) \bmod P(x) \quad (2)$$

A Rabin fingerprint can be computed in linear time, and efficiently implemented by a LFSR [9]. However, Rabin's algorithm is not secure against malicious attacks. An adversary can easily obtain the key (e.g., the seed value in a LFSR), and modify a message without changing its fingerprint. Cryptographic hash functions generally serve as higher quality fingerprint functions. However, they are much more costly, and lack proven guarantees on collision probability.

Fingerprints are widely used for deduplication in communication. For example, before re-loading a large file, a web browser first fetches the fingerprint of the file, compares it to that of an old copy, determines if the large file has any update, and only re-loads the file if an update exists [15].

III. Fingerprint-Based Detection of Malicious Programs in Hardware

A. Supply Chain Attack Model

A supply chain adversary is an insider who is involved in the design and the manufacturing of a hardware device. His tampering capability is based on his role in the supply chain, specifically his read and write permission in the design and the manufacturing process of a specific device. An IP provider or a designer for a specific module may have limited access to the design, while a foundry or a chip-level integration designer has access to the whole device design. The general lack of access control in today's supply chain further facilitates an adversary to gain knowledge of a design and launch attacks. Besides, based on his role in the supply chain, a supply chain adversary may gain further knowledge of a design by probing, testing, side-channel analysis, or reverse engineering. As a result, a supply chain adversary may have read and write permission to the whole design of a particular device.

A supply chain adversary may install a hardware Trojan that is triggered at system runtime [12], [25], [69]. A hardware Trojan can be a logic bomb that compromises hardware computation integrity by altering the authentic computation result, or a back door that compromises hardware data confidentiality by leaking out secrets or confidential information [12], [30], [57], [70]. A back door may launch an attack by performing more (e.g., in leaking out information) or less (e.g., in bypassing existing security checks) than expected, while keeping the authentic computation result intact. Because a logic bomb can be detected by online monitoring or concurrent checking, we focus on detecting back doors in this paper.

Specifically, we consider the security threat that a supply chain adversary (a designer, an IP provider, a CAD vendor, or a foundry) may install a hardware Trojan on an IC which provides an information leaking back door, once activated. Such a hardware Trojan may not be present in the design (for example installed at a foundry or hidden in an IP) such that it cannot be detected by design verification or formal verification. Further, it is very difficult to detect such a hardware Trojan by testing because 1) it is very difficult to activate it (for example, a hardware Trojan may be triggered by an IC aging sensor), and 2) there may be little trace even if the hardware Trojan is activated; for example, the hardware Trojan may leak information through a side channel or by steganography without affecting the authentic computation, without degrading the system performance, and without causing any noticeable power consumption increase.

However, such a hardware Trojan needs to have its own computation, for example, monitoring the authentic computation and finding data of interest from the intermediate computation results. Such a Trojan computation may not alter the authentic computation results. However, it may tamper with some of the runtime signals of a hardware system during the performance of a computation as long as it utilizes some of the existing hardware resources. A hardware Trojan that does not utilize any existing hardware resource has a larger footprint which makes it easier to be detected.

B. Fingerprint-Based Detection of Malicious Programs in Hardware

Our approach is to monitor the runtime signals of a hardware system during the performance of a computation to detect any Trojan computation, and protect the intermediate data during a computation (while before and after the computation, the data can be protected by encryption). We further compact the runtime signals into a fingerprint, and verify the fingerprint off-chip for computation integrity and tamper detection.

The proposed fingerprint-based malicious program detection scheme includes three components as follows.

1) Signal Sampling: We first sample a group of runtime signals in a hardware system. We will generate a fingerprint based on these signals, and verify the fingerprint. As a result, any Trojan computation which tampers with any of these sampled signals will be detected.

A Trojan computation tampers with the inputs and outputs of a module if the module is utilized for Trojan computation. By sampling any of the module input and output signals, we will detect any Trojan computation which utilizes this module. For example, by sampling the input and output signals of an instruction decoder, we will detect any Trojan computation which sends Trojan instructions to this instruction decoder. By sampling the inputs and outputs of an adder, we will detect any Trojan computation which utilizes this adder. By sampling the

memory access signals, we will detect any Trojan memory access (given that the authentic computation and memory access patterns are pre-determined, which we will elaborate on in the section on verification).

2) Fingerprint Generation: We generate a fingerprint which uniquely identifies a group of sampled signals.

We observe that the LFSR-based Rabin fingerprinting technique that is commonly applied in BIST for response compaction does not achieve tamper resistance. A supply chain adversary may generate a fingerprint for his tampered design given a Rabin fingerprinting scheme.

The Rabin fingerprint of the concatenation of two messages A and B has the following properties [9].

$$f(\text{concat}(A,B)) = f(\text{concat}(f(A),B)) \tag{3}$$

$$\begin{aligned} f(\text{concat}(A,B)) &= A(x) * x^l + B(x) \text{mod} P(x) \\ &= f(f(A) * f(x^l)) + f(B) \end{aligned} \tag{4}$$

Note that l is the length of B. Suppose that an adversary wants to insert a message T between messages A and B. Based on (3), it is guaranteed that $f(\text{concat}(A,T,B)) = f(\text{concat}(A,B))$ if $f(\text{concat}(A,T)) = f(A)$. Based on (4), an adversary can achieve the identical fingerprint after inserting a message T if $f(T) = f(A) + f(f(A) * f(x^l))$, where l is the length of message T. For a short message, there can be $T = f(T)$. For a long message, an adversary can concatenate a short message T' to an inserted message T to correct the fingerprint, e.g., $T' = f(T') = f(\text{concat}(A,T)) + f(f(\text{concat}(A,T)) * f(x^{l'}))$, where l' is the length of message T'.

