A PUF-Enabled Secure Architecture for FPGA-Based IoT Applications

Anju P. Johnson, *Student Member, IEEE*, Rajat Subhra Chakraborty, *Member, IEEE*, and Debdeep Mukhopadhyay, *Member, IEEE*

Abstract—The Internet of Things (IoT) is a dynamic, ever-evolving "living" entity. Hence, modern Field Programmable Gate Array (FPGA) devices with *Dynamic Partial Reconfiguration* (DPR) capabilities, which allow in-field non-invasive modifications to the circuit implemented on the FPGA, are an ideal fit. Usually, the activation of DPR capabilities requires the procurement of additional licenses from the FPGA vendor. In this work, we describe how IoTs can take advantage of the DPR capabilities of FPGAs, using a modified DPR methodology that does not require any paid "add-on" utility, to implement a lightweight cryptographic security protocol. We analyze possible threats that can emanate from the availability of DPR at IoT nodes, and propose possible solution techniques based on *Physically Unclonable Function* (PUF) circuits to prevent such threats.

Index Terms—Cryptographic protocol, dynamic partial reconfiguration, Internet of Things, field programmable gate arrays, hardware Trojans, physically unclonable functions

1 Introduction

Internet of Things (IoT) is a set of connected, uniquely identifiable, smart objects ("things"), based on and benefited by the Internet technology, which are expected to bring unprecedented improvement in human quality of life in the near future. The IoT technology has been shown to be useful in a wide range of fields, including medical and health-care [1], system automation [2], remote sensing [3], agriculture and food safety [4]. However, before the dream becomes a reality, several important concerns about the implementation, operations and applicability of IoT require satisfactory resolution and several issues needs to be addressed. Even setting aside the technical challenges, factors related to governance, quality of service, security, privacy, interoperability and other social and economic issues need to be resolved [5].

Cryptographic systems have become an integral part of our daily life through the need of security of many common activities such as communication, electronic money systems, disc encryptions etc. Today, most of the industrial sector use *Hardware Security Modules* (HSMs) for delivering dedicated cryptographic services, with dual emphasis on high performance and security. It is well known that the use of programmable hardware in system implementation can lead to performance improvements [6]. FPGAs are frequently used to implement cryptographic hardware, to provide secure authentication, and storage of secret data [7]. FPGAs have the added advantage of being reconfigurable, which increases their flexibility and makes them suitable candidates for IoT applications. While the relatively higher power dissipation of FPGAs in earlier generations used to be a challenge limiting their deployment in power sensitive application domains, ultra-low power FPGAs that are now commercially available [8] allows them to be used for IoTs.

A relatively recent enhancement to FPGA capabilities is *Dynamic Partial Reconfiguation (DPR)* or *Runtime Partial Reconfiguration (RPR)*. It is the ability to modify (mostly through the addition of functionality) the existing circuit on the FPGA, through "partial reconfiguration" (PR) of the FPGA at run time. DPR allows designer to use smaller devices, reduce power consumption and improve system upgradability. DPR-enabled FPGAs are thus ideal choices for IoT applications. However, it has been demonstrated that DPR-enabled FPGA based systems can be subjected to malicious circuit alterations, typically termed as *Hardware Trojan* insertion [9]. Interestingly, such attacks leverage the same DPR capabilities that are otherwise so valuable. Hence, proper defense strategies must be provided to counter such threats, while keeping the inherent physical constraints of IoT under consideration.

In recent years, *Physically Unclonable Function* (PUF) circuits have emerged as promising hardware security primitives to be used in low-overhead security applications [10]. The operating principle of PUF circuits is based on the utilization of nano-scale device–level process variation effects, from which unique, digital "fingerprints" for devices are derived [11]. Since the process–variation effects which are at the heart of PUF circuits, are uncontrollable and unpredictable at the current state-of-the-art of semiconductor manufacturing, the duplicate of a given PUF instance cannot be manufactured, hence the term "unclonable" in their nomenclature. The inherently low hardware overhead of PUFs again make them suitable for IoT applications, especially because IoT nodes often have too little computational resource to execute traditional mathematically intensive cryptographic algorithms. We describe a PUF based security protocol for DPR-enabled FPGAs that is resistant against hardware Trojan attacks.

• *The authors are with the Secured Embedded Architecture Laboratory (SEAL), Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur 721302, West Bengal, India. E-mail: {anjupj, rschakraborty, debdeep}@cse.iitkgp.ernet.in.*

Manuscript received 11 May 2015; revised 7 Sept. 2015; accepted 12 Oct. 2015. Date of publication 26 Oct. 2015; date of current version 11 Dec. 2015.
Recommended for acceptance by S. Ray, J. Park, and S. Bhunia.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TMSCS.2015.2494014

We summarise the contribution of this paper as follows:

1) Envisage a DPR-enabled FPGA-based secure IoT architecture, where the security is hardware-enabled. To establish network connectivity across the FPGAs, we took the assistance of the SIRC framework provided by Microsoft [12]. We mapped more than one closely related hardware cores on to the same FPGA, which are usually lined up in a crypto process cycle. At the same time a reasonable security provisions are also taken to avoid any kind of undesired mixing between the cores.
2) We investigate threats to this architecture by launching successful Hardware Trojan insertion attacks on the two main linchpins of security for the proposed architecture: the cryptographic hardware core and the *True Random Number Generator* (TRNG). For the former case, we were successful in attacking an 128-bit AES cryptographic core, leading to the recovery of the of the secret key by standard mathematical analysis in less than 15 minutes using a standard PC. For the later case, with very high probability, we were able to bias the output bitstream of the TRNG (described in [13]), such that the response of the TRNG becomes predictable, with a high probability of prediction success. These attacks are launched by malicious configuration bitstream transfer to the target FPGA using DPR techniques, which embed the hardware Trojan in the FPGA.
3) We describe a protection technique against the demonstrated attack, to secure the DPR-Enabled FPGAs in IoT applications from malicious modifications, while not compromising on the DPR capabilities of the FPGA. Our security protocol is based on the "XOR PUF" design [14]. Due to the composite nature of the PUF instance under consideration, it's behaviour is hard to model by an adversary who does not have any access to the intermediate PUF responses. However, if access to the intermediate PUF response is available (e.g., to the implementer), the PUF can be mathematically modelled, by following the approach described in [15], allowing its use in the secure communication protocol. Strong PUF instances can also be used as an identification tag for the FPGA devices.
4) We establish the effectiveness of the proposed protocol through experimental results.

The remaining sections of this paper is organized as follows. In Section 2, we describe the proposed DPR-enabled FPGA-based IoT architecture, and its possible applications. Section 3 describes and demonstrates the threats on the above mentioned architecture. In Section 4, we explore several options to prevent the attacks, including the details of a PUF based security protocol. In Section 5, we provide experimental results to establish the effectiveness of the defense strategy. We conclude in Section 6.

2 IoT Model for Dynamic Reconfigurable Hardware Modules

Over the last few years, attention has been given in standardizing the IoT architecture [16] and protocols [17] for connecting systems into a unified platform. Let us consider a set of cryptographic hardware solutions, which can be accessed by any authenticated application from anywhere at any time. By the term *cryptographic hardware solutions* (CHS), we mean commonly used hardware support for cryptography such as hardware security modules, secure crypto-processors, tamper-resistant security modules, crypto-accelerators, embedded crypto-engines, etc. Access to each CHS is provided on a time-shared basis. This model can be considered as a simple connected graph in which each application utilizing the cryptographic services are on one side, and the CHS on the other side.

Let us review the definition of IoT as given by Uckelmann et al: "Today, the Internet of Things is a foundation for connecting things, sensors, actuators, and other smart technologies, thus enabling person-to-object and object-to-object communications" [18]. This clearly means that in the context of IoT, every object has an active role, as IoT aims at increasing the ubiquity of the Internet by integrating every objects to communicate with human beings as well as other devices. Moreover, the dynamic nature of IoT demands online modifications to both the hardware and software components. Although at the current state-of-the-art software updates and upgradation to suit the dynamic needs of the IoT are relatively straightforward, the real bottleneck is in terms of having the same flexibility in the hardware components. This factor motivates us to develop an architecture which can enable dynamic flexibility in hardware as well.

