A Novel Tampering Attack on AES Cores with Hardware Trojans

Ayush Jain, and Ujjwal Guin

Dept. of Electrical and Computer Engineering, Auburn University

Emails: {ayush.jain, ujjwal.guin}@auburn.edu

Abstract—The implementation of cryptographic primitives in integrated circuits (ICs) continues to increase over the years due to the recent advancement of semiconductor manufacturing and reduction of cost per transistors. The hardware implementation makes cryptographic operations faster and more energy-efficient. However, various hardware attacks have been proposed aiming to extract the secret key in order to undermine the security of these primitives. In this paper, we focus on the widely used advanced encryption standard (AES) block cipher and demonstrate its vulnerability against tampering attack. Our proposed attack relies on implanting a hardware Trojan in the netlist by an untrusted foundry, which can design and implement such a Trojan as it has access to the design layout and mask information. The hardware Trojan's activation modifies a particular round's input data by preventing the effect of all previous rounds' key-dependent computation. We propose to use a sequential hardware Trojan to deliver the payload at the input of an internal round for achieving this modification of data. All the internal subkeys, and finally, the secret key can be computed from the observed ciphertext once the Trojan is activated. We implement our proposed tampering attack with a sequential hardware Trojan inserted into a 128-bit AES design from OpenCores benchmark suite and report the area overhead to demonstrate the feasibility of the proposed tampering attack.

Index Terms—Advanced Encryption Standard, Hardware Trojan, Tampering.

I. INTRODUCTION

The advancement in semiconductor manufacturing and testing has enabled the system-on-chip (SoC) design house to incorporate more functionality in modern SoCs that consists of millions of transistors. Consequently, the overall complexity of designing and manufacturing such integrated chips (ICs) has increased. As advanced technology nodes are adopted, building and maintaining a foundry requires large capital investment [1], resulting in a minimal number of foundries across the globe. Currently, integration of third-party intellectual properties (3PIP) with the original design and outsourcing to an offshore foundry for manufacturing and testing is a typical current trend for design-houses. However, this globalized and distributed supply chain model comes with ample scope to tamper the design by implanting hardware Trojans, malicious modifications into the IC, at the design and fabrication phases [2], [3]. Hardware Trojans can pose a severe security threat to the designs used for security-sensitive applications, such as cryptographic modules.

Cryptographic algorithms have widely been adopted as a critical means to provide the security of communication,

data, and other sensitive assets. The applications range from commonly used smart cards to highly critical defense or government applications, which rely on these algorithms implemented with dedicated hardware for confidentiality, end-point authentication, integrity verification, and non-repudiation [4]. A secure hardware implementation of a cryptographic system comprises of key dependant logic operations, where the secret key is stored in a tamper-proof memory. The plaintext inputs of a system go through a series of cryptographic computations dependent on the secret key value to produce the final ciphertext. Traditionally, such hardware implementations should be secure even if all the design details, except the secret key, is publicly available. Unfortunately, a hardware Trojan can leak this secret key to an adversary once it is activated.

The research community has been extensively studying the taxonomy of hardware Trojans, their implementation, and detection in cryptosystems (*i.e.,* AES, RSA, and ECC). These Trojans are designed to leak the secret key from the circuits, either through side-channels [5]–[7] or primary outputs [8]–[12]. Over the years, various detection and prevention methods have been proposed to address the threat originated from hardware Trojans. The detection methods can be categorized – (i) logic testing [13]–[15], (ii) side-channel analysis [11], [16]–[18]. On the other hand, prevention methods include modifications in the design [19]–[21] and split manufacturing [22]–[24]. All of these techniques have some drawbacks in terms of their feasibility, the type of Trojans that can be targeted, and adverse effects due to manufacturing process variations. Besides, sequential Trojans manifest their effect only when a particular time has elapsed after the trigger condition is met or when the Trojan is triggered multiple times in a row. This property of sequential Trojan makes their detection very difficult. As a result, this type of Trojan can become a prominent and suitable choice for an attacker to launch tampering attacks.

