Fault Attack on AES via Hardware Trojan Insertion by Dynamic Partial Reconfiguration of FPGA over Ethernet

Anju P. Johnson[1], Sayandeep Saha[1], Rajat Subhra Chakraborty[1],
Debdeep Mukhopadhyay[1] and Sezer Gören[2]
[1]Dept. of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, India
{anjupj,sahasayandeep,rschakraborty,debdeep}@cse.iitkgp.ernet.in
[2]Dept. of Computer Engineering, Yeditepe University, Istanbul, Turkey
sgoren@cse.yeditepe.edu.tr

ABSTRACT

We describe a novel methodology to exploit the widely used *Dynamic Partial Reconfiguration* (DPR) support in Field Programmable Gate Arrays (FPGAs) to implant a hardware Trojan in an Advanced Encryption Standard (AES) encryption circuit implemented on a FPGA. The DPR is performed by transferring the required partial configuration bitstream file over an Ethernet connection to the FPGA board, from an attacker's computer which can communicate with the FPGA over a network. The inserted Trojan launches a "fault attack" on the AES encryption circuit, which enables recovery of the secret key by standard mathematical analysis of the faulty ciphertext produced. To the best of our knowledge, this is the first reported attack which exploits DPR to break an AES hardware implementation on FPGA. Our implementation results establish this to be an extremely potent attack on AES at low hardware and computational overhead, while using the standard unlicensed FPGA design tools.

Keywords: Hardware Trojan · FPGA · Dynamic Partial Reconfiguration · AES ·Fault Attack

1. INTRODUCTION

The modern trend of outsourcing and procurement of design, tools and manufacturing services, to or from third party vendors in the form of synthesizable hardware intellectual property (IP) cores, computer-aided design (CAD) tools, or silicon integrated circuit (IC) fabrication facilities, makes a design vulnerable to malicious modifications. Malicious circuitry embedded as a result of such modifications, commonly referred to as Hardware Trojan Horses (HTHs), have been demonstrated to be potent threats [9]. The stealthy nature of HTHs help them to evade conventional post-manufacturing testing. Once deployed in-field, HTHs activate under certain rare conditions (depending on internal or external stimulus). The two most widely studied modes of action of HTHs are to compromise security by the leakage of system information, or to cause catastrophic system failure through the functional failure of circuits [24].

ESWEEK'14, October 12-17 2014, New Delhi, India

http://dx.doi.org/10.1145/2668322.2668323.

Recent research on HTHs mainly focuses on the modeling and detection of HTHs [7, 20–22, 24, 28–30]. Since the HTH triggering (i.e. activation) mechanism should be non-trivial, and a HTH should not be activated accidentally when not desired by the adversary, much current research effort is also being spent on the study of triggering mechanisms by means of HTH in an IC. However, all of these works deal with the classical attack model of HTHs, which relies on the untrustworthiness of the current semiconductor business practices. The basic assumption of this model is that the HTHs can only be inserted during the design and the manufacturing phases. The main limitation of this model is that it assumes the presence of malicious "insider" parties in the design or manufacturing teams. While this is feasible, and the influence of the human factor on the overall trustworthiness of electronic components can never be ruled out, an attack model which does not involve such malicious human agents, but can still implant a HTH in an otherwise correctly operating circuit, seems to be a greater threat. Another weakness of HTH insertion model is that it (either implicitly or explicitly) assumes that the validation mechanisms (primarily pre-manufacturing verification and post-manufacturing testing) are often insufficient in detecting well-designed HTHs, or can be easily bypassed. While this is true at the current state-of-the-art for many demonstrated HTHs, the attack model can be claimed to be stronger if *no validation or test mechanism currently exists, or is rarely enforced for a particular abstraction level* (behavioral, logic or physical) of the implemented circuit.

In this paper, we propose precisely such an attack model, and demonstrate its feasibility for FPGA based circuits. FPGAs seem to be the natural choice for such an attack model, because of their reconfigurable nature which allows change in design functionality any arbitrary number of times over its lifetime, something which is difficult to envisage is general Application Specific Integrated Circuits (ASICs). In the proposed attack model, the HTH is inserted in a FPGA "in-field", post-deployment. The FPGA is assumed to be connected to a network via Ethernet, which allows the circuit mapped on the FPGA to perform useful computations by real-time data transfer to-and-from other nodes in the network. The attack mechanism utilizes the *Dynamic Partial Reconfiguration* (DPR) capability of modern FPGAs, whereby modifications can be made to the circuit mapped

on a FPGA, usually to add additional useful functionality of the existing circuit, without needing to take the FPGA offline. The HTH is inserted by reconfiguring the dynamically reconfigurable part of the circuit by sending the required (partial) configuration bitstream over a live Ethernet connection to the FPGA. The widespread and ever-increasing use of FPGAs in modern system implementation, their increasing acceptance in implementing high-performance communication networks (especially temporary ones such as in military applications) and distributed computing [11, 25], and the desirability of the DPR facility which enhances system functionality, decreases power consumption and system downtime, make the proposed attack model extremely feasible and potent. The strength of the proposed attack model lies in the following:

1. The HTH insertion through DPR evades most conventional static and dynamic (runtime) testing techniques for Trojan detection in deployed circuits, as well as design validation mechanisms prior to deployment.