To achieve the above goal, in this section, **we define an IoT architecture which supports dynamic modifications to the hardware components**. This is achieved by deploying DPR-enabled FPGAs. Each FPGA is programmed with more than one CHS. We define *dynamic partitions* to incorporate additional hardware modifications, *on-the-fly* as required. **This DPR-enabled FPGA architecture is the basis of hardware sharing, hardware mixing and online hardware updates, and plays a vital role in IoT application as different enterprises can share the available resource**.

Next we describe the proposed architecture bottom-up, starting with the DPR-enabled FPGAs, which are the fundamental hardware building blocks of this architecture.

2.1 DPR-Enabled FPGA as IoT Building Blocks

DPR in the proposed architecture is performed by transferring the required partial configuration bitstream file over a live network connection to the FPGA board. The prime components of the proposed DPR setup are shown in Fig. 1. This setup makes it feasible to apply DPR to perform *on-the-fly* reconfiguration, enabling "add-on" functionality post initial reconfiguration, facilities for *hardware mixing* or *logic sharing* across the different "add-on" hardware modules, enabling high speed low overhead data transfers, etc. Enabling DPR on a FPGA requires partitioning the FPGA logic into *static* and *dynamic* partitions—the dynamically configured modules can reside only in the dynamic partition. The main work component is the *DPR controller*, which provides master supervision of all modules related to enabling DPR. The DPR controller generates the necessary control signals to the *Internal Configuration Access Port* (ICAP), which is a standard Xilinx hardware primitive for PR [19], which in turn performs the DPR in the specified dynamic partition. Equally crucial is the network connectivity of the FPGA,

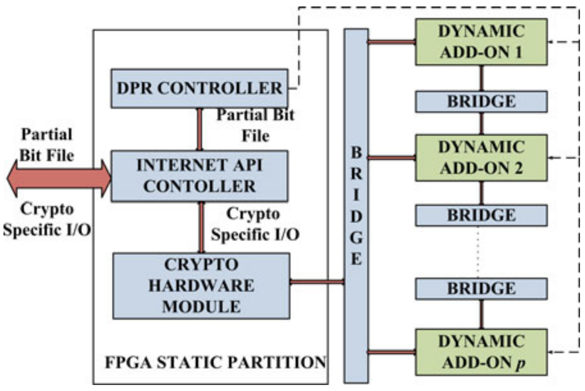

Fig. 1. The basic building block for DPR-enabled FPGA.

through which the secure communication targeted for the cryptographic applications are serviced, and the DPR files are transferred. This is established using an Internet 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 [20].

To manage the data (including the partial configuration bitstream) transfer functionality for the FPGAs (Internet API Controller), we propose to use a modified version of the widely used open-source *Simple Interface for Reconfigurable Computing* (SIRC) platform [12]. SIRC consists of both software and synthesizable hardware components, and makes it possible to seamlessly transfer arbitrary data to a FPGA through high-level API calls. This lets the application programmer concentrate on the *functionality* requiring the data transfer, without being burdened by the task of managing the complex network interactions. Being open source, the SIRC framework is also sufficiently customizable by design, and thus extremely suitable for an IoT environment. Along with the DPR controller and the Internet API controller, the cryptographic hardware module also resides in the static part of the design. The dynamic region of the FPGA is further divided into p dynamically reconfigurable partitions; presence of multiple dynamic partitions increases the flexibility and upgradability of the system. The logic interfaces between the static partitions and the dynamic "add-ons", and between the dynamic "add-ons" themselves, are also flexible and reconfigurable. In our implementation, simple circuit components available on FPGAs such as flip-flops, "bus macros" (predefined buses) and Lookup Tables (LUTs) serve as *communication bridges* between the modules. The design and strategy proposed by Wang et al. [21], includes such dynamically reconfigurable interconnect, and is proposed to be used for big data analysis. Similar strategies can be adopted for the proposed IoT technology.

2.2 FPGAs as IoT Building Blocks

For convenience of discussions, we illustrate the architecture with FPGAs providing solely cryptographic functionalities; however, in reality, the FPGAs can provide any necessary functionality, with a cryptographic module may or may not being present. A structural view of the DPR-enabled FPGA as presented in Section 2.1 is shown in Fig. 2. As illustrated, we have k distinct cryptographic cores arranged on an FPGA. Each crypto-core is configurable by a set of "add-ons". The ith crypto-core has provision for p_i dynamic "add-ons" (in p_i dynamic partitions), where the optimal values for parameters p_i and k are to be determined based on constraints set by technology, economy, security and other measures. To make the architecture symmetric, we dedicate equal number of "add-on" partitions (say, p) for all the crypto-cores on the FPGA. The detailed view of "add-on" set is shown to the right of Fig. 2. The only disadvantage of this scheme is that since FPGA resources are being allocated a-priori in the dynamic partitions, if one of more of such dynamic partitions are not used, it would incur a wastage of hardware resources. However, this shortcoming can be solved by providing expansion slots accommodating multiple FPGAs at the IoT nodes. Then, on demand, these auxiliary FPGAs can be configured to accommodate extra dynamic partitions. We term a networked *DPR-enabled FPGA node* as (DNODE); a DNODE uses the reconfigurable portions for functionality enhancement, as well as for interactions with other cores. The entire IoT is proposed to be composed of DNODES as shown here.

There are several fine points about this architecture which needs elaboration:

1) *Multiple CHSs sharing a single DNODE* is advantageous, as this enhances the efficiency and security of the DNODE. As an illustrative example, consider a DNODE with two cryptographic primitives: a TRNG

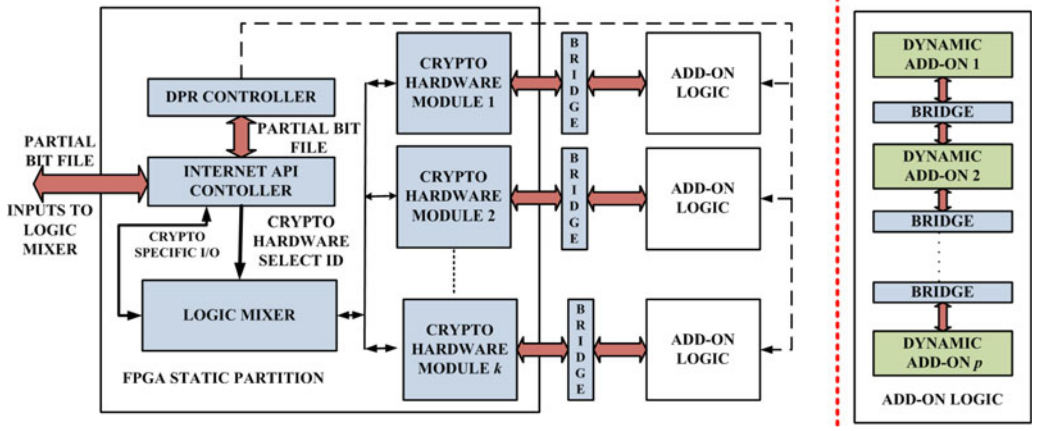

Fig. 2. The Basic Building block for DPR enabled FPGA considering *k* Crypto services incorporated in an FPGA.