In this paper, we show how an adversary can extract the secret key from different implementations of Advanced Encryption Standard (AES) by tampering the netlist with a hardware Trojan. AES performs a sequence of operations on the plaintext in multiple rounds that involves intermediate subkeys for each round, generated from the original secret key. Our attack relies on masking other intermediate subkeys' effect in an internal round through a sequential hardware Trojan. Once the Trojan is activated, it obstructs and modifies the data from all previous rounds. As a result, the input data

for the Trojan-affected round becomes all 1s if the Trojan's payload is an OR gate. We can also use an AND gate as the payload to make the input data all 0s. We refer this as an adversarial known value (*i.e.,* 0 or 1) because only the adversary pertains to the knowledge regarding this value and also the rare Trojan activation condition that would achieve this intentional alteration. The resultant output for this round can be observed directly from the primary output if a Trojan is implanted in the last round.

The contributions of this paper are described as follows:

- We propose a novel attack based on the malicious modifications of the hardware implementation of an AES core. The attack aims to modify the computation for an internal round and extract its corresponding intermediate subkey. For the same, we tamper the AES design with a sequential hardware Trojan. To the best of our knowledge, we are the first to demonstrate that the extraction of an intermediate key can be performed by inserting a sequential hardware Trojan, which can help an adversary for computing the original secret key. We propose to use the design for a sequential hardware Trojan due to its greater difficulty of detection during manufacturing tests and the normal functioning of the circuit. The addition of a state element (a counter) to the trigger of sequential Trojan requires triggered Q times consecutively, to deliver the payload.

- We demonstrate and validate our proposed attack on the OpenCores AES benchmark [25] synthesized in 32nm technology using Synopsys Design Compiler. The area and power overhead resulted from inserting a sequential hardware Trojan are negligible compared to the AES core.

The rest of the paper is organized as follows. First, we describe the AES structure in Section II. We present the proposed attack and its methodology on different AES implementations in Section III. Experimental results related to hardware Trojan and the proposed attack are shown in Section IV. Finally, we conclude the paper and provide future directions in Section V.

II. Background

Advanced Encryption Standard (AES) is a widely used block cipher for data encryption recommended by the National Institute of Standards and Technology (NIST) in November 2001 [26]. An adversary can tamper the AES core with a hardware Trojan as the implementation details of AES are publicly available. In this section, we provide a detailed description of the AES core and a hardware Trojan, which can be used to launch the tampering attack described in Section III.

A. AES Block Cipher

AES is the widely popular block cipher used almost in every secure application. It consists of multiple rounds of operations (*e.g.,* 10, 12, and 14) for different key sizes (*e.g.,* 128, 192, and 256, respectively). Each round (R_i with $i \in \{1, 2, \ldots, n\}$) consists of SubBytes (SB), ShiftRows (SR), MixColumns (MC), and AddRoundKey (AK) layers, except for final round without the MixColumns computation [26].

The intermediate round computations are usually represented by a 4×4 matrix, where each cell represents a byte. Note that the subscript for any variable represents the round number and superscript represents the accessible subgroups within that variable. The same notations are used and referred throughout the paper. We denote the input of the ith round by $A_i{}^j$, where $j \in \{0, 1, \ldots, 15\}$. The internal round keys (K_i) are generated from the key expansion modules. These intermediate keys corresponding to each round can be denoted as subkeys. These subkeys are bitwise XORed with the output of the MixColumns (or ShiftRows for the final round). The key bytes are arranged into a matrix with 4 rows and 4 (128-bit key), 6 (192-bit key) or 8 (256-bit key) columns. In this paper, we only focus on AES with a 128-bit key to demonstrate the tampering attack for simplicity. This same attack can be launched for AES with 196 and 256-bit keys as well without changing the attack methodology.

One can find the details for each round of AES in [26] and can be summarized in different layers described as follows:

1) SubBytes (SB): It is the nonlinear transformation step in AES, where each state byte is swapped with a precomputed value from a look-up table known as s-box.
2) ShiftRows (SR): This step rotates the 4×4 state matrix with different known offsets. Rows are shifted in a cyclic manner by 1, 2 and 3-bytes for the corresponding row number in the state matrix.
3) MixColumnns (MC): This step performs linear column-wise operations on the state matrix. Essentially, it is a matrix multiplication in the finite field of each column in the state matrix with a constant 4×4 matrix.
4) AddRoundKey (AK): It is the bitwise XOR of the state matrix with the corresponding subkey.
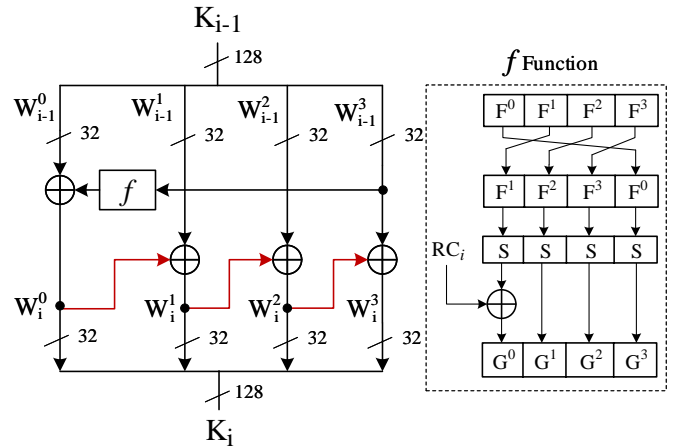

Figure 1: Key schedule module for 128-bit AES implementation.

The key schedule (KS) module generates subkeys for each rounds. As the AES primitive with 128-bit key (can be referred as AES-128) has 10 rounds, it is necessary to create 10 subkeys ($K_1, \ldots, K_{10}$) of 128-bit each. Following the notations, original key for AES K consisting of $\{k^0, k^1, \ldots k^{15}\}$, where each subgroup comprises of 8-bits respectively. Figure 1 shows

the implementation details of the key schedule module for the ith round, where the current round's subkey (K_i) can be computed from the previous round's subkey (K_{i-1}). The subkey operations are performed on a word length (32-bits or 4 bytes) subgroup. These subgroup words for each round can be represented as W_i^r, where r and i represent word index and round index, respectively. For i^{th} round, W_i^0 is comprised of $\{k_i{}^0, k_i{}^1, k_i{}^2, k_i{}^3\}$. Similarly, W_i^1 is formed with $\{k_i{}^4, k_i{}^5, k_i{}^6, k_i{}^7\}$ and so on.

As shown in the Figure 1, we can generalise the operation for the key schedule module and be described as:

$$W_i^0 = W_{i-1}^0 \oplus f(W_{i-1}^3); \qquad W_i^1 = W_i^0 \oplus W_{i-1}^1$$
$$W_i^2 = W_i^1 \oplus W_{i-1}^2; \qquad W_i^3 = W_i^2 \oplus W_{i-1}^3$$

where, f function can be formalised as:

$$G^0 = S(F^1) \oplus RC_i; \qquad G^1 = S(F^2)$$
$$G^2 = S(F^3); \qquad G^3 = S(F^0)$$

where, the values of RC_i can be found in [26].

Note that the detailed implementation of the key schedule module will help to compute the previous round's subkey (K_{i-1}) and finally the secret key K, if any of the subkey (K_i) is known.

B. Design for a Sequential Hardware Trojan

In this paper, we consider the design of a sequential Trojan to demonstrate the attack. Upon triggering, a sequential Trojan manifests it effect after the occurrence of a sequence or a period of time. Generally, Trojan comprises of a trigger and payload that can be activated through trigger inputs, which are taken from the primary inputs and/or internal nodes of a circuit. The Trigger inputs are selected such that the Trojan can evade manufacturing or production test patterns (*e.g.,* stuck-at fault tests, and delay tests) [27]–[29]. A *Type-p* Trojan comprises of p trigger inputs. The trigger is selected as an AND gate. However, any other combinational logic can also form the trigger which provides logic 1 when activated. Along with this AND gate, the sequential Trojan trigger includes a state element (Q-State counter). Upon availability of trigger inputs, the output of this AND gate becomes 1 (*i.e., $EN = 1$*) and the counter is incremented by 1. Upon triggering the Trojan Q-times consecutively, the counter reaches the maximum value and delivers the payload in the original circuit through the OR gate or XOR gate. The finite state machine (FSM) for the counter (CTR) is shown in Figure 2. The state transition occurs only when $EN = 1$, otherwise, it returns to the initial state, S_0. The output of the counter becomes 1, once EN is made to logic 1 consecutively for Q clock cycles. An adversary may choose any different design of a Trojan as well.