2. The relatively small size of the HTH payload to be transferred to implant the Trojan, allows it to be piggybacked with an authorized and benign bitstream to add useful functionalities to the existing circuit on the FPGA. The piggybacking does not require any extra data packet transfer over an Ethernet connection, and hence no extra communication overhead is added other than that is needed for a normal DPR.

3. Since analysis of a FPGA configuration bitstream to identify the representative circuits is difficult [8] and not widely explored at the present state-of-the-art, the HTHs inserted by DPR can potentially evade the existing simple bitstream validation mechanisms currently commonly deployed in a FPGA. For example, any configuration bitstream is subjected to *Cyclic Redundancy Code* (CRC) checking by dedicated hardware on the FPGA prior to it being allowed to actually make any change to the existing circuit functionality. However, this is only a checksum validation mechanism for the bitstream, and does not analyse the semantics of the bitstream contents.

4. The HTH on the FPGA is easily "erased" by the attacker after it serves its purpose, by overwriting it by another DPR procedure. Even if no more DPR is attempted, the HTH designed by us is "one-shot" in nature, whereby it goes to an de-activated state indefinitely after causing the circuit to malfunction once.

5. Finally, to launch the attack, the adversary need not access the common programming interface (usually JTAG) used by FPGAs for reconfiguration. The FPGA is remotely accessed over a regular Ethernet connection, the presence of which is necessary for the FPGA to perform data transfer with other communicating network nodes.

We have demonstrated the proposed attack model by implementing a prototype, whereby a "fault attack" is launched on the hardware implementation of 128-bit AES block cipher on a Xilinx Virtex-5 FPGA. Fault attacks induce a well-targeted transient fault on a cipher hardware. As a result of the circuit fault, faulty ciphertext is produced, and by standard mathematical analysis of relatively low computational complexity (manageable on an ordinary desktop PC), it is possible to extract the secret information (usually the cipher key). Fault attacks are particularly interesting as they require relatively less computational effort and are easy to launch [4,10,18,19,23,26]. Some significant examples can be found in the recent literature which utilize Trojans to perform fault attacks on AES [2, 3, 14]. In [2], the authors utilize specific nexus between parties associated with the design, manufacturing, testing, and deployment of cryptographic hardware to launch the attack. While the work demonstrated the attack to be potent, the dependence on a multi-party nexus to make the attack feasible, makes the attack model relatively weak. Another related work can be found in [3], where the authors proposed a simplified HTH design for the same purpose. However, both these works dealt with HTHs that had the following features:

1. Trojan was inserted during the design phase of bitstreams.

2. The Trojan remained undetected by the pre-deployment validation mechanism.

3. The Trojan infected design was mapped on the FPGA, i.e., the Trojan was present at all times in the deployment phase.

In contrast, our attack introduces the HTH during runtime, and does not need any nexus between multiple parties, which increases its feasibility. Applications of communication frameworks for HTH activation and HTH-based covert channels [13], or for capturing the information leakage [17] caused by HTH have also been described. However, these works are different from ours in the sense that all of them assumes that the circuit is Trojan infected at the point of its deployment, while in the proposed work the Trojan is inserted run-time through Ethernet communication interface. In this work, we employ a widely used open-source communication and synchronization framework called *Simple Interface for Reconfigurable Computing* (SIRC) [11, 25] distributed by Microsoft to facilitate the PC-FPGA intercommunication via Ethernet. A methodology to sent partial bitstreams to FPGA over an Ethernet connection has been described previously [5,6]. Compared to this work we found, an extension of the SIRC framework to be more promising for transferring the partial bitstream to the FPGA, as they provide sufficient isolation of the DPR controller from the Ethernet communication interface.

The rest of the paper is organized as follows. Section-2 describes the attack model in detail. Section-3 provides the necessary background of the DPR methodology and the SIRC infrastructure. In Section 4, we describe the details of the implemented prototype to demonstrate the attack. Experimental results are discussed at Section-5. Section-6 describes possible defense techniques against the proposed attack. Finally, we conclude in Section-7.

2. DETAILS OF THE ATTACK MODEL

Let us consider a FPGA connected to a network over a standard 1 Gbps Ethernet connection, providing real-time computational capabilities. One of the services the FPGA hardware provides to all users in the user group connected to the network is real-time private-key encryption of plaintexts

transferred to it via the Ethernet connection. The encryption key is either hard-coded in the cipher hardware core, or stored on a secure on-board memory module which the FPGA can access. The FPGA is DPR-enabled, and the facility can be utilized to add or modify the functionality of the existing circuit on the FPGA. Only a selected small set of "superusers" are authorized to perform DPR, by responding with any one among a selected set of passwords that the FPGA hardware challenges her to enter when she attempts DPR. Provided that the superuser enters the correct password, she is allowed to transfer the partial reconfiguration bitfile to an input buffer on the FPGA, utilizing SIRC API calls. Now the "superusers" have two choices:

1. Modify the existing "add-on"(she can reuse the "add-on" done by any previous DPR procedure by any "superuser"). This facility provides great degree of flexibility for evolving hardware. This will also lead to reduced reconfiguration time, since the new "add-on" is just an extention of the existing one.

2. Secondly, she has provision to erase some or all the previous "add-ons" and configure a new design with new "add-ons". This feature is enabled to facilitate the addition of hardware logic which do not have direct logical relation with the existing "add-ons".

