International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 04 Issue: 09 | Sep -2017 www.irjet.net p-ISSN: 2395-0072

VERIFICATION OF SECURITY FOR UNTRUSTED THIRD PARTY IP CORES

MANISHA P[1], SUDHAMAYI B[2], KIRANMAI M[3]

[1]PG Scholar in VLSI-SD, Nimra college of engineering and technology, vijayawada.
[2] Assistant professor in Department of ECE nimra college of engineering and technology,vijayawada.
[3] PG Scholar in VLSI-SD, Nimra college of engineering and technology, vijayawada.

Abstract - *Globalization of the system-on-chip (SoC) design flow has created opportunities for rogue intellectual property (IP) vendors to insert malicious circuits (a.k.a. hardware Trojans) into their IPs. We propose to formally verify third party IPs (3PIPs) for unauthorized hird party's. We validate our technique using Trojan benchmarks from the Trust-Hub*

keyWords: **sog,ip,Trojan,pip**

1.INTRODUCTION :

A. Motivation

Fabless System-on-a-Chip (SoC) designers integrate thirdpartyIntellectual Property (3PIP) cores with in-house IP coresto design SoCs. They outsource the fabrication and test phases.3PIP vendors, foundries and test companies are distributedworldwide. An SoC designer uses these services to meet the tight time-to-market deadlines and to reduce the design, fabrication, and test costs.

Not with standing its benefits, globalization of the SoC design flow has created opportunities for rogue elements within the supply chain to corrupt the ICs Rogue elements in a foundry can alter a design or include malicious circuits(called hardware Trojans) during fabrication. Similarly, rogue elements in the 3PIP companies can insert Trojans in theirIP. The inserted Trojans may be conditionally triggered or always on. When triggered, a Trojan may result ina deadlock or failure of the system (overt attack), or create a backdoor allowing the attacker to gain remote access to the system (covert attack) .

To build a trustworthy SoC design, it is necessary to ensure the trustworthiness of the 3PIPs. However since this is not always possible, the SoC integrator should ensure that all the security vulnerabilities in any of the 3PIPs are detected or their effects muted before they damage the system.

B. Previous work

Trojan-detection techniques in 3PIPs can be broadly classified into code/structural analysis and formal verification techniques.

Code/structural analysis techniques. Since 3PIPs are typically delivered as Register Transfer Level (RTL) VHDL / Verilog codes, code coverage analysis is performed on RTL codes to identify suspicious signals that may be a part of a Trojan . Even 100% coverage of the RTL code in a design does not guarantee that it is fault-free . Hence, code coverage analysis does not guarantee its trustworthiness. Alternately, an SoC integrator may automatically analyze the 3PIP code and mark suspicious signals using controllability and reachability values of signals . FANCI marks gates with low activation probability as suspicious . VeriTrust marks gates that are not driven by functional inputs as suspicious . The implicit assumption here is that those gates are driven by Trojans, as they do not perform any computation on functional inputs. The SoC integrator then manually analyzes the small number of suspicious gates to determine if they are part of a Trojan.

DeTrust exploits the limitations of FANCI and Veri Trustto design Trojans that bypass them . To bypass FANCI,DeTrust designs Trojans whose trigger vector arrives over multiple clock cycles. If the probability of activating a signal is below a pre-determined threshold, FANCI marks it as suspicious. For example, if a 128-bit trigger arrives in one clock cycle, the probability of activating the trigger signal is 2^{-128}, and FANCI marks it as suspicious. However, DeTrust makes the trigger signals arrive as four-bit nibbles over 32 clock cycles. Now, FANCI computes the probability of activating the trigger signal to be 2^{-4}. Since this value is significantly higher, FANCI does not mark this signal as suspicious. To bypass VeriTrust, DeTrust ensures that each gate in the Trojan is driven by a subset of functional inputs.The limitations of code/structural analysis techniques are: (1) they do not guarantee Trojan detection [10], (2) they burden the designer with manual analysis, and (3) they analyze only the combinational parts of the design.

Formal verification techniques. An SoC integrator and a 3PIP vendor can agree upon a pre-defined set of security properties that the IP should satisfy . The SoC integrator can check the 3PIP for these properties. To check if a design honors these properties, one converts the target design into a proof checking format (for example, Coq). This technique has been demonstrated to detect data leakage and malicious modifications to registers .