Cryptographic hash function-based fingerprinting prevents such attacks. Cryptographic hash functions include dedicated hash functions such as MD5 and SHA-1, and block cipher such as AES-based hash functions. Such cryptographic hash functions have preimage resistance or one-wayness, second preimage resistance or weak collision resistance, and collision resistance. Preimage resistance or one-wayness ensures that, given a hash value z, it must be computationally infeasible to find an input message x such that $z = h(x)$. Second preimage resistance or weak collision resistance ensures that, given a message x_1 and its hash value z_1, it must be computationally infeasible to find a different message x_2 with an equal hash value $z_1 = h(x_1) = h(x_2) = z_2$. Collision resistance ensures that it must be computationally infeasible to find two different messages x_1 and x_2 with equal hash values $z_1 = h(x_1) = h(x_2) = z_2$ [50]. As a result, given a cryptographic hash function-based fingerprinting scheme, a supply chain adversary cannot design a tampering scheme that alters the response without altering the fingerprint.

3) Fingerprint Verification: Given an authentic computation which we know, and many possible tampered computations which we do not know, we ensure the integrity and authenticity of a computation by generating a fingerprint based on sampled runtime signals, and verifying the fingerprint by comparing it with a pre-computed one, or by repeating the same computation in a separate system. For such a verification to be successful, we have a few conditions. First, the authentic computation needs to be pre-determined, i.e., the instruction sequence is pre-determined, and the input data are pre-determined. For certain simple programs such as a boot procedure, this condition is true. For a complex program, we can partition it into simple instruction sequences, each being free of branching, as in control flow checking [42], [46], [58]. Second, any tampered computation produces a different fingerprint. This condition can be achieved by choosing a fingerprint generation method of satisfiable collision resistance, and choosing a group of sampled runtime signals which a Trojan computation tampers with. Third, the verification cost is acceptable. This condition is not a problem for any repeated computation such as boot. A complex program can be partitioned into simple instruction sequences, and the sampled runtime signals can be selected such that they are expected to be the same for repeated computation, for example, as in control flow checking [42], [46], [58].

We implement the fingerprint verification scheme off the chip such that a hardware Trojan cannot tamper with the verification procedure. For example, to detect any Trojan installed by an ASIC foundry, we compute a fingerprint by RTL simulation or FPGA emulation. To detect any Trojan installed by an IP provider, we compute a fingerprint based on a function model of the IP or another IP of the same function.

A fingerprint may be verified concurrently to the computation under verification, or stored and verified at a later time.

C. Comparison to Concurrent Checking and BIST

The proposed fingerprint-based tamper detection method differs from the existing concurrent checking techniques as follows.

1) Tamper resistance—The existing concurrent checking techniques do not achieve tamper resistance. A hardware Trojan may tamper with information signals and check signals together, or invalidate any on-chip concurrent checking mechanism to avoid being detected.
2) Tamper evidence—Concurrent checking discards runtime signals instantly. In the proposed technique, a fingerprint may be stored for later verification, and multiple fingerprints may provide tamper diagnosis.
3) Efficiency—Compacting a vast amount of sampled signals into a fingerprint provides efficiency in storage or instant verification. In comparison, concurrent checking either checks each sample signal individually, e.g., in a DMR scheme [18], or checks a group of sample signals, e.g., based on an error-detecting code (EDC) [40], [41], or checks the signature of a code snippet in control flow checking [43]. Concurrent checking is performed in run time (on-the-fly), either in lock-stepping which checks internal states, or non-lock-stepping which checks only the final output of a program [45]. There are further synchronization and communication bandwidth problems between an on-chip function system and an off-chip checking system in lock-stepping concurrent checking.

Fingerprint-based tamper detection differs from BIST as follows.

1) BIST detects only any output difference for the given input patterns. Fingerprint-based tamper detection covers any sample signal for all input patterns.

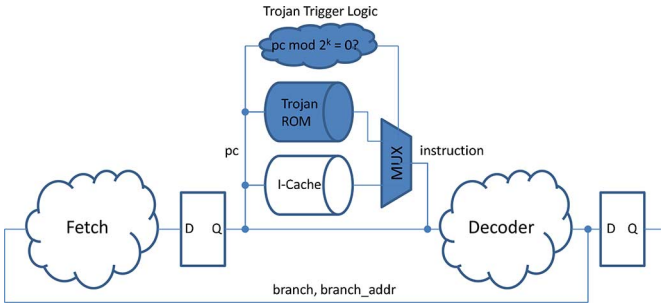

Fig. 2. A code injection hardware Trojan including a Trojan ROM, multiplexers, and trigger logic.

2) BIST achieves LFSR-based response compaction or Rabin fingerprinting. Fingerprint-based tamper detection achieves fingerprints of a higher quality based on cryptographic hash functions that are resistant to adversary tampering.

IV. Case Study

As a case study, we demonstrate application of the proposed fingerprint-based method to detect hardware-based code injection attacks in a processor.

A. Code Injection Hardware Trojan

Code injection is a major mechanism in software-based attacks. Code injection further provides increased efficiency and capacity for a hardware Trojan. For example, a hardware Trojan may launch a deputy attack (i.e., use the benign microprocessor in a malicious way) [20], and tamper with data in memory that are encrypted and authenticated such that only the authentic microprocessor has access to it. It has been shown that a hardware Trojan may tamper with the instruction memory, and direct a processor to execute malicious instructions, e.g., by inserting a JUMP instruction [27], or tampering a procedure return address stack through an overflown buffer [36]. Here we present a hardware Trojan which injects Trojan code from a Trojan ROM to an authentic instruction decoder.