Note that in the context of utilizing the services of the FPGA for encryption, a superuser is similar to any other member of the authorized user group in the network.

The DPR hardware controller module reads the configuration data stored in the input buffer which acts as the configuration memory, and then performs the reconfiguration. Only one "superuser" can perform DPR at any given time, and the requests of other "superusers" to perform DPR while a DPR operation is ongoing, are ignored by the interface hardware on the FPGA. Once the DPR operation is complete, the superuser releases the connection to the FPGA by another API call, and only then any other request for DPR by any other superuser are paid attention to. An ordinary (non-superuser) member of the user group can make a request for DPR, but this request, if authorized, must go through any one of the allowable superusers for the DPR to actually take place. A schematic of the network connectivity scenario for the FPGA is shown in Fig. 1, where A_is and U_is correspond to the superusers and non-superusers, respectively.

Every attempted DPR is logged by the Ethernet communication interface hardware on the FPGA, which is a modified version of the one provided by the SIRC framework, and the detailed log is stored on a tamper-resistant non-volatile memory module on the FPGA board. Hence, it is not possible for a malicious "superuser" to make arbitrary modifications on the existing FPGA hardware without revealing her identity. However, the superuser can wait till a pre-scheduled and pre-authorized DPR operation is to be performed, to add to or to modify the existing circuit on the FPGA, and then, piggyback the malicious component of the bitstream on the benign component bitstream to be mapped on the FPGA. The HTH that induces the fault attack on the cipher hardware is activated automatically after some time (possibly after some other DPR has taken place), depending on the output of a counter-like circuit or FSM. This makes it difficult to identify what exactly caused a single encryption operation to fail, and also difficult to determine with

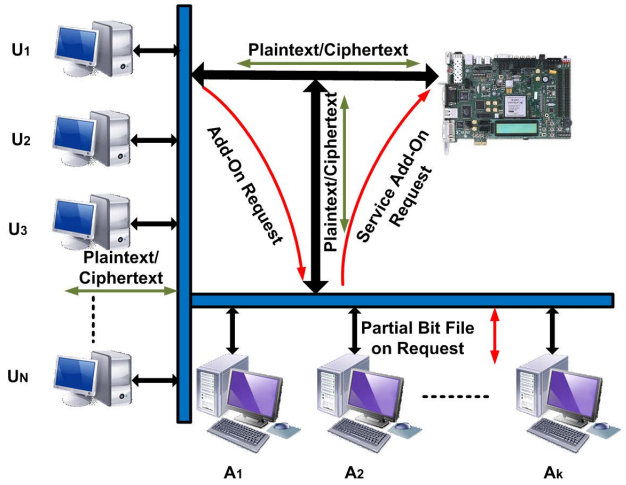

Figure 1: Network connectivity of the FPGA in the proposed attack.

certainty which DPR operation was exactly responsible to cause the failure. In addition, similar to the mechanism proposed in [2], the faulty ciphertext produced after the fault attack is altered by the inserted HTH. Because of this, no arbitrary party can recover the secret key from the garbled faulty ciphertext, but only an adversary having knowledge of how to calculate the original faulty ciphertext from the altered version, can launch the attack. As pointed out in the previous section, the proposed Trojan operates in an "one-shot" mode, i.e. it self deactivates after a fault is induced in the cipher hardware. This helps the HTH to masquerade as a random, non-directed circuit failure, the cause for which is difficult to determine.

3. DPR AND SIRC

3.1 Dynamic Partial Reconfiguration on Xilinx FPGAs

In this work, we target Xilinx Virtex-5 FPGA devices. Xilinx offers partial reconfiguration (PR) either through the *modular approach* [31] or the *difference based approach* [12]. In the modular approach, partitioned module instance descriptions written in a hardware description language (HDL) to define the new hardware functionalities, are translated into partial bitstreams. Xilinx supports module based PR through the licensed version of the *PlanAhead* software. Difference based PR flow is not widely used, as traditionally it is meant to allow only small and incremental design changes such as changing I/O standards, block RAM contents, and LUT programming [12].

In the present work, we generate the partial bitstreams following a difference based DPR technique [15, 16] using the Xilinx *FPGA Editor* software which does not need a special PR license. The difference based partial reconfiguration technique described in the *Xilinx Application Note* [12] uses FPGA Editor to make small logic changes to the already configured hardware which is not applicable here directly as the changes possible using it are too small for our purpose. Here we use an improved methodology called *"Difference based Partial Self Reconfiguration for Large Difference"* (DPSR-LD) [15, 16]. DPSR-LD, that has been proposed to