III. PROPOSED TAMPERING ATTACK ON AES WITH A HARDWARE TROJAN

The important aspect of cryptographic primitives is to encrypt the output in such a way that an adversary cannot find any key information at the output. In other words, no key information is leaked at the output and an adversary cannot
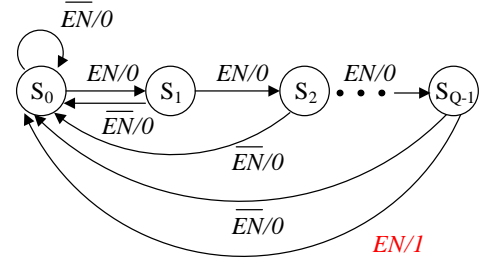

Figure 2: Finite state machine of the counter used in a sequential hardware Trojan.

determine the key by observing input/output responses of a system. In this section, we show how an adversary can extract the secret key using the proposed tampering attack with a sequential hardware Trojan.

A. Threat Model

The threat model is described to clearly identify the capabilities of an adversary. In this model, an untrusted foundry is considered as an adversary with the following capabilities:

- It has access to the netlist of the crypto primitives. The untrusted foundry has access to all the layout and mask information, which can be obtained from the GDSII or OASIS file. The netlist can be reconstructed from this information using reverse engineering [30].
- The attacker has the capability to modify the netlist so that it can tamper it with a hardware Trojan.
- The attacker has access to all the manufacturing test (*e.g.,* stuck-at fault and delay fault) patterns as it is common that production tests are performed at the foundry. The adversary can utilize these test patterns to design a Trojan which cannot be detected during manufacturing tests [27].

B. Attack Methodology

The proposed attack relies on tampering the netlist by an untrusted foundry with the aim of exposing the secret key. Once the secret key is exposed, the security of AES no longer exists. With this aim, an efficient two-step methodology is proposed that involves a sequential Trojan. The proposed attack can be described as follows:

- *Step 1*: The first step is to implement the hardware Trojan and place its payload in the netlist. Once activated, the Trojan masks the information obtained from previous round computations and nullify the impact of previous subkeys. The AddRoundKey (AK) layer is our primary area of interest while tampering the circuit with a hardware Trojan and modify intermediate round state matrix (A_i) to all 1s or 0s depending on the payload. In this paper, we treat the payload as OR gates, and thus A_i becomes all 1s. Once the response is collected at the primary output, an adversary then computes the secret key using *Step 2*.
- *Step 2*: The computation of the secret key is performed in this step from the Trojan activated response. As the design for the key schedule module is publicly available, an
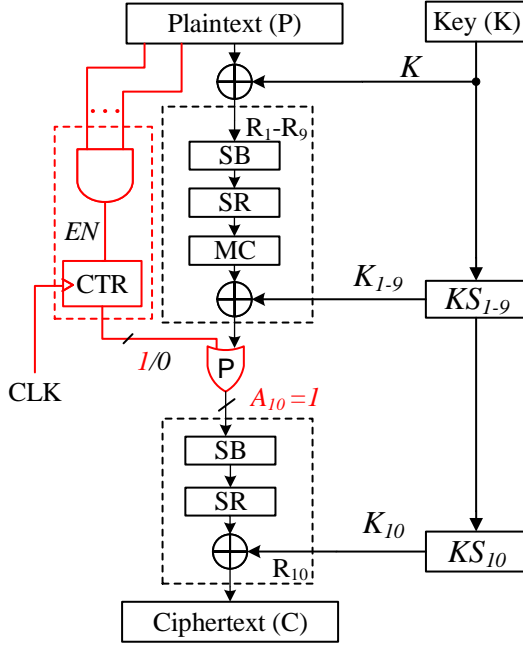
Figure 3: Tampering attack on AES design implanted with a sequential hardware Trojan, delivering the payload at the data input (A_{10}) of the last round (R_{10}).

adversary can compute the input data of the key schedule module from its output (see the computation details in Algorithm 1 presented in Section III-C). With this, any subkey can be traced back to retrieve the previous round key (K_{i-1}) and finally, the original secret key (K).