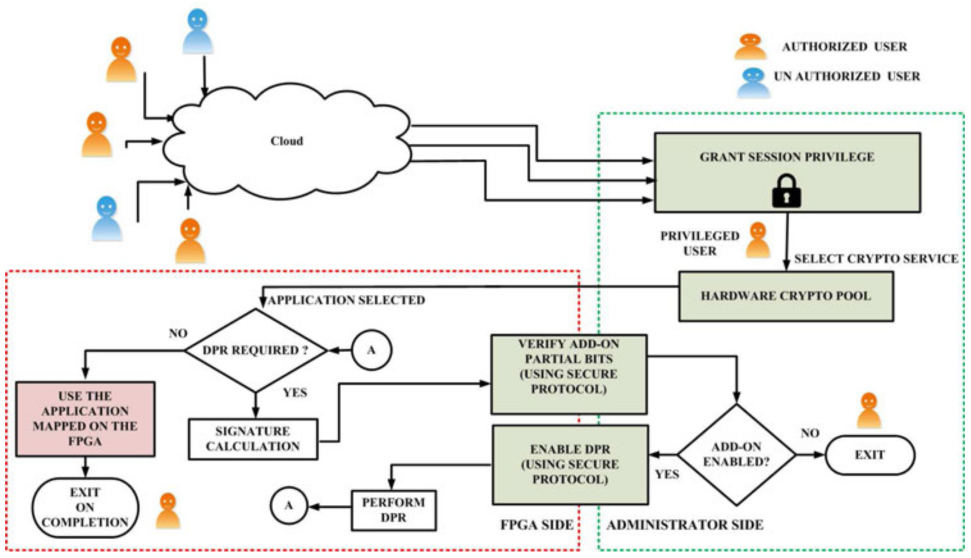

Fig. 3. Process of selecting DPR partition and enabling DPR.

and a symmetric-key encryption hardware module. Consider a user application willing to perform data encryption using the encryption module, whose security key is generated by the TRNG. If the two modules resides in an FPGA, the key generated from the TRNG need not be transmitted from other nodes over the network. The key cannot be transmitted in plain-text over the network, as an eavesdropper might then trivially compromise the security of the system after noting down the key. However, the usage of public key protocols for key establishment would incur an overhead not sustainable in resource-constrained IoTs. Although it is possible to have crypto-cores which have the key generator combined with the encryption engines, implementing them as separate modules gives more chance of exploring the benefits of DPR, by enabling required modifications targeting specific CHSs, as and when required in future. Also, the existence of multiple functional modules on a single FPGA allows the time-sharing of the single FPGA intercommunication service among the multiple modules, thereby decreasing the communication overhead.

2) *The distribution of the CHSs on the FPGAs* is an optimization problem, targeting two objectives: (a) maximizing the interactions between the partitions, and (b) minimizing the number of cross-node interconnections. To achieve the first goal, closely related applications should be mapped on the same FPGA, while to meet the second goal, a graph partitioning algorithm [22] might be used.
3) Authentication of the DNODE and its components is performed using PUFs. Recall that PUF circuit instances have an instance-specific behaviour, and hence can act as identifier generator circuits for the FPGAs (more details in Section 4.2). We propose to use a scheme where each FPGA has an internal "master PUF", and each functional core (CHS) has an associated "secondary PUF". An internal free-running low overhead *Linear Feedback Shift Register* (LFSR) generates the challenges to be applied for the PUF instances. The FPGA is identified by the responses of the master PUF, while each CHS is identified by the responses of the secondary PUFs.

2.3 Performing DPR on the FPGA

We now describe the whole process of usage and partial reconfiguration of the IoT DNODEs based on requests from privileged users. The whole process is shown in Fig. 3. The *Administrator* provides session privilege to an authorized user on time shared bases. A *privileged user* has the provision to select the required CHS from the pool of available resources. Now the application user is entitled to get service of the CHS (with/without DPR-based "add-on"). The PUF based identification scheme is employed for selecting the targeted DNODE and the target CHS. The privileged user has two options: either to use the CHS without modifications, or with permitted modifications. By the term "modification" we mean either addition of new hardware, or removal of existing hardware in the dynamic "add-on". We discuss two ways for enabling DPR: "Restrictive Mode" and "Non-restrictive Mode", more details are presented in Section 4. In non restrictive mode the PR bitstream signature is compared with a golden value stored on tamper-resistant non-volatile memory at the IoT node, and if a match is obtained, the DPR is granted. For IoT applications, non-restrictive mode is more advantageous as it provides more flexibility and security. As shown in Fig. 3 if a user requests the need for "add-on" services, she has to pass the PR bitstream verification using a secure PUF based protocol. The details of the authenticated DPR protocol is described in Section 4. There are provisions for t number of DPR to be performed, whose value needs to be fixed by the service provider. The user is not permitted to use the CHS if there is any failure in "add-on" security checking. Once the user is satisfied with the available "add-ons", she uses the CHS and exits. Every attempted DPR is automatically logged, with both the user details as well the currently configured CHS behaviour for future references.

This infrastructure can be expanded considering hardware sharing across different sets of enterprises, so as to

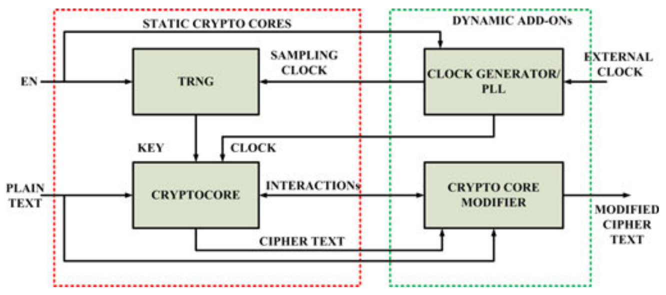

Fig. 4. Hardware “add-ons” for cryptographic hardware core and TRNG.

increase the utilization of available hardware resources, in a cloud like environment, where it is required to guarantee the security of the shared hardware. In such environment the need for secure protocol for DPR enabling is a prime consideration, since security of all enterprises involved may get adversely affected in case of any malicious circuitry being added *on-the-fly*. In other words, the processing of corporate data in an infected circuit seams to be greater threat.

Several previous works [23] has described alternative architectures for DPR that might be exploited by an adversary for HTH insertions; however they have several shortcomings compared to the architecture proposed by us. The shortcomings of the DPR architecture described in [23] compared to our proposed DPR architecture has been described below:

1) It makes an assumption that the partial configuration file is already available, without any description of the methodology to generate the necessary configuration file.
2) There is no mention on what type of DPR is targeted, i.e., whether *Modular* or *Difference based* partial reconfiguration. This is an important piece of information, as the type of partial reconfiguration to be performed has a direct impact on the structure and size of the partial configuration file.
3) The architecture is claimed to be “ultra-light”, but the components used in the design (microprocessor based design), shows that there is room for further reduction in hardware resource utilization
4) The work basically gives more emphasis on the speed of transferring the reconfiguration bitstream over the network, rather than providing a comprehensive DPR base design methodology.

2.4 An Example Application

Putting it all together, an example design as illustrated in Fig. 4, was implemented by us. A Xilinx Virtex-5 FPGA is connected to a network over a standard 100-Mbps Ethernet connection, providing real-time computational capabilities. Let the CHS employed in the FPGA be a cryptographic core and a *True Random Number Generator*. DPR facility is enabled to provide additional facilities to enhance the model in the crypto-core side (e.g., support for a specific mode of encryption like CBC, OFB, CFB, *tweakable encryptions*, etc.), and say, to modify the clock generation for synchronizing the CHS with the application using it. Considering the IoT model, one of the DNODE in the network consist of a cryptocore and a key generator. The partial reconfiguration bitstreams for the “add-ons” are generated by following the difference based DPR flow described [24] using the Xilinx FPGA Editor software which does not need a special partial reconfiguration license. This makes the approach well fitted for low-cost IoT applications. Difference based PR bitstream generation considers the difference between the existing “add-on” configuration with the new one. Hence, this leads to smaller PR bitstreams to be transferred over the network which is also another advantage for low bandwidth IoT applications (Section 5.2). The *Phase Locked Loop* (PLL) primitive in Virtex-5 is used in the “add-on” circuits for generating various clocks in the design. The cryptographic core (specifically AES) efficiency is strengthened by a cryptocore modifier “add-on”, specifically an efficient *Mix Column* implementation [25]. The “add-ons” are allowed to be reconfigured on demand *on–the–fly* by transferring the required partial configuration bitstream over an Internet connection to the FPGA board, from an authorized application running on remote system which can communicate with the FPGA over the network.