The limitations of this technique are: (1) One can check if a design satisfies pre-defined properties, but not if the design has additional vulnerabilities while satisfying these properties . (2) Lack of automation to convert VHDL/Verilog to Coq format. (3) The VHDL/Verilog and Coq

representations of the target design may not be equivalent; If a Coq representation of a design is considered trustworthy, it does not necessarily mean that the corresponding VHDL/Verilog representation is trustworthy.

Jasper uses a proprietary "taint propagation" technology to identify if secret data can be leaked by exploiting design bugs However, it does not target Trojans. The authors of identified and verified three security properties of SoCs using taint-propagation techniques: (1) the firmware should not read the hardware encryption keys, (2) the firmware should not modify the access privileges of memory and input/outputs, and (3) the host software should not modify the SoC security monitor. However, these techniques target unintentional design bugs, but not Trojans.

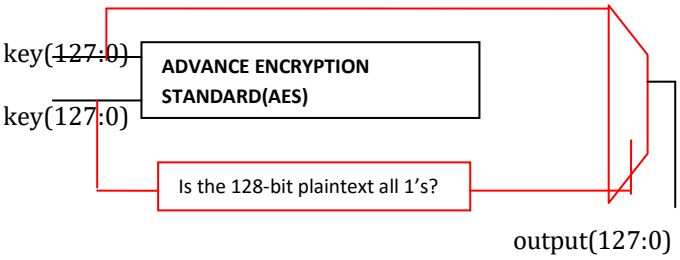

Fig. 1: AES design with a hardware Trojan. The Trojan leaks the secret key through the output when the 128-bit plaintext is all 1's.

The Trojan components are shown in red.
Recently, formal verification techniques to detect corruption of critical registers by Trojans have been proposed . However, these techniques do not detect Trojans that cause information leakage.

C.Contribution: Formally verifying 3PIPs for the presence of information leaking trojans

We propose to detect Trojans in 3PIP that leak sensitive information (cryptographic key, plaintext, or intermediate computation).

We use model checking to detect Trojans. The input to the model checker is the target property to be checked and a formal description of the design in temporal logic, which is a representation of the design as a sequence of states. The output of the model checker is a set of states which satisfy the given property, or a witness of a sequence which violates the property. The property that we will check is "does the design leak any sensitive information?" If a design violates this property, it is infected with information-leaking Trojans; otherwise, it is free from such Trojans. In addition to identifying Trojan infected designs, the model checker generates the witness (set of inputs over several clock cycles) that triggers the Trojan.

The advantages of this technique are: (1) It can be used on any (cryptographic) design; (2) It guarantees detection of information leaking Trojans and produces the trigger condition for the Trojan, if the 3PIP has one; (3) It can be used in conjunction with other property-based Trojan detection techniques .

D. Motivational example

A Trojan which leaks the critical data stored in register R, through an output port O is formally defined as

$$\exists\, i_{trigger} \in I \ni D \models (R == O) \,||\, (R == \neg O)\ \forall r \in R$$

where I is the set of possible input patterns and $i_{trigger}$ is the set of input pattern that triggers the Trojan, and r is the value of the register R. D is the design. On applying $i_{trigger}$, the contents of register R is leaked through the output port O.

Example 1: Consider the AES design shown in Figure 1. The Trojan leaks the secret key through the output when the 128- bit plaintext is all 1s. Here, the Trojan maps each bit of the key to the corresponding bit of the output. Based on the trigger, a Trojan can be (i) always on (i.e. no trigger), (ii) triggered by only current inputs, (iii) triggered after a specific number of clock cycles, and (iv) triggered by inputs arriving over multiple clock cycles . FANCI and VeriTrust deal with Trojans of type (i) and (ii). DeTrust designed Trojans of type (iii) and (iv) to defeat them. Our technique can detect data-corrupting Trojans of all types.

E. Organization of the paper

The rest of the paper is organized as follows: Section II describes the threat model, prior work, and background on model checking. Section III derives properties to detect Trojans that leak information. Results are provided in Section IV. Section V concludes the paper.

