Embedded Reconfigurable Logic for ASIC Design Obfuscation Against Supply Chain Attacks

Bao Liu* and Brandon Wang†
*University of Texas at San Antonio, 1 UTSA Circle, San Antonio, TX 78249
†Cadence Design Systems, Inc., 2655 Seely Avenue, San Jose, CA 95134

***Abstract*— Hardware is the foundation and the root of trust of any security system. However, in today's global IC industry, an IP provider, an IC design house, a CAD company, or a foundry may subvert a VLSI system with back doors or logic bombs. Such a supply chain adversary's capability is rooted in his knowledge on the hardware design. Successful hardware design obfuscation would severely limit a supply chain adversary's capability if not preventing all supply chain attacks. However, not all designs are obfuscatable in traditional technologies. We propose to achieve ASIC design obfuscation based on embedded reconfigurable logic which is determined by the end user and unknown to any party in the supply chain. Combined with other security techniques, embedded reconfigurable logic can provide the root of ASIC design obfuscation, data confidentiality and tamper-proofness. As a case study, we evaluate hardware-based code injection attacks and reconfiguration-based instruction set obfuscation based on an open source SPARC processor LEON2. We prevent program monitor Trojan attacks and increase the area of a minimum code injection Trojan with a 1KB ROM by 2.38% for every 1% area increase of the LEON2 processor.**

I. Introduction

Hardware is the foundation and the root of trust of any security system. In recent years, a growing number of software-based security solutions have been migrated to hardware-based security solutions for much enhanced resistance to software-based security threats. Such systems range from smartcards to specialized secure co-processing boxes, wherein hardware provides the source of security and trust for a number of security primitives [3], [23]–[25], [36], [37].

At a closer analysis, all the existing cryptographic primitives have proofs of security based on two assumptions: (1) read-proof hardware; that is, hardware that prevents an enemy from reading anything about the information stored within it; and (2) tamper-proof hardware; that is, hardware that prevents an enemy from changing anything in the information stored within it. In particular, existing cryptographic schemes consist of an algorithm which the adversary knows, but cannot change (i.e., stored in tamper-proof hardware), and a secret key, which the adversary does not know and cannot change (i.e., stored in hardware which is both read-proof and tamper-proof) [17].

However, in recent years, it has been brought into light that hardware is also subject to a number of security threats, which make hardware neither read-proof nor tamper-proof. The existing techniques mostly focus on information leak from a hardware system: An adversary may extract cryptographic keys and confidential information from a system by testing [1], [41], reverse engineering [13], or side-channel analysis [7], [21], [22], [39].

More critical threats come from the supply chain and compromise hardware integrity. In today's global IC industry, a supply chain adversary, such as an IP provider, an IC design house, a CAD company, or a foundry may have access to the source code of the design, and may easily tamper a hardware system by planting time bombs which compromise hardware computation integrity, or creating back doors which enable information leak, bypassing access control mechanisms at higher (e.g., OS and application) levels [20]. The recently-released Comprehensive National Cyber Security Initiative has identified this supply chain risk management problem as a top national priority [28].

A supply chain adversary's capability is rooted in his knowledge on the hardware design. Successful hardware design obfuscation would severely limit a supply chain adversary's capability if not preventing all supply chain attacks. However, not all designs are obfuscatable in traditional technologies. We propose to achieve ASIC design obfuscation based on embedded reconfigurable logic which is determined by the end user and unknown to any party in the supply chain. Combined with other security techniques, embedded reconfigurable logic can provide the root of ASIC design obfuscation, data confidentiality and tamper-proofness.

The rest of this paper is organized as follows. We present the theoretical results on obfuscation and introduce the existing VLSI obfuscation techniques in Section II. We present a supply chain adversary attack model, and an analysis on the existing VLSI obfuscation techniques, and present our proposed reconfigurable logic-based VLSI obfuscation in Section III. As a case study, we evaluate hardware-based code injection attacks and reconfiguration-based instruction decoder obfuscation in an open source LEON2 processor in Section IV. We conclude in Section V.