3 Possible Threats to DPR-Enabled FPGA-Based IoT Infrastructure

Since the practical deployment of IoTs is already severely challenged from security threats, it is natural to think that adding a feature such as DPR, which provides a relatively simple mean to modify the hardware running on a FPGA, will only increase the vulnerability of FPGA-based IoT. But the multitudinous benefits offered by the DPR technology far outweighs this disadvantage. This section is intended to demonstrate possible threats associated with insecure usage of DPR on the reconfigurable platform, and proves the need for better security measures when deploying DPR in building IoTs. Cryptographic security relies mainly upon two components: the cryptographic algorithm and the cryptographic key. The following attacks in this section demonstrate the exploitation of DPR capabilities to adversely effect these two components, specifically through the insertion of hardware Trojans through DPR. In the next section, we describe defense strategies to mitigate these threats.

3.1 Attack on *Advanced Encryption Standard* (AES) Hardware

Advanced Encryption Standard is the global standard for symmetric key encryption [26]. The proposed hardware Trojan helps to recover the secret key of an AES-128 bit cryptohardware, and is shown in Fig. 5 [9]. Once inserted in the circuit, the hardware Trojan gets triggered only when it receives some specific predetermined bit patterns (not necessarily identical) in the plain-text for some predefined number of times (not necessarily in consecutive plain-texts). Both the bit pattern, and the number of matchings required to activate the Trojan, are parts of the Trojan design specification, and decided by the adversary. Here we introduce three metrics to quantify the characteristics of the inserted Trojan: “Triggering Bit Pattern Length” (TBPL), “Triggering Bit Pattern Count” (TBPC) and “Trojan Activation Time” (TAT). The length of the bit pattern to be recognised as trigger bit pattern is called *TBPL*, and the number of times it has to appear to trigger the Trojan is called *TBPC*. We

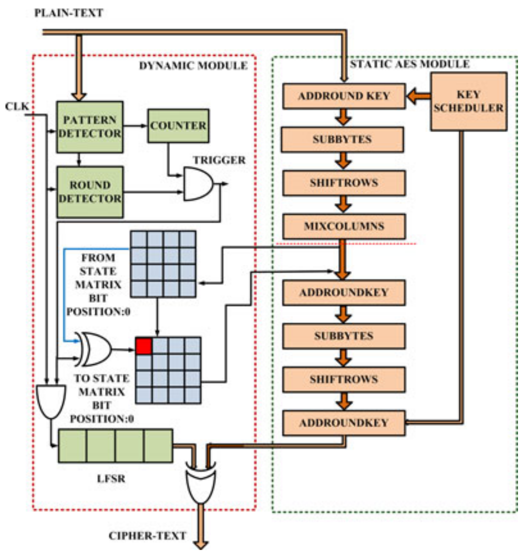

Fig. 5. Detailed design of the implemented hardware for attack on AES.

define the expected number of encryption operations done by an AES circuit infected by the Trojan before the Trojan triggers as *TAT*. Once we have a quantitative estimate of *TAT*, corresponding *TBPL* and *TBPC* can be determined by fixing one of these metrics. Note that the Trojan triggering event is non-deterministic, which simplifies the Trojan activation mechanism by removing the need for remote triggering, at the cost of the possibility of a scenario where the Trojan is never triggered. Interestingly, both extremely high and extremely low activation probabilities can be shown to be undesirable from the adversary's perspective in this scenario; the first one may increase the probability of the Trojan being detected during pre-deployment logic testing, while the second one may keep the Trojan dormant indefinitely. Hence, the adversary needs to set the TBPL and the TBPC parameters at appropriate levels, through correct design of the Trojan, to make sure that the Trojan activation probability reaches a moderate non-zero moderate value. In the implementation presented in [9], the Trojan is activated only when it receives a specific 20-bit pattern, nine times in succession; then, the probability of the Trojan accidentally triggering during normal operations in $\sim 10^{-7}$.

On activation, the Trojan synchronizes with the start of a new plain-text 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 0th 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 cipher-text. Note that, once triggered the Trojan never gets reactivated until the FPGA is powered-off and powered-up again (at which point it starts detecting the input plain-text again for possible triggering), 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} [27], which can be performed in less than 15 minutes using a standard desktop PC. The brute-force search size can be reduced further to 2^{8} [28].

Most of the existing Trojan detection techniques attempt Trojan detection before the IC is deployed for operation. Since in this approach, the Trojan is inserted *after* the FPGA starts to work, the Trojan can evade pre-deployment detection techniques. Combined with the low activation probability of Trojan, we can conclude that these HTHs can elude functional built-in-self-test (BIST) or authentication [29]. Power or delay analysis based Trojan detection methods also fail to detect it, because of the low impact on these side-channel parameters (as would be confirmed by our experimental results).

Since every attempted DPR is logged by the on the FPGA, It is not possible for a malicious "privileged user" to make arbitrary modifications on the existing FPGA hardware without revealing her identity. However, the "privileged user" 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, piggy-back 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 certainty which DPR operation was exactly responsible to cause the failure. In addition, similar to the mechanism proposed in [30], the faulty cipher-text 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 cipher-text, but only an adversary having knowledge of how to calculate the original faulty cipher-text from the altered version, can launch the attack. Also, 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. More details about this attack can be found in [9].

3.2 Attack on TRNG

True Random Number Generator circuits produce randomness by exploiting the intrinsic randomness of a physical process. There are a wide variety of TRNGs used in secure system designs, which utilizes different sources of randomness and follows different randomness extraction methodologies. Many of them have encountered criticisms and are prone to attacks [31], [32], [33], [34]. This section is intended to demonstrate a practical attack on the ring oscillator (ROSC) based TRNG using "Beat Frequency Detection" [13]. The hardware Trojan designed by us generates a patterned sampling clock and is inserted in the dynamic part of the circuit utilizing the DPR facility of the FPGA. Since the sampling clock generator is an inevitable part of the TRNG design, the design of the proposed Trojan, superimposed on this sample clock, reduces the hardware overhead associated with the Trojan design. Also this facilitates the use of smaller sized partial bit files to be transferred over the network for inserting the Trojan infected "add-on" design.

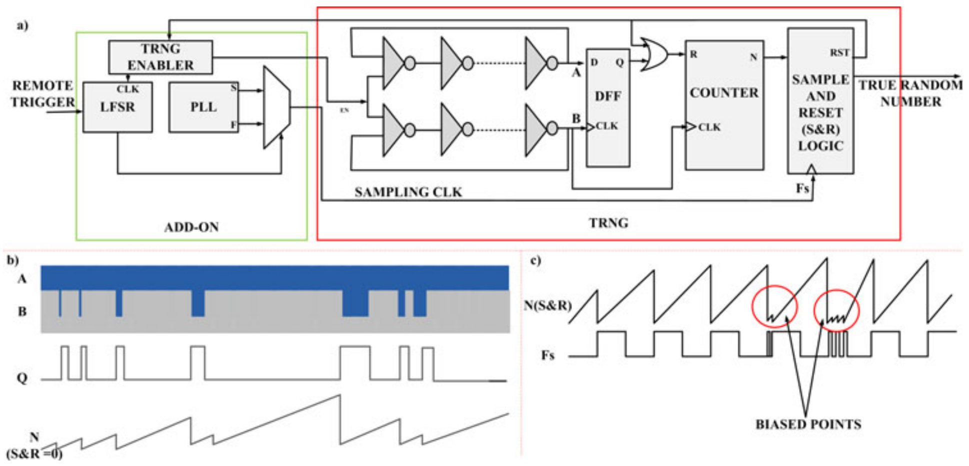

Fig. 6. a) Detailed design of the proposed hardware Trojan to bias the response of *Beat Frequency TRNG* circuit [13] implemented on FPGA. b) Free running response of the circuit in the absence of Trojan (Sampling and Reset is not performed). c) Response of TRNG in the presence of activated Trojan (with Sampling and Reset functioning).