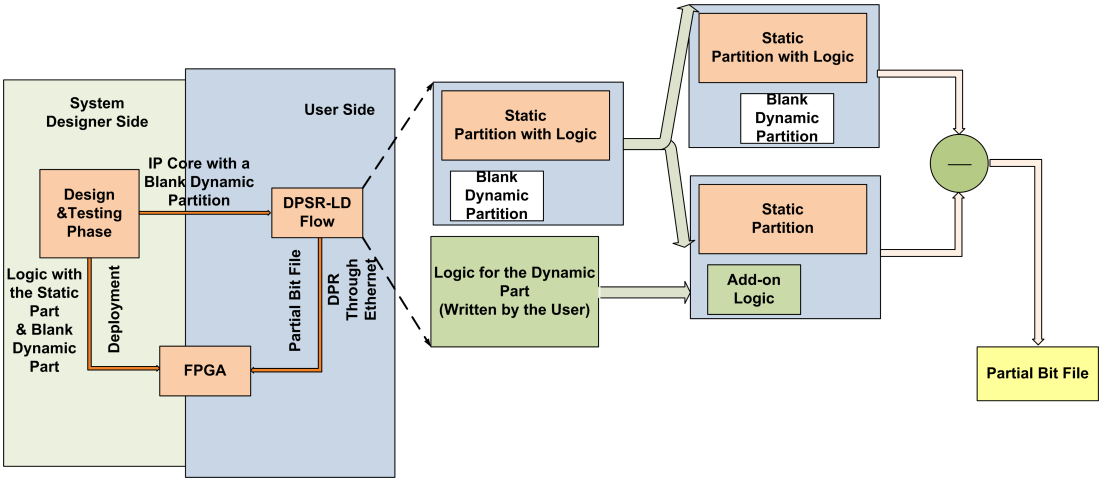

Figure 2: Overall scheme and design partitioning for dynamic partial reconfiguration.

protect FPGA configuration bitstreams from cloning and reverse engineering. The DPSR-LD design methodology is also a difference based approach, but it is more powerful because it works well even if there are large differences between the partial bitstream files. Although it was classically used for self reconfiguration, we use it here only to implement normal DPR for large difference. This approach works well even for low cost FPGAs which do not have PR tool support. The DPSR-LD design flow is hence a cost-effective solution, and can be used to provide dynamic reconfiguration capabilities at no extra cost to the user. The proposed attack adopts this technology for partial bitstream generation. Similar to the modular approach, the DPSR-LD flow also requires partitioning the floorplan. The whole FPGA floorplan is partitioned into static and dynamic regions, with the static region of the FPGA being configurable only once (at the beginning). Suitable setup constraints are used to avoid logic mix between the partitions, and to avoid routing across dynamic region and static region [15, 16]. Note that the partitioning of the original circuit has to be done inside the production chain during the design phase itself, to support DPR. This design is assumed to be provided to the user in the form of an IP core, described in a standard HDL such as *Verilog* or *VHDL*. The overall scheme and design partitioning used here is shown in Fig. 2.

For our particular attack, the architecture of the circuitry on the FPGA is shown in Fig. 3. In our present example, the static partition contains the basic AES core, the Ethernet controller, a PR controller and a communication interface between the partitions called a "bridge". The PR controller logic block controls DPR at the FPGA side. It contains the *Internal Configuration Access Port* (ICAP) which is a standard Xilinx hardware primitive for DPR on Virtex-5 platform. The dynamic partition is divided into p dynamically reconfigurable partitions to increase flexibility and upgradability. This way of partitioning is a promising solution for module based DPRs for evolving hardware designs [27]. We use a similar way of partitioning technique for the difference based DPRs. Dynamically reconfigurable "add-ons" in this work communicate with the static part through the communication interface either directly or via another "add-on". The "add-on" logics can be combined to and built a bigger

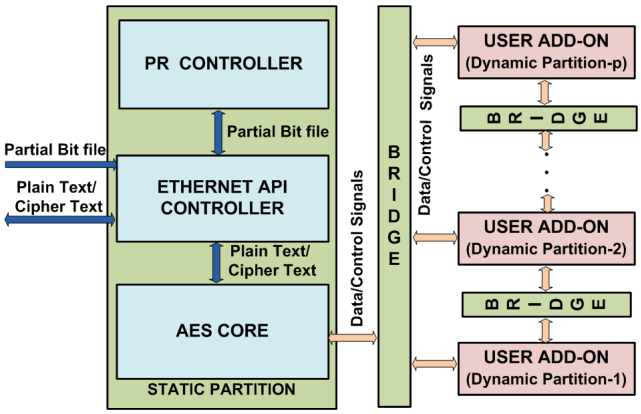

Figure 3: Architecture of the hardware implemented on FPGA.

evolving hardware as per the requirement. This facilitates the use of smaller sized partial bit files to be transferred for building a larger modification for some future purposes, and hence reduces the reconfiguration time for the larger design. Once the bitfile for partial reconfiguration is received on the FPGA through Ethernet, the PR controller generates necessary signals to the ICAP which in turn performs the DPR in the specified dynamic partition. The "bridge" logic consists of a set of flip-flops with fixed positions on the FPGA. If a user wants to use the design situated at the static part only (original AES core), she may use it without any modification. However, a specific user depending on her requirements, who wants additional functionalities, (e.g. support for a specific mode of encryption like CBC, OFB, CFB, *tweakable encryptions*, etc.), may need to make subtle modifications to the design using the basic AES hardware core. This can be performed using DPR which is done using the flow described in this work (see Fig. 2). The motivation of using DPR in this context stems from the fact that with the large variety and applications of such modes, it is unlikely that all the features would be needed simultaneously, and hence the implemented design can operated with a just enough logic, thus saving power, requiring less area, and a

