IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received February 5, 2022, accepted April 14, 2022, date of publication April 20, 2022, date of current version May 2, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3168991

Ransomware Attack as Hardware Trojan: A Feasibility and Demonstration Study

FELIPE ALMEIDA, MALIK IMRAN, (Student Member, IEEE), JAAN RAIK, (Member, IEEE), AND SAMUEL PAGLIARINI, (Member, IEEE)

Centre for Hardware Security, Department of Computer Systems, Tallinn University of Technology, 12616 Tallinn, Estonia

Corresponding author: Felipe Almeida (felipe.almeida@taltech.ee)

This work was supported in part by the European Union through the European Social Fund through the Project "ICT Program," and in part by the Estonian Research Council under Grant MOBERC35.

ABSTRACT The integrated circuit (IC) ecosystem, today, is widely distributed. Usually, a handful of companies is involved in the development of a single chip – an environment that presents many opportunities for malicious activities such as the insertion of hardware trojan horses. This work presents a specialized form of a hardware trojan that is able to mount a hardware-based ransomware attack, an attack that previously only existed in the software domain. This attack is therefore termed a hardware ransomware and is the main contribution of this work. As case studies, two architectures of the hardware ransomware are presented, along with a silicon demonstration in 65nm CMOS. In order to discuss the detectability of the malicious logic, the hardware ransomware is inserted in a complex system on chip (SoC). The experimental results show how an adversary can effortlessly insert the ransomware logic: the baseline SoC has a representative area utilization factor of 59.97% and, after the trojan is inserted, the area utilization factor increases by 0.73% to 60.70%. The inserted logic is also responsible for an increase of approximately 2% in static power – well within process variation margins. Finally, this paper discusses the implications of such an attack at length, showing that from the implementation and technological side, there are no barriers for an adversary to devise a hardware ransomware.

INDEX TERMS Ransomware attack, hardware security, hardware trojan horse, malicious logic, ASIC.

I. INTRODUCTION

Today, there are enormous challenges to protect networks, servers, personal computers and devices. It is estimated that the (cyber)security segment is a 100 billion dollar industry [1], with security solutions for hardware and infrastructure receiving a lot of increased attention. Adversaries are typically interested in accessing, modifying, or destroying information they are not privy to, but may also be interested in applying/causing a denial of service (DoS), leaking information, or extorting money from users. The latter type of threat is often termed as a *ransomware* and is the subject of this paper.

Ransomware is a type of malicious software (i.e., malware) that is designed to encrypt or limit the access of user data [2]. Typically, a ransomware is triggered by the user himself or by a timing event, which is then followed by an encryption of all of user data. Finally, a ransom is demanded to be paid for the decryption, typically handled through the anonymity of a Bitcoin transaction. In recent years, ransomware became a widespread concern. For instance, WannaCry infected tens of thousands of computers in over 150 countries [2], [3] in an attack in which the attackers demanded $300 per infected computer. Cryptolocker [4], which is another infamous example, was responsible for tens of millions of dollars in extortion when it first emerged. Several large organizations, such as England's NHS, have been affected by ransomware attacks [5].

It must be highlighted that ransomware attacks are often possible due to software and network vulnerabilities. Meanwhile, there is a whole array of other vulnerabilities and attacks that are studied in the domain of hardware security (e.g., backdoor insertion [6] and hardware trojan horses [7]). Backdoors and trojans are malicious logic that is inserted by an adversary. Backdoors can attempt to give an adversary privileged access to a given functionality of the integrated circuit (IC), while trojans might just attempt to corrupt some

The associate editor coordinating the review of this manuscript and approving it for publication was Aneel Rahim.

computation. Then, the question we are interested in answering in this paper is the following: *can an adversary mount a ransomware attack as a hardware trojan?*

In this feasibility study, we assert that, by combining cryptographic hardware and a key-generation scheme, a ransomware attack can indeed be mounted in hardware. We term this attack a **hardware ransomware**, which can be considered a specialized case of a hardware trojan. No demonstration of such an attack exists prior to our work, although it has been hypothesized in [8].

A. FEASIBILITY OF A HARDWARE RANSOMWARE

Before discussing the technical details of our work, we must address the feasibility of such an attack. Differently from software, the attacker has to balance a **much more complicated risk-reward equation**. First, not all systems are good candidates for being *infected* by hardware ransomware. The attack only makes sense if the ransomware targets a system that carries persistent data that has value to the user, such that the victim would be motivated to pay a ransom. On the other hand, if this system also runs software and an Operating System, the software ransomware attack vector remains more practical since it can be mounted by a much less capable adversary. An external hard drive is an example of a system that could be the target of hardware ransomware.

In the case of hardware ransomware, we have some stringent limitations that do not exist for its software counterpart, especially regarding how to communicate with the victim. In software, a ''popup'' warns the victim that the attack has taken place and provides instructions for payment. In hardware, a similar communication channel can be established if the ransomware can write files to the storage system. This, in turn, incurs a significant overhead since the malicious logic has to *understand* the filesystem of the storage. We address this limitation in our discussion.

On the detection side, both software and hardware versions of ransomware have to be stealth – it must **not** become obvious for a victim that his/her system is infected before the ransomware attack is executed. Let us assume an attacker has managed to successfully insert ransomware-like logic into an IC that was mass-produced. The attacker has to be insightful when orchestrating the trigger condition for this ransomware. Otherwise, once the first victim assigns blame to the infected IC, other potential victims would proceed to perform data backups or simply replace the IC altogether. For this reason, the authors of [8] make an argument that attackers can benefit not only from straightforward payment of ransom, as well as from causing the stock value of a targeted company to drop. On that premise, our focus on this paper has been to show the technical feasibility of a ransomware attack in hardware.

B. COMPARISON TO EXISTING THREATS: SOFTWARE RANSOMWARE AND HARDWARE TROJANS

1) SOFTWARE RANSOMWARE:

The most obvious comparison to be made is against the existing threat of a software ransomware. To guide this comparison, we note that a software ransomware has three distinct components: trigger, cryptographic payload, and user interface. The trigger is the condition at which the attack starts to execute and, in many cases, it is linked to an action by the user of a system (e.g., when downloading an infected file). For a hardware-based ransomware attack, this is not the case. The malicious logic has to already be in place from the time the circuit is fabricated.

Regarding the cryptographic payload, here the difference between software and hardware is non-existent: there are known open-source implementations of cryptographic protocols that an adversary can select from, both in software and in hardware.

From the user interface point of view, a software-based ransomware has much more flexibility when demanding the ransom payment from the user. The same cannot be said about hardware; this remains the most important difference between the two and has a severe impact on the feasibility as already discussed.

2) HARDWARE TROJAN:

Generally speaking, a hardware trojan is a malicious modification of the circuitry of an IC. Among the many types of hardware trojans described in the literature, the functional type aims to corrupt some internal data of the IC. We will show that the herein described hardware ransomware has the same characteristic since the user data is encrypted with an unknown key, thus appearing to be corrupted from the user point of view. Therefore, the main difference between a traditional hardware trojan and a hardware ransomware is the ransom demand and the subsequent reversal of the attack via decryption.

Traditional hardware trojans are also characterized by a trigger and a payload, where the trigger can be a long sequence of events that have to take place before the payload acts on the circuit. This characteristic remains in the hardware ransomware since it is a specialized case of a hardware trojan. However, the payload of a hardware ransomware is a cryptographic core that requires a non-negligible amount of area.

C. CONTRIBUTIONS

We summarize the contributions of this paper as follows:

- Proposal of two hardware ransomware architectures, one optimized for latency and another that is area-optimized. No software/processor components are utilized in either.
- A **silicon demonstration** of a hardware ransomware as a standalone ASIC. Our demonstration is optimized for low leakage power and small footprint.
- Presentation and analysis of measurement data from the parts fabricated in a 65nm CMOS technology.
- Implementation of both architectures in a Field Programmable Gate Array (FPGA)-specific form.