The TRNG under consideration consists of two similar ring oscillators, where the the response of one ROSC (say, $ROSC_A$) is sampled using the other (say, $ROSC_B$), and a counter is designed to capture their frequency difference. The output of the counter is sampled using a sampling clock of frequency F_s on reaching the *beat frequency interval*, where the beat frequency interval is being determined by the frequency difference between the two ROSCs. After sampling, the circuit is reset and is enabled again for determining the next random number. We exploit this sampling and reset functionality for injecting the hardware Trojan.

The main component of the Trojan design is the *Phase Locked Loop* circuit, which is necessary for generating the required clock for sampling, as well as for other circuity in the FPGA. The PLL primitive is configured to generate the required clock as well as an additional fast Trojan clock. Since the existence of PLL circuitry is an inevitable part of the FPGA design, and the hardware Trojan fast clock is superimposed on the PLL circuitry, this main component of the hardware Trojan do not incur any additional hardware footprint in the device. In addition to the PLL circuit, a “Linear Feedback Shift Register” circuit is added as a part of the hardware Trojan. A multiplexer selects one out of the two possible clock signals (the slow sampling clock or the fast Trojan clock), based on the LSB of the LFSR output, when triggered remotely. Based on the LFSR output, the sampling point varies in time. Since the Trojan clock is much faster compared to the counter clock, when the Trojan gets is activated, the counter outputs gets sampled before the counter gets incremented. This action biases the TRNG response to “zero”. After this, the circuit is reset automatically for the next sampling phase. This is shown in Fig. 6. Since the LFSR produces a sequence where a 0 or 1 is equiprobable, it is evident that at half the times, there is a possibility for the attacker to bias the response to 0, and the remaining 50 percent values are determined by the randomness of the device.

The design of the Trojan clock is crucial for biasing the response of the counter to zero. Here, the important design metric is the determination of *Trojan Sampling Clock* (T_S). The effect of T_S is reflected in the Probability Distribution Function (pdf) of the TRNG response, whereby the original true Response of the TRNG (which follows a Gaussian distribution) splits into three functions. In this case the resulting function may or may not be random and the degree of randomness is decided by the composite functions given by $f(x) = as(x) + bt_1(x) + ct_2(x)$, where $s(x)$ corresponds to the Gaussian pdf of the original sampling clock, $t_1(x)$ represents the symmetric/asymmetric partial Gaussian distribution due to the fast Trojan clock, and $t_2(x)$ represents the constant function x $= 0$ (also due to fast Trojan clock), and a, b, c are (non-negative) constant mixing weights, with $a + b + c = 1$ and $a = (b + c) = \frac{1}{2}$ (presence of LFSR in the design splits the uninfected distribution and the distribution due to HTH into components of equal weights). The success or failure of statistical tests of randomness depends on the actual mixture model. The randomness of the TRNG output bitstream is dependent on this mixture model. Hence, the generated output bitstreams passes or fails the NIST test depending on this resultant distribution. An intelligent adversary would design T_S such that $b = 0$, and hence he can predict the “zero” biased points accurately by knowing the initial LFSR configuration. Our observation also points out that *glitches* (i.e., extremely fast triggers) in the sampling clock can also lead to “zero biased” points in the TRNG responses produced. Thus the Trojan circuit can eventually be simplified to a glitch generator circuit, with further reduction in hardware overhead and effect on circuit parameters, in addition to the reduction in size of PR bitstreams to be transferred over the network. As would be shown by our implementation results, the hardware overhead and impact on circuit parameters of the propose Trojan are negligible.

3.3 Other Threats

The number of possible HTH instances in the HTH design space is very large, and unrestricted DPR provides insertion of any kind of these HTHs to the unsecured dynamic partition. Firstly, let us consider a combined attack of the one discussed in Sections 3.1 and 3.2. The architecture of the circuitry implemented on the FPGA is shown in Fig. 4. A combined DPR attack can be launched in this system by performing DPR in both the dynamic “add-on” partition. Here the attacker can leverage the combined support of both the Trojans, viz. the Trojan targeted towards the TRNG, and the one in AES Crypto core, so as to further reduce the complexity of key recovery attack. These kinds of combined attacks have greater threat, as the single HTH instance might not be sufficiently harmful to the system, whereas the combined HTH severely compromises system security by leaking the secret information.

Malicious dynamically reconfigurable instruction execution unit in a evolvable instruction set processor is yet another case of potential threat via DPR. In such processor addition of a new instruction can be executed via a malicious path leading to leakage of critical informations under rare triggering conditions. Dynamically changing an existing instruction in such processor may also lead to similar threat to the system.

Next we describe defense strategies against the proposed attacks.

4 Defense Strategies

The main idea we propose for defending against the above mentioned attacks is enabling *selective authentication* of the DPR capabilities. We propose two different modes of DPR: the *restricted mode* and the *non-restricted mode*.

4.1 Restricted Mode DPR

Since analysis of a FPGA configuration bitstream to identify the representative circuits is difficult [35] 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. 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, it is reasonable to assume that only a limited number of possible modifications might be performed for it through DPR. In such a scenario, the partial bitstreams might be generated in advance for these “pre-authorized modifications”, along with an associated signature to verify its authenticity. In our implementation we used the cryptographic hash function SHA-3 to calculate the verification signature. A “signature verifier” hardware logic module is also incorporated in the static design, so as to compare the signature of the incoming partial bitstreams against a stored “golden signature”. **The partial bitstream would be allowed to modify the dynamic part of the circuit, only if the signature calculated for it matches with one of the golden signatures. The security properties of the cryptographic hash function would ensure that the creation of a bitstream different to any of the valid bitstreams, but having a valid signature is computationally infeasible**. This simple but effective technique provides satisfactory protection at small hardware overhead. The only potential disadvantages are: (a) the lack of relative flexibility compared to an unrestricted DPR mode allowing arbitrary modifications, and (b) potential threat of attacks on the stored signatures on the FPGA.

4.2 PUF Based Security Protocol for DPR Enabled IoT Applications on FPGAs

The limitations of the restricted mode of DPR can be avoided if a secure protocol can be developed to authorize the DPR without resorting to explicit storage of golden signatures at the IoT nodes. The alternative technique should also be flexible to allow arbitrary modifications through DPR. Our proposed PUF based protocol aims at resolving this issue.

A PUF instance in general can be modeled as a n-bit input, m-bit output Boolean function $\gamma : \{0,1\}^n \rightarrow \{0,1\}^m$. In our case we consider PUF instances with $n = 64$ and $m = 1$. We employ the *Exclusive-OR PUF* (XORPUF) [14] in the static part of the FPGA, for secure DPR enabling. The XORPUF is built by XOR-ing the outputs of several *Arbiter PUF* (APUF) circuits. The APUF is a classic low-overhead PUF circuit that is built by cascading several path-switching stages (each stage is typically composed of a pair of 2:1 multiplexers). The standalone APUF is vulnerable to Machine Learning based *Modeling Attacks*, e.g., using the *Logistic Regression* (LR) technique [15], which enables an adversary to predict the response of a given PUF instance for an arbitrary challenge with high probability of success. However, the XORPUF is resistant to practical modeling attacks if the number of APUFs being XORed is larger than six [15]. Each PUF instance must be 100 percent reliable (i.e., have perfectly replicable input-output behavior over time), which can be enforced by having error correction circuitry associated with them.

