UNIVERSITY of York

This is a repository copy of *Remote dynamic partial reconfiguration: A threat to Internet-of-Things and embedded security applications*.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/id/eprint/117858/

Version: Accepted Version

Article:
Johnson, Anju Pulikkakudi orcid.org/0000-0002-7017-1644, Patranabis, Sikhar, Chakraborty, Rajat Subhra et al. (1 more author) (2017) Remote dynamic partial reconfiguration: A threat to Internet-of-Things and embedded security applications. Microprocessors and Microsystems. pp. 131-144. ISSN: 0141-9331

https://doi.org/10.1016/j.micpro.2017.06.005

Reuse
This article is distributed under the terms of the Creative Commons Attribution-NonCommercial-NoDerivs (CC BY-NC-ND) licence. This licence only allows you to download this work and share it with others as long as you credit the authors, but you can't change the article in any way or use it commercially. More information and the full terms of the licence here: https://creativecommons.org/licenses/

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

White Rose university consortium
Universities of Leeds, Sheffield & York

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

Remote Dynamic Partial Reconfiguration: A Threat to Internet-of-Things and Embedded Security Applications

Anju P. Johnson[a], Sikhar Patranabis[b], Rajat Subhra Chakraborty[b], Debdeep Mukhopadhyay[b]

[a] *Department of Electronic Engineering, University of York, UK – YO10 5DD*
[b] *Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, INDIA – 721302*

Abstract

The advent of the *Internet of Things* has motivated the use of *Field Programmable Gate Array* (FPGA) devices with *Dynamic Partial Reconfiguration* (DPR) capabilities for dynamic non-invasive modifications to circuits implemented on the FPGA. In particular, the ability to perform DPR over the network is essential in the context of a growing number of *Internet of Things* (IoT)-based and embedded security applications. However, the use of remote DPR brings with it a number of security threats that could lead to potentially catastrophic consequences in practical scenarios. In this paper, we demonstrate four examples where the remote DPR capability of the FPGA may be exploited by an adversary to launch *Hardware Trojan Horse* (HTH) attacks on commonly used security applications. We substantiate the threat by demonstrating remotely-launched attacks on Xilinx FPGA-based hardware implementations of a cryptographic algorithm, a true random number generator, and two processor based security applications - namely, a software implementation of a cryptographic algorithm and a cash dispensing scheme. The attacks are launched by on-the-fly transfer of malicious FPGA configuration bitstreams over an Ethernet connection to perform DPR and leak sensitive information. Finally, we comment on plausible countermeasures to prevent such attacks.

Keywords: Internet of Things, Dynamic Partial Reconfiguration, Field Programmable Gate Array, Hardware Trojan Horse, Hardware Security

1. Introduction

Internet of Things (IoT) today is a network of physical objects or *things*, embedded with electronics, software, sensors, and network connectivity, that can be sensed and controlled remotely across the existing communication network infrastructure. This ensemble of uniquely identifiable objects creates opportunities for more direct integration between the physical world and computer-based systems, resulting in improved efficiency, accuracy, and economic benefit [1]. Since IoT applications are often constrained by the dual requirements of high performance and parsimony of resources, the use of reconfigurable hardware has gained popularity. FPGAs are frequently used in IoT applications today due to their highly reconfigurable properties [2].

A recent extension to flexible reconfiguration capabilities of FPGAs is in the form of *Dynamic Partial Reconfiguration* (DPR). DPR allows dynamic, energy-efficient non-invasive modification of the existing circuit on the FPGA, mostly to enhance functionality in the form of added plug-ins. The "plug-and-play" philosophy is particularly suitable for IoT applications since it supports multiple functions at a very large scale without the need for having dedicated hardware available at all times. DPR-enabled FPGAs are thus ideal choices for IoT applications. DPR-enabled FPGAs have already become the platform of choice for certain distributed applications (e.g. deep neural network based image search), due to lower power consumption, lower reconfiguration latency and lower memory bandwidth requirements [3].

Email addresses: anju.johnson@york.ac.uk (Anju P. Johnson), sikharpatranabis@cse.iitkgp.ernet.in (Sikhar Patranabis), rschakraborty@cse.iitkgp.ernet.in (Rajat Subhra Chakraborty), debdeep@cse.iitkgp.ernet.in (Debdeep Mukhopadhyay)

DPR of FPGAs has been around in academia for more than two decades and have started to be employed in industrial applications. Recently, Xilinx has come up with solutions for cost sensitive applications targeting Industrial IoT (IIoT) [4, 5]. Intel and Microsoft, have started using FPGAs for industrial applications. Emerging IoT applications, such as Smart City infrastructure, intelligent factory automation, Smart Grid, and data center acceleration, already use Intel (Formerly Altera) FPGAs to take advantage of hardware and software programmability, security, and high-performance features [2]. Intel *Xeon E5* server processors chips combined with Altera's FPGAs is a recent enhancement for IIoT applications. Also, Intel's collaboration with Alibaba uses FPGAs for accelerating business applications [6]. Microsoft has also reported the applications of FPGAs in IIoT Neural Networks [7].

In this paper, we focus on *remote* DPR, where a bitstream is transferred to the FPGA remotely over the network to reconfigure one or more applications embedded on the FPGA. The fact that IoT applications often require a large number of devices to communicate remotely makes this a very powerful tool. However, despite the many advantages that remote DPR provides, it could lead to security threats that, unless addressed properly, have potentially catastrophic implications in practical IoT environments. According to survey results presented in [8], close to half (49%) of the respondents felt that IoT would roughly have the same level of security issues as the previous waves of technology. Of those who thought differently, slightly more were optimistic (21%) than pessimistic (17%). This survey also says that "The IoT is and will be a great security challenge and an opportunity for new ways of thinking about ecologies of security". To the best of our knowledge, the idea of using remote DPR for inserting Hardware Trojan Horse (HTH) was first reported in [9]. This work implants an add-on HTH remotely via DPR by transferring bitstreams corresponding to a malicious HTH. The newly implanted HTH extracts secret information from a 128-bit AES cryptographic core. A similar attack on a *True Random Number Generator* (TRNG) has been demonstrated in [10], in which an HTH circuit was introduced in the dynamic partition of the FPGA via DPR to bias the response of the TRNG. However, both of these above-mentioned attacks can be prevented by a bitstream validation mechanism, since the bitstream corresponding to the add-on explicitly encodes the logic for the inserted HTH. In contrast, our proposed approach does not directly insert an HTH circuit in the dynamic partition of the circuitry. Instead, it inserts a malicious controller for DPR in the on-chip clock management circuitry. This controller, in turn, invokes single/multiple DPRs for configuring various clock signals. The resulting patterned clock due to the observations listed in Section 3, behaves maliciously by posing a threat to the cryptographic system. This paper is an extension of the work published in [11].

The main contributions of this paper can be summarized as follows:

1. In this paper, we demonstrate that remote DPR-enabled FPGA-based systems can be subjected to malicious circuit alterations, typically termed as *Hardware Trojan Horse* (HTH) insertion via transmission of configuration bitstreams over the network, to compromise the security of one or more applications. This work demonstrates a remote fault injection technique on an FPGA using configuration bitstreams which invoke multiple DPRs in the FPGA.
2. We demonstrate successful remote attacks launched on Xilinx FPGA-based hardware implementations of a cryptographic algorithm, a true random number generator, and two processor based security applications - namely, a software implementation of a cryptographic algorithm and a cash dispensing scheme. The ability to remotely insert the HTH using the non-invasive property of remote DPR makes the attack model potent, easy to launch and difficult to counter. The demonstrated attacks imply that cryptographic hardware implementations in particular, which are omnipresent for applications such as secure communication, electronic fund transfer, etc. are extremely vulnerable if implemented on DPR-enabled FPGAs. Thus, remote DPR, although efficient, may not be the most secure choice for IoT today unless combined with appropriate countermeasures.

The rest of the paper is structured as follows. Section 2 describes a practical target architecture for our attacks and points out some of the pros and cons of remote DPR in the context of secure IoT and embedded applications. Section 3–4 realizes these threats in the form of concrete case stud-

ies on security critical applications mapped on the FPGA. Section 5 presents actual experimental results establishing the effectiveness of the proposed attacks. Section 6 proposes plausible countermeasures to prevent attacks via DPR. Finally, Section 7 concludes the paper.

2. Remote DPR: Architecture, Advantages and Threats

A number of present day IoT applications use FPGAs programmed with *Cryptographic Hardware Solutions* (CHSs). CHS includes commonly used hardware support for cryptography, e.g. hardware security modules, secure cryptoprocessors, tamper-resistant security modules, crypto-accelerators, embedded crypto-engines and other crypto-primitives. FPGAs programmed with CHS are potential targets for attacks by malicious agents, and are therefore of utmost importance in the context of overall IoT security. In this section, we present a detailed description of our target IoT architecture. It includes an FPGA with dynamic partial reconfiguration properties, and we assume the FPGA is programmed with one or more cryptographic hardware primitives. We then focus specifically on remote DPR based threats to this target architecture that could compromise its security.