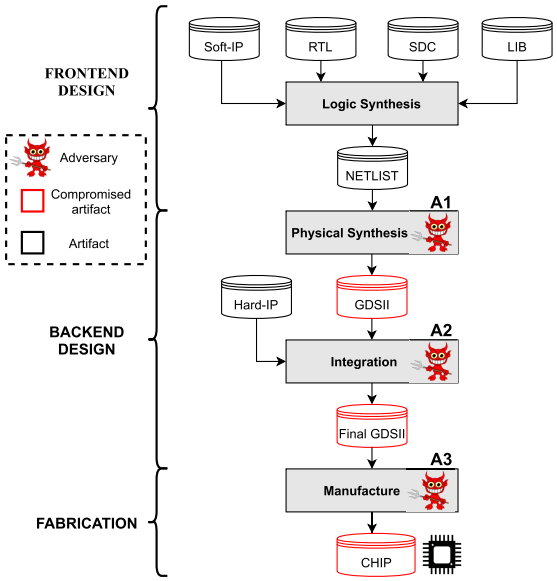


FIGURE 1. Representative ASIC design flow, possible attackers, and compromised design artifacts.

- A case study of the insertion of the low-footprint hardware ransomware in a complex System on Chip (SoC) and the related discussion on detection.

The remainder of this work is organized as follows: in Section II, a background pertaining to attackers and the IC design flow is presented. The proposed hardware architectures for ransomware are described in Section III. Pre-silicon and post-silicon results for the fabricated design are given in Section IV, while FPGA-based results are presented in Section V. A case study of ransomware insertion in a complex SoC design is given in Section VI. In Section VII, a comparison with known trojans is given, as well as a discussion on related limitations and avenues for future research. Section VIII concludes the paper.

II. BACKGROUND: ATTACKERS VS. DESIGN FLOW

The IC ecosystem, today, is widely distributed, both geographically and logistically. Usually, a handful of companies is involved in the many steps an IC goes through, including design, fabrication, test, packaging, etc. As a result of this spread, there are many opportunities for malicious attackers to act.

Application Specific Integrated Circuits (ASICs), in their vast majority, are developed following a standard cell-based design flow that is carried out by a set of Computer-Aided Design (CAD) tools. In Fig. 1, we show a design flow as a series of transformations that the design goes through: from a natural language specification to a netlist of standard cells to a physical layout. The representative design flow presents the locations of three distinct attackers that can mount a hardware ransomware. The attackers are labeled as A1-A2-A3. Their capabilities are discussed in Section II-A.

TABLE 1. Possible attackers and characteristics.

Attacker	Location	Role	Capability
A1	Design house	Block designer	Modify block layout
A2	Design house	Integration	Modify any layout but respect pinout
A3	Untrusted foundry	Manufacture or mask prep	Modify portions of a layout

A. THREAT MODEL

We consider three different attackers and, for the sake of brevity, name them A1, A2, and A3. Their locations in the design flow are indicated in Fig. 1. Their characteristics are given in Table 1. The goal of all considered attackers is to insert the hardware ransomware logic without detection. For all attackers, we can also make the assumption that they are **rogue elements** within their organizations.

Attacker A1 is an IC designer that is responsible for a given block. He/she does not enjoy chip-level visibility. He/she has no control over the top-level floorplan.

Attacker A2 is an IC designer that is responsible for the integration. He/she enjoys chip-level visibility. He/she has full control over the top-level floorplan, except for pinout.

Attacker A3 is a foundry engineer. He/she enjoys chip-wide visibility, albeit in a finalized layout form. He/she has no control over the top-level floorplan.

Generally speaking, any malicious modification of the circuitry of an IC can be referred to as a hardware trojan.[1] Trojans can be characterized by their physical representation and behavior. Moreover, there are various works where surveys and taxonomies for hardware trojans have been catalogued/proposed [9]–[11]. We have adopted the classification proposed by Rajendran *et al.* [10] where trojans are categorized based on: 1) insertion phase, 2) abstraction level, 3) activation, 4) effect, and 5) location on the design.

Regarding *insertion phase*, attackers A1 and A2 perform insertion at design time. Attacker A3 performs fabrication-time insertion. Consequently, the *abstraction level* for attacker A1 is register-transfer level (i.e., meaning the attacker can modify code to instantiate the malicious logic). Attacker A2, on the other hand, has a netlist description of the ransomware and treats it as another piece of the system. Attacker A3 is a rather special case: our assumption is that he/she leverages the engineering change order (ECO) flow of a commercial physical synthesis tool, meaning that the system, as he/she sees it, is a finalized layout and the malicious logic is a netlist. The attacker's job is to insert the netlist into a finalized layout with minimal invasiveness, which is exactly the motivation for the use of an ECO flow. The feasibility of this type of attack was first shown in [12] for relatively small designs.

[1]The terminology varies from author to author, with subtle differences between the definition of backdoors and trojans.

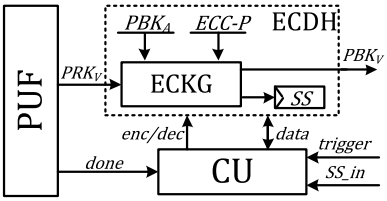


FIGURE 2. Block diagram of the proposed hardware ransomware.

For all considered attackers, having an internal trigger is preferred for *activation*. With respect to the *effect* of the hardware ransomware, the closest definition it can be mapped to is DoS. Finally, the *location* of the hardware ransomware changes according to the attacker: for A1, the ransomware logic becomes part of an existing block; for A2, the ransomware logic is placed at the system-level as a block of its own; finally, for A3, the ransomware logic lives in the gaps between the standard cells of a placed layout, i.e., the locations of filler cells. It should be evident that A2 has a privileged position and a relative ease to alter the logic of an IC.

III. PROPOSED HARDWARE RANSOMWARE DESIGN

Now that we have covered the design flow and associated threats, we shift the discussion to the specifics of our proposed ransomware circuit. A simplified block diagram of the proposed ransomware is given in Fig. 2. The design consists of three units: a Physical Unclonable Function (PUF), an encryption/decryption unit that employs an Elliptic Curve Diffie Hellman (ECDH) protocol, and a dedicated Control Unit (CU) that is based on a Finite State Machine (FSM).

In short, our ECDH-based architecture makes use of two private/public key pairs (denoted as PRK_A, PRK_V, PBK_A, and PBK_V) and a shared secret (SS). The PUF unit is responsible for providing a chip-side key (PRK_V) as an input to the ECDH unit and for notifying the CU through the *done* signal when PRK_V has been transmitted.

As shown in Fig. 2, *trigger* is a one-bit signal which is used for the purpose to activate the ransomware. Therefore, an external trigger was implemented (in our design) to make the silicon demonstration more controllable (and easy to bring-up). However, in a realistic attack scenario, the attacker would not enjoy device access in order to assert the trigger. Being so, a **more sophisticated trigger mechanism** would be mandatory. A version of our ransomware using an internal trigger is detailed in Section IV-A.

Once the ransomware is activated by an external event (notice the input signal named *trigger* in Fig. 2), it works by itself without any intervention by the attacker. Then, the CU asserts the *enc* signal for the ECDH unit to perform data encryption. Conversely, the CU asserts the *dec* signal for decryption operation **only when** the value of *SS_in* (meaning the SS key applied externally) and the SS key generated internally are a match. The scheme works for the adversary has decided the ECC parameters, the constant value of PBK_V, and the also constant (and related) PRK_A. Once the adversary receives PBK_V, calculating SS is trivial, i.e., the user of the system sends PBK_V along with the hypothetical payment and, in exchange, receives SS back. For the sake of clarity, Fig. 2 does not capture the adversary side of the attack where he/she calculates SS. [2]

It is worth noting that the software version of a ransomware typically employs known cryptographic functions such as the Advanced Encryption Standard (AES) [13], Rivest–Shamir–Adleman (RSA) [14], and Elliptic Curve Cryptography (ECC) [15]. In particular, CTB_locker, Petya, and TeslaCrypt use ECC, which is also the strategy we employ in our ECDH block. ECC-based crypto cores are often regarded as very efficient (with respect to RSA), especially in terms of area, and therefore are a good fit for our design. Another aspect of software ransomware that is worth mentioning is that the ransom addresses can be unique to each infected machine, as is the case in the Locky family of ransomwares [16]. In our approach, we make use of PUFs to the same end.