II. Background

A. Theoretical Results on Obfuscation

Obfuscation is a long-standing problem in computer security and cryptography. To obfuscate a function f is to create an implementation of f that reveals nothing about f except its input-output behavior. Intuitively, a circuit obfuscator $\mathcal{O}$ is an efficient algorithm that, given a circuit C implementing some function f, outputs another circuit $\mathcal{O}(C)$ such that (i)

978-3-9815370-2-4/DATE14/©2014 EDAA

(preserving functionality) it computes (perhaps approximately) the same function as f, (ii) (polynomial slowdown) its size is within a polynomial factor of c, and, (iii) ("virtual black-box" property) for any efficient adversary that computes some predicate on $\mathcal{O}(C)$, there exists an efficient simulator that computes the same predicate with black-box access to an oracle that evaluates f [4], [9], [27].

Software and hardware design obfuscation has long been studied as a potentially powerful tool against design tamper. Recent theoretical studies show that, (1) there exist functions that cannot be obfuscated, and (2) there exist functions that can be obfuscated. Barak et al. showed the existence of (contrived) classes of functions which are not obfuscatable, or, a general purpose obfuscator does not exist [4]. Goldwasser and Kalai showed that there exist many natural classes of functions that cannot be obfuscated with respect to auxiliary input (intuitively, one may think auxiliary input as the history or previous executions of the circuit) [18]. In contrast, Hohenberger et al. presented an obfuscation scheme of a public key-based re-encryption program [19]. Besides, the only positive obfuscation result is of point functions, which are Boolean functions that return 1 on exactly one input, for example, a password check program. Canetti and Wee separately showed how to obfuscate a point function based on a random oracle, e.g., a hash function that hides all details [9], [40]. An obfuscated point function queries the random oracle on an input, and compares the answer with a stored value. For example, a password check program encrypts an input, and compares the encryption result with a stored value, which is an encrypted password. As a result, it achieves the virtual black box property of obfuscation. This scheme is based on a weaker definition of obfuscation, which says that there is a negligible probability to distinguish an adversary circuit based on the obfuscated scheme and a simulator based on a black box of the function. As a result, this obfuscation scheme of point functions cannot be extended to obfuscate arbitrary Boolean functions [27].

B. Existing VLSI Logic Encryption/Locking Techniques

The state-of-the-art VLSI logic encryption/locking techniques include combinational logic locking and finite-state machine (FSM) locking. Combinational logic locking augments a combinational logic network with an additional group of lock inputs such that the augmented combinational logic network has the same function as the original combinational logic network only if a specific vector (aka a valid key) is applied to the lock inputs [31]. This is achieved, for example, by inserting a group of XOR logic gates or LUTs [6] or an additional logic cone [11] to the combinational logic network. FSM locking augments an FSM by introducing a group of extra finite states, which form an obfuscated mode. Only a correct sequence of inputs transit the FSM out of the obfuscated mode and set the FSM to the correct initial state in the normal operation mode [2], [10]–[12], [14]. These techniques hide combinational logic or FSM functionality by introducing additional inputs or finite states.

III. VLSI Obfuscation Against Supply Chain Attacks

A. Attack Model

A supply chain adversary is an insider who is involved in the design and the manufacturing of a hardware device. His tamper 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. For example, commercial tools are available to help convert a layout to a gate-level netlist, and further convert a gate-level netlist to a higher-level abstraction. To locate an adder in a gate-level netlist, one can re-synthesize the logic based on a revised cell library which includes a zero-cost adder cell. As a result, a supply chain adversary (e.g., in a foundry) 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. 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. 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 results intact. Such a back door cannot be detected by testing or concurrent checking of the computation results.

B. Analysis on Existing VLSI Logic Encryption/Locking Techniques

We observe that while the existing VLSI logic encryption/locking techniques succeed in preventing an adversary from operating a hardware system, they cannot guarantee to prevent a supply chain adversary from understanding and tampering a VLSI design, as it has been proven that a generic obfuscator does not exist [4]. We present a more detailed analysis as follows.

An adversary can understand and tamper the design once he has the logic encryption/locking key. An adversary at a foundry may gain knowledge of the key if the foundry needs to receive a key and activates an IC to perform manufacturing test. Or, an adversary having knowledge of the function of the chip can obtain the key. Such an adversary can be anyone having an activated IC in his possession. We assume that the adversary has the knowledge of the design, for example, by reverse engineering, and has the capability to test the chip, e.g., by applying stimuli and observing responses.

We categorize the state-of-the-art combinational logic encryption/locking techniques as follows.