C. Tampering Attack on AES Core

Figure 3 shows the proposed tampering attack on 128-bit the AES implementation. The netlist is implanted with *Type-p* sequential hardware Trojan, which consists of *p*-input trigger and a payload OR gate. The trigger inputs may come from the input depicted as plaintext (P) in the figure. The trigger circuit comprises of a AND gate and counter (CTR) with the maximum count number of Q. One can find the detailed description of this sequential Trojan in Section II-B. We propose to place the payload OR gate(s) before the last round (R_{10}) computation, *i.e.,* after the add round key, *AK* layer in R_9. Once the Trojan is activated, the output of the payload (P) OR gate(s) becomes 1. As a result, all the bits in the input state matrix for last round $A_{10} = 1$, consequently, masking the effect of all previous round key information ($K_1 - K_9$) and round computations ($R_1 - R_9$). Due to availability of the implementation details, we can compute the K_{10} from the observed ciphertext C, and described as follows:

1) The activated Trojan delivers the payload to modify the input to the last round with $A_{10} = 0\mathrm{x}FF\ldots FF$.
2) The output of SubBytes (SB) layer of R_{10} can be computed as:
$$Y_1 = SB(A_{10})$$
$$= 0\mathrm{x}1616\ldots1616$$

3) Once Y_1 is known, the output of ShiftRows (SR) layer can be computed as:
$$Y_2 = SR(Y_1)$$
$$= 0\mathrm{x}1616\ldots161616$$

Since the last round R_{10} does not perform MixColumn (MC) operation, the output of ShiftRow (SR) gets XORed with subkey (K_{10}) in AddRoundKey (AK) step which is the final ciphertext (C) at the primary output. From this output bitwise XOR being a symmetric operation, we can calculate the subkey (K_{10}) as:
$$K_{10} = C \oplus Y_2$$
$$= C \oplus (0\mathrm{x}1616\ldots1616)$$

Once K_{10} is retrieved, an adversary can recover all the previous subkeys and the original secret AES key. In the following, we will show how the subkey K_9 can be recovered from K_{10}. Here, W_{10}^0, W_{10}^1, W_{10}^2 and W_{10}^3 are known as the value of K_{10} has been evaluated from the ciphertext previously using activating the hardware Trojan.

1) *Step 1*: Computation of W_9^3 can be performed from XORing the W_{10}^3 with W_{10}^2 as the XOR operation is reciprocal.
$$W_9^3 = W_{10}^3 \oplus W_{10}^2$$

2) *Step 2*: Once W_9^3 is known, one can compute W_9^0 using the following equation.
$$W_9^0 = W_{10}^0 \oplus f(W_9^3)$$

3) *Step 3*: Finally, W_9^1 and W_9^2 can be evaluated using the following equations.
$$W_9^1 = W_{10}^1 \oplus W_{10}^0; \qquad W_9^2 = W_{10}^2 \oplus W_{10}^1$$

The general process for evaluating the secret key K from K_{10} is described in Algorithm 1. The round subkey K_{10} is provided as input to the algorithm and the original key K will be returned as the output. The algorithm starts by selecting the subkey from which the previous round subkey is to be calculated (Line 1). The subkey (K_i) is divided into 4 subgroups of 32-bits each, namely $[W_i^0, W_i^1, W_i^2, W_i^3]$, from the 128-bit key using the *assign* function (Line 2). The 4 subgroups for $(i-1)^{th}$ key is calculated from the ith key (refer Figure 1 in Section II) (Lines 3-6). Finally, 4 different 32-bit subgroups (*i.e.,* $W_0^0, W_0^1, W_0^2, W_0^3$) are obtained for the original key which are concatenated together and the algorithm reports the original key K (Lines 8-9). During these operations, function f is used which takes inputs as the 32-bit subgroup word W (Line 17). Function f performs the SubByte operation on 8-bits of keys using the sbox (S) and RC_i (corresponding to each round) and returns the concatenated result (Lines 18-22).

Algorithm 1: Reverse Key Schedule

Input: SubKey (K_{10}) of round R_{10}
Output: Original Key (K)

1 **for** $i = 10$ **to** 1 **do**
2   $\quad [W_i^0, W_i^1, W_i^2, W_i^3] \leftarrow assign(K_i)$;
3   $\quad W_{i-1}^3 \leftarrow W_i^3 \oplus W_i^2$;
4   $\quad W_{i-1}^2 \leftarrow W_i^2 \oplus W_i^1$;
5   $\quad W_{i-1}^1 \leftarrow W_i^1 \oplus W_i^0$;
6   $\quad W_{i-1}^0 \leftarrow W_i^0 \oplus f(W_{i-1}^3)$;
7 **end**
8 $K \leftarrow \{W_0^0 \parallel W_0^1 \parallel W_0^2 \parallel W_0^3\}$;
9 Report K ;