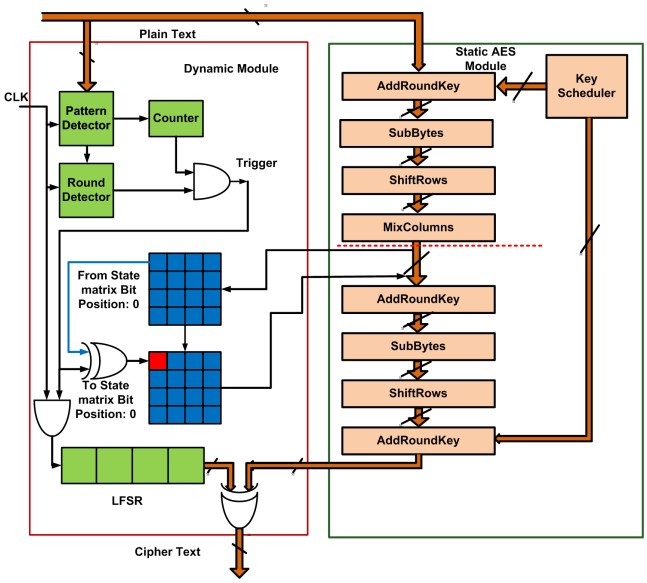

Figure 4: Detailed design of the implemented HTH to cause fault-attack in the AES encryption circuit implemented on FPGA.

smaller critical path. The user originally needs only the default services provided by the "static part" of the IP core of the partitioned architecture, with the pre-designated blank "dynamic part". The "add-on" logic can be inserted in the dynamic part on-demand. Both the cores with and without the add-on logic in the dynamic partition can be synthesized and implemented fully on *Xilinx ISE* creating the *Native Circuit Description (NCD)* files, *Physical Constraints File (PCF)* files and the *FPGA Programming Bitfiles*. The NCD files, the PCF and the PGA programming bitfiles are given to BITGEN as input and the partial bitfile for the add-on is created by taking difference between these files from the two design cores. Active Reconfiguration switch is enabled while generating the partial bitfile. The "add-on" logic may be designed by the designer himself or by some other party including the designer of the basic IP core. Once created, this partial bitfile can be downloaded on the FPGA any time it is needed through the Ethernet from any node in the network.

3.2 The Simple Interface for Reconfigurable Computing (SIRC) Framework

The SIRC is a free, open-source software-hardware communication framework running on Windows, for the Xilinx Virtex series of FPGAs (Virtex-5 and Virtex-6), developed by Microsoft Research [1,11]. It consists of mainly two components: (a) a library of C++ API calls that an application developer can utilize to communicate with the FPGA over an Ethernet connection, and (b) a custom Ethernet interface hardware module to be mapped on the FPGA, which acts as the intermediary between the software application running on the host PC, and the "user logic" hardware accelerator mapped on the FPGA. It frees the application developer from the burden of developing her own custom driver software or interface hardware to manage and synchronize the data transfer between the host PC and the FPGA. It has been widely used in the research community, mainly for high performance distributed reconfigurable computing [25]. The SIRC framework proved ideal in the implementation of our attack, as its architectural features allowed that the user application (a cipher circuit in our case) can be isolated from the complex interactions of the DPR controller hardware and the Ethernet communication interface hardware.

4. IMPLEMENTATION DETAILS

4.1 HTH Insertion Using DPSR-LD flow and SIRC Framework

As mentioned in Section 3.1, the user can design "add-on" logic for additional functionality, to be lodged only in the dynamic partition of the IP-core. While this facility increases the design flexibility to a certain extent, it also poses a serious threat. The attacker may design the HTH, create the partial bitfile for the HTH design, and then download it on the FPGA via the Ethernet module resident at the static part. In this paper, the HTH is intended to launch a fault attack on the AES module which is included in the static part of the circuit. The details of the HTH implementation and fault attack on AES hardware is discussed in next few subsections.

4.2 The HTH Design

The design of the HTH implemented here is shown in Fig. 4. The design is similar to those described in [2] and [3], with a few modifications to make it simpler but more potent. Once inserted in the circuit, the HTH gets triggered only when it receives some specific predetermined bit pattern in the plaintext for some predefined number of times, not necessarily consecutively, both of which are decided by the adversary while designing the HTH. The length of the bit pattern is called *Triggering Bit Pattern Length* ($TBPL$) and the number of times it has to appear to trigger the HTH is called *Triggering Bit Pattern Count* ($TBPC$). Note that the HTH triggering event is non-deterministic, which simplifies the HTH activation mechanism by removing the need for remote triggering, at the cost of the possibility of a scenario where the HTH is never triggered. Interestingly, both extremely high and extremely low *activation probabilities* are bad in this scenario, as the first one may increase the probability of the HTH being detected, while the second one may keep the HTH dormant indefinitely. Hence, the adversary needs to set the $TBPL$ and the $TBPC$ properly, to make it sure that the HTH activation probability reaches a non-zero moderate value. In our present implementation, the HTH is activated only when it receives a specific pattern of 20-bits 9 times. The reason behind such a selection is explained in Section 4.3. The triggering bit pattern is detected using an equality checker circuit, and the number of times the sequence occurs is counted using a counter.