2.1. Remote DPR Enabled IoT Architecture

Figure 1 depicts the architecture we are considering. The architecture consists of a DPR-enabled FPGA, connected to a network over a standard 100 Mbps or 1 Gbps Ethernet connection, providing real-time computational capabilities. This remote DPR capability of the FPGA is pivotal to on-chip, hardware modifications/enhancements and online hardware updates. This capability is vital to IoT applications since it allows optimal and scalable resource sharing. We further assume that our target FPGA is programmed with some cryptographic hardware solution (CHS). CHS includes commonly used hardware support for cryptography such as hardware security modules, secure cryptoprocessors, tamper-resistant security modules, crypto accelerators, embedded crypto-engines and other crypto primitives. FPGAs programmed with CHS are potential targets for attacks by malicious agents, and are therefore of utmost importance in the context of overall IoT security.

In the target architecture, an authorized user is allowed to perform remote DPR by transferring the required partial configuration bitstream file over a live Ethernet connection to the FPGA. Enabling DPR on an FPGA partitions the FPGA logic into two major parts - *static* and *dynamic*. The dynamic partition essentially comprises of the dynamic *add-on* modules to add or modify existing functionality. The configuration of the dynamic partition can be modified by DPR, while that of the static partition remains unmodified for the entire lifetime of the application being executed on the FPGA. In our target FPGA set-up, we assume that the static partition comprises of four major components - the application core (e.g. a cryptographic core or a processor), the *Digital Clock Manager* (DCM) module residing in the clock management tiles of the FPGA, and two controller modules - a DPR controller and an Ethernet API controller. The working of the DPR-enabled dynamic partition tiles is supervised by the master DPR controller, which generates the necessary control signals for DPR to the *Internal Configuration Access Port* (ICAP) [12]. The Ethernet API controller provides the much needed remote access channel for secure communication to cryptographic applications, as well as the remote transfer of DPR bitstreams. The other important aspect of our target architecture is the management of the data (both DPR and cryptographic) that is transferred to the FPGA via the Ethernet API controller. A popular choice is the open-source *Simple Interface for Reconfigurable Computing* (SIRC) platform [13, 9, 10]. SIRC consists of both software and synthesizable hardware components and facilitates the seamless transfer of arbitrary data to an FPGA via high-level C++ API calls. Moreover, SIRC is highly customizable and is hence an ideal choice for IoT applications.

The feasibility of DPR methodology described in the present paper has to be analysed in three aspects: Firstly, the hardware support for achieving DPR, secondly, the DPR bitstream generation and finally, an infrastructure for remote bitstream transfer. The main hardware support for DPR is an ICAP controller, a PR primitive for Xilinx FPGAs. Apart from the ICAP controller, the designer needs to implement a DPR controller which generates necessary signals for the ICAP controller. In our design, the DPR controller occupies 14 slices of the FPGA, proving it to be a lightweight and hence suitable for IoT application. Secondly, the bitstream generation is performed using Xilinx tools which do not require any special paid license, enabling this to be an easy target for all DPR ap-

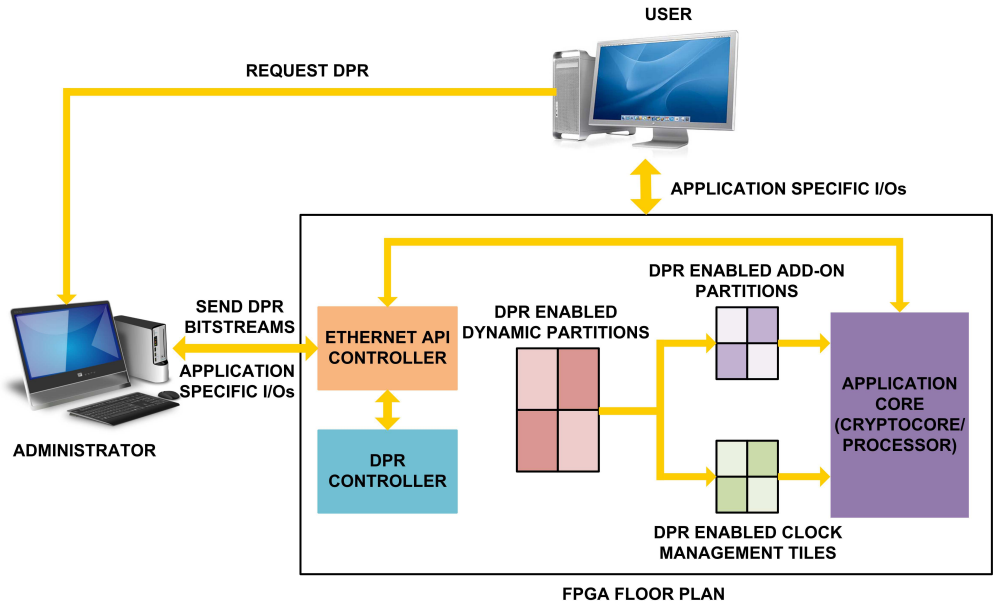

Figure 1: The DPR enabled target architecture for demonstration of attack.

plications. Equally crucial is the network connectivity of the FPGA, through which DPR files are transferred. This is established using an Ethernet API Controller. This is usually not an extra requirement, as it is relatively straightforward to design and deploy FPGA boards with wireless (and wired) connectivity, in fact, many such IoT moats with wireless connectivity are already commercially available [14]. In our application, we have demonstrated the remote bitstream transfer using an open source SIRC platform from Microsoft.

The fundamental advantage of remote DPR lies in the fact that it enables a number of devices to communicate remotely with each other to regulate the functioning of a number of FPGA-based functional modules, without the need for physical proximity. This results in better connectivity and easier management of a vast network of devices functioning simultaneously, especially in an IoT context. In the following section, we examine in greater detail how DPR may be used to address this crucial issue of dynamic clock reconfiguration.

2.2. DPR for Custom Clock Synthesis

In this section, we focus on a specific scenario in which DPR proves helpful in the context of our target architecture. At any given point of time, the dynamic partition of the FPGA consists of a number of applications that are plugged in and reconfigured

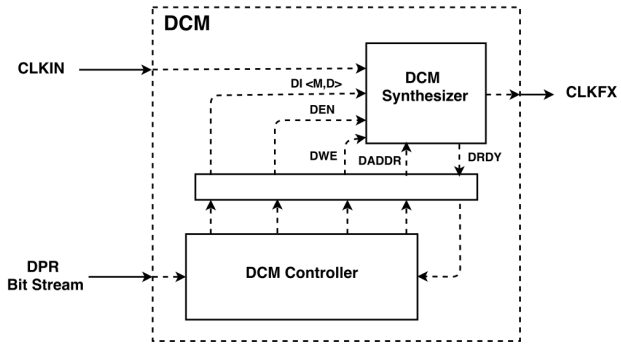

Figure 2: Control signals for dynamic partial reconfiguration of the digital clock manager.

on the fly. These applications may have varying requirements of the clock frequency, which are usually not known a priori. Hence, fixed clocking schemes are not sufficient to cater to such varying requirements. DPR, on the other hand, provides a way to address this issue by generating custom clocks on the fly depending on the requirements for various applications. The usual techniques to generate such custom clocks is to use some clock generation circuitry such as the *Phase Locked Loop* (PLL) module or the DCM module. In this paper, we focus on the use of DCM due to its ease of use. The DCM module is enabled in the *Clock Management*

Tiles (CMTs) of the FPGA. We denote by F_{CLKIN} the input clock signal to the DCM and by F_{CLKFX} the corresponding synthesized clock signal. We further define two major attributes of the DCM module - the $CLKFX_{MULTIPLY}$ attribute with value M and the $CLKFX_{DIVIDE}$ attribute with value D. Given these attributes, the relation between the input and output clock signals is given by:

$$F_{CLKFX} = F_{CLKIN} \times \frac{M}{D} \tag{1}$$

The DPR capability of the FPGA allows modification of the M and D values during runtime to synthesize various clock frequencies depending on the requirements of various applications. Figure 2 shows the control signals via which dynamic partial reconfiguration of the DCM takes place. The first step is to remotely pass on to the FPGA via an Ethernet connection, a configuration file containing the specific bitstream corresponding to the DCM controller. The DCM controller then generates the necessary control signals to the *Dynamic Reconfiguration Port* (DRP) of the DCM module to write the target M and D values, which are passed on via the data input bus DI. The M and D values are then written in the specific reconfiguration address provided by the $DADDR$ input bus of the DCM, provided that the write enable control signal DWE and the dynamic reconfiguration enable signal DEN are both high, and so is the $DRDY$ signal indicating that the DCM is ready for reconfiguration. The possible ranges for the M and D are specific to the FPGA family used. For the target FPGA in our architecture, which belongs to the Virtex-5 family, the range of the expression $\frac{M}{D}$ is between 2 and 32. This means that using DPR, one can synthesize a clock up to 32 times faster than the input clock.

2.3. An Example Application of DPR: Flexible AES Encryption Circuit

We now demonstrate an example of an application running on remote DPR enabled IoT architecture discussed previously. Let us assume that the static partition of the FPGA is mapped with a 128 bit AES encryption circuitry. Remote DPR can be used to reconfigure the cryptographic circuit to switch between specific modes of encryption such as cipher block chaining (CBC), output feedback (OFB), cipher feedback (CFB), and even tweakable encryptions.

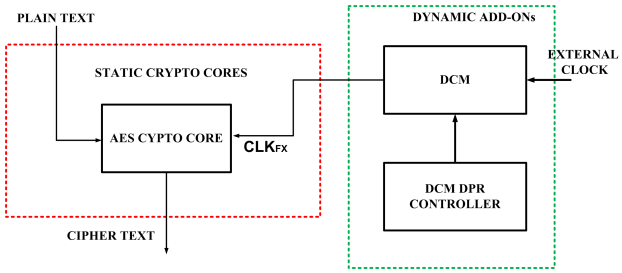

Figure 3: Architecture for DCM DPR through remote bitstream transfer for the AES encryption circuit.