10 **Function** $assign(K_i)$ **:**
11   $\quad [k_i^1, k_i^2 \ldots k_i^{16}] \leftarrow K_i$;
12   $\quad W_i^0 \leftarrow \{k_i^0 \parallel k_i^1 \parallel k_i^2 \parallel k_i^3\}$;
13   $\quad W_i^1 \leftarrow \{k_i^4 \parallel k_i^5 \parallel k_i^6 \parallel k_i^7\}$;
14   $\quad W_i^2 \leftarrow \{k_i^8 \parallel k_i^9 \parallel k_i^{10} \parallel k_i^{11}\}$;
15   $\quad W_i^3 \leftarrow \{k_i^{12} \parallel k_i^{13} \parallel k_i^{14} \parallel k_i^{15}\}$;
16   $\quad$ Return $[W_i^0, W_i^1, W_i^2, W_i^3]$;

17 **Function** $f(W)$ **:**
18   $\quad G^0 \leftarrow S(k^{13}) \oplus RC_i$;
19   $\quad G^1 \leftarrow S(k^{14})$;
20   $\quad G^2 \leftarrow S(k^{15})$;
21   $\quad G^3 \leftarrow S(k^{12})$;
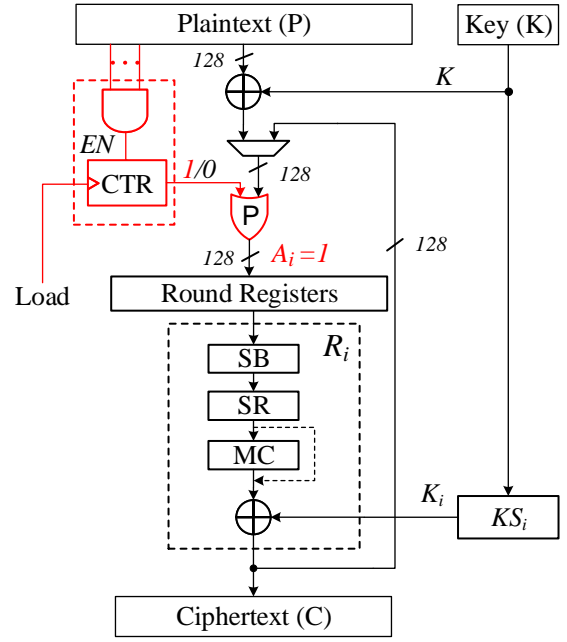22 $\quad$ Return $[G^0, G^1, G^2, G^3]$;

Figure 4: Tampering attack on OpenCores AES benchmark with a sequential hardware Trojan.

$K_{10} = C \oplus (0x1616 \ldots 1616)$. Finally, the original key (K) can be retrieved using the Algorithm 1. Note that the attacks on AES are explained using a sequential Trojan. One can use other types of existing hardware Trojan designs to launch the attack as well.

IV. RESULTS AND DISCUSSIONS

To validate the effectiveness of our proposed attack, we implemented our proposed hardware Trojan in the OpenCores AES benchmark [25]. The sequential Trojan Trigger comprised of a AND gate followed by a counter. The Trigger inputs to the AND gate were taken directly from the plaintext input to the AES core. The trigger pattern was selected in such a way so that it does not belong to the manufacturing test patterns (*e.g.,* stuck-at fault patterns) [27]–[29].

Table I: Area overhead analysis.

Max Count (Q)	2	4	8	16	32	64
Area Overhead (%)	0.51	0.53	0.55	0.58	0.60	0.63

The area for a hardware Trojan can vary based on the trigger input, trigger design, and the type of Trojan selected. For our experimentation, we implemented a sequential hardware Trojan with 5-input AND gate, counter with maximum count Q, and 128 payload OR gates. To estimate the area and power overhead, the Trojan inserted AES benchmark is synthesized with 32nm technology [31] using Synopsys Design Compiler [32]. Table I shows the percentage area overhead obtained by comparing the Trojan-free and Trojan inserted AES benchmark. The overall area overhead for counter with different maximum count value Q. The higher value of Q increases the difficulty of detecting a hardware Trojan using a logic test, as it is increasingly difficult to trigger the Trojan

D. Tampering Attack on OpenCores AES Benchmark

The utilization of the hardware resources can be reduced by adopting multicycle designs, that re-use the hardware or functional blocks in a design. For the OpenCores AES [25] implementation, the plaintext (P) is provided as the input, and the internal result after every round is stored in round registers, which is then fed back to the SubByte layer of the next round. The result in the round register after 10th round (R_{10}) is the ciphertext (C) output of the AES core.