A code injection Trojan can be very small. For example, it may only need to include a Trojan ROM containing the Trojan instructions, a few multiplexers at the instruction fetch unit inputs, and a trigger logic module (Fig. 2). The Trojan trigger logic module monitors the program count (pc) in the instruction fetch unit. When the trigger condition is met, for example, the lower n bits of the next program count are all zeros, the Trojan multiplexers direct the instruction fetch unit to fetch instructions from the Trojan ROM other than from the instruction cache. Because the Trojan ROM is very small, it can be addressed by the lower n bits of the program count. The Trojan instruction sequence starts by saving the program count and the other processor internal states, and ends by restoring the processor internal states including the program count. When the low n bits of the program count equal the address of the last Trojan instruction (that restores the program count), the Trojan multiplexers direct the instruction fetch unit to fetch instructions from the instruction cache. This action resumes the authentic operation.

To evade manufacturing tests, we further include an IC aging sensor in the Trojan trigger logic such that the Trojan can only be triggered after the IC is sufficiently aged. Such an IC aging sensor can be in the form of a 6T SRAM cell which has a minimum footprint, and consumes no static power (Fig. 3) [52]. The only difference between this sensor and a regular 6T SRAM cell is that the PMOS transistor P1 is sized up a bit. This sensor works in two modes: polling, and tracking. In polling, or when the sensor is powered up with the pass transistors turned off, the PMOS transistors fight to determine the sensor output. Initially, as P1 is stronger, the bitline outputs logic 1. In tracking, or when the sensor is powered on, as the PMOS transistor P1 is constantly subject to a negative gate-to-source voltage V_{gs}, the Negative Biased Temperature Instability (NBTI) effect increases the threshold voltage of P1. After the system is aged to the point that P1 is weaker than P2, a polling will flip the sensor output, and enable Trojan activation.

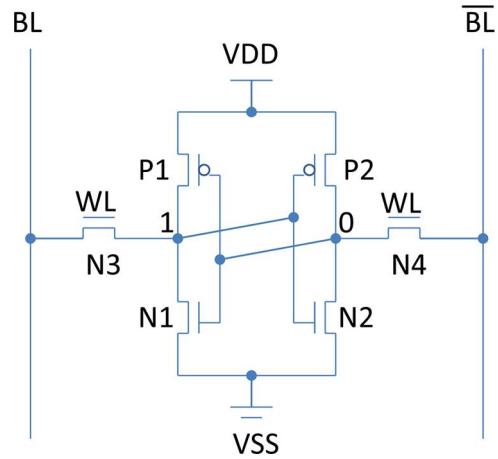

Fig. 3. A SRAM cell-based NBTI sensor.

Such a Trojan cannot be detected by a static code integrity check, because the Trojan instructions are not in the memory. The Trojan cannot be detected by non-lock-stepping concurrent checking [4], [5], [19], [56], [67] because the authentic computation results are intact. But the Trojan may be detected by lock-stepping concurrent checking [13], [42], [43], [46], [47], [58], [59]. However, if a lock-stepping concurrent checking module resides on the same chip as the function system, a supply chain adversary such as a foundry or a chip-integration designer can easily tamper with the checking mechanism. If a lock-stepping concurrent checking mechanism resides on a different chip, it would be difficult to achieve synchronization, and only a limited number of signals can be monitored. As a result, a supply chain adversary may tamper with the system while keeping the sampled signals intact.

B. Tamper-Evident Architecture

Our fingerprint-based tamper detection is based on a tamper-evident architecture (TEA) that generates fingerprints for sampled signals (Fig. 4). By sampling the instruction fetch unit inputs, we can detect Trojans that send Trojan signals such as branch target addresses to the instruction fetch unit. While at a higher cost, a supply chain adversary may create his own Trojan instruction fetch unit that inserts Trojan instructions to the instruction decoder unit. By sampling the instruction decoder unit inputs, we can detect Trojans that send Trojan instructions to the