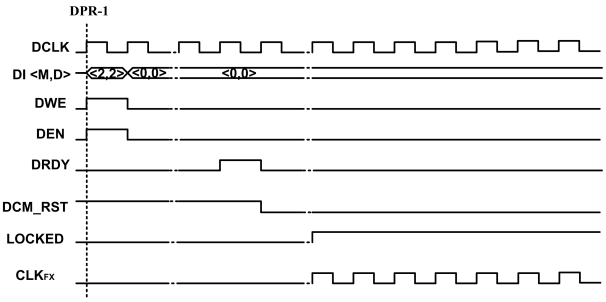

Figure 4: Waveform details for DCM controller to dynamically reconfigure an 100 MHz input clock signal to AES-128 encryption module.

For normal working of the 128-bit AES Encryption module mapped on to the FPGA, a clock signal has to be provided as input. This can be provided from clock generation circuitry such as *Phase Locked Loop* (PLL) or *Digital Clock Management* (DCM) module. In our implementation, we use DCMs for producing different clocks to all applications residing in the static or dynamic partition of the design. In this particular test case application, let us assume that remote DPR is configured to introduce a DCM DPR controller in the dynamic partition of the FPGA. This DPR controller for DCM generates the necessary signals which configure the output clock of DCM to 100 MHz, which is necessary for functioning of AES Encryption. The simplified representation of the example application is shown in Figure 3. The timing diagram to demonstrate the effect of the inserted bitstream on the working of the DCM controller is demonstrated in Figure 4. Other control signals DWE and DEN are set to logic high level for one cycle to initiate the DCM DPR phase. Once the clock is ready to be used, the $DRDY$ signal goes high. During the DPR cycle, DCM is held in the reset state by asserting DCM_RST. The setup requires a one-time

transfer of a partial reconfiguration file to the DCM controller of the FPGA using the remote connection, which in turn initiates a DPR instance via the DRP ports of the DCM to reconfigure the clock signal output to AES.

2.4. The Attack Model

Consider an 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 all users in the user group connected to the network is real-time private-key encryption of plaintexts transferred to it via the network 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" (Administrator) 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 bit file to an input buffer on the FPGA. 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 a great degree of flexibility for evolving hardware. This will also lead to reduced reconfiguration time since the new "add-on" is just an extension 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 does not have direct logical relation with the existing "add-ons".

Note that in the context of utilizing the services of the FPGA for encryption, and a superuser is similar to any other member of the authorized user group in the network. A "superuser" may be malicious and may attempt vulnerable hardware updations via DPR.

The IoT is a dynamic, ever-evolving entity. Hence, it requires flexibility in both hardware and software. It is common to have software updates/upgrades for devices connected to the Internet. On the contrary, it is far less common to update or upgrade hardware over a network. However, with the increasing use of FPGAs in mainstream computing, and the widespread availability of dynamic reconfiguration capabilities of such hardware, such situations are becoming more common. As an example, any fielded device can be upgraded with a new encryption algorithm. The need might be due to compatibility with new applications. Considering a cryptographic application, the existing algorithm may be broken. For example, an AES algorithm was created when *Data Encryption Standard* (DES) algorithm was broken. Similarly, an AES algorithm may be enhanced using specific encryption modes such as Cipher Block Chaining (CBC), Electronic Codebook (ECB), Cipher Feedback (CFB), Output Feedback (OFB), and Counter (CTR) using DPR. When attempting a DPR, the attacker gets limited access the static design which is used as a back-door for HTH insertion.

2.5. Exploitation of DPR for Remote HTH Insertion

Although DPR-enabled FPGAs provide us with a range of options and flexibility due to its ability to synthesize a wide variety of clocks by reconfiguring the DCM, they are vulnerable to security threats. When the FPGA is programmed with CHS modules that use the clock output of the DCM, a malicious adversary can reconfigure the DCM to launch attacks that leak secret information from cryptographic modules. This is HTH insertion in essence. We assume that every attempted DPR is logged on the FPGA, and it is not possible for a malicious user to make arbitrary modifications to the existing FPGA hardware without revealing her identity. However, the user can wait till a prescheduled and preauthorized DPR operation 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 can be remotely triggered using DPR to maliciously alter the working of the cryptographic circuit on the fly, without the need for any dedicated conditional trigger circuit to be present on the FPGA. This reduces the attack overhead, makes the attack model more practical, efficient and challenging to debug and defend against** [9, 10]. In

this paper, we focus on two such instances of remote Trojan insertion that remotely configure the output of the DCM to compromise the security of cryptographic modules:

- The first category of attacks we focus on in this paper are *clock glitch* based fault attacks. Clock glitches introduce faults in the normal functioning of the cryptographic core of the FPGA by causing *set-up time violations* in the critical path of the circuit. A number of low-cost clock glitch based fault attacks have been demonstrated in cryptographic literature [15]. Remote DPR, on the other hand, makes the threat of clock glitch based fault injections even more potent by allowing the adversary to simply transmit malicious DPR bitstreams over a remote connection to dynamically reconfigure the output clock of the DCM. However, traditional clock glitch based fault injection techniques are invasive, in the sense that the adversary must have physical access to the clock port of the target FPGA [15, 16]. **Remote DPR, on the other hand, makes the threat of clock glitch based fault injections even more potent by allowing the adversary to simply transmit malicious DPR bitstreams over a remote connection to dynamically reconfigure the output clock of the DCM, without being in physical proximity of the FPGA being attacked.**

- The second category of threats arises from alterations of DCM clock signals that are used to generate uniformly random distributions by sampling one or more random signals. The HTH can maliciously alter the frequency of the sampling clock signal to bias the resulting distribution towards a specific value or set of values. This, in turn, compromises the security of one or more cryptographic primitives (e.g. TRNG) that use this distribution for their computations.

- The third category of hardware trojan horse insertion via remote DPR targets a processor implemented on an FPGA. We implement a 128 bit AES encryption algorithm, in the processor. The clock frequency of the processor is adjustable by remote DPR. We HTH deliberately inserts fast clock leading to instruction skipping attacks there by leaking sensitive information on the AES Encryption key. We implement two different methodologies of instruction skip, which can lead to the recovery of AES Key.

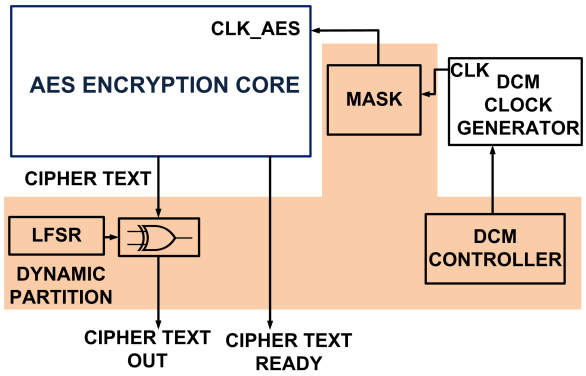

Figure 5: Hardware Trojan Insertion on AES Encryption Algorithm Implemented on an FPGA

- Final text-case of attack targets processor based implementation of cash machine algorithm. This enables malicious advisory to receive money or assets from a bank or other financial institution. This is implemented by skipping password checking in electronic machines using remote DPR targeting clocking circuits to the processor. This kind of attack particularly targets insecure retail outlets and cash machines.

In the next sections, we present two practical case studies to demonstrate how remote DPR capabilities can be leveraged to compromise the security of applications mapped on an FPGA.

3. Attacks on Cryptographic Primitives through Remote DPR

3.1. Case Study I: Remote DPR based Fault Attack on AES-128 Cryptographic Core

The first attack proposed attack targets an AES encryption circuit running on the FPGA and recovers the secret encryption key with a single remote fault injection using clock glitches. The second attack targets the TRNG on the FPGA, and significantly biases its output distribution, thus compromising its *true randomness* property.

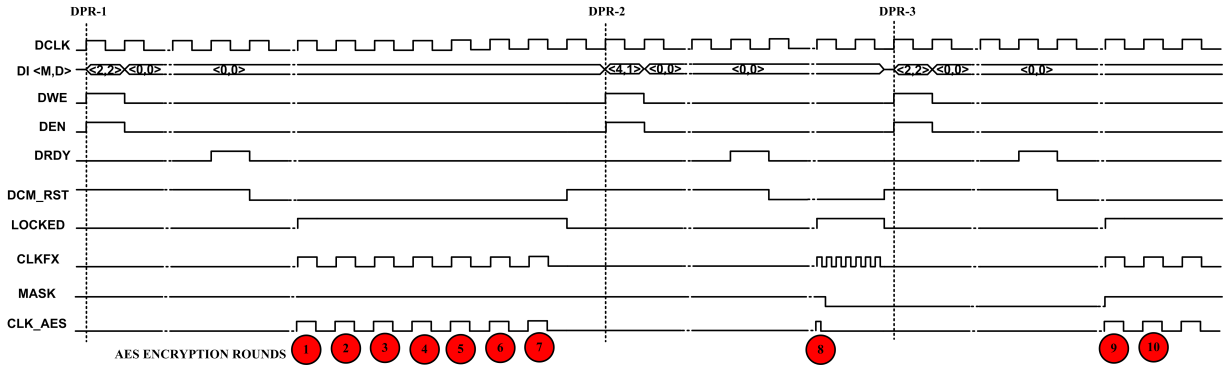

Figure 6: Timing diagram for DCM controller operation during Fault Attack on AES-128 encryption circuit using remote DPR. The red circles indicate AES encryption rounds. CLK_AES is the clock to the AES encryption module. Using the mask signal the fast HTH clock is selected exactly once in the eighth round (out of ten rounds) of AES encryption.