Figure 4 shows the hardware implementation of a Open-Cores AES benchmark. The design has a *load* input pin which loads the input plaintext for encryption. The total encryption process takes 13 clock cycles once the *start* signal is assigned and the ciphertext is observed at the output when done signal becomes 1 [25]. The proposed tampering attack can also be extended to this OpenCores AES design, which is tampered with a sequential Trojan described in section II-B. The counter (CTR) of the hardware Trojan uses the *load* signal as a clock. In other words, the Trojan counter will increase its value when the *load* signal is high as well as the plaintext matches the trigger input pattern (*i.e.,* $EN = 1$). It is necessary to trigger the Trojan Q times consecutively to launch the attack. Once activated, the payload is delivered to make A_i all 1's. The Trojan will remain activated during the entire encryption process and all the internal round computation will be modified as well. The ciphertext (C) observed at the output will be used to determine K_{10}, which can be computed as:

Q times consecutively. The overhead is minimal and is less than 1%. For example, it is only 0.55%, when we choose Q of 8. Note that the majority of the overhead comes from the payload as we require 128 OR gates. Since the Trojan remains quiet during normal operation, it does not have any switching power. The leakage power for the Trojan would only contribute to the power overhead. However, the Trojan's leakage power is in order of magnitude less than the switching power of the counter and can be negligible.

V. Conclusion

Hardware Trojans can pose a severe threat to our critical infrastructure that relies on AES for encrypting sensitive data. We presented a novel tampering attack on AES core to extract the secret key by implanting a sequential hardware Trojan. The attack mainly relies on modifying the input for an internal round using the Trojan's payload, to mask the previous round's information. The Trojan helps an adversary to compute the last round subkey from the observed ciphertext. We present an algorithm to retrieve the original key from the last round's subkey. The sequential Trojan presented in the paper requires triggering of Q consecutive times, which fulfills the requirement of increased difficulty in detecting such Trojans. It is extremely difficult to identify such Trojans using logic testing as the attacker only knows the trigger condition derived from the input plaintext, and apply it repeatedly.

Acknowledgment

References

[1] Age Yeh, "Trends in the global IC design service market," DIGITIMES Research, 2012.

[2] M. Tehranipoor and C. Wang, *Introduction to hardware security and trust*. Springer Science & Business Media, 2011.

[3] S. Adee, "The hunt for the kill switch," *iEEE SpEctrum*, vol. 45, no. 5, pp. 34–39, 2008.

[4] E. Barker, "NIST Special Publication 800-175B NIST Special Publication 800-175B Cryptographic Standards in the Federal Government: Cryptographic Mechanisms," 2016.

[5] L. Lin, M. Kasper, T. Güneysu, C. Paar, and W. Burleson, "Trojan side-channels: Lightweight hardware trojans through side-channel engineering," in *International Workshop on Cryptographic Hardware and Embedded Systems*, 2009, pp. 382–395.

[6] L. Lin, W. Burleson, and C. Paar, "MOLES: malicious off-chip leakage enabled by side-channels," in *IEEE/ACM International Conference on Computer-Aided Design-Digest of Technical Papers*, 2009, pp. 117–122.

[7] Y. Zhao, J. Song, X. Wu, L. Wu, and X. Zhang, "A Novel Trojan Side Channel For Attacking Masking," in *IEEE International Conference on Anti-counterfeiting, Security, and Identification*, 2018, pp. 151–154.

[8] S. Kutzner, A. Y. Poschmann, and M. Stöttinger, "Hardware trojan design and detection: a practical evaluation," in *Proceedings of the Workshop on Embedded Systems Security*, 2013, pp. 1–9.

[9] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware trojans," in *Int. Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2013, pp. 197–214.

[10] S. Bhasin, J.-L. Danger, S. Guilley, X. T. Ngo, and L. Sauvage, "Hardware Trojan horses in cryptographic IP cores," in *Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2013, pp. 15–29.