The aforementioned units (PUF, ECDH, and Control Unit) are further described in the subsequent subsections.

A. PUF - PHYSICAL UNCLONABLE FUNCTION

A PUF is a structure that derives values from the physical characteristics of the IC. It can generate signatures that leverage (undesirable) manufacturing variability such as shifts in gate delay, threshold voltages, and many other physical characteristics [17]. The physical randomness creates a fingerprint – a unique value – for each device. Our PUF block, depicted in Fig. 2, is actually a wrapper that harvests this signature from a randomness source.

PUFs are divided into two categories: strong and weak. This classification is based on the number of Challenge-Response Pairs (CRPs) of a PUF. A strong PUF has an immeasurable number of CRPs while a weak PUF has one (or a limited number) of CRPs. Importantly, for the ransomware scenario, a weak PUF is sufficient. The most common source of weak PUFs are Static Random Access Memory (SRAM) and Ring-oscillator (RO) devices [18]. The use of SRAM as a PUF exploits the positive feedback loop in the SRAM bitcell.

Interestingly, despite FPGA devices being often SRAM-based, there is a limitation to use the SRAM bits of the FPGA fabric as a PUF: SRAM bits are often erased on reset or during the programming. Therefore, the SRAM bits have their 'randomness' lost after a power-on challenge. Effectively, although SRAM bits are plenty in an FPGA, they cannot be used for PUF purposes. On the other hand, an RO is another kind of weak PUF that relies on gate delay manufacturing variability [19]. Since FPGA devices are also

[2]The adversary side is not bound to any hardware limitations, so he/she may make use of the same infrastructure that a software ransomware adversary does.

ICs, they suffer from variability that can be exploited in an RO PUF.

In short, both SRAM and ROs can be utilized for generating unique signatures that enable the hardware ransomware attack. An attacker is **not bound** to using one or the other PUF. The use of an SRAM PUF is discussed in Section IV, while the use of an RO PUF is discussed in Section V.

B. ECDH – ELLIPTIC CURVE DIFFIE HELLMAN

In our architecture, the ECDH unit and its elliptic-curve-key-generator (ECKG) sub-unit are responsible for implementing the ECDH protocol. As shown in Fig. 2, the ECKG generates PBK_V and SS. To generate PBK_V, the ECKG unit takes elliptic-curve parameters (*ECC-P*) and PRK_V as inputs. To generate SS, the ECKG unit takes the attacker's public key (PBK_A) and PRK_V as inputs. Notice that PRK_V and SS never leave the architecture, while PBK_V does. We also highlight that we have used a predefined constant value for PBK_A, chosen by the attacker.

The employed ECKG unit is over $GF(2^m)$ with $m = 163$, i.e., all keys are 163 bits long. The initial elliptic-curve parameters, i.e., *ECC-P*, have been selected from NIST recommendations [20]. The ECKG unit consists of a register file unit, pipeline registers, and an elliptic-curve arithmetic unit. It is a modified version of [21], now containing an additional protocol layer and an interface to the PUF.

We have provided two different solutions to generate shared keys for the attacker and victim. The first solution, termed low-latency ECDH core (LL-ECDH), requires a reduced number of clock cycles at the expense of area. The second solution, termed low-area ECDH core (LA-ECDH), requires fewer hardware resources but the computation takes longer to complete. The key generation procedure implemented requires a total of 3426 and 162512 clock cycles for the LL-ECDH and LA-ECDH cores, respectively.

C. CONTROL UNIT

The final module of our proposed architecture is the CU, an FSM with four states (idle, keygen, encrypt, decrypt) that orchestrates the operation of the ransomware. As we previously alluded to, the ransomware starts the key generation and subsequent encryption after the *trigger* signal is asserted, which causes a transition from the idle to keygen state. Conceptually, a ransomware can be activated by the occurrence of an event inside or outside of the system it resides in. Both approaches can be of interest for an attacker: an external trigger provides better control over the attack, while an internal trigger is likely to be more difficult to detect. It is important to note that *trigger* (in our fabricated chip – this will be described later in detail in Section IV) is an external single bit signal that activates the ransomware on demand. This choice is purely for practical demonstration reasons.

The keygen state of the CU FSM handles the calculation of two keys: PBK_V, and SS. The PUF module generates the PRK_V and sends it to the ECDH module along with the *done* signal informing that PRK_V is ready – the ECDH block can now calculate SS. After that, the FSM transitions to the encrypt state. The CU starts to read address by address from the storage system. The data being read is sent to the ECDH block to encrypt using SS as the key. Next, the encrypted data is sent to the CU to be written back at the same address on the storage system. We assume a generic storage system (or its interface) is in place, such that the the adversary will be able to identify it and connect malicious logic to it. The encryption/decryption takes 32-bit chunks of data at a time.

Once the encryption process is concluded, the CU goes again into idle mode until the hypothetical ransom payment is executed. The attacker is able to calculate SS externally and provides this value back to victim upon payment. If the user provides the right SS key (using the *SS_in* signal, shown in Fig. 2) to the circuit, the CU notifies ECDH to start the decryption process by driving the *dec* signal. Here, the procedure is identical to the encryption: addresses are read one by one, decrypted, and written back to the storage, 32 bits at a time. This process is repeated until the last address is processed. The attack ends.

However, it is conceivable that the victim will turn the system on and off once the attack takes place. The ideal PUF for a hardware ransomware should provide the same response every time. It is not important that the PUF remains stable over a long period of time like in other applications, so concerns with lifetime degradation of the PUF are eased.

IV. ASIC IMPLEMENTATION AND SILICON DEMONSTRATION

In order to demonstrate that a hardware ransomware attack is technically feasible, we designed and manufactured an ASIC. The following assumptions were made with respect to the design: 1) the trigger is external to the device; 2) the storage is emulated [3] as an SRAM memory; 3) the adversary can connect his malicious logic to the address and data buses of the storage. In practice, this last assumption holds true if the adversary has system-wide visibility, as is the case for attackers A2 and A3. The motivation for carrying out this exercise in silicon was to demonstrate a **minimal working example** where the targeted system has only one block other than the malicious logic, i.e., the storage itself.

The ransomware silicon demonstration is carried out in a 65 nm CMOS technology. The design emulates an external hard drive as our case study application. It can be repurposed for any application that contains a storage system that holds **persistent user data**. In Fig. 3, we show a high-level diagram of the three main blocks of our design. The malicious logic on the left portion of the image corresponds to the ransomware architecture previously detailed in Section III, with the exception that SS is exposed to a debug unit. The storage system on the right side of the image corresponds to a single SRAM instance that is generated from a memory compiler provided by a partner silicon foundry. In the center of the image,

[3] An actual storage should be non-volatile and contain information that is of value to the user. SRAM memory does not present such characteristic, so the keyword here is **emulation**.

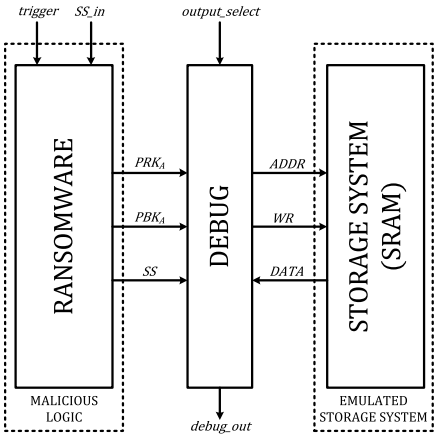


FIGURE 3. Top-level diagram (simplified) of our ASIC design that implements a hardware ransomware.

we highlight the use of a debug structure. Conceptually, this debug structure is not part of the system or the ransomware, but it is added to our demonstration so we can have visibility of the many circuit internals for validation purposes.

For coding and verification efforts, we have chosen the verilog language. The top-level design was synthesized using Cadence Genus and a foundry-provided 65 nm standard cell library. The resulting netlist was used for physical implementation in Cadence Innovus. For physical verification (DRC and LVS), we have used Calibre from Mentor Graphics. The design implements the LL-ECDH variant of our elliptic curve arithmetic unit.