In a practical IoT environment, the FPGA can be envisaged to be programmed to execute cryptographic algorithms, including symmetric and public key encryption algorithms. The current global standard for symmetric key block ciphers is AES, which is used to provide bulk data encryption for several modern day applications. For our first case study, we program the FPGA with an AES-128 encryption circuit, which in conjunction with the supporting architecture for remote DPR, resides within the static partition of the FPGA floorplan described in Figure 1. The AES circuit is implemented in an iterative fashion with ten rounds, with a nominal working clock frequency of 100 MHz supplied from the DCM. Our aim is to recover the AES encryption key using an efficient fault attack that uses a single fault injection. For this, we employ the most efficient *Differential Fault Analysis* (DFA) attack on AES-128 proposed in the cryptographic literature [17], that uses only a single pair of faulty and fault-free ciphertexts. This DFA attack reduces the search space of the secret key from 2^{128} to 2^8. The attack requires a single byte fault injection at the start of the eighth round *MixColumns* operation of AES. This attack is extremely efficient and requires less than 10 minutes on a standard desktop PC. For the desired single byte fault injection, we use the remote DPR capability of the FPGA to transfer a malicious bitstream that reconfigures the output clock of the DCM and causes set-up time violations along the critical path of the deployed AES encryption circuit. The effect of the malicious bitstream on the working of the DCM controller is demonstrated in Figure 6. We assume that the adversary can distinguish between AES encryption rounds, and hence is able to switch the clock frequency to create a glitch at the appropriate round. The first seven rounds of AES are configured to run at 100 MHz using the first DPR at the DCM end. For this, the D and M values are each set to the nominal value 2 (as per equation. (1)). At the beginning of the eighth round, the DCM is reset and reconfigured to generate a 400 MHz clock signal by a second DPR at the DCM end. The corresponding design parameters D and M used are 1 and 4 respectively. The DCM is allowed to output this clock frequency for some cycles, after which it is brought back to 100 MHz using another DPR at the DCM end.

The DCM circuitry generates fast clock (400MHz) for a few clock cycles. We require a single fast clock cycle at the start of eighth round encryption to introduce a set up time violation. Generating a fast clock for exactly one round using DCM is non-trivial, and is accomplished by using a mask logic to filter out the unwanted fast clock cycles. The clock to the AES circuitry is provided from the output of a mask logic filter whose input is the output of DCM circuitry. A mask logic filter selects the fast clock for a single round (eighth round). This allows us to generate a clock pattern in which only the eighth round clock is fast. This introduces the fault in the eighth round via set-up time violations [18]. Once the fault is injected, the DCM is brought back to its original output frequency generation configuration, and the AES encryption completes (after the tenth round) with a faulty output ciphertext, as desired by an adversary. It is desirable to an adversary that no user except her, in the network would

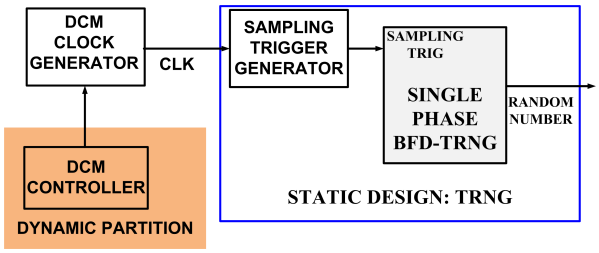

Figure 7: Hardware trojan insertion on single phase BFT TRNG implemented on an FPGA

be able to deduce the secret key from the faulty ciphertext. The faulty ciphertext output is garbled using the output values obtained from a "Linear Feedback Shift Register" (LFSR) circuit added as a part of the HTH in the dynamic partition. The adversary, knowing the initial configuration of the LFSR could easily recover the key. The design implemented on the FPGA is depicted in Figure 5. **The attack requires a one-time transfer of a partial reconfiguration file to the DCM controller of the FPGA using the remote connection, which in turn autonomously initiates three DPR operations via the DRP ports of the DCM to reconfigure the clock signal output to the AES encryption module, as described in Figure 6. This makes the attack extremely potent**.

3.2. Case Study II: Remote DPR based Attack on TRNG using Beat Frequency Detection

A TRNG is a hardware module that generates statistically independent random number sets by leveraging the inherent randomness in physical sources. A good TRNG must possess two major cryptographic properties - (a) an adversary should not be able to predict its response, based on knowledge of an unlimited number of previously generated bits, and (b) an adversary should not be able to bias its response towards a specific value (0 or 1), i.e., in an ideal TRNG, 0s and 1s should be equiprobable in the output bitstream.

In our case study, we assume that the target FPGA has a ring oscillator based TRNG that uses beat frequency detection mechanism to extract noise for random number generation (BFT TRNG) [19]. We briefly describe its working here. Assume that the target TRNG circuit consists of two quasi-identical ring oscillators, namely $ROSC_A$ (time period T_1) and $ROSC_B$ (time period T_2), with similar physical construction and placement. The difference in frequency of oscillation between the two oscillators at each time instance, resulting from random process variations, is sampled and recorded using a D flip-flop (DFF). As a result, the DFF outputs 1 at random beat frequency intervals, corresponding to specific *capture events*. The output clock signal of $ROSC_B$ drives a binary counter, whose output ramps up to random values before being reset each time the DFF outputs 1. The output of the counter is sampled by a sampling clock before it reaches its maximum value. The sampled response is then serialized to obtain the desired random bitstream. For more details on the working of the TRNG circuit, please refer to [19].

We use the remote DPR capability of the FPGA to design an HTH that biases the distribution of the TRNG output towards producing 0. Initially, the device is configured to operate at slower sampling clock (golden sampling clock), which produces a sampling trigger at the n-th $n = 162$ in our experiment) cycle of golden sampling clock. This produces a random distribution with mean m ($m = 170$ in our experiment). The counter output is sampled using a sampling trigger signal and is serialized to be used as a random bitstream. The design implemented on the FPGA is depicted in Figure 7.

The inserted HTH generates a patterned sampling clock which is superimposed on the DCM circuitry required to synthesize the clock for sampling the TRNG counter output. The DCM circuit is controlled using a controller module that is implanted in the dynamic part of the circuit utilizing the remote DPR facility of the FPGA, during an authorized DPR process. The HTH uses the remote DPR capability to configure the DCM controller using a malicious bitstream. The DCM controller, in turn, generates control signals that configure the DCM dynamically to introduce a fast sampling clock. The fast HTH sampling clock generates a sampling trigger in lesser time than that would be if run by a golden sampling clock. As a result, the TRNG counter gets hardly any opportunity to increment before being reset by the sampling trigger signal. Consequently, the TRNG response is biased towards lower counter values, which in turn biases the output bitstream towards 0.

The attack requires a one-time transfer of a partial reconfiguration file to the DCM controller of the FPGA using the remote connection, which in turn autonomously initiates a DPR operation via the DRP ports

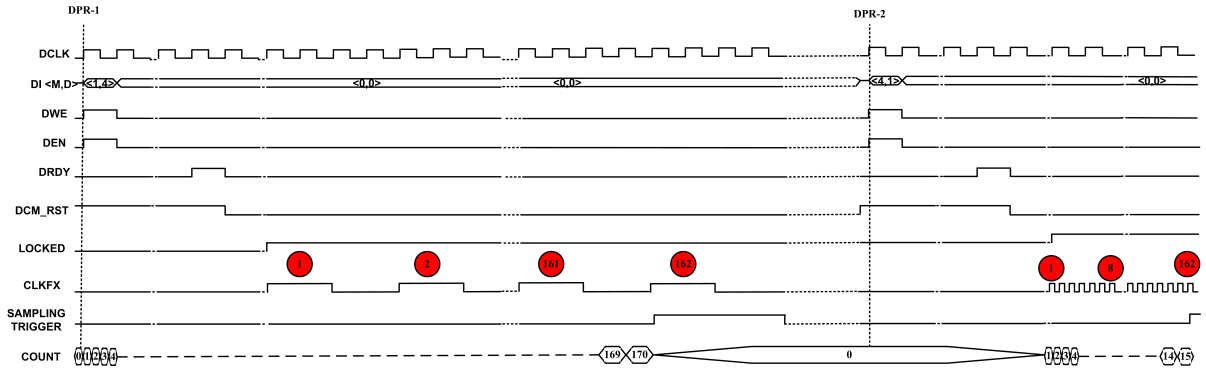

Figure 8: Timing diagram for DCM controller operations during the HTH attack on TRNG using remote DPR. The red circles indicate cycles of DCM output clock signals. Exactly at the 162^{th} cycle of the DCM clock, a sampling trigger is generated to sample the counter output.

of the DCM whenever is biasing is desired to reconfigure the clock signal output to the TRNG sampler. The overall attack is practical and relatively easy to launch and exploits the remote DPR capability of the FPGA to remotely trigger the HTH. Since the sampling clock generator is an indispensable part of the TRNG architecture, superimposing the proposed HTH on this sampling clock management circuitry reduces the hardware overhead associated with the HTH design. The working of the HTH, along with other control signals required to remotely activate the DPR in the DCM module, is pictorially depicted in Figure 8.