XOR/XNOR-Based: The simplest combinational logic locking technique is to insert XOR and XNOR gates into a combination logic network [6], [31]. An adversary knows which inputs are functional inputs and which inputs are lock inputs. He can then identify the lock gates connected to the lock inputs. If a total of M lock gates are inserted in a combinational logic network, the complexity for an adversary to find the correct logic may not be 2^M. For example, for a logic output, if its fanin cone contains m_i lock gates, the complexity to find the correct logic function for that logic output is at most 2^{m_i}.

MUX-Based: Another combinational logic locking technique is to insert multiplexers or combine logic functions based on Shannon expansion [11]. The reason is as follows. If a lock input is connected to a lock gate that is not a XOR or XNOR gate, the key to the lock input is implied to be the non-controlling logic value of the lock gate. An adversary could easily obtain the key, unless the lock input is connected to multiple lock gates and is implied to have conflicting logic values (e.g., the lock input is connected to a group of AND gates and a OR gate which have the same function as a XOR or XNOR gate), or the gate is hidden in reconfigurable logic as in [6]. A more general design paradigm is to have a lock input connected to multiple logic cones which provide different logic functions. These logic cones need to be combined by multiplexers or based on Shannon expansion at the logic outputs with lock inputs providing select signals. For example, one can partition a combinational logic network into 2^n sub-networks, where n is the number of lock inputs, and combine the sub-networks by a multiplexer with the lock inputs providing the select signals. Each sub-network needs to take all the logic inputs, otherwise, an adversary can identify that sub-network is not valid. This increases the size of the combinational logic network exponentially. For cost reduction, the sub-networks may share common logic cones. This gives the next paradigm.

Permutation-Based: An extension of the previous paradigm is to permute the logic inputs and the logic outputs (e.g., in [32]). A permutation logic block is a group of multiplexers, wherein each output is given by any of the inputs based on the select signal. A further extension is to cut a combinational logic network into two parts, and permute the signals crossing the cut line.

Reconfigurable Logic Barrier-Based: Another technique cuts a combinational logic network into two parts with all the inputs in one part and all the outputs in another part, and implements all the gates in the cut line in reconfigurable logic, i.e., forming a reconfigurable logic barrier [6].

An adversary has the following techniques to unlock a locked combinational logic network.

Key Propagation: An adversary may apply an input vector and propagate a bit in the key to a logic output based on an ATPG algorithm. Improved locking techniques can limit the adversary advantage to nothing more than brute force [30].

Path Analysis: Similarly, in testing, an adversary flips one bit at the logic input, and observes a flipped bit at the logic output. He then finds the signal propagation path(s) in the combinational logic network. The inversion of the signal propagation path must match the inversion between the flipped logic input and the flipped logic output. If there is a single XOR or XNOR gate in the signal propagation path, the lock input or the side input of the XOR or XNOR gate is determined by the inversion of the path. If there are multiple XOR or XNOR gates in the signal propagation paths, the adversary needs to find more signal propagation paths to determine the logic values of the lock inputs. If there is a multiplexer in a signal propagation path, the select signal is implied by the signal propagation path. If multiple signal propagation paths cross a multiplexer, any signal propagation path of an incorrect inversion can be eliminated. A logic network with permuted inputs and outputs [32] may not be unlocked by this technique if the adversary cannot observe any internal signal. But it can be unlocked by graph isomorphism analysis as follows.

Graph Isomorphism: Since an adversary knows the function of the logic network (e.g., by testing), he can synthesize a combinational logic network of the same function based on the same cell library, compare with the locked combinational logic network, and find any isomorphic graph. An isomorphic graph can be found by, for example, first identifying the node with the largest degree of connection, and proceeding to its neighboring nodes and so on (e.g., as in formal verification [8]). This technique is capable to unlock any locked combinational logic network with XOR/XNOR gates, multiplexers, permutation blocks, or reconfigurable logic barriers. For a reconfigurable logic barrier [6], graph isomorphism analysis gives the function of the reconfigurable logic barrier. Although the key may still be unknown, an adversary can proceed to reconfigure the logic barrier for the same logic function, which will remove the lock from the design. An adversary can unlock a locked FSM similarly.

C. Industry Practice on VLSI Obfuscation

Although a generic obfuscator does not exist, the semiconductor industry has certain effective hardware obfuscation techniques which prevent supply chain tampering based on a specific manufacturing technology. For example, in 3D manufacturing, the interposer can be manufactured at an untrusted foundry, while the logic chips are manufactured at a trusted foundry and mounted on the interposer at a later stage. The untrusted foundry has only black box access to the logic chips and cannot tamper them. This technology however assumes the existence of a trusted foundry.