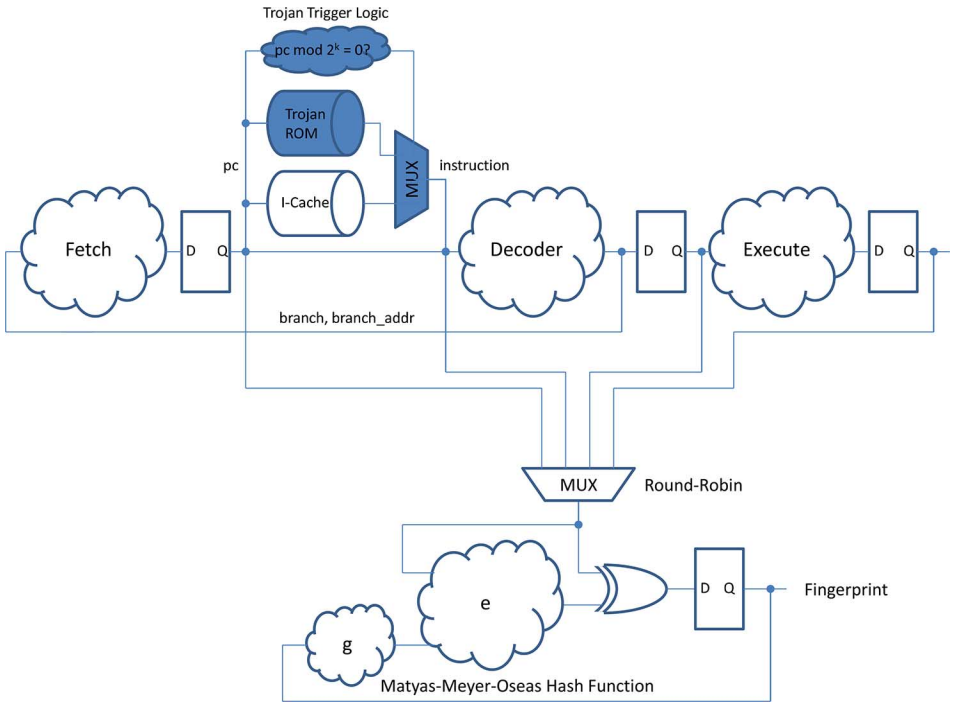

Fig. 4. A code injection hardware Trojan which consists of a Trojan ROM, multiplexers, and trigger logic (colored); and a tamper-evident architecture (TEA) which samples runtime signals in a round-robin scheme based on multiplexers, and computes a fingerprint based on a Matyas-Meyer-Oseas hash function (below the pipeline) in a processor.

decoder unit. While at a higher cost, a supply chain adversary may create his own Trojan instruction decoder unit that sends control signals and data to the function units in the execute stage. To defeat such an attack, we need to further sample the execute stage inputs. While at an even higher cost, a supply chain adversary may create his own Trojan function units. To defeat such an attack, we need to further sample the register files and the memory inputs.

We implement a round-robin scheduling algorithm to sample the signals for a fingerprint generator based on a hash function that accepts fixed-length messages. We sample any specific signal once in every k clock cycles, which guarantees to detect any inserted Trojan instruction sequence that takes more than k clock cycles. A Trojan instruction sequence cannot be too short because it needs to start by saving the processor internal states and end by restoring the processor internal states to keep the authentic computation result intact. We have $k = 5$ in our implementation.

We implement a Matyas-Meyer-Oseas hash function for fingerprint generation (Fig. 4), while several other hash functions such as Davies-Meyer and Miyaguchi-Preneel are equally effective [50]. We divide a message x into blocks x_i of a fixed size such as 128 bits. A block cipher such as AES encrypts each message block x_i, while taking a mapping $g(H_{i-1})$ from the previous output H_{i-1} as the key input to the cipher. If the previous output H_{i-1} and the key input have the same length, $g(.)$ can be the identity mapping. After encryption, we XOR the encrypted message block $e_{g(H_{i-1})}(x_i)$ to the original message block x_i. The function can be expressed as $H_i = e_{g(H_{i-1})}(x_i) \oplus x_i$. The last output value computed is the hash of the whole message x_1, $x_2, \ldots, x_n$, i.e., $H_n = h(x)$ [50].

A particularly strong attack against the TEA and fingerprint-based tamper detection method is for a hardware Trojan to initiate a precise interrupt. A precise interrupt does not affect any program output, but increases the runtime of an on-the-fly program. Similarly, a hardware Trojan may 1) freeze the on-the-fly authentic computation and fingerprint generation process, for example, by clock gating; 2) save the system internal state; 3) perform the Trojan program; 4) restore the system internal state; and 5) resume the on-the-fly authentic computation and fingerprint generation process. Such an attack (as a precise interrupt) does not tamper with the authentic computation result or the fingerprint. However, it leads to significant performance degradation, which makes it easy to be detected, e.g., by checking the program runtime in clock cycles.

C. Fingerprint Generation and Verification

We generate cryptographic hash-based fingerprints by RTL simulation assuming the RTL design is tamper-free while a foundry or an IP provider may install a hardware Trojan which is not present in the RTL design. A simple input-free program such as a boot sequence has a unique fingerprint s. The TEA is expected to output the fingerprint s in a given number t of clock cycles. At the start of a program, we insert a no-op instruction giving in its operand field a check value $v = f(s, t)$, where f is a function that is not on-chip and unknown to any supply chain adversary. At the end of a program run, an off-chip verifier that knows the check function $f(s, t)$ computes its check value $v' = f(s', t')$ based on the actual fingerprint s' and the actual runtime t', and compares v to v'. Any interrupt time needs to be deducted from the program runtime. If a hardware Trojan fakes an interrupt, the interrupt occurrence record provides tamper

evidence. If a hardware Trojan eavesdrops a set of (v, s, t), and launches a replay attack, i.e., by giving v first then s after t cycles to cover a Trojan program run of t cycles, an off-chip verifier can detect this change by checking if the program of fingerprint s has been re-launched.

We partition a more complex program including branching instructions into branch-free instruction sequences as in path signature analysis [46]. We generate a fingerprint s and a check value $v = f(s, t)$ for each branch-free instruction sequence which takes t cycles. To prevent a hardware Trojan from injecting a Trojan branch-free instruction sequence, and covering it with a set of (v, s, t) in a replay attack, we link each branch-free instruction sequence to its two successor branch-free instruction sequences as follows. We insert two no-op instructions at the end of a branch-free instruction sequence carrying the check values for the next branching, and non-branching instruction sequences, respectively. As a result, the program cannot branch to a Trojan instruction sequence.

D. Evaluation

The proposed TEA provides computation integrity verification and intermediate data protection against hardware Trojans or malicious programs.

If a hardware Trojan launches a malicious program which tampers with an existing hardware module from which signals are sampled, then the TEA will produce an incorrect fingerprint, leading to tamper detection. Alternatively, performing all the Trojan program based on dedicated Trojan hardware leads to an increased hardware footprint, and makes the Trojan easier to be detected.