II. THREAT MODEL AND BACKGROUND

A. Threat model

A 3PIP vendor or a rogue element in the 3PIP vendor company is the attacker. The attacker seeks to subvert the security of the SoC that is using his IP. He introduces hardware Trojans in the IP to corrupt critical data. He only inserts Trojans whose trigger and payload characteristics are "digital." He cannot design a Trojan that depends on the physical characteristics of the SoC as these characteristics are determined by the design-synthesis constraints; the 3PIP vendor has no control over the design constraints imposed on the SoC by the SoC integrator. If a 3PIP has "non-volatile" components, a designercan identify them as he needs to

manufacture them; thus, we consider Trojans that do no use non-volatile components.

The SoC integrator is the defender. His objective is to detect Trojans, if any, in the 3PIP. We assume that the defender has access to the RTL/gate-level netlist of the 3PIP and hence can verify its function. Furthermore, he knows the functionality of the input and output ports of the 3PIP from the specification.

Protocol to verify trustworthiness of an IP. We follow the protocol outlined in below. The SoC integrator and the IP vendor agree upon a set of security properties for the design. For instance, one property can check for the valid ways to update a stack pointer in a processor. An SoC integrator performs functional verification. The attacker in the IP design house can insert Trojans that corrupt critical data, while satisfying the agreed upon security properties, and passes functional verification. The task of the SoC integrator is twofold: (1) check if the design satisfies the agreed upon security properties, and (2) check if the design has Trojans that corrupt data without violating these properties.

B. Formal methods in hardware design

Formal verification is an approach to ensure that safetycritical components in a design are exhaustively tested for correctness . Quality criteria may be specified using properties described in temporal logic and its variations . In linear time temporal logic (LTL), the notion of time is that of a linearly ordered set (this can be thought of as a possible sequence of states).

Model checking is the process of analyzing a design forthe validity of properties stated in temporal logic. A model checker takes the Verilog code along with the property written as a Verilog assertion and derives a Boolean satifiability (SAT) formulation for validating/invalidating the property. This SAT formulation is fed to a SAT engine, which then searches for an input assignment that violates the property .

Bounded model checking. In practice, designers know the bounds on the number of steps (clock cycles) with in which a property should hold. In Bounded Model Checking (BMC), a property is determined to hold for at least a finite sequence of state transitions. The Boolean formula for validating/invalidating the target property is given to a SAT engine, and if a satisfying assignment is observed within T clock cycles, that assignment is a witness against the target property. We develop properties to detect Trojans that corrupt critical data and verify the target design for satisfaction of these properties using a bounded model checker.

C. Formal methods in hardware security

SAT-based techniques can be used to detect fault attacks . Satisfiability Modulo Theory-based techniques can evaluate the strength of software countermeasures against side channel attacks . These techniques do not target Trojans.

III.FORMALLY DETECTING INFORMATION LEAKAGE

A. A first attempt at a property to detect information leakage

$$\exists\, i \in I \ni D \models (s == o) \quad (1)$$

To detect an information-leaking Trojan, the property should check if there exists an input assignment such that the secret is mapped to an output port for all possible values of the secret. Equation (2) lists this property. The SAT engine searches for an input assignment that violates the property in the target design. In this case, the SAT engine searches for an input assignment that leaks the secret, s, to the output o, for all possible values (S) of the secret key. If there exists an assignment, then the secret key can be leaked. The input assignment, if exists, is the trigger vector for the Trojan.

However, this simple property has several disadvantages. First, one needs to check for all possible values of the secret key. If the key has N bits, there exist 2N possible values. For large N (say 100), a defender cannot check for all possible values. Second, instead of leaking the entire key, an attacker may leak only its subset. If the key has N bits, there exist 2^N subsets. Thus, it is computationally infeasible check for the leakage of all possible subsets for large N.

Example 2: Consider the Trojan in AES-T100 in the Trust-Hub benchmark suite . This Trojan leaks only the least significant 8 bits of the secret key. The initial property will not detect this Trojan as its most significant 120 bits are not leaked. If one modifies the initial property to check for the leakage of subset of the key bits, there exist 2^{128} possible subsets, making it computationally infeasible to check for all possible subsets.