D. Proposed VLSI Obfuscation Method Based on Embedded Reconfigurable Logic

We propose to achieve hardware design obfuscation based on embedded reconfigurable logic in ASIC technology. For example, the dominant technology such as FPGA achieves reconfigurable logic based on lookup tables (Figure 1). A lookup table includes a 2^n-to-1 multiplexer and 2^n configuration memory cells. We construct a n-input gate of specific logic by loading configuration data bits to the configuration memory

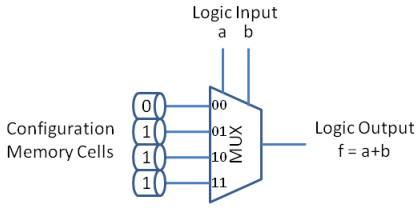

Fig. 1. LUT-based reconfigurable logic as in FPGA.

cells. A multiplexer also provides reconfigurable interconnect. Such a reconfigurable logic technology is fully compatible with the ASIC technology, i.e., reconfigurable logic modules can be embedded on an ASIC chip without any change in the manufacturing process.

In this technology, we do not assume any trusted party in the design and manufacturing process for the reconfigurable logic. The foundry can perform manufacturing test based on any implementation of the correct function of a reconfigurable logic block, while the end user determines the final implementation of the reconfigurable logic blocks in the hardware system after the design and the manufacturing process, such that nobody in the design and manufacturing process has any knowledge on the implementation of the reconfigurable logic functions and cannot tamper the reconfigurable logic blocks.

A supply chain adversary has only "black box" access to a reconfigurable logic module. He may know the function of a reconfigurable logic module based on his role in the design and manufacturing process, but he does not know the exact implementation of the function or the internal structure of a reconfigurable logic module. He cannot perform reverse engineering, run testing or probe internal signals of a reconfigurable module because the reconfigurable logic module has not been finalized. The reconfigurable module can be configured to achieve the required functionality such that the entire system can be verified at a design house or tested at a foundry, while the end user determines the final configuration for operation. Only an embedded hardware Trojan or a field engineer may have access to the final configuration of the reconfigurable module. A hardware Trojan has only limited intelligence or resource. To tamper a reconfigurable module, one needs to locate an internal signal in the module. For that a hardware Trojan needs to probe all the nodes in a reconfigurable module, apply stimuli, collect responses and analyze them. This requires a very large area or power consumption for a Trojan to stay stealthy. A field engineer may gain no knowledge on the reconfigurable logic if reconfiguration is applied right after the field engineer's visit, effectively achieving the "virtual black-box" property of the reconfigurable logic module. As a result, a reconfigurable logic implementation of a function f is an obfuscated implementation because it possesses the three properties of obfuscation: (i) preserving functionality, (ii) polynomial slowdown, and (iii) "virtual black-box."

We do not propose to implement a security system entirely in reconfigurable logic, because reconfigurable logic has a higher implementation cost in area, power consumption, and performance compared with ASIC, and reconfigurable design

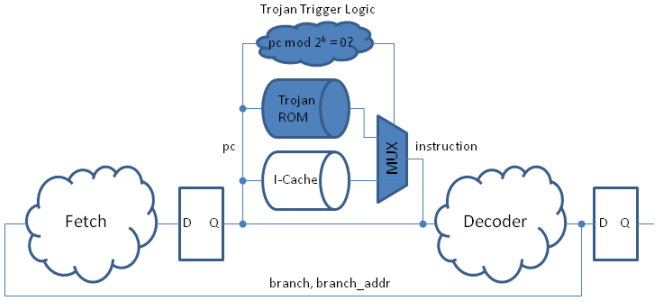

Fig. 2. Code injection Trojan including ROM, multiplexers, and trigger logic.

data also faces security threats and needs to be protected, for example, by encryption and authentication. We observe that in many cases, a security scheme has layers. We propose to achieve by reconfiguration a minimum obfuscated module which provides the root of security for the entire system. As a case study, we present obfuscation of an instruction decoder in a processor as follows.

IV. Case Study

A. A Code Injection Trojan

It is well known that a supply chain adversary may tamper a processor by inserting a hardware Trojan, which compromises authentic computations or leaks out confidential information once it is triggered by a specific instruction or data pattern [38], a specific interrupt or the count of a specific event [33]. We present here a hardware Trojan which injects Trojan code at runtime to the instruction decoder of a processor. Such a code injection Trojan can perform more complex tasks than other hardware Trojans and may easily bypass existing security checks as it hijacks the authentic microprocessor in a deputy attack, for example, in case that only the microprocessor has the decryption key to access the encrypted memory.