On activation, the Trojan synchronizes with the start of a new plaintext encryption, and then waits for seven clock cycles, before enabling an XOR gate with logic-1 as *trigger pulse* at one of its input. The other input of this XOR gate, connected to the 0-th bit of the AES state matrix gets flipped by the Trojan. Thus, a fault is induced in the static AES hardware at the beginning of the 8-th round during encryption which causes the AES to generate a faulty output ciphertext. Note that, once triggered the Trojan never gets reactivated until the FPGA Trojan circuit is reset, which makes its tracking and detection highly challenging. It has

been already proved that with a single fault injection point and with a single faulty encryption the key can be derived with a brute-force search of size 2^{32} [18], which can be performed in less than 15 minutes using a standard PC. The brute-force search size can be reduced further to 2^8 [26].

It is desirable to an adversary that no user except her, connected to the network would be able to deduce the secret key from the faulty ciphertext. The faulty ciphertext output can be garbled using the output values obtained from a "Linear Feedback Shift Register" (LFSR) circuit added as a part of the HTH. This LFSR is enabled by the same trigger signal which enables the HTH, and remains active for the time taken to complete a single encryption by the AES. The recovery of the ciphertext is only possible by the adversary who knows its configuration and initial state [2]. Moreover, the extraction of the output ciphertext from the modified ciphertext by brute force, also requires operations of workload $\binom{128}{P} \cdot 2^{32}$ (where P is the number of bit positions in the faulty ciphertext modified by the HTH, 10 in our implementation), which is also practically infeasible.

4.3 Effectiveness of the Attack

As mentioned in the previous subsection, the *activation probability* of the Trojan is a critical quantity which is to be decided by the attacker. We define the expected *number of encryption operations* done by an AES circuit infected by the Trojan before the Trojan triggers, as the "*Trojan Activation Time*" (TAT). Once we have a quantitative estimate of TAT, we can formally construct the problems of finding the corresponding $TBPL$ and $TBPC$ as follows:

Problem 1. *Given the TAT in the circuit and the $TBPL$, what should be value of the $TBPC$ so that the activation probability reaches a specific probability value p_c within a duration TAT after the Trojan is inserted, and,*

Problem 2. *Given the TAT in the circuit and the $TBPC$, what should be value of the $TBPL$ so that the activation probability reaches a specific probability value p_l, within a duration TAT after the Trojan is inserted.*

Any one of the above mentioned problems can be solved to design an effective Trojan, assuming some moderate values of p_c or p_l. It is reasonable to assume that the activation probability of the trojan is dependent on the three parameters TAT, $TBPC$, and $TBPL$. Let us assume that within TAT plaintext, arrival of a triggering plaintext is considered as success and arrival of a normal plaintext is considered as failure. Then, each of the plaintext arrival can be modeled as independent Bernoulli trials and the probability that $TBPC$ number of success occurs can be found using a binomial distribution $B(n, p)$ with parameters $n = TAT$ and the probability of a single success which is given by $p = 1/2^{TBPL}$. Now as n is sufficiently large and p is sufficiently small and the product np is a constant, the binomial distribution can be approximated as a *Poisson distribution* with parameter $\lambda = np$. So for sufficiently large values of TAT and $TBPL$, assuming a uniform distribution of the plaintexts, the activation probability follows a *Poisson distribution* with probability density function $\Pr(\lambda, TBPC) = \frac{e^{-\lambda}\lambda^{TBPC}}{TBPC!}$, with mean of the distribution $\lambda = \frac{TAT}{2^{TBPL}}$. In this work, we have designed the Trojan to have $TAT = 10^7$ ($\sim 2^{24}$) plaintexts, and $TBPL = 20$. Then, to achieve $p_c \approx 10\%$ we set $TBPC = 9$.

Table 1: Xilinx XPOWER Analyzer report on total on chip power

Total On-Chip Power		
Golden Reference (Blank Dynamic Partition) (mW)	**Trojan Infected Circuit (mW)**	**Increase in Power w.r.t. (Golden %)**
2923.97	2924.95	0.0335

Table 2: Xilinx Timing Analyzer Report on Critical Path Delay

Critical Path Delay		
Design Without Trojan (ns)	**Design With Trojan (ns)**	**Increase in Critical Path Delay (%)**
9.502	9.502	0.00

Table 3: Hardware Overhead

Device Utilization	Golden Reference Design	Trojan Infected Design	Increase in Hardware Overhead w.r.t Golden (%)
Slice	1576	1582	0.38
SliceReg	1742	1748	0.34
LUTs	3733	3739	0.16

5. RESULTS

The hardware design (both the AES cipher core as well as the HTH) was performed using Verilog HDL. The designs were synthesized and implemented using *Xilinx ISE 14.5*, and simulated using *Xilinx Isim*. The power estimation of the circuit was carried out using *Xilinx XPower Analyzer* and delay estimation using *Xilinx Timing Analyzer*. Table 1 shows the percentage increase in the total power consumption of the design before and after Trojan insertion. Table 2 compares the critical path delays before and after Trojan insertion in the dynamic partition. Table 3 compares the hardware overheads for the golden and the Trojan-infected designs. The simulated power traces obtained from the Trojan-free and the Trojan-inserted design did not show any significant variation. From these results, it is clear that the Trojan insertion can be realized using minimal hardware overhead, and has negligible effect on the power and delay. Moreover, the Trojan implemented by us is much more lightweight than the Trojan described in [2]. The small size of the Trojan results in a negligible payload to be transferred to implant the Trojan which allows it to be piggybacked with a benign design. Finally, as expected, the faulty ciphertext obtained under the influence of the activated Trojan, led to the recovery of the cipher key by 15 minutes of computation (following the theory described in [18]) on a PC with 2 GB of main memory and a 2 GHz CPU.