We evaluate a few possible code injection hardware Trojans and the proposed fingerprint-based tamper detection scheme based on a TEA on a five-stage in-order open source SPARC V8 architecture LEON2 processor [17]. We have designed a minimum code injection hardware Trojan including a few multiplexers, a trigger logic network, and a 1 KB Trojan ROM. We have further designed two TEAs which compute signatures of guaranteed preimage and second preimage resistance based on two block ciphers AES and PRESENT, respectively [7], [50]. We achieve logic synthesis of these designs by Synopsys Physical Compiler based on the 45 nm Nangate open source cell library [22]. Table I compares their hardware cost in terms of area, power consumption, and critical path delay. The LEON2 processor is configured to include a five-cycle multiplier, a 35-cycle divider, a floating-point unit, a memory management unit, a PCI interface, and a network unit with no co-processors. Compared with the LEON2 processor, the minimum code injection hardware Trojan with a 1 KB ROM leads to a layout area increase of only 2.5%. In the presence of a TEA, the hardware Trojan cannot utilize any existing hardware resources, and needs to have its own instruction fetch unit, instruction decode unit, function units, memory, and write back logic. This condition leads to a layout area increase of 36.1% for the LEON2 processor, which makes it much easier to detect the hardware Trojan. While an AES-based TEA leads to a 45.1% layout area increase, moving to the lightweight block cipher PRESENT leads to a tamper-evident architecture of less than twice the area of a code injection hardware Trojan with a 1 KB ROM, or only a 4.5% layout area increase for the LEON2 processor.

TABLE I
HARDWARE OVERHEAD OF A LEON2 PROCESSOR, A CODE INJECTION HARDWARE TROJAN INCLUDING A 1KB ROM, ANOTHER CODE INJECTION HARDWARE TROJAN INCLUDING A 1KB ROM AND AN INTEGER UNIT, AND TWO TEAS BASED ON AES AND PRESENT, RESPECTIVELY

	Area (μm^2)	Power (mW)	Delay (ns)
LEON2	4.52×10^4	2.25	7.79
Trojan w/ 1KB ROM	1.12×10^3	2.22×10^{-3}	0.41
Trojan w/ 1KB ROM & IU	1.63×10^4	0.26	6.79
AES-Based TEA	2.04×10^4	0.46	1.92
PRESENT-Based TEA	2.03×10^3	0.19	0.39

V. TAMPER DIAGNOSIS

The proposed fingerprint-based tamper detection method can be extended for tamper diagnosis. For example, we generate multiple fingerprints for different groups of sampled signals concurrently. Our experimental results (Table I) show that this result is possible to achieve even under a tight area constraint based on a lightweight block cipher such as PRESENT.

We may generate fingerprints for each pipeline stage in a processor. For more thorough diagnosis, one may arrange the sequential elements to sample in a 2-D array, and generate a fingerprint for each row and each column. A sequential element of a tampered with signal would be located by the row and the column of an incorrect fingerprint. A more efficient scheme is to generate fingerprints for selected sampling points in a way that is similar to Hamming code construction. For example, for 4 information bits, 3 check bits are generated by respectively taking XOR for bits 1,3,5,7, for bits 2,3,6,7, and for bits 4,5,6,7. In general, for n information bits, $\log(n)$ check bits are needed to construct a single-error-correctable Hamming code. Similarly, we need $\log(n)$ fingerprints for n sampled signals to locate a single tampered with signal.

VI. CONCLUSION

In this paper, we propose a fingerprint-based method to detect malicious programs in hardware. We propose a TEA which generates fingerprints that each uniquely identifies a sequence of runtime signals in a hardware system during the performance of a computation. We generate cryptographic hash-based fingerprints such that a hardware Trojan cannot tamper with any sampled signal without leaving tamper evidence such as a missing or incorrect fingerprint. We verify fingerprints off-chip such that a hardware Trojan cannot tamper with the fingerprint verification process. To avoid being detected by the proposed method, a hardware Trojan cannot launch a malicious program based on any existing hardware module from which signals are sampled.

This result leads to an increased Trojan footprint or degraded system performance, and makes Trojan detection easier. As a case study, we apply this technique to detect hardware-based code injection attacks in a LEON2 processor. Logic synthesis based on the 45 nm Nangate open cell library shows that, based on a lightweight block cipher PRESENT, a TEA requires only a 4.5% area increase for a LEON2 processor; while avoiding being detected by the TEA increases the area of a code injection

hardware Trojan with a 1 KB ROM from 2.5% to 36.1% of a LEON2 processor. The small overhead of fingerprint-based tamper detection further enables concurrent generation of multiple fingerprints for more advanced tamper diagnosis techniques.

REFERENCES

[1] A. T. Abdel-Hamid, S. Tahar, and M. Aboulhamid, "A survey on IP watermarking techniques," *Design Autom. Embed. Syst.*, vol. 9, pp. 211–227, 2004.

[2] M. Agrawal, S. Karmakar, D. Saha, and D. Mukhopadhyay, "Scan based side channel attacks on stream ciphers and their counter-measures," in *Proc. Int. Conf. Cryptology in India (INDOCRYPT)*, 2008, pp. 226–238.

[3] Altera, "Anti-tamper capabilities in FPGA designs," [Online]. Available: http://www.altera.com/literature/wp/wp-01066-anti-tamper-capabilities-fpga.pdf/

[4] T. M. Austin, "DIVA: A reliable substrate for deep submicron microarchitecture design," in *Proc. Annu. Int. Symp. Microarchitecture (MICRO)*, 1999, pp. 196–207.