Security against modeling attacks and hardware overhead are the key parameters in deciding the number of parallel Arbiter PUFs to be employed. *Replay attacks* are prevented by using different sets of CRPs for every attempted authentication. The administrator, for every FPGA based IoT node, maintains a database of challenges that have been previously used for PUF based DPR enabling/disabling, and disregards every old challenge used. However, for challenge set size of (say) q in each authentication attempt, with each challenge being n-bit, the probability of repeated challenges occurring termed as the “Failure Probability” $P_{fail} \approx \exp(\frac{-q(q-1)}{2^{n+1}})$, which for typical sets of values of $q = 1{,}000$ and $n = 64$, is almost zero (quantitative estimate of the failure probability of PUF based protocol due to challenges being reused). Hence, in our protocol descriptions, we do not mention this challenge management issue.

We now describe two protocols for secure identification of the requesting device and authentication of partial bitstreams corresponding to a new “add-on” (Protocol-1), and for enabling the DPR if Protocol-1 passes (Protocol-2). Note that the strength of the proposed protocols depends critically on the computational difficulty of the PUF modeling problem, and we provide a formal proof of security for the protocols later in Section 4.3. Both the protocols exploit the inherent ability of PUFs to support security applications, without the need for explicit storage of secret keys.

Protocol-1. PUF Based Device Authentication and Bitstream Validation

Objective

1: **Entity Authentication**: The Administrator **A** (Party-1) verifies the identity of device requesting DPR (Party-2, the FPGA **F**)

2: **PR bitstream Validation**: **A** verifies the genuineness of the PR bitstreams received from **F**.

Prerequisites

1: An n-bit input, 1-bit output XOR PUF P is reconfigured in the static partition of the FPGA **F**

2: A model M of P resides with **A**

3: **F** and **A** have agreed on a fixed encoding scheme $E(\cdot)$ and a decoding scheme $D(\cdot)$, such that for any binary string x, $E(\cdot)$ and $D(\cdot)$ are injective, $X = E(x)$ and $D(X) = x$.

Output

A value in variable $Flag$ to show success ($Flag = 1$) or failure ($Flag = 0$)

Steps

1: **F** chooses: q independent random bitstrings XOR-ed with the state of a *Linear Feedback Shift Register*, to form challenges from the received PR bitstreams

2: **F** characterizes: $R_{pi} = P(C_i)$, $\forall\ i \in 1, 2, \dots q$. The ith challenge $C_i = (C_{i_k}, C_{i_{(k+1)}} \dots, C_{i_{(k+n)}})$ is formed by taking n consecutive bits with starting address k from the received PR binary file with cryptographic hash value FID

3: **F** sends to **A**: $S = E(FID\ \|(\|_{i=1}^{q}(i_k\ \|R_{pi})))$

4: **A** computes: $D(S) = (FID\ \|(\|_{i=1}^{q}(i_k\ \|R_{pi})))$

5: **A** computes: $R_{mi} = M(C_i)\ \forall\ C_i$

6: **A** computes: $N = (1 - \sum_{i=1}^{q} (R_{p_i} \oplus R_{m_i})/q)$

7: If $N \geq 0.99$, **A** declares device **F** authenticated, else sets $Flag \leftarrow 0$ and exits

8: If step-7 is successful, **A** compares FID with the list of available signatures for device **F**

9: If step-8 comparison succeeds for any available signature, **A** sets $Flag \leftarrow 1$, else sets $Flag \leftarrow 0$ and exits

Protocol-1: Device Authentication and Bitstream Validation

This is a two-player protocol between FPGA based IoT node (denoted by **F**) and the Administrator (denoted by **A**). The objective of this protocol is to ensure secure device authentication and verification of PR "add-on" bitstreams. *Protocol-1* consists of the following steps:

1) **F** generates q different random unsigned integers. Collection of bits of size n with each of the selected random number as starting address, XOR-ed with the state of a *Linear Feedback Shift Register*, are considered as challenges for the PUF P configured in the static partition of **F**.
2) The above selected challenges are applied to the PUF P. The respective q responses are collected.
3) An encoded[1] string consisting of the concatenation of following three parts is sent from **F** to **A**: (a) hash value of the received PR bitstream (signature of the PR bitstream); (b) q random numbers corresponding to the challenge location in the PR bitstream file, and, (c) corresponding q responses of P.
4) On receiving S, **A** decodes the message and retrieves the string S.
5) The retrieved challenge locations are mapped to the PR file with signature FID. **A** applies them to the model M and collects the corresponding responses.
6) **A** determines the matching between the received responses and the generated responses.
7) If 99 percent or more match[2] is found between the generated and received responses, the device **F** is declared to be authenticated by **A**; otherwise A declares an error by reseting a flag, and exits to call Protocol-2.
8) **A** then compares the obtained signature FID in its database, with all the available signatures for **F**.
9) If FID matches a signature, A sets a flag and calls Protocol-2. takes necessary decision to enable/disable the attempted DPR.

Protocol-2: PUF based Scheme for Enabling/Disabling DPR

Protocol-2 is very similar to *Protocol-1*, except that, the roles of **A** and **F** gets reversed. **A** communicates back the decision by sending the model generated (actual) or complement (false) responses to F depending upon the authentication of device and applicability of the reconfiguration file. If comparison with the PUF characterization data fails at **F**, the DPR attempt is invalidated. **Note that an explicit ENABLE/DISABLE command from A to F is avoided to increase the security of the protocol**. Every time a DPR is attempted, the two protocols are invoked to authorize the attempt.

Protocol-2. PUF Based Scheme for Enabling/Disabling DPR

Objective

1: **Enable/Disable DPR**: The Administrator **A** (Party-1) communicates back to the FPGA (Party-2) **F** certain data which disables or enables DPR

Prerequisites

Same as *Protocol-1*, value of $Flag$ obtained from *Protocol-1*

output

A value in variable EN to show successful DPR ($EN = 1$) or failure ($EN = 0$)

Steps

1: **A** chooses: l independent random challenges from the PR file with hash value FID

2: **A** characterizes: $R_{mi} = M(C_i)$, $\forall\ i \in 1, 2, \dots l$. The ith challenge $C_i = (C_{i_k}, C_{i_{(k+1)}} \dots, C_{i_{(k+n)}})$ is formed by taking n consecutive bits with starting address k from the received PR binary file with cryptographic hash value FID

3: If $Flag$==1, **A** sends to **F**: $S = E(FID\ \|(\|_{i=1}^{l}(i_k\ \|R_{mi})))$, else, **A** sends $S = E(FID\ \|(\|_{i=1}^{l}(i_k\ \|(1 - R_{mi}))))$

4: **F** computes: $D(S) = (FID\ \|(\|_{i=1}^{l}(i_k\ \|R_{mi})))$

5: **F** computes: $R_{pi} = P(C_i)\ \forall\ C_i$

6: **F** computes: $N = (1 - \sum_{i=1}^{l} (R_{p_i} \oplus R_{m_i})/l)$

7: If $N \geq 0.99$, $EN \leftarrow 1$, else $EN \leftarrow 0$ and exits.

4.3 Formal Proof of Security

We now provide a formal proof of security for the two proposed protocols, following a methodology that is standard

1. Since IoT uses low bandwidth communication, the use of encoding scheme mainly aids in correcting errors in the received string S.

2. The model can be built extremely accurate [15], still we provide 1 percent tolerance, as the modeling error is non-zero.

TABLE 1
Partial Reconfiguration File Size

Test Case	Complete Bitstream Size (kB)	Partial Reconfiguration File (P1-to-Blank) (kB)	Partial Reconfiguration (Blank-to-P1) File (kB)
Test case-1	3,890.087	23.460	23.460
Test case-2	3,890.087	15.367	15.367

in cryptography to establish protocol security [36]. First, we define a *negligible function* as follows: a function $f(\cdot)$ defined over natural numbers is termed *negligible* if for every polynomial $p(\cdot)$, there exists a positive value M such that for all integers $n > M$, $f(n) < \frac{1}{p(n)}$. A function that is not negligible is called *non-negligible*.