4. Attacks on Processor through Remote DPR

Cryptography is the fundamental component for securing the Internet traffic, particularly in an IoT environment. Most of the cryptographic algorithms demand enormous processing power. This introduces a bottleneck in high-speed network applications. To fully utilize the network bandwidth, the cryptographic algorithm implementation needs to achieve high processing rate. Additionally, IoT applications operate in an ever changing environment. To incorporate the modifications in cryptographic algorithms and standards, their implementation must be upgradable in-field, contrarily, offline upgrades results in excessive cost. A solution to this problem would be implementations of secure features in an adaptive processor that can provide software-like flexibility with hardware-like performance, where FPGA-based implementation is an excellent choice. This section is intended to demonstrate practical attacks on applications mapped on a processor implementation on an FPGA. The idea is to exploit DPR capability of FPGA devices to recover secret information from a cryptographic encryption algorithm implemented on a processor. We also target to weaken the security of a cash machine scheme implemented on the processor. Both of these attacks are based on instruction skipping and pose a severe threat to the security of applications mapped on an FPGA-based processor. The attacks are demonstrated on a *Xilinx PicoBlaze Processor* [20]. The concept works fine for other processor implementation on the FPGA also. PicoBlaze is an 8-bit microcontroller, which requires two clock cycles per instruction execution.

4.1. Case Study III: Remote DPR based Attacks on Processor Implementation of AES-128 Encryption Algorithm

The AES encryption algorithm is implemented on a Xilinx PicoBlaze processor using the following subroutines and instruction:

1. LOADKEY (A) (loads the AES 128-bit key to the processor state matrix position-1) (Scratchpad RAM for PicoBlaze processor),
2. LOADPLAINTEXT (B) (reads the AES 128-bit plaintext to the processor state matrix position-2),
3. XORROUNDKEY (C) (performs bitwise XOR operation on contents in processor state matrix position-1 and position -2 and stores the result in state matrix position-2),

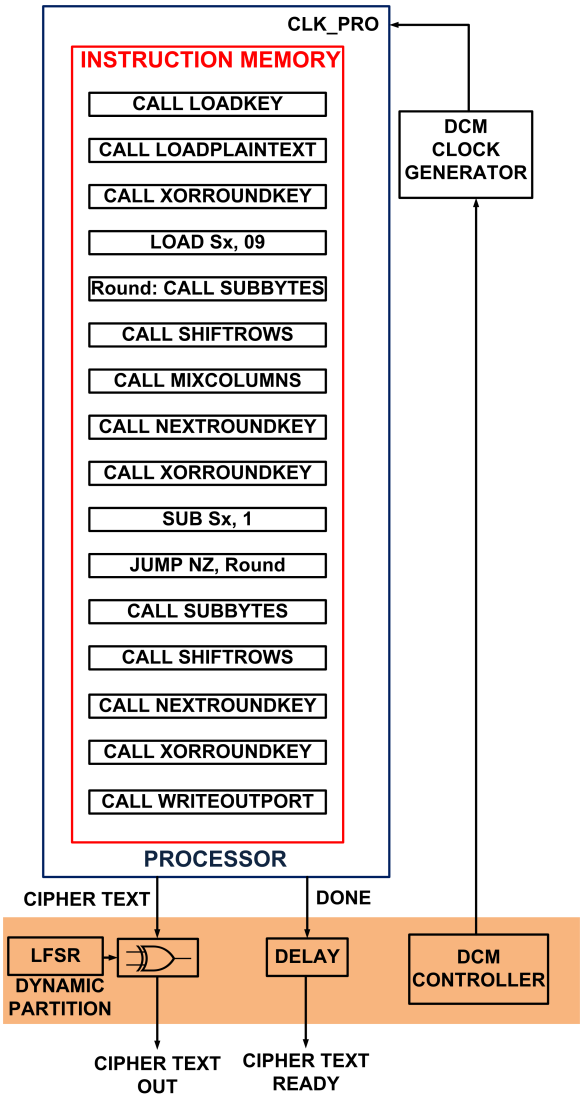

Figure 9: Hardware trojan insertion on AES encryption algorithm implemented on a processor.

4. WRITEOUTPORT (K) (writes the content of processor state matrix position-2 to the output port of the processor),
5. SUBBYTES (E) does the substitution operation,
6. SHIFTROWS (F) cyclically shifts the bytes in each row by a certain offset,
7. MIXCOLUMNS (G) together with F provides diffusion in the system,
8. NEXTROUNDKEY (H) derives a sub key in the system,
9. LOAD Sx,9 (D) (Load register with value 9),
10. SUB Sx,1 (I) (Subtract 1 from the content of Sx) and
11. JUMP NZ Round (J) (To incorporate round loop).

The flow of AES encryption implementation on the processor is depicted in Figure 9.

The attack is implemented by following the strategy described below: Skip all the instruction that follow the first "CALL XORROUNDKEY" instruction, except the "CALL WRITEOUTPORT" instruction. This requires 12 instructions to be skipped and requires 24 consecutive fast clocks.

This method is straightforward as the HTH operates on consecutive sets of fast clock. This requires only a single DPR to induce a fast clock of 24 clock cycles and another DPR to bring the system back to the original operating frequency of 100 MHz. The key can be trivially recovered using a single "bitwise XOR" operation between the ciphertext and the plaintext. This methodology produces the faulty ciphertext with a small number of clock cycles (only a few instruction is actually executed by the encryption algorithm).This might cause suspicions from outside and hence we introduce the necessary delay before delivering the cipher text. It is desirable to an adversary that no user except her, in the network would be able to deduce the secret key from the faulty ciphertext. The faulty ciphertext output is garbled using the output values obtained from a "Linear Feedback Shift Register" (LFSR) circuit added as a part of the HTH in the dynamic partition. The adversary, knowing the initial configuration of the LFSR could easily recover the key. The control signals required for generating the fast clocks is provided by the malicious DCM controller residing in the dynamic partition of the device. The whole circuit arrangement is shown in Figure 9. The working of HTH along with other control signals required to remotely activate the DPR in the DCM module, is pictorially depicted in Figure 10. **The attack requires an one-time transfer of a partial reconfiguration file to reconfigure the DCM controller, LFSR and the Delay logic residing in the dynamic partition of the design of the FPGA using the remote connection, which in turn autonomously initiates three DPR operations via the DRP ports of the DCM to reconfigure the clock signal output to the processor, as described in Figure 10. This makes the attack extremely potent.**

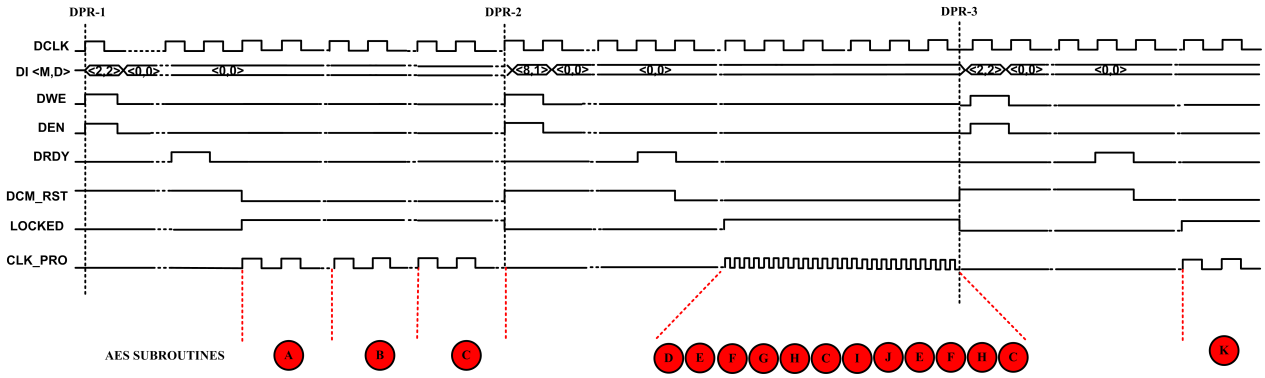

Figure 10: Timing diagram for DCM controller operation during instruction skipping attack on AES-128 encryption algorithm implemented on a PicoBlaze processor using remote DPR. The red circles indicate AES algorithm subroutines. *CLK_PRO* is the clock to the processor module.

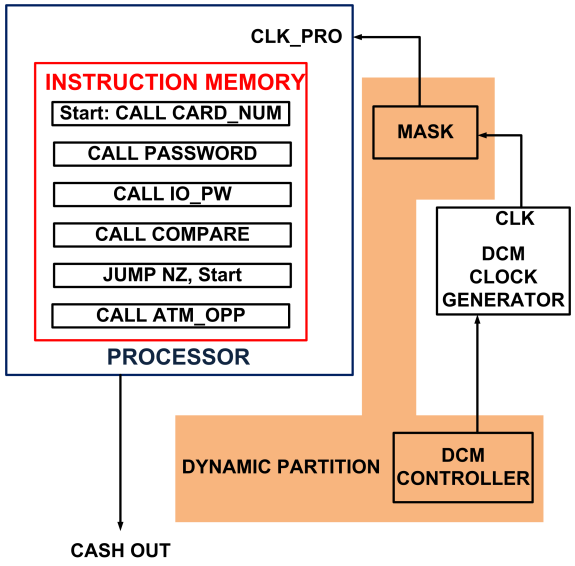

Figure 11: Hardware trojan insertion on cash machine scheme implemented on a processor.