In Fig. 4, we show the layout of our chip in which the major structures are highlighted. Some routing layers are removed for the sake of clarity. The visible structures are the SRAM PUF on the upper right corner, the storage system on the lower left, the ransomware core in the center of the chip (a sea of standard cells), and the debug module on the lower right corner. Both SRAMs have an exclusive power ring around them, so no standard cells are allowed in their vicinity. The PUF uses an SRAM with 32 address and 6-bit data lines. The storage system is also an SRAM but with 64 address and 32-bit data lines. IO cells are visible on the chip periphery, while power stripes are gridded and routed across the entire chip, horizontally and vertically.

The GDSII file was submitted to the foundry through a broker. The design was completed in March 2020, underwent fabrication in April-May, and parts were delivered a few months later when they were also bench tested at our in-house lab. A total of one hundred chips was fabricated, but only twenty were packaged in a Dual-In-Line-28 (DIP-28) form factor. In Fig. 5, we show a die shot of an unpackaged chip taken with the aid of a microscope. It is possible to recognize the same power routing stripes and IO cells as in the layout. A black circle is used to facilitate the identification of the lower right corner of the chip in both Fig. 4 and Fig. 5.

A custom PCB was fabricated to aid in the validation of our chip. The packaged chip is placed on a PCB with a DIP-28

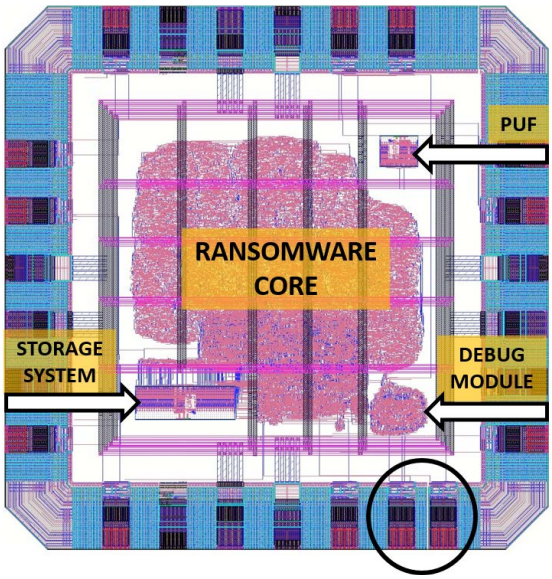


FIGURE 4. Screenshot of the chip layout generated in Cadence Virtuoso. Chip dimensions are 960μm x 960μm, but the ransomware core only requires a fraction of the area ($0.14\,\mathrm{mm}^2$). Major structures are highlighted. The ransomware is the LL-ECDH variant.

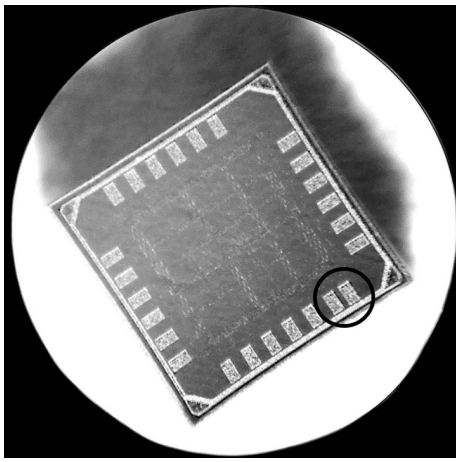


FIGURE 5. Microscope view of an unpackaged die where we can identify the same IOs and some power stripes on the top metal layer.

socket. The PCB also provides power to the core logic (1.2V) and to the IO cells (2.5V). A handful of decoupling capacitors is utilized for each power supply. A Zedboard FPGA [22] is utilized to drive/read the signals to/from the PCB. Moreover, the FPGA drives the trigger, as well as the clock, and collects the outputs of the chip. A Universal Asynchronous Receiver/Transmitter (UART) is used to communicate with a workstation that performs the same encryption/decryption process for verification of correctness of the chip outputs. The setup is shown in Fig. 6, also highlighting a high-precision multimeter for leakage measurement and an oscilloscope to aid in debugging.

The experiments with our fabricated ransomware show that after a trigger, all data in the storage are encrypted as

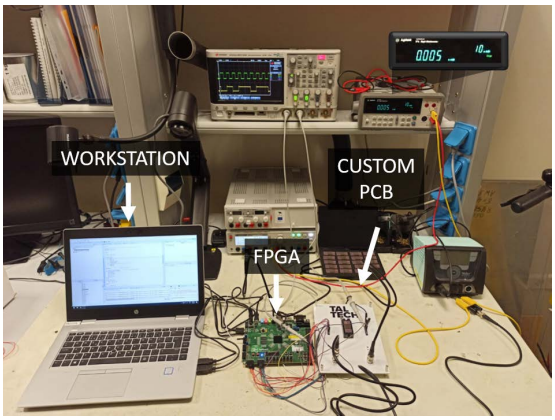


FIGURE 6. Setup utilized to validate our fabricated ransomware ASIC.

expected. Data are only decrypted after providing a correct key back to the chip, as expected. Switching between these different actions is managed by the *output_select* signal shown in Fig. 3, which can take the values of BYPASS ('00'), DUMP ('01'), and ACT ('10'). In BYPASS mode, the ransomware is completely ignored and a direct connection between signals *key_in* (input) and *debug_out* (output) is created. This is useful to verify that the fabricated parts are alive before more complex tests are performed. In DUMP mode, the debug module exposes the content of the storage system directly on the output of the chip. Finally, in ACT mode, the ransomware is allowed to come into operation.

When the debug module is active, it drives the *debug_out* output signal, which is a single bit. In practice, the information of interest has to be serialized out of the chip. The SRAM is read address by address, and each data element has a header and a tail appended to it. In other words, the debug module generates a packet composed of {HEADER + DATA + TAIL}. The same packet concept is applied when serializing PRK, PBK, and SS. This artifice facilitates the identification of data and keys when the debug module is utilized for validation/bring-up of the chip, while still keeping the pin count under control – the actual implementation by an adversary would not have such costly features.

The attack begins after the activation of the *trigger* signal. Next, the ransomware generates the SS key for encryption using the PUF as a seed. However, before initiating the encryption, we perform an initial DUMP to get all the values from the memory. We collect the memory content and all keys, which serve as a golden model to verify if the encryption and decryption modules are working correctly. Then, the ECDH block encrypts the user data, one address at a time. We again perform a DUMP to verify if all the data is encrypted. Next, the user is expected to provide a key back via the *SS_in* input. The use of the right key triggers the decryption. Finally, a third DUMP of the storage is performed and the values are compared (i.e., the storage content after decryption should match the plain-text used during encryption).

TABLE 2. Dynamic and static power values reported from physical synthesis.

Power (μW)	SS corner	TT corner	FF corner
Internal	2506.5	2992.2	3625.6
Switching	2802.4	3285.2	3751.5
Leakage*	11.8**	**9.8**	18.5
Total	5320.7	6287.3	7395.7

* The leakage power values reported here are for VDD and VDDIO supplies, thus they are higher than measurements that consider VDD only.
** The utilized technology suffers from temperature inversion, i.e., the SS corner is not necessarily the lowest leakage corner.

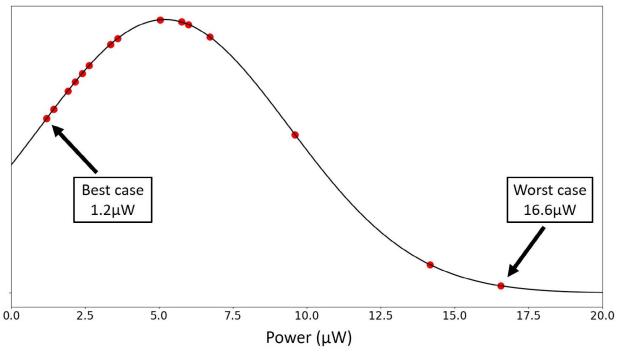


FIGURE 7. Static current measurements plotted as a normal distribution. Each red circle corresponds to a single die. Best ($1.2\mu W$) and worst ($16.6\mu W$) values for static power are highlighted.