B. Refinement 1: Check for leakage of a subset of the secret key

$$\exists\, i \in I, s_x \in S_{N-1}\ D \models (s_0 == o) \quad (2)$$

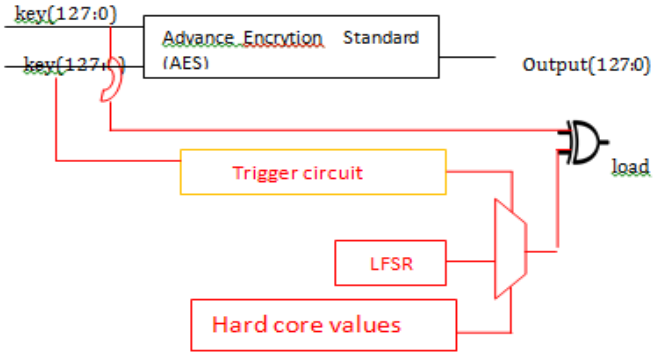

Fig. 2: AES design with a hardware Trojan. The Trojan leaks the least significant 8 bits of the secret key through the load output. One needs to apply four pre-selected plaintexts (1 through 4) to trigger the Trojan. Once the Trojan is triggered, the key is XORed with a pre-defined constant. Otherwise, the key is XORed with the contents of a linear feedback shift register (LFSR). Refinement 1 (Equation 3) checks for a Trojan which leaks a target bit (bit s0) of the secret key. While checking for this property, the BMC will assign values to the other bits of the secret key and the inputs such that this property is satisfied.

If there exists an assignment, then the property finds a Trojan. By targeting the individual key bits, instead of the subset of the key bits, the number of properties checked by the defender reduces from 2^N to N for an N-bit key. Furthermore, only two possible values — logic 0 and 1 — can be assigned to the target bit s0. Thus, one needs to check for 2xN possible values and not 2^N values.

C. Refinement 2: Check for leakage in the presence of a multiple clock cycle trigger

There exist Trojans whose trigger vector arrives over multiple clock cycles. The initial property and refinement 1 cannot detect such Trojans.

Example 3: Consider the Trojan in AES-T800 in the *Trust-hub* benchmark suite shown in Figure 2. The Trojan leaks the least significant eight bits of the secret key through the load output. One needs to apply four pre-selected plaintexts (1 through 4) consecutively to trigger the Trojan. The Trojan will not be triggered if these plaintexts do not arrive in sequence. Once the Trojan is triggered, the key is XORed with a pre-defined constant. Otherwise, the key is XORed with the contents of a linear feedback shift register (LFSR).

To detect such Trojans, one needs to determine the assignments to the input over multiple clock cycles. Following refined property 2 (Equation 4) uses BMC. BMC unrolls the design for multiple clock cycles and tries to find a set of input assignments over those clock cycles that violates this property.

$$\exists\, i \in I\ , S_x \in S_{N-1} \ni D \models (s_0 == o) \qquad (3)$$

Example 4: In the case of AES-T800, if the design is unrolled for at least four clock cycles, the BMC can find the four plaintexts that triggers the Trojan.

D. Final property

A Trojan may not necessarily leak a key bit through an output port. A Trojan may compute a function of the key bits and leak the output of that function. An attacker can infer the key from this output, or, at the least, can gain information about the key, thereby reducing the number of brute force attempts in recovering it. For example, an attacker may leak the AND of two secret bits.

Example 5: Consider the AES-T800. Only when the value in the LFSR is all 0s, the least significant eight bits of the key are leaked through the load output. Consequently, it will be detected by refinement 2. However, when the value in the LFSR is all 1s, the keys are inverted, and the inverted value is leaked. Consequently, the Trojan will not be detected.

To detect such Trojans, one needs to check for the leakage of all possible functions of secret bits. For an N-bit key, there are 2^{2n} Boolean functions. It is computationally infeasible to detect the leakage for all possible functions, for large values of N (say 100). However, since we are checking for the leakage of individual key bits, there are only four possible functions for a key bit s: { 1,0,s, ¬s}An attacker cannot get information about the key bit when the Trojan leaks constant 1 and 0; he can gain information only when s and/or :s is leaked. The final property checks if the target key bit or its compliment leaks constant 1 and 0; he can gain information only when s and/¬s or s is leaked. The final property checks if the target key bit or its compliment leaks.

$$\exists i \in I, S_{N-1} \in S_{N-1} \ni D \models (s_0 == o) \vee (\neg s == o) \forall s_0 \in \{0,1\} \ldots (4)$$

Example 6: In the case of AES-T800, when the least significantbit of the LFSR is 1, the complement of key bit is leaked at the load output. Otherwise, the key bit is leaked. Thus, the final property will detect this Trojan irrespective of the value in the LFSR.