4.2. Case Study IV: Remote DPR based Attacks on Processor Implementation of Cash Machine Scheme

The cash machine scheme is implemented on a processor using the following subroutines and instructions: CALL CARD_NUM (A) reads the number of the inserted bank cash machine card. CALL PASSWORD (B) reads the password corresponding to the inserted card from the secure password storage memory and stores the details in processor state matrix position-1. CALL IO_PW (C) reads the password entered by the user and stores the details in processor state matrix position-2. COMPARE (D) subroutine compares the result stored in position-1 and position-2 of the scratch pad memory. This operation effects the zero flag of the processor. The cash machine scheme continues to the transaction stages only if the above instruction is successful. This is tested using the JUMP instruction (E). Once successful, the subroutine ATM_OPP is performed (F), which leads to withdrawal of the desired amount of cash from the machine. This flow is depicted in Figure 11.

The cash machine scheme can be attacked by using a single fast clock targeting the instruction JUMP (E). The clock to the processor is provided from the output of a mask logic filter whose input is the output of DCM DCM circuitry. A mask logic filter selects the fast clock for two rounds . Since generating a fast clock for exactly two cycle is nontrivial, we use a mask logic filter to filter out unwanted fast clock cycles. This generates a patterned clock in which exactly the clock corresponding to instruction "E" is fast, leading to the corresponding instruction skip. Once the fast clock is induced, the DCM is brought back to its original output frequency of 100 MHz using another DPR. **The attack requires a one-time transfer of a partial reconfiguration file to configure the DCM controller and the mask logic. The DCM controller in turn autonomously initiates three DPR operations via DPR ports of the DCM to reconfigure the clock signal output to the processor, as described in Figure 12.**

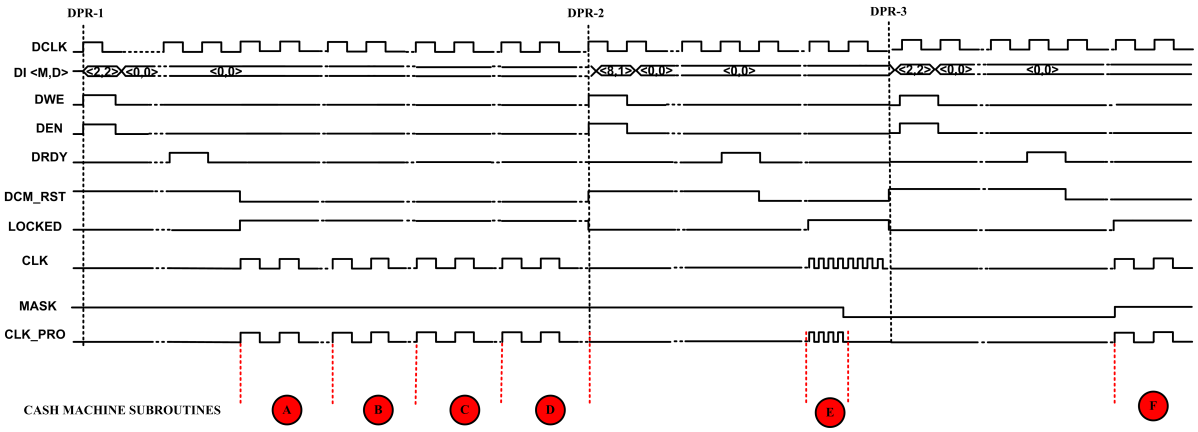

Figure 12: Timing diagram for DCM controller operation during instruction skipping attack on a cash machine scheme implemented on a PicoBlaze processor using remote DPR. The red circles indicate cash machine scheme subroutines. CLK_PRO is the clock to the processor module.

5. Experimental Study and Results

We first describe the experimental set-up to realize the attacks on the cryptographic cores and the processor. This is followed by a presentation of experimental results illustrating the efficiency of the proposed attack methods.

5.1. Experimental Setup

The test case designs for our experiments, including the cryptographic primitives and the HTH, were described in *Verilog*, synthesized and implemented using *Xilinx ISE 14.7* for the *Xilinx XUP Virtex-5-LX110T* FPGA target platform, and were simulated using *Xilinx Isim*. For generating the DPR bitstreams, we used the difference based methodology proposed in [21, 22] using *Xilinx FPGA Editor* software to reduce transmission overhead over the Ethernet, and since it does not require any special CAD tool design license. Remote access capabilities for the FPGA were incorporated by appropriately modifying the *SIRC* framework from *Microsoft* [13]. We monitored the power dissipation of the circuit using the *Xilinx XPower Analyzer* tool, while delay estimation was accomplished using *Xilinx Timing Analyzer*. The computations for recovering the secret key were performed on a PC with 2 GB of main memory and a 2 GHz CPU.

5.2. Attack on AES Encryption Circuit

An iterative implementation of AES-128 with ten rounds per encryption. The maximum operating frequency for the standalone AES implementation was found to be 100 MHz. Any value higher than this is expected to produce a faulty ciphertext. In our experiments, the fast clock frequency used to introduce glitches via setup time violation was 400 MHz, using the DCM parameters $M = 4$ and $D = 1$. In order to ensure that the fault spreads to all bytes of the final AES state, the clock glitch was remotely triggered at the beginning of the 8-th encryption round. The remote triggering was done using malicious DPR bitstream transmitted to the DCM controller via Ethernet, forcing it, in turn, to generate control signals to the DRP ports of the DCM, to switch to the faster clock, and then return back to the original clock frequency.

5.3. Attack on TRNG

The ring oscillators for the TRNG module were designed as "hard macros", with a target (nominal) time period of 38.00 ns. The actual time periods were observed to be $T1 = 37.9256$ ns and $T2 = 38.0000$ ns respectively. The frequency difference of 0.2959% between the two oscillators was due to process variation effects. The maximum value that the counter could reach was 255 (for an 8-bit counter), while the golden sampling clock frequency was set such that the sampled value was around 170 (sampling value). The fast HTH clock was designed according to the number and position of bits of the TRNG output that were to be biased to 0. For instance, suppose that we aim to bias the four MSB bits of the TRNG output to 0 in one random number generation cycle. The four LSB bits of the counter, on the other hand, are ran-

dom. The maximum possible value of the counter, in this case, is 15, which we refer to as $count_{max}$. Let $T = (\text{sampling value}) * T2$, where T is the total time required to generate the sampling trigger when the counter reaches the desired sampling value. For our TRNG design, we have $T = 6460$ ns. Let S be the total number of cycles of the golden sampling clock (T_g=25 MHz, see Figure 8, $M = 1$ and $D = 4$) required to sample at this rate. Then, we have $S = \frac{T}{T_g}$ (~ 162). Finally, if T_{fast} is the infected sampling clock triggered by the HTH with frequency F_{fast}, the following relation must hold:

$$T_{fast} \leq \left\lfloor \frac{(count_{max}) * T_2}{S} \right\rfloor \quad (2)$$

For our experiments, we intended to have $T_{fast} \leq 3$ ns (conversely, $F_{fast} \geq 333.333$ MHz). So we set F_{fast} to be 400 MHz, corresponding to $M = 4$ and $D = 1$. This leads to a biased distribution for the four MSB bits of the TRNG output towards 0. Following similar calculations, increasing the clock frequency even further to 500 MHz using remote dynamic controls to the DCM biased each bit of the entire TRNG output towards 0.

5.4. Attack on Processor based AES Encryption Algorithm

We used a Xilinx PicoBlaze processor to program an iterative implementation of AES-128 with ten rounds per encryption. PicoBlaze processor requires two clock cycles for an instruction execution. The processor works correctly for a clock frequency of 100 MHz. Processing speed higher than this is expected to produce a skip in instruction execution. In our experiments, the fast clock frequency used to introduce instruction skip was 800 MHz, using the DCM parameters $M = 8$ and $D = 1$. The DCM controller was designed to trigger fast clocks at the start of instructions to be skipped. The malicious DPR bitstream was transmitted to the DCM controller via Ethernet, forcing it, in turn, to generate control signals to the DRP ports of the DCM, to switch to the faster clock, and then return back to the original clock frequency. The attack is feasible by a distributed fast clocks as well as consecutive fast clocks. Since PicoBlaze is an 8-bit micro controller (cipher text is output in 16 times, 8-bits each), an 8-bit LFSR is used to scramble the resultant faulty cipher text.

5.5. Attack on Processor based Cash Machine Algorithm

The same Xilinx PicoBlaze processor to program the cash machine algorithm. The attack was relatively simple to launch as it requires only two cycles of the fast clock. The attack requires only a single instruction skip. In our experiments, the fast clock frequency used to introduce instruction skip was 800 MHz, using the DCM parameters $M = 8$ and $D = 1$. The DCM controller was designed to trigger fast clocks at the start of instructions to be skipped. The malicious DPR bitstream was transmitted to the DCM controller via Ethernet, forcing it, in turn, to generate control signals to the DRP ports of the DCM, to switch to the faster clock, and then return back to the original clock frequency.