Such a code injection Trojan can be very small. For example, it may only include a Trojan ROM containing the Trojan instructions, a few multiplexers at the instruction fetch unit inputs, and trigger logic (Fig. 2). The Trojan trigger logic monitors the next program count (npc) in the instruction fetch unit. When the trigger condition is met, for example, the lower n bits of the next program count are all zero's, the Trojan multiplexers fetch instructions from the Trojan ROM other than from the instruction cache. Since 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 to the address of the last Trojan instruction (that restores the program count), the Trojan multiplexers fetch instructions from the instruction cache. This resumes the authentic operation.

Such a code injection Trojan cannot be detected by static code integrity check, because the Trojan instructions are not in the memory. Further, it cannot be detected by testing or non-lock-stepping concurrent checking which checks the final output of a program [26] because the authentic computation

results are intact. Lock-stepping concurrent checking which checks the internal states [26] may detect such a Trojan. 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 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 the system while keeping the sampled signals intact.

B. Reconfiguration-Based Instruction Decoder Obfuscation

Instruction set randomization (ISR) has been proposed to prevent software-based code injection attacks [5], [15], [35]. One of the simplest ISR technique is to have a reconfigurable opcode encoding while keeping the same instruction set and the same instruction formats. Or, the bits of an instruction may be permuted or XOR'ed before being sent to the instruction decoder unit (IDU). We achieve ISR by obfuscating the IDU by implementing it in reconfigurable logic, concealing the opcode check logic or any instruction bit permutation or XOR logic. ISR has its limitation in preventing software-based code injection attacks [15], [35]. We believe ISR is more effective in preventing hardware-based code injection attacks because a hardware Trojan has much less computation resource. We further prevent a hardware Trojan from monitoring an authentic computation. For example, an XOR operation in RC5 is associated with the cryptographic key, and a Trojan may be triggered by an XOR operation to leak the key [33], [38]. ISR prevents a Trojan from identifying an XOR operation and achieving the key.

A supply chain adversary has only black box access to the obfuscated IDU. Consequently, he has no knowledge on the instruction set and cannot tamper the instruction decoder. A Trojan may carry out a deputy or circumvention attack [15] as follow, but cannot stay stealthy at the same time. To carry out a deputy attack or send Trojan instructions to the obfuscated IDU, a Trojan needs knowledge of the instruction set, and it further needs to be reconfigurable to translate the Trojan instructions to the local dialect. A Trojan may apply stimuli to the instruction decoder, collect responses and analyze them for knowledge on the instruction set. To probe an internal signal (e.g., the opcode check logic output), a Trojan needs to probe all the nodes in the reconfigurable IDU, which has a prohibitive cost. Or, a Trojan may probe the other signals which are in the hardware system but are not in the obfuscated module. For a microprocessor, these signals include the program count, the ALU inputs, the memory stage inputs, the register file inputs, etc. The cost is also prohibitive for a Trojan to stay stealthy. For example, an ALU add operation may be caused by an add, subtract, multiple, load, store, jump, or return instruction. To carry out a circumvention attack or bypass the instruction decoder, a Trojan needs to duplicate the instruction decoder unit, which would be too costly. Or, a Trojan may reconfigure the IDU, which requires that the Trojan includes all the configuration bits for the IDU. There would further be significant performance degradation when the Trojan reconfigures the IDU.

TABLE I

HARDWARE OVERHEAD OF THE LEON2 PROCESSOR, A CODE INJECTION TROJAN WITH 1KB ROM, THE IDU, AN OBFUSCATED IDU, AND THE LEON2 PROCESSOR WITH AN OBFUSCATED IDU, 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
IDU	9.26×10^2	1.40×10^{-2}	6.79
Obfuscated IDU	1.57×10^4	1.46×10^{-1}	18.01
Obfuscated LEON2	6.09×10^4	2.40	18.01

C. Evaluation

We evaluate the code injection Trojan and reconfiguration-based IDU obfuscation based on an five-stage in-order open source SPARC processor LEON2 [16], which 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. We perform logic synthesis based on the Synopsys Design Compiler and the 45nm Nangate open cell library [34]. For the lookup table-based reconfigurable technology (Fig. 1), we modify the 45nm Nangate cell library such that a n-input logic gate has the same area as a 2^n-input multiplexer plus 2^n latches.