Avoiding high leakage current is a concern since the attacker mounting a hardware ransomware attack wants to defeat detection. Thus, power optimization was used to achieve the lowest leakage possible (along with the extensive use of HVT transistors). Table 2 presents a comparison among different corners: SS (Slow-Slow) at 125^oC and 1.08V, TT (Typical-Typical) at 25^oC and 1.2V, and FF (Fast-Fast) at 0^oC and 1.32V. The leakage portion is very low compared with the dynamic power (Internal and Switching). We argue that the presence of the ransomware would not be noticeable in a design with a significant amount of logic such as in a modern SoC. In Section VI, we expand on this argument when we present the insertion of the ransomware on a SoC design.

In Fig. 7, we plot a normal distribution of the leakage power measurements from 20 packaged chips. We remind the reader that leakage power is the static consumption of a circuit when it is effectively idle. The average leakage power is $5.2\mu W$ and the standard deviation is $4.2\mu W$. Measurement points are plotted as red dots. The measurement results are in line with the pre-silicon results from Table 2, well within the expected process margins. These relatively low values are only possible due to the extensive use of HVT transistors and power optimizations during synthesis. An adversary can, without any restrictions, make these same design decisions.

In our manufactured IC, we have utilized an SRAM PUF with 32 addresses and 6 bits per address. Since this is a compiled memory, not all sizes and ratios are valid. The

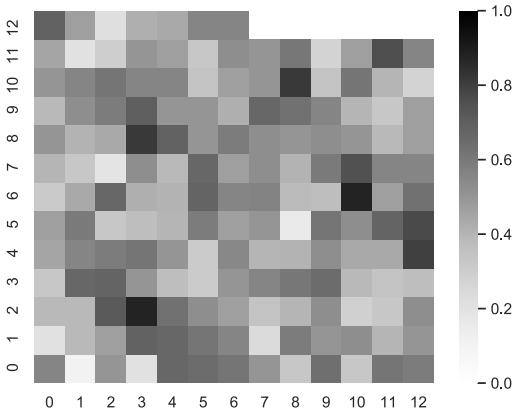


FIGURE 8. Heatmap representation of the 163 bits collected from the SRAM PUFs of 20 chips. The darker the pixel, the higher the ocurrence of '1s'.

32 × 6 configuration is the least wasteful arrangement to generate the 163 PRK bits we need for our **minimal working example**. The remaining 29 bits can be utilized for enhancing the quality of the PUF response. In Fig. 8, we show a heatmap of the 163 PUF bits from the same 20 ICs. The occurrence of '1s' is slightly higher (55%) than '0s' (45%). Each chip was power cycled 10 times, revealing that 17% of the bits showed instability, i.e., intra PUF variation. On average, 22 bits per chip have different values at power-on. This value is in line with the findings of other works that also make use of a commercial SRAM IP as a PUF [23]. Further error correction schemes and helper data would be necessary to guarantee that the SRAM PUF response – after correction – has the same deterministic value at every challenge [24], [25]. Alternatively, the attacker can repurpose the foundry SRAM IP to promote better stability [26] (and therefore require less resources for error correction). An attacker can also make use of PUF that requires no error correction codes at all, such as the self-testing approach described in [27].

The die size is 0.960mm x 0.960mm, but additional structures are necessary for fabrication. The insertion of a seal ring increases the die size to 1mm x 1mm. However, when excluding all the extra structures like storage system, debug module, seal ring, and IO cells, which conceptually are not part of the ransomware, the actual **area of the ransomware logic then becomes 0.1411mm^2**. This represents the actual amount of logic an attacker would have to insert. In Table 3, we show a breakdown of the components of the chip, from which it becomes clear that the biggest area and leakage power consumption comes from the ECDH block. We separate the PUF contribution in two lines, where the first refers to the SRAM instance itself while the second line refers to standard cell logic that is required to interface with the memory ('PUF wrapper').

A. ALTERNATIVE IMPLEMENTATION

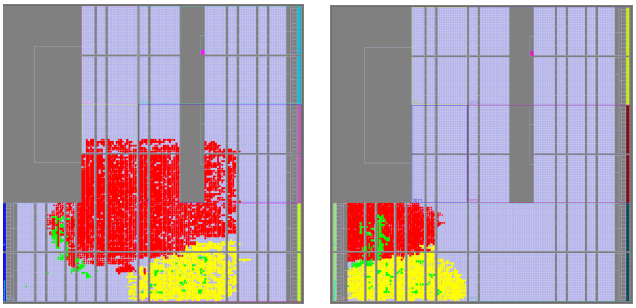
In this section, we provide a discussion on an alternative implementation of our ransomware that has an internal trigger and uses the LA-ECDH variant of our elliptic curve arithmetic unit. This version was *not* fabricated and the results herein reported are from synthesis using the exact same libraries as in the fabricated version.

TABLE 3. Ransomware area report after physical synthesis. This version uses an external trigger and the LL-ECDH core.

Block	Inst. Count	Total Area (mm^2)	Leak. Power (μW)	Total Power (μW)
PUF	1	0.0037	0.0510	3.440
PUF wrapper	719	0.0074	0.0902	8.223
Control Unit	663	0.0027	0.0292	2.754
LL-ECDH	55727	0.1310	1.1889	81.237
Ransomware	57109	0.1411	1.3083	92.214

TABLE 4. Ransomware area report after physical synthesis. This version uses an internal trigger and the LA-ECDH core.

Block	Inst. Count	Total Area (mm^2)	Leak. Power (μW)	Total Power (μW)
PUF	1	0.0037	0.0510	3.440
PUF wrapper	719	0.0074	0.0902	8.223
Control Unit	663	0.0027	0.0292	2.754
LA-ECDH	10222	0.0368	0.3751	15.910
Internal trigger	25	0.0009	0.0090	0.600
Ransomware	11629	0.0478	0.5035	27.487



(a) Ransomware using LL-ECDH core (b) Ransomware using LA-ECDH core

FIGURE 9. FPGA placement solutions of the ECDH implementations. In red the ECDH version, in yellow the RO, and in green the FSM.

First, this version is expected to have a much smaller area footprint, which is shown in Table 4. Figures are also provided for the numbers of gates and power consumption. The LA-ECDH version of our ransomware has reduced the number of gates by 81.66%, area by approximately 71.91%, and total power by 80.42%. However, the ransomware using the LA-ECDH has a degraded performance: the number of clock cycles required to generate the SS is 47 times bigger than in the LL-ECDH core. This might still be a welcomed trade-off for an attacker that is interested in defeating detection above all.

The internal trigger module developed for this version uses a counter register that is incremented every time a specific address is accessed on the storage system. Every time the specific address is utilized for writing or reading, the counter is

incremented. When any other address is accessed, the counter resets to zero. When the counter reaches a threshold (in our implementation the threshold is set to 10), the internal trigger is activated. This type of counter-based trigger can easily be extended for any threshold and adapted for any target address and target width. This type of trigger is extremely relevant to an attacker as he or she can devise a malicious software that performs read/write operations that would lead to the trigger being activated on purpose. For any other piece of software, it would be unlikely that the trigger condition would be met. The proposed internal trigger is very small, and requires only 4 flip-flops. Attackers A1, A2, and A3 are equally capable of devising this internal trigger.

The results presented so far show that the system-level increase in area and power would be insignificant if the ransomware is inserted in a complex SoC design with millions of gates. It should also be taken into account that the ransomware would be fabricated without all the debug structures, which further contributes to a reduction in area and power. These values would be easily masked in the variation of manufacturing technology, which we show later in Section VI.

V. FPGA IMPLEMENTATION

As discussed in Section III, we described two ECDH cores, one optimized for area (LA-ECDH) and another for latency (LL-ECDH). The most contrasting difference from the previous ASIC versions is the need to use a PUF type other than SRAM.

For this purpose, we have utilized an RO PUF which is also a weak PUF just like the SRAM PUF. The utilized RO PUF is adapted from [28]. A pair of ROs can be used to generate a single value based on differences in gate delay caused by variation in the manufacturing process (i.e., '0' if RO1 is faster, '1' if RO2 is faster). Therefore, ROs have a limited number of CRPs. A total of 163 RO pairs were used for competing with each other to generate 163 random bits to be used as the PRK. In other words, 163 ROs pairs are necessary to provide 163-bit CRPs. The PRK_A is still utilized as a seed in the ECDH block to generate SS which is then used to encrypt and decrypt.