[11] M. Muehlberghuber, F. K. Gürkaynak, T. Korak, P. Dunst, and M. Hutter, "Red team vs. blue team hardware Trojan analysis: detection of a hardware Trojan on an actual ASIC," in *Proceedings of the International Workshop on Hardware and Architectural Support for Security and Privacy*, 2013, pp. 1–8.

[12] S. Bhunia, M. S. 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.

[13] P. Kitsos, D. E. Simos, J. Torres-Jimenez, and A. G. Voyiatzis, "Exciting FPGA cryptographic Trojans using combinatorial testing," in *International Symposium on Software Reliability Engineering*, 2015, pp. 69–76.

[14] X. Wang, M. Tehranipoor, and J. Plusquellic, "Detecting malicious inclusions in secure hardware: Challenges and solutions," in *IEEE Int. Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 15–19.

[15] H. Salmani, "COTD: Reference-free hardware trojan detection and recovery based on controllability and observability in gate-level netlist," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 2, pp. 338–350, 2016.

[16] B. Hanindhito and Y. Kurniawan, "Hardware Trojan Design and Its Detection using Side-Channel Analysis on Cryptographic Hardware AES Implemented on FPGA," in *International Conference on Electrical Engineering and Informatics (ICEEI)*, 2019, pp. 191–196.

[17] Y. Liu, K. Huang, and Y. Makris, "Hardware Trojan detection through golden chip-free statistical side-channel fingerprinting," in *Proceedings of the Annual Design Automation Conference*, 2014, pp. 1–6.

[18] J. He, Y. Zhao, X. Guo, and Y. Jin, "Hardware trojan detection through chip-free electromagnetic side-channel statistical analysis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2939–2948, 2017.

[19] L. Wu, X. Wang, X. Zhao, Y. Cheng, D. Su, A. Chen, Q. Shi, and M. Tehranipoor, "AES design improvement towards information safety," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2016, pp. 1706–1709.

[20] K. Xiao, D. Forte, and M. Tehranipoor, "A novel built-in self-authentication technique to prevent inserting hardware trojans," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 1778–1791, 2014.

[21] P.-S. Ba, S. Dupuis, M. Palanichamy, M.-L. Flottes, G. Di Natale, and B. Rouzeyre, "Hardware trust through layout filling: A hardware Trojan prevention technique," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 254–259.

[22] K. Vaidyanathan, B. P. Das, and L. Pileggi, "Detecting Reliability Attacks During Split Fabrication Using Test-Only BEOL Stack," in *Proc. of Design Automation Conf.*, 2014, pp. 1–6.

[23] J. J. V. Rajendran, O. Sinanoglu, and R. Karri, "Is Split Manufacturing Secure?" in *Proc. Conf. Design, Automation and Test in Europe (DATE)*, 2013, pp. 1259–1264.

[24] Y. Wang, P. Chen, J. Hu, and J. J. Rajendran, "The Cat and Mouse in Split Manufacturing," in *Proceedings of Design Automation Conference*, 2016, pp. 1–6.

[25] Opencores, https://opencores.org/projects/aes_crypto_core.

[26] NIST-FIPS, "Announcing the Advanced Encryption Standard (AES)," *Federal Information Processing Standards Publication*, vol. 197, no. 1-51, pp. 3–3, 2001.

[27] Z. Zhou, U. Guin, and V. D. Agrawal, "Modeling and test generation for combinational hardware Trojans," in *VLSI Test Symposium (VTS)*, 2018, pp. 1–6.

[28] A. Jain, Z. Zhou, and U. Guin, "TAAL: Tampering Attack on Any Key-based Logic Locked Circuits," *arXiv preprint arXiv:1909.07426*, 2019.

[29] A. Jain, U. Guin, M. T. Rahman, N. Asadizanjani, D. Duvalsaint, and R. S. Blanton, "Special Session: Novel Attacks on Logic-Locking," in *IEEE VLSI Test Symposium (VTS)*, 2020, pp. 1–10.

[30] R. Torrance and D. James, "The state-of-the-art in IC reverse engineering," in *International Workshop on Cryptographic Hardware and Embedded Systems*, 2009, pp. 363–381.

[31] Synopsys 32/28nm Generic Library for teaching IC design, Available: https://www.synopsys.com/community/university-program/teaching-resources.html.

[32] RTL Design and Synthesis: Next Generation RTL Design for Advanced Nodes, Synopsys, https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test.html.