E. Limitation

We use BMC to execute the G operator. In BMC, the number of clock cycles to unroll is fixed and is specified by the user. The simulation complexity increases with the increase in the number of clock cycles. Hence, one can perform BMC only for a limited for a number of clock cycles. Let M be the maximum number of clock cycles for which BMC can be performed. The

final property can detect Trojans only if they leak the secret key in the first M clock cycles. For clock cycles greater than M, our technique does not guarantee the trust worthiness of the design. One solution is the SoC integrator restarts the design once the number of clock cycles exceeds M.

IV. RESULTS

A. Experimental Setup We generated Verilog assertions for the information leakage property in Section III for the designs in the Trust-Hub benchmark suite . These assertions were embedded into the respective designs and provided as input to the BMC engine of the SMV tool from Cadence. Each design is unrolled for twelve clock cycles. We used an Intel(R) Xeon E5-2450L 32 cores CPU with 128GB memory operating at 1.80GHz to run the simulations. We used only the benchmarks in which the Trojans leak information and are triggered by digital inputs. This is because (i) the technique targets Trojans that leak information and (ii) in our threat model, the malicious 3PIP vendor has no control over the design constraints imposed on the SoC by the SoC integrator.

The first three columns in Table I show the characteristicsof the Trojans. The Trojans leak either the entire or a subset of the secret key of AES and RSA. The trigger condition varies from always on (e.g., AES-T100) to trigger spanning multiple clock cycles (e.g., AES-T800) to a trigger arriving after a specific number of clock cycles (e.g., AES-T900).

B. Detection capability

A design is infected with an information-leaking Trojan even if one bit of the secret key or its compliment is leaked through an output. Columns 4-8 in Table I shows the detection capability of the properties. The initial property does not detect any Trojan as it is computationally infeasible to enumerate all possible keys.

Refinement 1 detects Trojans that leak only some of the bits of the secret key. However, it does not detect the Trojans in AES-T800, AES-T1100, AES-T2000, and RSA-T300, because the trigger for these Trojans arrives over multiple clock cycles. AES-T900 and AES-T1200 designs are special cases. In these designs, the Trojan is triggered after 2^{128} -1 clock cycles. The Trojans consist of a 128-bit counter to count the number of clock cycles. To generate a counterexample that violates refinement 1, SMV initializes all the flip-flops of this counter to 1, thereby forcing the value of 2^{128} -1. This triggers the Trojan, thereby leaking a part of the secret key.

Refinement 2 and final property detect Trojans whose trigger vectors arrive over multiple clock cycles. All Trojans were detected by these two properties. The final property additionally considers the leakage of a key bit or its complement.
The technique is oblivious to the structure of the Trojan. For example, AES-T600 leaks the key through an inverter, while AES-T700 leaks the key through an XOR-gate where the other input of the XOR gate is fed by an LFSR. BMCbasedformal verification detects both these Trojans by setting the plaintext (and LFSR seed) to an appropriate value that triggers the Trojan. The technique is also independent of the underlying algorithm; it detects Trojans in both AES and RSA.

The last two columns in Table I show the memory usage and the time taken. The memory usage is high because BMC makes multiple copies of the design for the number of clock cycles unrolled. However, the memory usage is within the limits of a modern processor, thus making it feasible to check for several hundred clock cycles. Furthermore, all the Trojans were detected within 100 seconds. Checking for the final property did not result in any falsenegatives as it detected all the Trojans. To check for false positives, we checked for secret key leakage on Trojan-free AES and RSA designs from the same benchmark suite. Our technique did not flag these designs as Trojan-infected.