[5] D. Bernick, B. Bruckert, P. D. Vigna, D. Garcia, R. Jardine, J. Klecka, and J. Smullen, "NonStop advanced architecture," in *Proc. Int. Conf. Dependable Systems and Networks (DSN)*, 2005, pp. 12–21.

[6] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Proc. Annu. Int. Cryptography Conf. Advances in Cryptography*, 1997, pp. 513–527.

[7] A. Bogdanov, L. R. Knudsen, G. Le, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An ultra-lightweight block cipher," *Cryptograph. Hardw. Embed. Syst.*, pp. 450–466, 2007.

[8] D. Boneh, R. A. Demillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in *Proc. Int. Conf. Theory and Application of Cryptographic Techniques (Eurocrypt)*, 1997, pp. 37–51.

[9] A. Z. Broder, "Some applications of Rabins fingerprinting method," in *Sequences II: Methods in Communications, Security, and Computer Science*. New York, NY, USA: Springer-Verlag, 1993, pp. 143–152.

[10] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Norwell, MA, USA: Kluwer, 2000.

[11] A. E. Caldwell, H.-J. Choi, A. B. Kahng, S. Mantik, M. Potkonjak, G. Qu, and J. L. Wong, "Effective iterative techniques for fingerprinting design IP," *IEEE Trans. Comput.-Aided Design*, vol. 23, no. 2, pp. 208–215, 2004.

[12] R. S. Chakraborty, S. Narasimhan, and S. Bhunia, "Hardware Trojans: Threats and emerging solutions," in *Proc. IEEE HLVDT Workshop*, 2009, pp. 166–171.

[13] S. F. Daniels, "A concurrent test technique for standard microprocessors," in *Dig. Papers Compcon Spring 83*, 1983, pp. 389–394.

[14] R. Elbaz, D. Champagne, C. Gebotys, R. B. Lee, N. Potlapally, and L. Torres, "Hardware mechanisms for memory authentication: A survey of existing techniques and engines," in *Transactions on Computational Science IV*. Berlin, Germany: Springer-Verlag, 2009, pp. 1–22.

[15] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, 2000.

[16] R. A. Frohwerk, "Signature analysis: A new digital field service method," *Hewlett-Packard J.*, vol. 28, no. 9, pp. 2–8, 1977.

[17] Aeroflex Gaisler, LEON SPARC V8 Processors [Online]. Available: http://www.gaisler.com/

[18] M. Göessel, V. Ocheretny, E. Sogomonyan, and D. Marienfeld, *New Methods of Concurrent Checking*. New York, NY, USA: Springer, 2008.

[19] M. A. Gomaa, C. Scarbrough, T. N. Vijaykumar, and I. Pomeranz, "Transient fault-recovery for chip multiprocessors," in *Proc. Int. Symp. Computer Architecture*, 2003, pp. 98–109.

[20] N. Hardy, "The confused deputy (or why capabilities might have been invented)," *ACM SIGOPS Operat. Syst. Rev.*, vol. 22, no. 4, pp. 36–38, 1988.

[21] D. Hely, F. Bancel, M.-L. Flottes, and B. Rouzeyre, "Test control for secure scan design," in *Proc. Eur. Test Symp.*, 2005, pp. 190–195.

[22] Silicon Integration Initiative, "Nangate open cell library," [Online]. Available: http://www.si2.org/openeda.si2.org/projects/nangatelib/

[23] C. E. Irvine and K. Levitt, "Trusted hardware: Can it be trustworthy?," in *Proc. ACM/IEEE Design Automation Conf.*, 2007, pp. 1–4.

[24] A. K. Jain, L. Yuan, P. R. Pari, and G. Qu, "Zero overhead watermarking technique for FPGA designs," in *Proc. Great Lakes Symp. VLSI*, 2003, pp. 147–152.

[25] Y. Jin, N. Kupp, and Y. Makris, "Experience in hardware Trojan design and implementation," in *Proc. IEEE Int. Workshop Hardware-Oriented Security and Trust (HOST)*, 2009, pp. 50–57.

[26] Y. Jin and Y. Makris, "Hardware Trojan detection using path delay fingerprint," in *Proc. IEEE Int. Workshop Hardware-Oriented Security and Trust (HOST)*, 2008, pp. 51–57.

[27] Y. Jin, M. Maniatakos, and Y. Makris, "Exposing vulnerabilities of untrusted computing platforms," in *Proc. IEEE Int. Conf. Computer Design*, 2012, pp. 131–134.

[28] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Constraint-based watermarking techniques for design intellectual property protection," *IEEE Trans. Comput.-Aided Design*, vol. 20, no. 10, pp. 1236–1252, 2001.

[29] A. B. Kahng, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Robust IP watermarking methodologies for physical design," in *Proc. ACM/IEEE Design Automation Conf.*, 1998, pp. 782–787.

[30] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware Trojans," *IEEE Comput.*, vol. 43, no. 10, pp. 39–46, Oct. 2010.

[31] R. Karri, K. Wu, and P. Mishra, "Fault-based side-channel cryptanalysis tolerant architecture for Rijndael symmetric block cipher," in *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 2001, pp. 427–435.

[32] R. Karri, K. Wu, P. Mishra, and Y. Kim, "Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers," *IEEE Trans. Comput.-Aided Design*, vol. 21, no. 12, pp. 1509–1517, 2002.

[33] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. Int. Cryptography Conf. Advances in Cryptography*, 1999, pp. 388–397.