5.6. Effectiveness of the Difference based DPR Methodology

This ensures that the reconfiguration bitstream files have small sizes, that can be transmitted easily over a remote network with very little cost. Table 1 demonstrates that the required partial bitstream sizes for our attacks are relatively very small. We assume that the FPGA dynamic partition is blank before the first partial reconfiguration is performed. Hence, to configure a new "add-on" it is required to transfer PR bitstreams to the FPGA (Blank-to-P1). In order to remove a particular "add-on", it is required to transfer bitstreams which erase the "add-on" logic (P1-to-Blank). The ICAP primitive of the targeted FPGA board is configured to run in an 8-bit configuration at a clock frequency of 100 MHz. This ensures that DPR of the "add-on" DCM controller can be performed efficiently in order of micro-seconds time.

5.7. Results of Remote Attack on AES Core using Clock Glitches

Table 2 summarizes the percentage increase in the total on-chip power consumption of the AES design (Case Study I), before and after HTH insertion, while Table 3 (Case Study I) compares the critical path delays before and after HTH insertion in the dynamic partition. Also, Table 4 (Case Study I) compares the hardware overhead for the golden and the HTH-infected AES designs. From these results, it is clear that the HTH insertion can be realized using minimal hardware overhead, and a negligible effect on the power consumption as well as the circuit delay. Also, no significant variation was observed

Table 1: **Partial Reconfiguration File Size**

Test Case Attack	Complete Bitstream Size (kB)	Partial Reconfiguration File (P1-to-Blank) (kB)	Partial Reconfiguration (Blank-to-P1) File (kB)
AES Cryptographic Core	3890.016	44.368	44.368
TRNG	3890.017	27.681	27.681
Processor based AES Algorithm	3889.964	65.959	65.959
Processor based Cash Machine Algorithm	3889.964	59.275	59.275

Table 2: **Power Overhead from HTH Insertion**

Design	Golden Reference (Blank Dynamic partition)(W)	Trojan Infected Circuit(W)	Increase in Power w.r.t.Golden (%)
AES Cryptographic Core	3.571	3.712	3.9485
TRNG	3.628	3.628	0.00
Processor based AES Algorithm	3.799	3.892	3.3226
Processor based Cash Machine Algorithm	3.371	3.454	2.4622

in the simulated power traces obtained from the Trojan-free and the Trojan-infected designs. This establishes the fact that our proposed attack is indeed practical, and can be used to obtain the secret key of AES remotely in less than ten minutes.

5.8. Results of Remote Attack on TRNG using a Faster Sampling Clock

Our remote attack on the 8-bit TRNG module involves biasing distribution of the output bits. In particular, we performed an attack that biased the four MSB bits of the TRNG output to 0, while the four LSB bits were allowed to take any random value. The golden sampling clock frequency F_g for the TRNG was set such that the output counter value was around 170, while in the biased scenario the clock frequency was so modified to F_{fast} as to allow a maximum counter value 15. For our experiments, we set F_{fast} to be 400 MHz (corresponding to $M = 4$ and $D = 1$ from equation (1)). This allowed us to bias distribution for the four MSB bits of the TRNG output towards 0. Increasing the clock frequency even further to 500 MHz biased each bit of the entire TRNG output towards 0. The faster sampling clock frequencies were obtained by remote DPR of the DCM controller using maliciously transferred bit streams.

Table 2 (Case Study II) shows the percentage increase in the total on-chip power consumption of the TRNG design before and after Trojan insertion, while Table 4 (Case Study II) compares the hardware overheads for the golden and the HTH-infected designs. From these results, it is clear that the HTH insertion can be realized using minimal hardware overhead. The probability of occurrence of a 0 at any of the output bit positions is expected to depend on the number of instances of random number generation cycle that are run using the fast sampling clock. This was precisely observed in our experimental results, where we were able to bias the TRNG response to 0 at predefined positions using high-frequency sampling clock. At these positions, the probability of zeros ($P(0)$) in the generated TRNG response was close to 0.75, which is higher than the expected ideal value of 0.5 for an ideal TRNG.

We also performed NIST statistical tests [23] on the output samples produced after the attacks. We collected 20 million samples from the golden TRNG design, and the infected TRNG design. For a block size of 100, the golden reference design passed all the NIST tests (P-value $\chi^2 > 0.01$ and Proportion ≥ 0.96), whereas the infected design failed all NIST tests.

5.9. Results of Remote Attack on processor based implementation of AES Encryption Algorithm

Table 2 summarizes the percentage increase in the total on-chip power consumption of the processor based AES encryption algorithm (Case Study III), before and after HTH insertion, while Table 3 (Case Study III) compares the critical path delays before and after HTH insertion in the dynamic partition. Also, Table 4 (Case Study III) compares the hardware overhead for the golden and the HTH-infected AES designs. From these results, it is clear that the HTH insertion can be realized using minimal hardware overhead, and a negligible effect on the power consumption as well as the circuit delay. Also, no significant variation was observed in the simulated power traces obtained from the Trojan-free and the Trojan-infected designs. This establishes the fact that our proposed attack is indeed practical, and can be used to obtain the secret key

Table 3: **Timing Overhead from HTH Insertion (Minimum Period)**

Design	Design Without Trojan (ns)	Design With Trojan (ns)	Increase in Critical Path Delay (%)
AES Cryptographic Core	10.194	10.194	0.00
TRNG	7.275	7.275	0.00
Processor based AES Algorithm	11.684	11.684	0.00
Processor based Cash Machine Algorithm	11.699	11.699	0.00

Table 4: **Hardware Overhead from HTH Insertion**

Design	Device Utilization	Golden Reference Design	Trojan Infected Design	Hardware Overhead w.r.t Golden (%)
AES Cryptographic Core	Slice	3386	3403	0.5021
	SliceReg	5217	5239	0.4217
	LUTs	7988	8019	0.3881
TRNG	Slice	2342	2348	0.2562
	SliceReg	4214	4227	0.3085
	LUTs	6446	6454	0.1241
Processor based AES Algorithm	Slice	2343	2380	1.00
	SliceReg	4153	4326	4.165
	LUTs	6591	6689	1.486
Processor based Cash Machine Algorithm	Slice	2771	2792	0.7578
	SliceReg	4271	4302	0.7258
	LUTs	6591	6645	0.8193

of AES remotely using a single bitwise XOR operation.

5.10. Results of Remote Attack on processor based implementation of Cash Machine Algorithm

Table 2 summarizes the percentage increase in the total on-chip power consumption of the processor based AES encryption algorithm (Case Study IV), before and after HTH insertion, while Table 3 (Case Study IV) compares the critical path delays before and after HTH insertion in the dynamic partition. Also, Table 4 (Case Study IV) compares the hardware overhead for the golden and the HTH-infected AES designs. From these results, it is clear that the HTH imposes a negligible impact on every measurable circuit parameter. The HTHs do not show any impact on delay, even the latest reported delay based trojan detection methodology [24, 25] fails in this case.

6. Countermeasures against the proposed attack

Although remote hardware updates are vital to IoT applications, as demonstrated in this work, straightforward enabling of DPR on DPR-capable FPGAs is a double-edged sword, with severe security vulnerabilities as they are prone to serious malicious hardware updations.

On closely analyzing the proposed attacks, it should be clear that the targeted IoT architecture requires two layers of security, one at the bitstream insertion level and second at the CMT end. Malicious bitstream insertions can be avoided if DPR is permitted restrictively in the system. For this, one possible solution is to pre-store signatures (e.g. cryptographic hashes) of possible hardware modifications to the system after successful passing of the design through HTH validations mechanisms. Any attempted DPR is allowed to modify the hardware if the signature of the partial bitstream matches with any of the pre-stored values [9].

To prevent attacks via fast HTH clock insertions by DPR at the CMT end, pre-storing clock modeling parameters based on the analysis of maximum and minimum operating conditions of the logic residing in the static partition of the design might be adopted. For example, only restrictive modification of M and D values for the DCM can be allowed. Applications targeting restricted mode of DPRs using CMTs has been reported in [26].

However, although attaining security of IoT devices and embedded security applications is extremely important, compromising remote DPR capabilities by only restrictively enabling DPR, the system may not be a wise choice, as this will not allow the capability of remote DPR to be harnessed to its fullest extent. To avoid DPR being exploited by attacks, and to permit its advantages to the fullest possible extent, careful protocol design to transfer only trusted DPR bitstreams to the FPGA is needed as well. A security protocol for unrestricted

DPR using*Physical Unclonable Functions* (PUFs) for FPGA authentication and bitstream validation is a wise choice [27]. Our future research efforts would be directed towards designing a secure IoT protocol that permits unrestricted DPR to be performed on the system. The technique proposed in [28] for integrity protection and authentication of DPR bitstreams can also be adopted. The "Xilinx controller for encrypted partial reconfiguration" can be used to prevent arbitrary unauthorized modifications to the bitsreams [29].

Recent researches on HTH insertions via DPR points out the inevitable need of post-deployment anomaly detection for circuits residing in the dynamic partitions of the FPGA [9, 10, 11]. For this, it is required to incorporate *Design for Testability* (DFT) in the dynamic partition. One way to achieve this is to modulate the single large dynamic partition into reasonable sized smaller partitions. Access to these small sized partitions from the static testing circuit in the FPGA shall be allowed using bus-macros or Flip-flops. We'll be looking for the design of secure dynamically reconfigurable architecture in our future works.

The attack described in Section 3.1, falls under the category of DFA attacks, where a single byte fault is induced to reduce the AES Key. DFA countermeasure is an excellent choice for attacks of this nature particularly, Concurrent Error Detection [30]. Higher-order masking schemes, detection and infection countermeasures, and their combinations are also other choices [31]. A survey on existing DFA countermeasures is presented in [32] with insight into suitable DFA prevention techniques.