Benchmark characteristics			Detection capability						
Name	Key bits leaked	Trigger condition	Property				Detected?	Memory (GB)	Time (s)
			Initial	Refinement 1	Refinement 2	Final			
AES-T100	7-0	Always on	No	Yes	Yes	Yes	Yes	3.91	96.1
AES-T200	7-0	Always on	No	Yes	Yes	Yes	Yes	3.92	94.23
AES-T600	127-0	Plaintext =128'hf	No	Yes	Yes	Yes	Yes	3.96	96.8
AES-T700	7-0	Plaintext = 128'h00112233445566778899aabbccddeeff	No	Yes	Yes	Yes	Yes	3.94	96.33
AES-T800	7-0	Plaintext = (1) 128'h3243f6a8885a308d313198a2e0370734 (2) 128'h00112233445566778899aabbccddeeff (3) 128'h0 (4) 128'h1	No	No	Yes	Yes	Yes	3.95	96.86
AES-T900	7-0	At clock cycle 2^{128}	No	Yes	Yes	Yes	Yes	3.93	96.12
AES-T1000	7-0	Plaintext = 128'h00112233445566778899aabbccddeeff	No	Yes	Yes	Yes	Yes	3.94	96.55
AES-T1100	7-0	Plaintext = plain texts consecutively (1) 128'h3243f6a8885a308d313198a2e0370734 (2) 128'h00112233445566778899aabbccddeeff (3) 128'h0 (4) 128'h1	No	No	Yes	Yes	Yes	3.95	96.57
AES-T1200	7-0	At clock cycle 2^{128}	No	Yes	Yes	Yes	Yes	3.94	96.87
AES-T2000	127-0	Plaintext = (1) 128'h3243f6a8885a308d313198a2e0370734 (2) 128'h00112233445566778899aabbccddeeff (3) 128'h0 (4) 128'h1	No	No	Yes	Yes	Yes	3.98	94.92
RSA-T100	31-0	Plaintext = 32'h44444444	No	Yes	Yes	Yes	Yes	0.27	2.31
RSA-T300	31-0	Every alternate clock cycle	No	No	Yes	Yes	Yes	0.28	2.48

TABLE I: Detection capability of the properties against the Trojans in Trust-Hub benchmark suite . The number within parentheses in the trigger condition column indicates the number of clock cycles.

C. Number of clock cycles for which the property is checked For the designs in the Trust-Hub benchmark suite, the maximum number of clock cycles over which the trigger vectors arrive is four. Since we performed BMC for 12 clock cycles, we were able to detect the Trojans. However, if one performs the BMC for only 3 clock cycles, then Trojans in AES-T800, AES-T1100, AES-T2000 would not be detected. Hence, it is necessary to perform BMC for the maximum possible number of clock cycles.

Table II shows the maximum number of clock cycles for which SMV can unroll a design and check for the final property. For this experiment, we set the maximum memory usage to be 16GB. In case of AES designs, one can unroll for more than thousand clock cycles. However, in case of RSA designs, one can unroll for only a few hundred clockcycles. This is because memory usage of the Cadence SMVtool increases with the increase in state variables. RSA designs have more state variables than AES designs. Nevertheless, all the Trojans were detected.

If these designs do not have Trojans, we guarantee the trustworthiness of the designs for the number of clock cycles unrolled. Beyond this, we do not offer any security guarantees. To be prudent, the SoC integrator has to reset the design.

TABLE II: Maximum number of clock cycles for which the information-leakage property is checked once the number of clock cycles exceeds this value. Sincewe unrolled the design for several hundred clock cycles, there to grater needs to reset it every several hundred clock cycles, leading to a throughput penalty of less than 1%.

D. Number of leakage paths

A Trojan can leak a key bit through multiple output ports. One may classify a design as Trojan-infected if atleast one of the leakage paths is detected. However, detecting all these leakage paths demonstrate the effectiveness of averification technique.

TABLE II

Name	Direct			Compliment		
	Max. # of clock cycles	Memory (GB)	Time (s)	Max. # of clock cycles	Memory (GB)	Time (s)
AES-T100	4400	10.65	379.12	2535	12.32	155.06
AES-T200	7605	11.83	174.61	2251	13.79	298.33
AES-T600	1042	12.45	151.75	1042	12.45	152.41
AES-T700	1203	10.56	198.15	815	11.18	276.98
AES-T800	1483	13.16	155.48	1583	13.52	160.36
AES-T900	740	13.20	1368.93	3540	14.18	169.36
AES-T1000	1640	12.26	152.52	1640	12.26	153.51
AES-T1100	1655	13.76	162.0	1655	13.76	161.85
AES-T1200	870	13.49	1094.01	840	13.77	1385.64
AES-T2000	970	12.46	180.16	1002	12.80	165.5
RSA-T100	220	13.49	1247.77	-	-	-
RSA-T300	160	6.45	394.99	-	-	-

Figure 3 shows the number of leakage paths detected for both the key bits and their compliments over multiple clock cycles. For most designs, the number of leakage paths is 64, because the least significant 8 bits of the key are leaked through 8 output ports. In case of AES-T600 and AEST2000, the number of leakage paths is 1280, as the 128-bit key is leaked through 10 output ports. Most of the leakage paths are detected in the second clock cycle. This is because the latency of the Trojan in these designs is two clock cycles.