Both implementations were deployed in a Zedboard FPGA based on Xilinx Zynq®-7000 SoC [22]. Vivado development kit was used for synthesis, implementation, and bitstream generation. In Fig. 9, we show the placement of the two versions after the implementation phase. In Fig. 9a, we can observe the ransomware with LL-ECDH and, in Fig. 9b, the ransomware with LA-ECDH and internal trigger. The portion of the design highlighted in red corresponds to the ECDH while the yellow portion corresponds to the RO PUF. It should be clear, even from visual inspection, that the overhead of an SRAM PUF in ASICs (see Table 4) is much smaller than the overhead of having an equivalent size PUF with RO pairs in FPGAs.

In Table 5, we show the instance count for each ransomware implementation in terms of LUTs, registers, and muxes. In a separate line, we show results for the RO PUF only. The ransomware using the LL-ECDH version requires 32.84% of the FPGA resources. The ransomware using the LA-ECDH, on the other hand, only requires 17.56% of the resources. This relatively high density is due to our FPGA device of choice. Currently, there are modern FPGAs like Intel® Arria®10 [29] that have versions with 1.15M LUTs and 1.7M Registers. The Zedboard FPGA used for our implementation has only 53.2K LUTs and 106.4K registers. Therefore, for a modern FPGA, the density increase caused by our ransomware would be smaller. Graphically, however, using the Zedboard FPGA allows us to make a visual comparison of the implemented designs and their relative sizes.

TABLE 5. FPGA resources required by both ECDH versions.

Module	LUTs	Registers	F7 Muxes	F8 Muxes
LL-ECDH	13815	5586	184	8
LA-ECDH	5202	5923	353	15
RO PUF	2293	3432	22	9

VI. CASE STUDY: COMMON EVALUATION PLATFORM

The Common Evaluation Platform (CEP) [30] is an SoC design based on the Freedom U500 RISC-V Core with two levels of cache memory and a collection of open-source cores. The CEP is composed of crypto cores, Digital Signal Processing (DSP) cores, and Global Positioning System (GPS) logic. It also interfaces with a variety of protocols and memories, including support for an SD card functionality. The platform is meant to allow users to test a variety of tools and techniques in a realistic SoC.

To demonstrate the insertion of our ransomware in this SoC design, we chose a simplified version of the CEP. Using 5 crypto cores (AES-192, Triple-DES, MD5, SHA-256, and RSA), two Fourier transform cores (Discrete Fourier Transform (DFT) and Inverse Discrete Fourier Transform (iDFT)), and two filters (Finite Impulse Response (FIR) and an Infinite Impulse Response (IIR)). Without loss of generality, we have removed the GPS code generator (too small) and the cache memories (too big).

The ransomware variant chosen to be inserted into the CEP was the LA-ECDH core with the internal trigger. First, for the sake of establishing a baseline, we have synthesized, logically and physically, a ransomware-free version of the SoC. Later, we perform the same exercise but include an instance of the ransomware on the top level description of the SoC. Referring to Fig. 1, this type of insertion would be feasible for an adversary that is involved in the integration process (i.e., we act as attacker A2). It is also conceivable that an adversary in the manufacture process could perform the ransomware insertion, but certainly the challenge would be much higher for this adversary as he/she has to find enough gaps in the placement to insert his malicious logic and then find enough routing resources to connect the inserted cells [31]. We will also show how this type of fabrication-time attack is possible (i.e., we act as attacker A3).

Both the ransomware-free SoC and the ransomware logic were synthesized using Cadence Genus. The netlists were taken from Genus and loaded into Cadence Innovus. The same technology and standard cell library from our fabricated ransomware demonstration was employed in this experiment. Next, in Innovus, all memory macros and top-level pins were placed in the exact same position in the floorplan of all designs. This is important both for the sake of fairness but also to represent a more realistic attack scenario. We assume attackers **cannot** change the position of the top level pins. The position of the memory macros guides the placement of the many modules of the CEP, i.e., the memories serve as a seed for the floorplan. Finally, all standard cells are placed and routed.

In Fig. 10, we illustrate the floorplan (amoeba view) and the placed and routed (physical view) of three versions of the CEP. The leftmost design is the original design, without any malicious logic. Notice how the memory macros, small rectangular blocks highlighted in green and pink colors, are placed in the design's periphery. Also notice how high-congestion areas form around the AES and RSA modules.

We also consider a modified design where the attacker is A2. Since here the attacker is in charge of the integration process, he can insert the ransomware and place it as if it was a regular module of the design. The ransomware logic is highlighted in purple in the second panel from left to right. Observe that the AES module is placed in almost the same position in both versions, implying that the presence of the ransomware is not aggressively disturbing the placement solution. The RSA module, on the other hand, was moved downwards to create space for the ransomware logic on the upper right corner. This move did not compromise the performance of either the AES or RSA modules. Still regarding the RSA module, notice how it forms 6 very distinguishable regions of high congestion. It is very clear to see the move of these regions in the physical views. A careful reader can appreciate that RSA and AES modules are the congestion bottlenecks for this design, which is evidenced by the higher usage of pink lines (metal 7). Visually, the congestion level in M7 remains very similar before and after the ransomware insertion.

Finally, a third version of the design is considered where the attacker is A3. Notice how the floorplan of this version is nearly identical to the floorplan of the original design. This is a **guarantee** of the ECO flow that we have employed, which will perform the insertion of the malicious logic without perturbing the existing logic. It is possible to appreciate that the ECO placer was able to find a region with many gaps between standard cells in the same top right corner of the floorplan.

All three versions were implemented using the same core area, i.e, all versions have the same size of $2.165mm$ by $2.164mm$. The baseline implementation of the CEP has a placement utilization factor of 59.97% (meaning that 59.97% of the area is covered by logic). After the insertion of the ransomware by attacker A2, the utilization factor was increased by 0.73% to 60.70%. We have performed timing analysis and extraction on both designs, which revealed that the additional capacitance due to the insertion of the ransomware is insignificant, causing no performance loss to the SoC. The results also show that the total power consumption was increased by 0.36% and 0.15% for attacks mounted by A2 and A3, respectively. The ransomware-free baseline burns 344.97mW, while the compromised versions burn 346.20mW and 345.49mW.

As mentioned before, a higher than expected leakage current/power is indicative that some malicious logic was inserted into the design. When considering all three design corners, the leakage power of the ransomware-free CEP is $89.19\mu W$ (SS), $22.78\mu W$ (TT), and $66.69\mu W$ (FF). In practice, the real leakage of a fabricated IC is a normal distribution centered around the typical (TT) value and with a deviation proportional to the other corners (SS and FF). Now, we have to reason about how the ransomware insertion affects this value. According to Table 4, the leakage power associated with our LA-ECDH ransomware is $0.5035\mu W$, roughly 2% of the SoC leakage. Effectively, a victim would have no direct way of differentiating the expected variation from the insertion of our malicious logic. We emphasize that the ransomware insertion has not increased chip area and/or impacted timing. Moreover, since the trigger events are rare, an IC with a hardware ransomware virtually behaves as a trojan-free circuit.

VII. DISCUSSION AND COMPARISON

To the best of our knowledge, there are no hardware implementations of a ransomware prior to our work. Our literature review unveiled only one reference that hypothesizes on the topic [8], but no circuits were built or simulated, therefore comparisons are not possible. From this point of view, this type of attack is poorly understood. In the interest of clarity, we revisit some limitations of the attack in this section.

First, the SRAM-based PUF, as implemented off a commercial IP, would not generate a stable key. An attacker would have to carefully design a mechanism for error correction or similar measure. However, the literature contains many solutions that can be leveraged. The use of a correction mechanism would incur further overheads that are not desirable from the point of view of the attacker. However, the attacker is not concerned with long term reliability issues in PUFs (e.g., aging). The allowed time window between the trigger and the ransom payment can be defined by the attacker, i.e., the PUF should provide a reliable response during challenges within the time window.