Let $MPUF$ be the mathematical model of the XORPUF possessed by the adversary, corresponding to the actual XORPUF hardware at the IoT node, which is characterized by a mathematical mapping PUF. Let $\mathbf{c}$ be an arbitrary n-bit challenge. Then, the fact that an XORPUF with six or more APUFs cannot be modelled accurately by efficient computational means, is stated as the following:

$$Pr[(MPUF(\mathbf{c}) \oplus PUF(\mathbf{c})) = 0] \leq \frac{1}{2} + negl(n), \tag{1}$$

where $negl(n)$ is a negligible function.

We now demonstrate that the problem of breaking our proposed protocols is at least as difficult as the problem of accurately modeling an arbitrary XORPUF instance. In particular, we show that the problem of breaking the proposed protocols by an adversary leads to a mathematical relationship that violates inequality-(1). Consider q arbitrary challenges $\mathbf{c_1}, \mathbf{c_2}, \ldots \mathbf{c_q}$ which are used by the administrator and the FPGA during the authentication phase. Define the q-bit vector $\mathbf{v} = (v_1, v_2, \ldots v_q)$, where $v_i = \overline{MPUF(\mathbf{c_i}) \oplus PUF(\mathbf{c_i})}$, which denotes how many of the applied challenges produced the same response from the XORPUF and its model in possession of the adversary. Suppose, an adversary can successfully break Protocol-1 and Protocol-2. This would imply (again invoking the notion of security expressed in Inequality-(1)):

$$Pr[HW(\mathbf{v}) \geq 0.99q] = \frac{1}{2} + \epsilon(q), \tag{2}$$

where $HW(\mathbf{v})$ denotes the *Hamming Weight* of a binary vector $\mathbf{v}$, and $\epsilon(\cdot)$ is a non-negligible function. Let $\mathcal{S} \subseteq \{1, 2, \ldots q\}$ denote the set of indices for the challenges for which $v_i = 1$, i.e., the predicted value from the XORPUF and the actual XORPUF hardware match. Then, since the adversary successfully broke the protocols, $|\mathcal{S}| \geq 0.99q$. Assuming each challenge is applied independently, Eqn. (2) implies, for a given set of indices $\mathcal{S}$:

$$\prod_{i \in \mathcal{S}}^{|\mathcal{S}|} Pr[v_i = 1] = \prod_{i \in \mathcal{S}}^{|\mathcal{S}|} Pr[(MPUF(\mathbf{c}) \oplus PUF(\mathbf{c})) = 0] = \frac{1}{2} + \epsilon(q). \tag{3}$$

Let $p = \max_i\{Pr[v_i = 1]\}$. Then, the above equation leads to:

$$p^{|\mathcal{S}|} \geq \frac{1}{2} + \epsilon(q). \tag{4}$$

TABLE 2
Power Overhead for AES Encryption/Decryption Circuit

Golden Reference (Blank Dynamic partition) (mW)	Trojan Infected Circuit (mW)	Increase in Power w.r.t.Golden (%)
2,923.97	2,924.95	0.0335

Since $0 < p < 1$, and $|\mathcal{S}| \geq 0.99q > 1$, hence $p^{|\mathcal{S}|} < p$. Thus, the above inequality leads to:

$$p > \frac{1}{2} + \epsilon(q). \tag{5}$$

Note that the above inequality implies than for *any arbitrary*-choice of a set of q challenges, there is always one challenge for which the response obtained from the adversary's mathematical model and that obtained from the XORPUF hardware match, with probability greater than $\frac{1}{2}$ by a non-negligible amount. This contradicts the fundamental assumption about the security of XORPUF described in Eqn. (1). Hence, we prove that the proposed protocols are secure, assuming the computational difficulty of the XORPUF modelling problem.

5 Experimental Results

5.1 Experimental Setup

The hardware architectures (including the cryptographic primitives and the hardware Trojan) were designed using Verilog HDL. The designs were synthesized and implemented using *Xilinx ISE 14.5* for the Xilinx Virtex-V target platform, and simulated using *Xilinx Isim*. The DPR methodology was implemented using the technique described in [9]. Power estimation of the circuits was carried out using *Xilinx XPower Analyzer* and delay estimation using *Xilinx Timing Analyzer*. The computations to recover the secret key were carried out on a PC with 2 GB of main memory and a 2 GHz CPU.

5.2 Effectiveness of Difference Based DPR Methodology

Table 1 shows the required partial bitstream file to be loaded via the network for erasal and implementation of an "add-on" Trojan discussed in Section 3. For both the test cases (AES Trojan and TRNG Trojan), the complete reconfiguration file is 3,890.087 kB. From Table 1, it is evident that the required partial bitstream sizes for adding and removing the Trojans are relatively very small, which is a direct consequence of the difference based DPR methodology. This in turn leads to lesser data transfer between the remote PC and the deployed FPGA. The *Internal Configuration Assess Port* (ICAP) (Virtex-5 primitive for PR) of the targeted FPGA board is configured to run in 8 bit configuration at a clock frequency of 100 MHz. Hence DPR for the required PR files are performed in the order of micro-seconds.

5.3 Effectiveness of Inserted Trojan

Table 2 shows the percentage increase in the total on-chip power consumption of the AES design before and after Trojan insertion, while Table 3 compares the critical path delays before and after Trojan insertion in the dynamic partition. Table 4 compares the hardware overheads for the golden and the Trojan-infected AES designs. The simulated power

TABLE 3
Timing Overhead for AES Encryption/Decryption Circuit

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

TABLE 4
Hardware Overhead for AES Encryption/Decryption Circuit

Device Utilization	Golden Reference Design	Trojan Infected Design	Hardware Overhead w.r.t Golden (%)
Slice	1,576	1,582	0.38
SliceReg	1,742	1,748	0.34
LUTs	3,733	3,739	0.16

TABLE 5
Power Overhead for TRNG Circuit

Golden Reference (Dynamic partition with PLL) (W)	Trojan Infected Circuit (W)	Increase in Power w.r.t.Golden (%)
1.314	1.395	6.1644

TABLE 6
Hardware Overhead for TRNG Circuit

Device Utilization	Golden Reference Design	Trojan Infected Design	Hardware Overhead w.r.t. Golden (%)
Slice Reg	2,086	2,094	0.3835
Slice LUT	3,711	3,715	0.1078
Occupied Slice	1,277	1,286	0.7048
LUT Flip-Flop Pairs	4,236	4,323	2.0538
BUFG/BUFGCTRLS	3	4	0.3333

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 here is much more lightweight than the Trojan described in [30]. 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 cipher-text obtained under the influence of the activated Trojan, led to the recovery of the cipher key by 15 minutes of computation time. (following the theory described in [27]).

Table 5 shows the percentage increase in the total on-chip power consumption of the TRNG design before and after Trojan insertion, while Table 6 compares the hardware overheads for the golden and the Trojan-infected designs. From these results, it is clear that the Trojan insertion can be realized using minimal hardware overhead. The power profile shows 6.164 percent increase in power after Trojan insertion. This is due to the fast Trojan clock. The probability of occurrence of $ones$, $P(1)$ in the generated random numbers is shown in Table 7. Due to 50 percent biasing of response to “zero”, it is expected to get $P(0)$ to be $(3/4)$, in the Trojan infected design, which is evident from the results. We collected 20 million LSB bits from the golden BFD-TRNG [13] design and the Trojan infected design. NIST [37] statistical tests were performed

TABLE 7
Frequency Distribution of Generated Random Numbers

Design Parameter	Trojan Infected Experimental	Trojan Infected Theoretical	Golden Reference Experimental	Golden Reference Theoretical
Frequency	0.2488	0.2500	0.4992	0.5000
Percentage Deviation	0.468%		0.16%	
Percentage Deviation Infected from golden (Experimental)	50.1522%			

TABLE 8
Hardware Utilization Incorporating Signature Verifier

Module Name	Slice	Slice Reg	LUT	LUTRAM	BRAM/FIFO
Ethernet API Controller	1,059	1,188	2,355	11	111
Bridge	9	36	0	0	0
Reconfiguration Controller	8	12	15	0	0
Signature Verifier	1,298	2,247	2,891	1	32
Input FIFO	10	6	24	8	0

TABLE 9
Hardware Utilization for 8-XOR PUF with 64-bit APUFs

Module Name/ Components	Path Swapping Switches	Arbiter	Register/ Flip-Flop	Balanced XOR tree	Total
LUT	1,024	8	0	3	1,035
Flip-Flip	0	0	8	0	8

on the samples. 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). The infected design failed all NIST tests except “Rank” and “Linear Complexity”. NIST statistical tests were performed on the samples for the Golden TRNG circuit and the proposed HTH design on TRNG for various Trojan sampling clock frequencies.This is provided in Table 10. As discussed in Section 3.2, the TRNG output LSB bitstreams pass or fail NIST statistical tests based on the type of the resultant distribution.