Most of the existing Trojan detection techniques, targets on Trojan detection before FPGAs are deployed for operation. Since we insert Trojans after FPGA starts to work, the Trojan can evade pre-deployment detection techniques. As discussed earlier, this Trojan is triggered only once in its lifetime by regular users accidentally with some moderate activation probability even for relatively large number of encryption operations ($\sim 10^7$). So, it is almost impossible to trace the actual attacker even inside a small user

group. Hence, we can conclude that these hardware Trojans can elude functional built-in-self-test (BIST) or authentication [30]. Power or delay analysis based Trojan detection methods also fail to detect it, because of the low impact on these side-channel parameters.

6. DEFENSE STRATEGY AGAINST THE PROPOSED ATTACK

The proposed post-deployment "in-field" Trojan insertion strategy evades most traditional static and dynamic Trojan detection techniques. As mentioned previously, simple-minded validation of the partial bitstream by techniques such as CRC checking, before allowing the actual DPR to happen, will not be effective here as it cannot check the semantics of the an arbitrary bitstream to deduce the logic implemented by it. However, a restrictive mode of DPR can be implemented that can prove effective in preventing Trojan insertion, as follows. For any application mapped on the FPGA, if it is decided that only a limited number of possible modifications might be performed for it through DPR, the partial bitstreams might be generated in advance for these "pre-authorized" modifications. For each of these pre-authorized modifications, a golden signature (e.g. a CRC calculated for the partial bitstream corresponding to it) would also be generated, to act as an identifier. At the same time, the static part of the basic circuit should include a "signature verifier logic module", which will calculate the signature for any partial bitstream. The partial bitstream would be allowed to affect the dynamic part of the circuit, only if the signature calculated for it matches one of the golden signatures. Although extremely simple, the prevention technique provides reasonable protection at small hardware overhead, while having the following disadvantages:

1. The restricted DPR mechanism would lack flexibility, and would prevent the utilization of the dynamic reconfiguration capability to its full strength.
2. There would always be a finite, non-zero (albeit small) probability that the signature verifier would fail to detect any malicious add-on HTH logic, because of aliasing of the checksum calculated. The aliasing probability can be made smaller by increasing the CRC length.

In our attack model, all the "superusers" have access to the communication channel between PC and the FPGA, but only one of the "superuser" is allowed to perform FPGA dynamic reconfiguration. In such a model there is a finite probability for some multi-party (multiple "superusers") involvement in modifying the bitstream before it reaches the reconfigurable device. So better solution is to use authenticated signed encrypted bitstreams. In this case the FPGA will accept only bitstreams from authenticated source with "pre-authorized" modification. Since we use, Ethernet framework to sent bit-streams to the FPGA, the same channel can be used to provide information on the success or failure of the partial bit-stream update.

7. CONCLUSIONS

Dynamic Partial Reconfiguration is a useful feature in modern high-end FPGAs which allows the addition of extra functionality, or modification of existing functionality to a FPGA during runtime. We have presented a novel, lightweight and hard-to-detect HTH which exploits DPR capability and Ethernet connectivity of a FPGA to cause malicious modifications to the existing circuitry. We demonstrated the attack model by inserting a low overhead HTH to launch a fault attack on AES cipher hardware mapped on Xilinx Virtex-5 FPGA platform, aided by the SIRC PC-FPGA interfacing framework, thereby leading to the recovery of the secret cipher key. Although the flexibility and benefits offered by DPR reach across many industries and applications, it is time to carefully scrutinize its security implications, as evident from this work.

8. REFERENCES

[1] *Available online at "http://research.microsoft.com/en-us/downloads/d335458e-c241-4845-b0ef-c587c8d29796/".*

[2] Ali, S. S., Chakraborty, R. S., Mukhopadhyay, D., and Bhunia, S. Multi-level attacks: An Emerging Security Concern for Cryptographic Hardware. In *Proc. of DATE conference* (2011), IEEE, pp. 1–4.

[3] Bhasini, S., Danger, J., Guilley, S., Ngo, X. T., and Saurvage, L. Hardware Trojan Horses in Cryptographic IP Cores. In *Proc. of Workshop on FDTC* (2013), pp. 15–29.

[4] Blömer, J., and Seifert, J. Fault based cryptanalysis of the advanced encryption standard (AES). In *Financial Cryptography* (2003), Springer, pp. 162–181.

[5] Bomel, P., Crenne, J., Ye, L., Diguet, J., and Gogniat, G. Ultra-fast downloading of partial bitstreams through ethernet. In *Proc. of Architecture of Computing Systems* (2009), M. Berekovic, C. MÃijller-Schloer, C. Hochberger, and S. Wong, Eds., pp. 72–83.

[6] Bomel, P., Gogniat, G., and Diguet, J. A networked, lightweight and partially reconfigurable platform. In *Proc. of Reconfigurable Computing: Architectures, Tools and Applications* (2008), pp. 318–323.

[7] Chakraborty, R. S., Narasimhan, S., and Bhunia, S. Hardware trojan: Threats and emerging solutions. In *Proc. of IEEE Int. Workshop on HLDVT* (2009), IEEE, pp. 166–171.