[34] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," Advances in Cryptology CRYPTO96, Lecture Notes in Computer Science, vol. 1109, pp. 104–113, 1996.

[35] J. Lee, M. Tehranipoor, C. Patel, and J. Plusquellic, "Securing designs against scan-based side-channel attacks," *IEEE Trans. Depend. Secure Comput.*, vol. 4, no. 4, pp. 325–336, 2007.

[36] R. B. Lee, D. K. Karig, J. P. Mcgregor, and Z. Shi, "Enlisting hardware architecture to thwart malicious code injection," in *Proc. Int. Conf. Security in Pervasive Computing*, 2003, pp. 237–252, Springer Verlag.

[37] R. B. Lee, P. C. S. Kwan, J. P. McGregor, J. Dwoskin, and Z. Wang, "Architecture for protecting critical secrets in microprocessors," in *Proc. Int. Symp. Computer Architecture*, 2005, pp. 2–13.

[38] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architecture support for copy and tamper resistant software," in *Proc. Int. Conf. Architecture Support for Programming Languages and Operating Systems (ASPLOS-IX)*, 2000, pp. 168–177.

[39] L. Lin, M. Kasper, T. Guneysu, C. Paar, and W. Burleson, "Trojan side-channels: Lightweight hardware Trojans through side-channel engineering," *Cryptograph. Hardw. Embed. Syst.*, pp. 382–395, 2009.

[40] B. Liu, X. Chen, and F. Teshome, "Resilient and adaptive performance logic," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 8, no. 3, 2012, No. 22:1-22.

[41] B. Liu and L. Wang, "Minimum logic of guaranteed single soft error resilience based on group distance-two code," in *Proc. Int. Conf. IC Design and Technology (ICICDT)*, 2012, pp. 193–196.

[42] D. Lu, "Watchdog processors and structural integrity checking," *IEEE Trans. Comput.*, vol. 31, no. 7, pp. 681–685, 1982.

[43] A. Mahmood and E. J. McCluskey, "Concurrent error detection using watchdog processors—A survey," *IEEE Trans. Comput.*, vol. 37, no. 2, pp. 160–174, 1988.

[44] Microsoft, "Next-generation secure computing base," [Online]. Available: http://www.microsoft.com/resources/ngscb/default.mspx

[45] S. Mukherjee, *Architecture Design for Soft Errors*. San Mateo, CA, USA: Morgan Kaufmann, 2008.

[46] M. Namjoo, "Techniques for concurrent testing of VLSI processor operation," in *Proc. IEEE Int. Test Conf.*, 1982, pp. 461–468.

[47] M. Namjoo and E. J. McCluskey, "Watchdog processors and capability checking," in *Dig. Papers 12th Annual Int. Symp. Fault Tolerant Computing, FTCS-12*, 1982, pp. 245–248.

[48] National Security Council, "The comprehensive national cybersecurity initiative," [Online]. Available: http://www.whitehouse.gov/cybersecurity/comprehensive-national-cybersecurity-initiative