On the crypto front, we have utilized ECC augmented by a Diffie-Hellman protocol for key exchange. Other solutions can be sought, perhaps with lower overheads. Furthermore, an adversary can reuse existing pieces of logic from the original design to conceal his malicious logic. For instance, our case study SoC already contains many crypto cores, which an adversary could have wrapped with his customized ECDH-like protocol. We argue that the insertion discussed

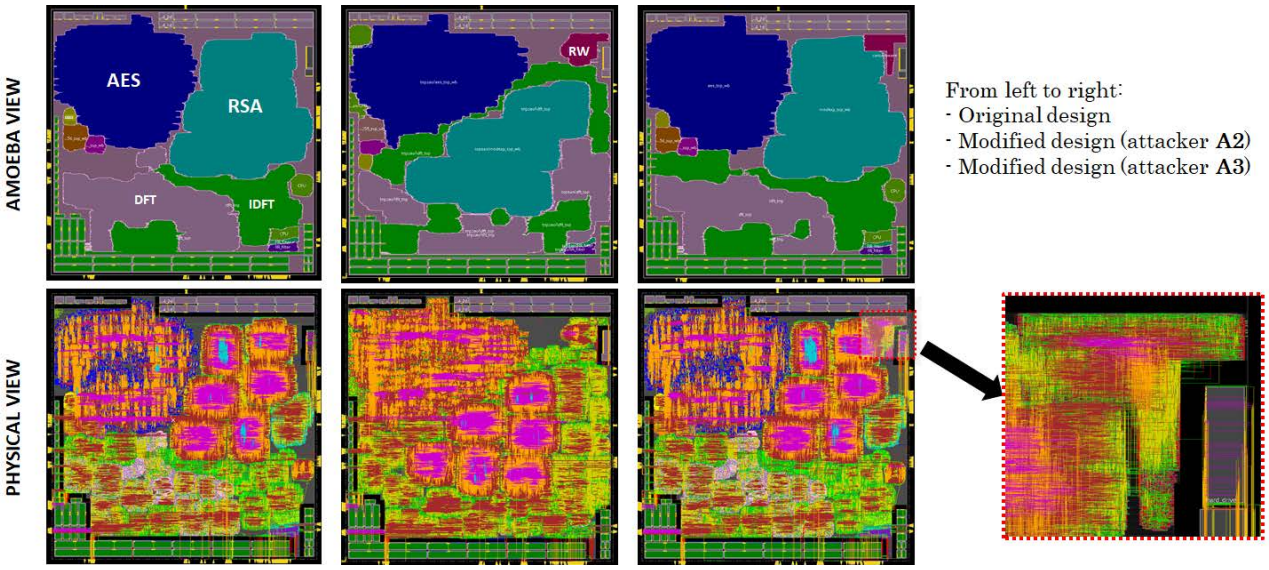


FIGURE 10. Amoeba and physical views detailing the floorplan of the original CEP design and two variants with ransomwares inserted. The image inset shows the congestion increase promoted by the ECO-inserted ransomware, which is localized and does not affect the performance of the other modules.

along with our case study is already hard to detect, but would become even harder if the adversary is now capable of reusing logic from the original design. This approach remains a formidable avenue of research for future attacks.

Without a doubt, designing the trigger condition and the associated user interface are the hardest aspects of a (successful) hardware ransomware. The adversary has to reason about verification, so the trigger is not detected at design time. The adversary also has to reason about Test to make sure the trigger is not detected post fabrication. The adversary also has to reason about the victim's usage of the *infected* device and the implications of the attack becoming public. It is unlikely that an adversary will attempt to design malicious logic that understands a filesystem, as the logic would be very complex. Albeit beyond the scope of our work, the adversary may benefit from a companion malicious software that handles the interface aspect (and perhaps the trigger aspect as well). The most compelling solution is probably for an adversary to completely ignore the communication with the victim, and instead make sure the attack becomes public at a time of his choosing and only at that time (i.e., a time-bomb attack).

Finally, we highlight that even the area optimized implementation is not cost-free. Our ransomware has more cells than other published trojans due to its complexity. We provide a comparison of our ransomware to other works in Table 6. A key differentiating characteristic of our attack is that it must 'communicate' with the victim and that it is reversible, i.e., user data has to be encrypted and decrypted. Other trojans typically do not have these concerns.

TABLE 6. Comparison of our hardware ransomware with published hardware trojans.

Trojan	# Cells	Timing critical?
A2 Analog [32]	2*	X
A2 Digital [32]	91	√
Shadow mode [33]	959**	√
Memory access [33]	1341**	√
This work	11629	X

* Since this is an analog trojan, this table entry is not given in standard cells.
** The authors use the term 'FPGA logic gates'.

A. DETECTION AND PREVENTION

Our concerns with detection relate to changes in the characteristics of the targeted IC when the attack is not yet taking place. In other words, both dynamic and static power consumption can be utilized as proxies for detection, akin to a side-channel analysis. Yet, the victim does not have two versions of the design, with and without the ransomware, to compare one against the other. In practice, for large SoCs, the ransomware detection has to be strong enough to overcome process variation in a statistically-sound manner. For instance, the authors of [34] propose a test generation approach for trojan detection by augmenting their side-channel traces. For the approach to work, identifying rare nodes of a circuit is necessary. However, for the ransomware version with the internal trigger, it is assumed it is connected to data and address buses. These signals, by definition, do not have rare switching probabilities. The counter-based nature of the internal trigger already prevents traditional test practices from succeeding in detecting the trojan.

In terms of physical inspection, the changes to the design highlighted in Fig. 10 are all possible to spot. Modern reverse engineering practices [35] can, albeit with some challenges in scalability, identify features in the 10 nm range. However, physical inspection is not an ordinary step in the design

flow or life-cycle of an IC. For it to be executed, the victim should already suspect of the presence of the compromised logic, which can be minimized if the attack has a time-bomb characteristic.

Regarding prevention, the specialized literature contains many examples of techniques that counter the insertion of hardware trojans. In BISA [36], the authors develop a scheme where filler cells are given functionality, which in turn prevents them from being easily removed from a design. The approach is not practical for large SoCs that have placement utilization factors in the range of $\sim$ 60%, as is the case in the experiment depicted in Fig. 10. In practice, covering the remaining gaps in the floorplan with functional cells would also increase the leakage power by $\sim$ 40%. Even if this approach is deemed necessary, it cannot stop attacker A2 since he/she can insert BISA after the ransomware insertion, thus nullifying the technique. Attacker A3 could be stopped (or severely discouraged) if a technique like BISA was a common practice in IC design.

VIII. CONCLUSION

In this paper, we have investigated the possibility of a hardware ransomware attack. The feasibility of this attack, previously exclusive to the software domain, is discussed at length. We also provide detailed steps on how to map the design to different platforms, namely FPGA and ASIC. Two versions of the ransomware logic were implemented: one aiming low area and another aiming high performance. A clever adversary can improve either implementation to further meet specific malicious criteria he/she might have. We hope that a demonstration of this type of attack can shed light on this out of the ordinary hardware-based attack. We believe that a capable adversary, with the right resources, could probably address the various limitations we have highlighted and from there and construct a harder-to-detect version of the ransomware. On the opposite side, we hope that the feasibility analysis herein described can raise security concerns in the design and fabrication of ICs, therefore raising the bar even higher for this type of attack.

REFERENCES

[1] Gartner. *Gartner Forecasts Worldwide Security and Risk Management Spending Growth to Slow but Remain Positive in 2020*. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2020-06-17-gartner-forecasts-worldwide-security-and-risk-managem

[2] Q. Chen and R. A. Bridges, "Automated behavioral analysis of malware: A case study of WannaCry ransomware," in *Proc. 16th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2017, pp. 454–460.

[3] K. Liao, Z. Zhao, A. Doupe, and G. J. Ahn, "Behind closed doors: Measurement and analysis of CryptoLocker ransoms in Bitcoin," in *Proc. eCrime Res. Summit, eCrime*, 2016, pp. 1–13.