[8] Chakraborty, R. S., Saha, I., Palchaudhuri, A., and Naik, G. K. Hardware Trojan Insertion by Direct Modification of FPGA Configuration Bitstream. *IEEE Design & Test of Computers 30*, 2 (apr 2013), 45–54.

[9] DARPA. *TRUST in Integrated Circuits (TIC).* [Online]. Available: http://www.darpa.mil/MTO/solicitations/baa07-24., 2007.

[10] Dusart, P., Letourneux, G., and Vivolo, O. Differential fault analysis on AES. In *Applied Cryptography and Network Security* (2003), Springer, pp. 293–306.

[11] Eguro, K. SIRC: An Extensible Reconfigurable Computing Communication API. In *Proc. of 18th IEEE Annual Int. Symposium on FCCM* (2010), pp. 135–138.

[12] Eto, E. Difference-based partial reconfiguration, 2003.

[13] Farag, M. M., Lerner, L. W., and Patterson, C. D. Interacting with Hardware Trojans over a network. In *Proc. of IEEE Intl Symposium on Hardware-Oriented Security and TRUST* (2012), pp. 69–74.

[14] Gallais, J., Grosssschädl, J., Hanley, N., Kasper, M., Medwed, M., Regazzoni, F., Schmidt, J., Tillich, S., and Wójcik, M. Hardware Trojans for inducing or amplifying side-channel leakage of cryptographic software. In *Trusted Systems*. Springer, 2011, pp. 253–270.

[15] Gören, S., Ozkurt, O., Yildiza, A., Ugurdag, H. F., Chakraborty, R. S., and Mukhopadhyay, D. Partial bitstream protection for low-cost FPGAs with physical unclonable function, obfuscation, and dynamic partial self reconfiguration. *Computers & Electrical Engineering* (2012).

[16] Gören, S., Turk, Y., Ozkurt, O., Yildiz, A., and Ugurdag, H. F. Achieving modular dynamic partial reconfiguration with a difference-based flow. In *Proc. of ACM/SIGDA int. symposium on FPGA* (2013), ACM, pp. 270–270.

[17] Jin, Y., and Makris, Y. Hardware Trojans in wireless cryptographic ICs. *IEEE Design & Test of Computers 27*, 1 (2010), 26–35.

[18] Mukhopadhyay, D. An improved fault based attack of the advanced encryption standard. In *Progress in Cryptology–AFRICACRYPT*. Springer, 2009, pp. 421–434.

[19] Piret, G., and Quisquater, J. A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In *Proc. of CHES*. Springer, 2003, pp. 77–88.

[20] Potkonjak, M., Nahapetian, A., Nelson, M., and Massey, T. Hardware Trojan horse detection using gate-level characterization. In *Proc. of 46th ACM/IEEE DAC* (2009), pp. 688–693.

[21] Rad, R. M., Wang, X., Tehranipoor, M., and Plusquellic, J. Power supply signal calibration techniques for improving detection resolution to hardware Trojans. In *Proc. of IEEE/ACM ICCAD* (2008), pp. 632–639.

[22] Salmani, H., Tehranipoor, M., and Plusquellic, J. New design strategy for improving hardware Trojan detection and reducing Trojan activation time. In *Proc. of IEEE Intl. Symposium on Hardware-Oriented Security and TRUST* (2009), pp. 66–73.

[23] Takahashi, J., and Fukunaga, T. Differential Fault Analysis on the AES Key Schedule. *IACR Cryptology ePrint Archive 2007* (2007), 480.

[24] Tehranipoor, M., and Koushanfar, F. A Survey of Hardware Trojan Taxonomy and Detection. *IEEE Design and Test of Computers 27*, 1 (2010), 10–25.

[25] Thomas, K. L., and Thompson, M. S. Performance Modeling of Reconfigurable Distributed based on OpenSPARC FPGA Board and the SIRC Communication Framework. In *Proc. of Int. conference on ReConFig* (2013).

[26] Tunstal, M., Mukhopadhyay, D., and Ali, S. Differential fault analysis of the advanced encryption standard using a single fault. In *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication*. Springer, 2011, pp. 224–233.

[27] Upegui, A., and Sanchez, E. Evolving hardware by dynamically reconfiguring xilinx fpgas. In *Evolvable Systems: From Biology to Hardware*, vol. 3637 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2005, pp. 56–65.

[28] Wang, L., Xie, H., and Luo, H. Novel Analysis Method of Power Signal for Integrated Circuits Trojan Detection. In *Proc. of Int. IEEE Symposium on IPFA* (2013), pp. 637–640.

[29] Wang, X., Salmani, H., Tehranipoor, M., and Plusquellic, J. Hardware Trojan Detection and Isolation Using Current Integration and Localized Current Analysis. In *Proc. IEEE Int. Symposium on DFTVS* (2008), pp. 87–95.

[30] Xiao, K., and Tehranipoor, M. BISA: Built-in self-authentication for preventing hardware Trojan insertion. In *Proc. of IEEE Intl. Symposium on Hardware-Oriented Security and TRUST* (2013), pp. 45–50.

[31] Xilinx Inc. *Partial Reconfiguration User Guide UG702 (v14.5)*. www.xilinx .com, April April 2013.