Our technique detects all the direct leakage paths reported in the documentation of the benchmark designs. While the documentation does not report the paths for complimentary leakage, our technique detects them.

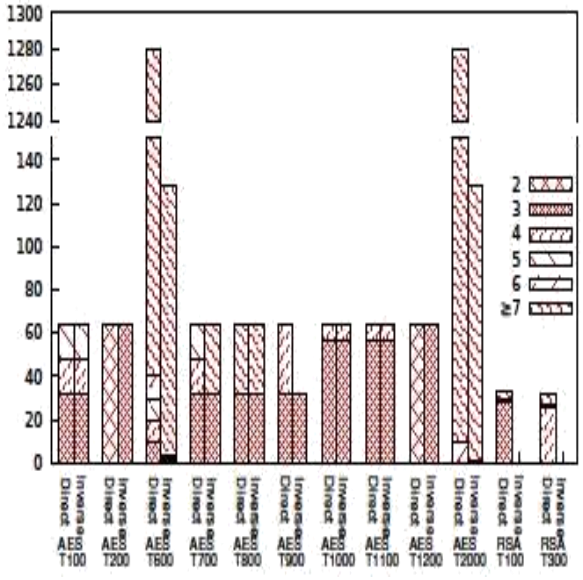

Fig. 3: Number of leakage paths across different clock cycles

V. CONCLUSION

We proposed a property to detect information-leaking Trojans, provided the information is leaked within the maximum number of clock cycles for which the design is

unrolled. We do not offer any security guarantees for clock cycles greater than this number.

Hence, a designer has to reset the design when the number of clock cycles hits this limit, thereby reducing the throughput. However, as seen in Table II, one can unroll the designs for more than hundred clock cycles, leading to a throughput reduction of less than 1%. Nevertheless, one can increase the number of clock cycles for which the design is checked by using an automatic test pattern generator (ATPG), instead of a BMC, to check for the property. This is because an ATPG consumes less memory than a BMC. One can develop similar properties to detect Trojans that corrupt registers that hold critical information and Trojans that change the functionality of a design.

REFERENCES

[1]M. Tehranipoor and F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," IEEE Design and Test of Computers, vol. 27, no. 1, pp. 10–25, 2010.
[2]"Defense Science Board (DSB) study on High PerformanceMicrochip Supply," http://www.acq.osd.mil/dsb/reports/ADA435563.pdf, 2005.
[3]S. Bhunia, M. Hsiao, M. Banga, and S. Narasimhan, "Hardware Trojan Attacks: Threat Analysis and Countermeasures," Proceedings of the IEEE, vol. 102, no. 8, pp. 1229–1247, 2014.
[4]X. Zhang and M. Tehranipoor, "Case study: Detecting hardware Tro-jans in third-party digital IP cores," IEEE Intentional Symposium on Hardware Oriented Security and Trust, pp. 67–70, 2011.
[5]M. Banga and M. Hsiao, "Trusted RTL: Trojan detection methodology in pre-silicon designs," IEEE International Symposium on Hardware-Oriented Security and Trust, pp. 56–59, 2010.
[6]J. Jou and C. J. Liu, "Coverage analysis techniques for HDL design validation," IEEE Asia Pacific Conference on Chip Design Languages, 1999.

BIOGRAPHIES

Ms SUDHAMAYI B is presently working as a Assistant Professor in ECE department.Nimra Institute of Engineering & Technology, Vijayawada. she has obtained M.tech from JNTUK.s he has published several research papers in various national and international Journals .

Ms MANISHA P is a Postgraduate student of Nimra Institute of Engineering & Technology, Vijayawada. she is presently pursuing her M.Tech., degree from Nimra Institute of Engineering & Technology Affiliated to JNTU, Kakinada. she has obtained B.Tech., degree from Sana college of Engineering & Technology Affiliated to JNTU, Hydeabad in the year 2014.

Ms KIRANMAI MOKA is a Postgraduate student of Nimra Institute of Engineering & Technology, Vijayawada. she is presently pursuing her M.Tech., degree from Nimra Institute of Engineering & Technology Affiliated to JNTU, Kakinada. she has obtained B.Tech., degree from Nova Institute of Engineering & Technology Affiliated to JNTU, Kakinada in the year 2013.