We have implemented a minimum code injection Trojan with a 1KB ROM, a few multiplexers at the instruction fetch unit input, and a trigger logic network. We have further implemented an IDU in the lookup table-based reconfigurable technology as in FPGA. We did not implement a deputy attack Trojan because of its complexity. We estimate the cost of a circumvention attack Trojan by summing up the cost of the minimum code injection Trojan and the standard IDU or the configuration memory cells of a reconfigurable IDU.

Table I gives the hardware overhead of these designs. Compared with the LEON2 processor, the minimum code injection Trojan with a 1KB ROM leads to a layout area increase of only 2.5%. Reconfiguration-based IDU obfuscation increases the IDU area from $926.3\mu m^2$ to $15718.3\mu m^2$, and increases the overall LEON2 processor area by 34.7%. For a circumvention attack, to reconfigure the IDU, including all the configuration memory cells would increase the minimum code injection Trojan area by a factor of 9.5 as the configurable memory cells take 67.7% of the area in our 2-input LUT-based reconfigurable logic. Alternatively, including an additional IDU would increase the minimum code injection Trojan area by 82.7%. For tradeoff between cost and security, we may implement $x\%$ of the IDU in reconfigurable logic starting from the inputs, which increases the Trojan area by $0.827x\%$ at the cost of $0.347x\%$ area increase for the LEON2 processor.

V. CONCLUSIONS

We propose to achieve ASIC design obfuscation based on embedded reconfigurable logic which is determined by the

end user and unknown to any party in the supply chain. While the existing obfuscation results are limited to a few specific functions, reconfiguration-based obfuscation is widely applicable to any logic function. Combined with other security techniques, embedded reconfigurable logic can provide the root of ASIC design obfuscation, data confidentiality and tamper resistance. As a case study, we evaluate hardware-based code injection attacks and reconfiguration-based instruction decoder obfuscation based on an open source LEON2 processor [16]. We prevent program monitor Trojan attacks and increase the area of a minimum code injection Trojan with a 1KB ROM by 2.38% for every 1% area increase of the LEON2 processor. Our future work include to enhance processors against a variety of supply chain attacks [29] based on this technique.

ACKNOWLEDGMENTS

We thank Dr. Ramesh Karri and Jeyavijayan Rajendran at NYU-Poly and anonymous reviewers for their valuable comments.

REFERENCES

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

[2] Y. M. Alkabani and F. Koushanfar. Active hardware metering for intellectual property protection and security. In *Proc. USENIX Security Symposium*, pages 291 –306, 2007.

[3] T. Alves and D. Felton. Trustzone: Integrated hardware and software security. *Information Quarterly*, 3(4):18–24, 2004.

[4] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *Proc. International Conference on Cryptography*, pages 1–18, 2001.

[5] E. G. Barrantes, D. H. Ackley, S. Forrest, and D. Stefanovic. Randomized instruction set emulation. *ACM Trans. on Information and System Security*, 8(1):3 – 40, 2005.

[6] A. Baumgarten, A. Tyagi, and J. Zambreno. Preventing IC piracy using reconfigurable logic barriers. *IEEE Design and Test of Computers*, pages 66 – 75, 2010.

[7] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In *Proc. Ann. Intl. Cryptography Conf. Advances in Cryptography*, pages 513–527, 1997.

[8] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, C-35(8):677 – 691, 1986.

[9] R. Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *Proc. International Conference on Cryptography*, pages 455–469, 1997.

[10] R. S. Chakraborty and S. Bhunia. Hardware protection and authentication through netlist level obfuscation. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 674–677, 2008.

[11] R. S. Chakraborty and S. Bhunia. Harpoon: An obfuscation-based SoC design methodology for hardware protection. *IEEE Trans. Computer-Aided Design*, 28(10):1493–1502, 2009.

[12] R. S. Chakraborty and S. Bhunia. Security against hardware Trojan through a novel application of design obfuscation. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 113–116, 2009.

[13] Chipworks. `http://www.chipworks.com/`.

[14] A. R. Desai, M. S. Hsiao, et al. Interlocking obfuscation for anti-tamper hardware. In *CSIIRW*, pages 1 – 4, 2012.