[4] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, "CryptoLock (and drop it): Stopping ransomware attacks on user data," in *Proc. IEEE 36th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2016, pp. 303–312.

[5] W. Smart. (2018). *Lessons Learned Review of the WannaCry Ransomware Cyber Attack*. [Online]. Available: https://www.england.nhs.U.K./wp-content/uploads/2018/02/

[6] A. Waksman and S. Sethumadhavan, "Silencing hardware backdoors," in *Proc. IEEE Symp. Secur. Privacy*, May 2011, pp. 49–63.

[7] S. Bhasin, J. L. Danger, S. Guilley, X. T. Ngo, and L. Sauvage, "Hardware trojan horses in cryptographic IP cores," in *Proc. 10th Workshop Fault Diagnosis Tolerance Cryptography (FDTC)*, 2013, pp. 15–29.

[8] H. Martin, P. Peris-Lopez, L. Entrena, and G. D. Natale, "Ransomware based on hardware trojans. Forget the typical ransomware, you should start worrying about hardware," in *Proc. USENIX Secur.*, 2017.

[9] M. Tehranipoor and F. Koushanfar, "A survey of hardware Trojan taxonomy and detection," *IEEE Design Test Comput.*, vol. 27, no. 1, pp. 10–25, Jan. 2010.

[10] J. Rajendran, E. Gavas, J. Jimenez, V. Padman, and R. Karri, "Towards a comprehensive and systematic classification of hardware Trojans," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2010, pp. 1871–1874.

[11] S. Moein, T. A. Gulliver, F. Gebali, and A. Alkandari, "A new characterization of hardware trojans," *IEEE Access*, vol. 4, pp. 2721–2731, 2016.

[12] T. Perez, M. Imran, P. Vaz, and S. Pagliarini, "Side-channel trojan insertion—A practical foundry-side attack via ECO," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[13] (2001). FIPS. *Standards Publication 197 Announcing the ADVANCED ENCRYPTION STANDARD (AES)*. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[14] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[15] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comput.*, vol. 48, no. 177, pp. 203–209, 1987.

[16] A. O. Almashhadani, M. Kaiiali, S. Sezer, and P. O'Kane, "A multi-classifier network-based crypto ransomware detection system: A case study of Locky ransomware," *IEEE Access*, vol. 7, pp. 47053–47067, 2019.

[17] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, "Physical unclonable functions and applications: A tutorial," *Proc. IEEE*, vol. 102, no. 8, pp. 1126–1141, Aug. 2014.

[18] I. A. Adames, J. Das, and S. Bhanja, "Survey of emerging technology based physical unclonable funtions," in *Proc. ACM Great Lakes Symp. VLSI (GLSVLSI)*, 2016, pp. 317–322.

[19] M. Gao, K. Lai, and G. Qu, "A highly flexible ring oscillator PUF," in *Proc. 51st Annu. Design Autom. Conf. Design Autom. Conf. (DAC)*, 2014, pp. 1–6.

[20] D. Boneh, *Digital Signature Standard—Encyclopedia of Cryptography and Security*. Boston, MA, USA: Springer, 2011.

[21] M. Imran, M. Rashid, A. R. Jafri, and M. Kashif, "Throughput/area optimised pipelined architecture for elliptic curve crypto processor," *IET Comput. Digit. Techn.*, vol. 13, no. 5, pp. 1–8, 2019.

[22] *ZedBoard*. Accessed: Dec. 20, 2021. [Online]. Available: http://zedboard.org/

[23] I. Karageorgos, M. M. Isgenc, S. Pagliarini, and L. Pileggi, "Chip-to-chip authentication method based on SRAM PUF and public key cryptography," *J. Hardw. Syst. Secur.*, vol. 3, no. 4, pp. 382–396, Dec. 2019.

[24] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede, "Helper data algorithms for PUF-based key generation: Overview and analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 6, pp. 889–902, Jun. 2015.

[25] Y. Wang and M. Orshansky, "Efficient helper data reduction in SRAM PUFs via lossy compression," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 1453–1458.

[26] L. T. Clark, S. B. Medapuram, D. K. Kadiyala, and J. Brunhaver, "Physically unclonable functions using foundry SRAM cells," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 3, pp. 955–966, Mar. 2019.

[27] M. Bhargava and K. Mai, "An efficient reliable PUF-based cryptographic key generator in 65 nm CMOS," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2014, pp. 1–6.

[28] W. Yan, C. Jin, F. Tehranipoor, and J. A. Chandy, "Phase calibrated ring oscillator puf design and implementation on fpgas," in *Proc. 27th Int. Conf. Field Program. Log. Appl. (FPL)*, 2017, pp. 1–8.

[29] Intel. *Intel Aria 10*. Accessed: Dec. 20, 2021. [Online]. Available: https://www.intel.com/

[30] MIT Lincoln Laboratory. *Common Evaluation Platform*. Accessed: Dec. 20, 2021. [Online]. Available: https://github.com/mit-ll/CEP

[31] T. Trippel, K. G. Shin, K. B. Bush, and M. Hicks, "ICAS: An extensible framework for estimating the susceptibility of IC layouts to additive trojans," in *Proc. IEEE Symp. Secur. Privacy (SP)*, Los Alamitos, CA, USA, May 2020, pp. 1742–1759.

[32] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester, "A2: Analog malicious hardware," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 18–37.

[33] S. T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou, "Designing and implementing malicious hardware," in *Proc. 1st USENIX Workshop Large-Scale Exploits Emergent Threats, Botnets, Spyware, Worms, More (LEET)*, 2008, pp. 1–8.
[34] Y. Huang, S. Bhunia, and P. Mishra, "Scalable test generation for Trojan detection using side channel analysis," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 11, pp. 2746–2760, Nov. 2018.
[35] Intelligence Advanced Research Projects Activity. *Rapid Analysis of Various Emerging Nanoelectronics*. Accessed: Dec. 20, 2021. [Online]. Available: https://www.iarpa.gov/index.php/research-programs/raven
[36] K. Xiao and M. Tehranipoor, "BISA: Built-in self-authentication for preventing hardware trojan insertion," in *Proc. IEEE Int. Symp. Hardware-Oriented Secur. Trust (HOST)*, Jun. 2013, pp. 45–50.

FELIPE ALMEIDA received the bachelor's degree in computer engineering from Pernambuco University and the master's degree in microelectronics from the Federal University of Rio Grande do Sul. He is currently pursuing the Ph.D. degree with the Centre for Hardware Security, Tallinn University of Technology (Taltech). His research interests include hardware security and radiation tolerant circuits.

MALIK IMRAN (Student Member, IEEE) received the bachelor's degree in computer engineering from the COMSATS Institute of Information Technology, Abbottabad, Pakistan, in 2011, and the M.S. degree in telecommunication and networks from Abasyn University, Islamabad, Pakistan, in 2015. He is currently pursuing the Ph.D. degree with the Center for Hardware Security, Tallinn University of Technology (TalTech), Tallinn, Estonia. Prior to joining TalTech, he was affiliated with different research laboratories at both national and international levels for development of efficient hardware solutions for intrusion detection systems and asymmetric cryptography.

JAAN RAIK (Member, IEEE) received the M.Sc. and Ph.D. degrees in computer engineering from the Tallinn University of Technology, in 1997 and 2001, respectively. He is currently a Professor of digital systems verification at the Department of Computer Systems, Tallinn University of Technology, and also the Leader of the Center for Dependable Computing Systems Design (DCSD). He has coauthored more than 200 scientific publications. He is a member of the IEEE Computer Society and HiPEAC, and a member of steering/program committees of several conferences.

SAMUEL PAGLIARINI (Member, IEEE) received the Ph.D. degree from Telecom ParisTech, Paris, France, in 2013. He has held research positions with the University of Bristol, Bristol, U.K., and with Carnegie Mellon University, Pittsburgh, PA, USA. He is currently a Professor of hardware security with the Tallinn University of Technology (TalTech), Tallinn, Estonia, where he leads the Centre for Hardware Security. He is also the Co-ordinator of the H2020 Project SAFEST. His current research interests include many facets of digital circuit design, with a focus on circuit reliability, dependability, and hardware trustworthiness.