[49] A. L. Oliveira, "Robust techniques for watermarking sequential circuit designs," in *Proc. ACM/IEEE Design Automation Conf.*, 1999, pp. 837–842.
[50] C. Paar and J. Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners*. Berlin, Germany: Springer-Verlag, 2010.
[51] T. Park and K. G. Shin, "Soft tamper-proofing via program integrity verification in wireless sensor networks," *IEEE Trans. Mobile Comput.*, vol. 4, no. 3, pp. 1–13, 2005.
[52] Z. Qi, J. Wang, A. Cabe, S. Wooters, T. Blalock, B. Calhoun, and M. Stan, "SRAM-based NBTI/PBTI sensor system design," in *Proc. ACM/IEEE Design Automation Conf.*, 2010, pp. 849–852.
[53] G. Qu, "Publicly detectable watermarking for intellectual property authentication in VLSI design," *IEEE Trans. Comput.-Aided Design*, vol. 21, no. 11, pp. 1363–1368, 2002.
[54] M. O. Rabin, Fingerprinting by Random Polynomials, 1981 Center for Research in Computing Technology, Harvard Univ., Cambridge, MA, USA, Rep. TR-15-81.
[55] G. B. Ratanpal, R. D. Williams, and T. N. Blalock, "An on-chip signal suppression countermeasure to power analysis attacks," *IEEE Trans. Depend. Secure Comput.*, vol. 1, no. 3, pp. 179–188, 2004.
[56] E. Rotenberg, "AR-SMT: A microarchitectural approach to fault tolerance in microprocessors," in *Proc. Annu. Fault-Tolerant Computing Systems (FTCS)*, 1999, p. 84.
[57] J. C. M. Santos and Y. Fei, "Designing and implementing a malicious 8051 processor," in *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, 2012, pp. 63–66.
[58] J. P. Shen and M. A. Schuette, "On-line self-monitoring using signatured instruction streams," in *Proc. IEEE Int. Test Conf.*, 1983, pp. 275–282.
[59] P. P. Shirvani and E. J. McCluskey, Fault-Tolerant Systems in a Space Environment: The CRC ARGOS Project , CRC Technical Report No. 98-2 (CSL TR No. 98-774), 1998.
[60] T. J. Slegel, R. M. Averill, M. A. Check, B. C. Giamei, B. W. Krumm, C. A. Krygowski, W. H. Li, J. S. Liptay, J. D. MacDougall, T. J. McPherson, J. A. Navaroo, E. M. Schwarz, K. Shum, and C. F. Web, "IBMs S/390 G5 microprocessor design," *IEEE Micro*, pp. 12–23, 1999.
[61] L. Spainhower and T. A. Gregg, "IBM S/390 parallel enterprise server G5 fault tolerance: A historical perspective," *IBM J. Res. Develop.*, vol. 43, no. 5/6, pp. 863–873, 1999.
[62] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "AEGIS: Architecture for tamper-evident and tamper-resistant processing," in *Proc. Int. Conf. Supercomputing*, 2003, pp. 160–171.
[63] G. E. Suh, C. W. ODonnell, I. Sachdev, and S. Devadas, "Design and implementation of a single-chip secure processor using physical random functions," in *Proc. Int. Symp. Computer Architecture*, 2005, pp. 25–36.
[64] R. Torrance and D. James, "The state-of-the-art in IC reverse engineering," Cryptographic Hardware and Embedded Systems—CHES 2009, Lecture Notes in Computer Science, vol. 5747, pp. 363–381, 2009.
[65] I. Torunoglu and E. Charbon, "Watermarking-based copyright protection of sequential functions," *IEEE J. Solid State Circuits*, vol. 35, no. 3, pp. 434–440, 2000.
[66] Trusted Computing Group, "TPM main, part 1, design principles," Mar. 2006 [Online]. Available: http://www.trustedcomputinggroup.org/downloads/specifications/tpm/tpm/
[67] T. N. Vijaykumar, I. Pomeranz, and K. Cheng, "Transient fault recovery using simultaneous multithreading," in *Proc. Int. Symp. Computer Architecture*, 2002, pp. 87–98.
[68] A. Waksman and S. Sethumadhavan, "Tamper evident microprocessors," in *Proc. IEEE Symp. Security & Privacy*, 2010, pp. 173–188.
[69] A. Waksman and S. Sethumadhavan, "Silencing hardware backdoors," in *Proc. IEEE Symp. Security and Privacy*, 2011, pp. 49–63.
[70] X. Wang, T. Mal-Sarkar, A. Krishna, S. Narasimhan, and S. Bhunia, "Software exploitable hardware Trojans in embedded processor," in *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, 2012, pp. 55–58.
[71] Z. Wang and R. B. Lee, "New cache design for thwarting software cache-based side channel attacks," in *Proc. Int. Symp. Computer Architecture*, 2007, pp. 494–505.
[72] C. Webb, "z6—The next-generation mainframe microprocessor," 2007 [Online]. Available: http://speleotrove.com/decimal/IBM-z6-mainframe-microprocessor-Webb.pdf
[73] B. Yang, K. Wu, and R. Karri, "Scan based side channel attack on dedicated hardware implementations of data encryption standard," in *Proc. IEEE Int. Test Conf.*, 2004, pp. 339–344.
[74] L. Yuan, R. Pari, and G. Qu, "Soft IP protection: Watermarking HDL source codes," in *Proc. 6th Information Hiding Workshop*, 2004, pp. 224–238.

Bao Liu (SM'11) received the B.S., M.S., and Ph.D. degrees in 1993, 1996, and 2003, respectively.

He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Texas at San Antonio, San Antonio, TX, USA. He has authored over 60 journal articles and conference papers, and holds three U.S. and international patents. His current research interests include hardware security, nanocomputing, and VLSI CAD, including physical design, statistical timing analysis and optimization, power rail and signal integrity analysis, reliable and resilient design, and delay testing.

Prof. Liu has served as the Chair of an invited session Emerging Nano-Circuits and System in Midwest Symposium on Circuits and Systems in 2010, the Chair of the Hardware and System Security track in International Symposium on Quality Electronic Design (ISQED) in 2015, the Co-Chair of the Emerging Design and Technology track in ISQED since 2006, and a Panelist on CAD for Nanoelectronics in International Symposium on Nanoscale Architectures in 2010. He was a recipient of the Best Paper Award in the International Conference on Computer Design in 2005, the Best Research Award in UCSD Research Review in 2002, the China ICCAD Best Member Award in 1996, and the China Mathematics Olympiad Honor Medal in 1988.

Ravi Sandhu (F'02) received the B.Tech. and M.Tech. degrees from IIT Bombay and Delhi, and the M.S. and Ph.D. degrees from Rutgers University, New Brunswick, NJ, USA.

He is Executive Director of the Institute for Cyber Security at the University of Texas at San Antonio, where he holds the Lutcher Brown Endowed Chair in Cyber Security. Previously he served on the faculty at George Mason University (1989–2007), and Ohio State University (1982–1989). A prolific and highly cited author, his research has been funded by NSF, NSA, NIST, DARPA, AFOSR, ONR, AFRL, and private industry. His seminal papers on role-based access control established it as the dominant form of access control in practical systems. His numerous other models and mechanisms have also had considerable real-world impact.

Dr. Sandhu is a Fellow of ACM and AAAS, and has received awards from IEEE, ACM, NSA, and NIST. He served as Editor-in-Chief of the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, and previously as founding Editor-in-Chief of *ACM Transactions on Information and System Security*. He was Chairman of ACM SIGSAC, and founded the ACM Conference on Computer and Communications Security, the ACM Symposium on Access Control Models and Technologies, and the ACM Conference on Data and Application Security and Privacy. He has served as General Chair, Steering Committee Chair, Program Chair, and Committee Member for numerous security conferences. He has consulted for leading industry and government organizations, and has lectured all over the world. He is an inventor on 30 security technology patents, and has accumulated over 28,000 Google Scholar citations for his papers. At the Institute for Cyber Security, his research projects include attribute-based access control, secure cloud computing, secure information sharing, social computing security, and secure data provenance. His web site is at http://www.profsandhu.com.