The attack described in Section 3.2 is possible due to the week design of the TRNG. This attack emphasis the need of TRNG designs with Built-In Tolerance or self-repair capabilities against HTH Attacks. Biasing effects can be minimized using post processing stage, for example in [33], the authors come up with the use of resilient functions to withstand attacks on TRNG.

Section 4 describes attacks on processor targeting instruction skipping operation. Integrity checks can be adopted to achieve code integrity [34, 35] in secure embedded processors. Recovery schemes based on checkpoint and rollbacks [36, 37] also prevents instruction skipping attacks, but have a high overhead in terms of storage space.

7. Conclusions

This work has demonstrated severe threats due to in-field remote HTH insertions to FPGAs, that have the potential to compromise the system security of entire IoT networks. We have demonstrated the leakage of cryptographic keys, biasing the probability of 0s and 1s in the output bitstream of a TRNG, and threats to processor implementations of secure applications. Cyber threat attacks have shown themselves to be capable of accommodating rapid technological changes, taking full advantage of the present of internet enabled technologies to develop new and powerful attack vectors. Knowledge of the threats, together with appropriate countermeasures, provides essential information for the design of secure IoT infrastructure that enables IoT to be benefited by remote hardware update methodology. Identifying the threats that are specific to the particular systems with DPR-capable FPGAs, prioritizing protection against them, and effective implementation of the protection schemes is very important and will require considerable research effort in future.

References

[1] O. Vermesan, P. Friess, Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems, River Publishers, 2013.

[2] Altera, [Online]. Available: https://www.altera.com/solutions/technology/ iot/overview.html, IoT Applications, accessed: 2015-11-01 (2015).

[3] C. Edwards, Growing Pains for Deep Learning, Communications of the ACM 58 (7) 14–16.

[4] Xilinx Extends its Cost-Optimized Portfolio Targeting a Wide Range of Applications Including Embedded Vision and Industrial IoT, [Online]. Available: `https://www.xilinx.com/news/press/2016/xilinx-extends-its-cost-optimized-portfolio-targeting-a-wide-range-of-applications-including-embedded-vision-and-industrial-iot.html`, accessed: 2017-04-06 (September 2016).

[5] Xilinx Zynq SoCs and MPSoCs Power Embedded Vision and IIoT Applications at ARM TechCon 2016, [Online]. Available: `https://www.xilinx.com/news/press/2016/xilinx-zynq-socs-and-mpsocs-power-embedded-vision-and-iiot-applications-at-arm-techcon-2016.html`, accessed: 2017-04-06 (October 2016).

[6] Intel Collaborates with Alibaba Cloud to Help Customers Accelerate Business Applications, [Online]. Available: `https://newsroom.intel.com/news/alibaba-collaborating-intel-fpga-based-solution-help-customers-accelerate-business-applications/?wapkw=xeon+fpga`, accessed: 2017-04-06 (March 2017).

[7] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, E. Chung, Accelerating deep convo-

lutional neural networks using specialized hardware, [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/CNN20Whitepaper.pdf (February 2015).

[8] J. Pescatore, Securing the Internet of Things Survey, in: A SANS Analyst Survey, 2014.

[9] A. P. Johnson, S. Saha, R. S. Chakraborty, D. Mukhopadhyay, S. Gören, Fault Attack on AES via Hardware Trojan Insertion by Dynamic Partial Reconfiguration of FPGA over Ethernet, in: 9th Workshop on Embedded Systems Security (WESS 2014), 2014.

[10] A. P. Johnson, R. S. Chakraborty, D. Mukhopadhyay, A Novel Attack on a FPGA based True Random Number Generator, in: 10th Workshop on Embedded Systems Security (WESS 2015), 2015.

[11] A. P. Johnson, S. Patranabis, R. S. Chakraborty, D. Mukhopadhyay, Remote Dynamic Clock Reconfiguration Based Attacks on Internet of Things Applications, in: 2016 Euromicro Conference on Digital System Design (DSD), 2016, pp. 431–438.

[12] Xilinx Inc, [Online]. Available: http://www.xilinx.com, Virtex-5 Libraries Guide for HDL Designs UG621 (July 2010).

[13] K. Eguro, SIRC: An Extensible Reconfigurable Computing Communication API, in: IEEE Symposium on Field-Programmable Custom Computing Machines (short paper), 2010.

[14] Xilinx Inc, [Online]. Available: http://www.xilinx.com, Xilinx UltraScale MPSoC Architecture, accessed: 2015-04-01 (February 2014).

[15] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, C. Whelan, The Sorcerers Apprentice Guide to Fault Attacks, Proceedings of the IEEE 94 (2) (2006) 370–382.

[16] T. Fukunaga, J. Takahashi, Practical fault attack on a cryptographic lsi with iso/iec 18033-3 block ciphers, in: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2009 Workshop on, IEEE, 2009, pp. 84–92.

[17] M. Tunstal, D. Mukhopadhyay, S. Ali, 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.

[18] N. Selmane, S. Guilley, J.-L. Danger, Practical setup time violation attacks on AES, in: Dependable Computing Conference, 2008. EDCC 2008. Seventh European, IEEE, 2008, pp. 91–96.

[19] Q. Tang, B. Kim, Y. Lao, K. Parhi, C. Kim, True Random Number Generator Circuits Based on Single- and Multi-Phase Beat Frequency Detection, in: Custom Integrated Circuits Conference (CICC), 2014 IEEE Proceedings of the, 2014, pp. 1–4.

[20] Xilinx Inc, [Online]. Available: http://www.xilinx.com, PicoBlaze 8-bit Embedded Microcontroller User Guide UG 129 (v2.1), accessed: 2016-11-23 (June 2011).

[21] S. Gören, O. Ozkurt, A. Yildiza, H. F. Ugurdag, R. S. Chakraborty, D. Mukhopadhyay, Partial Bitstream Protection for Low-cost FPGAs with Physical Unclonable Function, Obfuscation, and Dynamic Partial Self Reconfiguration, Elsevier, 2012.

[22] S. Gören, Y. Turk, O. Ozkurt, a. H. F. U. Abdullah Yildiz, Achieving Modular Dynamic Partial Reconfiguration with a Difference-based Flow, in: Proc. of ACM/SIGDA int. symposium on FPGA, ACM, 2013, pp. 270–270.

[23] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, A statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, Tech. rep., DTIC Document (2001).

[24] A. Nejat, S. M. H. Shekarian, M. S. Zamani, A Study on the Efficiency of Hardware Trojan Detection based on Path-delay Fingerprinting, Microprocessors and Microsystems 38 (3) (2014) 246 – 252.

[25] S. M. H. Shekarian, M. S. Zamani, Improving Hardware Trojan Detection by Retiming , Microprocessors and Microsystems 39 (3) (2015) 145 – 156.

[26] A. P. Johnson, R. S. Chakraborty, D. Mukhopadhyay, An Improved DCM-based Tunable True Random Number Generator for Xilinx FPGA, IEEE Transactions on Circuits and Systems II: Express Briefs PP (99) (2016) 1–1.

[27] A. P. Johnson, R. S. Chakraborty, D. Mukhopadhyay, A PUF-Enabled Secure Architecture for FPGA-Based IoT Applications, IEEE Transactions on Multi-Scale Computing Systems 1 (2) (2015) 110–122.

[28] Y. Hori, T. Katashita, H. Sakane, T. Kenji, A. Satoh, Bitstream Protection in Dynamic Partial Reconfiguration Systems using Authenticated Encryption, IEICE Transactions on Information and Systems 96 (11) (2013) 2333–2343.

[29] A. Zeineddini, J. Wesselkamper, PRC/EPRC: Data Integrity and Security Controller for Partial Reconfiguration, Xilinx Inc, [Online]. Available: http://www.xilinx.com, accessed: 2016-11-23 (June 2012).

[30] X. Guo, R. Karri, Invariance-based Concurrent Error Detection for Advanced Encryption Standard, in: Proceedings of the 49th Annual Design Automation Conference (DAC, ACM, 2012, pp. 573–578.

[31] V. Lomn and T. Roche and A. Thillard, On the Need of Randomness in Fault Attack Countermeasures - Application to AES, in: 2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, 2012, pp. 85–94.

[32] S. Ali, X. Guo, R. Karri, D. Mukhopadhyay, Fault Attacks on AES and Their Countermeasures, Springer International Publishing, 2016, pp. 163–208.

[33] B. Sunar, W. Martin, D. Stinson, A Provably Secure True Random Number Generator with Built-In Tolerance to Active Attacks, Computers, IEEE Transactions on 56 (1) (2007) 109–119.

[34] N. M. Huu, B. Robisson, M. Agoyan, N. Drach, Low-cost Recovery for the Code Integrity Protection in Secure Embedded Processors, in: 2011 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 2011, pp. 99–104.

[35] N. Moro, K. Heydemann, E. Encrenaz, B. Robisson, Formal Verification of a Software Countermeasure against Instruction Skip Attacks, Journal of Cryptographic Engineering 4 (3) (2014) 145–156.

[36] Y. Tamir, M. Tremblay, High-performance Fault-Tolerant VLSI Systems using Micro Rollback, IEEE Transactions on Computers 39 (4) (1990) 548–554.

[37] P. A. Bernstein, Sequoia: a Fault-tolerant Tightly Coupled Multiprocessor for Transaction Processing, Computer 21 (2) (1988) 37–45.