[15] D. Evans, A. Nguyen-Tuong, and J. Knight. Moving target defense: An asymmetric approach to cyber security. chapter Effectiveness of Moving Target Defenses. Springer, 2011.

[16] A. Gaisler. LEON SPARC V8 Processors. `http://www.gaisler.com/`.

[17] R. Gennaro, A. Lysyanskaya, T. Malkin, S. Micali, and T. Rabin. Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In *Theory of Cryptography Conf.*, pages 258–277, 2004.

[18] S. Goldwasser and Y. T. Kalai. On the impossibility of obfuscation with auxiliary input. In *Proc. IEEE Symp. Foundations of Computer Science*, pages 553–562, 2005.

[19] S. Hohenberger, G. N. Rothblum, A. Shelat, and V. Vaikuntanathan. Securely obfuscating re-encryption. In *Theory of Cryptography Conf.*, pages 233–252, 2007.

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

[21] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Proc. Intl. Cryptography Conf. Advances in Cryptography*, pages 388–397, 1999.

[22] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. *Advances in Cryptology - CRYPTO'96, Lecture Notes in Computer Science*, V. 1109:104–113, 1996.

[23] R. B. Lee, P. C. S. Kwan, J. P. McGregor, J. Dwoskin, and Z. Wang. Architecture for protecting critical secrets in microprocessors. In *Proc. International Symposium on Computer Architecture (ISCA)*, pages 2–13, 2005.

[24] 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. International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS-IX)*, pages 168–177, 2000.

[25] Microsoft. Next-Generation Secure Computing Base. `http://www.microsoft.com/resources/ngscb/default.mspx`.

[26] S. Mukherjee. *Architecture Design for Soft Errors*. Morgan Kaufmann Publishers, 2008.

[27] A. Narayanan and V. Shmatikov. On the limits of point function obfuscation, 2006. `http://eprint.iacr.org/2006/182`.

[28] National Security Council. The Comprehensive National Cybersecurity Initiative. `http://www.whitehouse.gov/cybersecurity/comprehensive-national-cybersecurity-initiative`.

[29] J. Rajendran, A. K. Kanuparthi, M. Zahran, S. K. Addepalli, G. Ormazabal, and R. Karri. Securing processors against insider attacks: A circuit-microarchitecture co-design approach. *IEEE Design & Test of Computers*, 30(2):35–44, 2013.

[30] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri. Security analysis of logic obfuscation. In *Proc. ACM/IEEE Design Automation Conf.*, pages 83 – 89, 2012.

[31] J. Roy, F. Koushanfar, and I. Markov. EPIC: Ending piracy of integrated circuits. In *Proc. Conference on Design Automation and Test in Europe*, pages 1069 – 1074, 2008.

[32] J. A. Roy, F. Koushanfar, and I. L. Markov. Protecting bus-based hardware IP by secret sharing. In *Proc. ACM/IEEE Design Automation Conf.*, 2008.

[33] J. C. M. Santos and Y. Fei. Designing and implementing a malicious 8051 processor. In *Proc. IEEE Intl. Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pages 63 – 66, 2012.

[34] S. I. I. (SI2). Nangate Open Cell Library. `http://www.si2.org/openeda.si2.org/projects/nangatelib/`.

[35] N. Sovarel, D. Evans, and N. Paul. Where's the FEEB? the effectiveness of instruction set randomization. In *14th USENIX Security Symposium*, 2005.

[36] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. Aegis: Architecture for tamper-evident and tamper-resistant processing. In *Proc. International Conference on Supercomputing*, 2003.

[37] Trusted Computing Group. Trusted Platform Module (TPM) Specifications. `http://www.trustedcomputinggroup.org/resources/tpm_main_specification`.

[38] X. Wang, T. Mal-Sarkar, A. Krishna, S. Narasimhan, and S. Bhunia. Software exploitable hardware trojans in embedded processor. In *Proc. IEEE Intl. Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pages 55 – 58, 2012.

[39] Z. Wang and R. B. Lee. New cache design for thwarting software cache-based side channel attacks. In *Proc. International Symposium on Computer Architecture*, pages 494–505, 2007.

[40] H. Wee. On obfuscating point functions. In *Proc. ACM Symp. the Theory of Computing*, 2005.

[41] B. Yang, K. Wu, and R. Karri. Scan based side channel attack on dedicated hardware implementations of data encryption standard. In *Proc. IEEE Intl. Test Conf.*, pages 339–344, 2004.