5.4 Hardware Overhead of Defense Strategies

The hardware utilization for the whole circuity implemented on the FPGA for restricted DPR mode is shown in Table 8, excluding the user application AES/TRNG with its “add-ons”. The restricted mode of DPR is implemented by calculating the SHA-3 hash value of the received PR bit streams followed by comparison with the stored hash values. Table 9 shows the hardware overhead of an 8-XOR PUF with 64-bit APUFs. Through our experimental results we found that our proposed secure DPR schemes incur minimal hardware overhead, and they are also minimally invasive by design.

6 Conclusions

IoT architectures need to trade-off between the essential flexibility and the inherent resource-constraints. Dynamic Partial Reconfiguration (DPR) is a powerful technique that adds immense flexibility to FPGAs, and is thus suitable for IoT applications, provided potent security threats are overcome. We have proposed and demonstrated the feasibility of implementation of two different low overhead secure DPR architectures targeted for IoT applications.

TABLE 10
Statistical Test Results for Golden and Infected TRNG

Test	Sampling Clock (Golden) 103.1992 KHz						Trojan Clock 135.2265 KHz						Trojan Clock 172.2531KHz					
LSB	LSB-1		LSB-2		LSB-3		LSB-1		LSB-2		LSB-3		LSB-1		LSB-2		LSB-3	
P-Val/Proportion	**P-Val**	**Prop.**	**P-Val**	**Prop.**	**P-Val**	**Prop.**	**P-Val**	**Prop.**	**P-Val**	**Prop.**	**P-Val**	**Prop.**	**P-Val**	**Prop.**	**P-Val**	**Prop.**	**P-Val**	**Prop.**
Frequency	0.1223	1.00	0.9114	1.00	0.0179	0.95	0.5341	1.00	0.7399	1.00	0.1223	1.00	F	F	F	F	F	F
BlockFrequency	0.3505	1.00	0.5341	1.00	0.4373	1.00	0.2133	1.00	0.5341	0.95	0.8343	1.00	F	F	F	F	F	F
CumulativeSums*	0.2133	1.00	0.4373	1.00	0.1626	0.95	0.2757	1.00	0.9114	1.00	0.9114	1.00	F	F	F	F	F	F
Runs	0.5341	1.00	0.3505	0.95	0.9114	1.00	0.7400	0.95	0.1626	1.00	0.5341	0.95	F	F	F	F	F	F
LongestRun	0.9114	1.00	0.9114	0.95	0.2133	1.00	0.0669	1.00	0.3505	1.00	0.1223	1.00	F	F	F	F	F	F
Rank	0.5341	1.00	0.5341	1.00	0.7399	1.00	0.0909	1.00	0.5341	1.00	0.2757	1.00	0.1626	1.00	0.2133	0.95	0.6371	1.00
FFT	0.5341	1.00	0.5341	0.95	0.8343	1.00	0.5341	1.00	0.8343	1.00	0.9643	1.00	F	F	F	F	F	F
NonOverlappingTemp.*	0.0043	0.90	0.0043	0.90	0.0179	0.90	0.0179	0.90	0.5341	F	0.0114	F	F	F	F	F	F	F
OverlappingTemplate	0.4373	0.95	0.5341	1.00	0.0909	1.00	0.3505	1.00	0.9915	1.00	0.8343	1.00	F	F	F	F	F	F
ApproximateEntropy	0.9915	1.00	0.4373	1.00	10.1626	0.90	0.1626	1.00	0.6371	1.00	0.0909	1.00	F	F	F	F	F	F
Serial*	0.3505	1.00	0.3505	1.00	0.1626	1.00	0.3505	1.00	0.1223	0.90	0.2133	1.00	F	F	F	F	F	F
LinearComplexity	0.9114	0.95	0.8343	1.00	0.0352	1.00	0.7399	1.00	0.3505	1.00	0.6371	1.00	0.3505	0.95	0.3505	1.00	0.9114	1.00

**Note: Tests with more than one subtest, the p-value and proportion shown here are the smaller values. F corresponds to cases of test failure.*

ACKNOWLEDGMENTS

A part of the current paper (Sections 2.1 and 3.1) is based on a paper published by the authors in the *ACM WESS'14* workshop.

REFERENCES

[1] A. J. Jara, M. A. Zamora-Izquierdo, and A. F. Skarmeta, "Interconnection framework for mHealth and remote monitoring based on the Internet of Things," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 9, pp. 47–65, Sep. 2013.

[2] S. Kelly, N. Suryadevara, and S. Mukhopadhyay, "Towards the implementation of IoT for environmental condition monitoring in homes," *IEEE Sensors J.*, vol. 13, no. 10, pp. 3846–3853, Oct. 2013.

[3] S. Wang, "Spatial data mining under Smart Earth," in *Proc. IEEE Int. Conf. Granular Comput.*, Nov. 2011, pp. 717–722.

[4] R. J. Lehmann, R. Reiche, and G. Schiefer, "Future internet and the agri-food sector: State-of-the-art in literature and research," *Comput. Electron. Agriculture*, vol. 89, pp. 158–174, 2012.

[5] V. Ovidiu, F. Peter, G. Patrick, G. Sergio, S. Harald, B. Alessandro, J. I. Soler, M. Margaretha, H. Mark, E. Markus, and D. Pat, "Internet of things strategic research roadmap," in *Internet of Things - Global Technological and Societal Trend*. Delft The Netherlands: River Publishers, 2011, pp. 9–52.

[6] G. De Michell and R. K. Gupta, "Hardware/software co-design," *Proc. IEEE*, vol. 85, no. 3, pp. 349–365, Mar. 1997.

[7] T. Wollinger, J. Guajardo, and C. Paar, "Security on FPGAs: State-of-the-art implementations and attacks," *ACM Trans. Embedded Comput. Syst.*, vol. 3, no. 3, pp. 534–574, 2004.

[8] ULTRA LOW-POWER iCE FPGAs. (2008) [Online]. Available: http://www.latticesemi.com/ /media/LatticeSemi/Documents/ApplicationNot es/UZ/UltraLow-PoweriCEFPGAs.PDF?document_id=44648, Lattice Semiconductor

[9] A. P. Johnson, R. S. Chakraborty, D. Mukhopadyay, and S. Gören, "Fault attack on AES via hardware Trojan insertion by dynamic partial reconfiguration of FPGA over ethernet," in *Proc. 9th Workshop Embedded Syst. Security*, Oct. 2014, pp. 1:1–1:8.

[10] M. van Dijk and U. Rührmair, "Physical unclonable functions in cryptographic protocols: Security proofs and impossibility results." *IACR Cryptology ePrint Archive*, vol. 2012, p. 228, 2012.

[11] D. Lim, "Extracting secret keys from integrated circuits," Master's thesis, MIT, Cambridge, MA, USA, 2004.

[12] K. Eguro, "SIRC: An extensible reconfigurable computing communication API," in *Proc. IEEE Symp. Field-Programmable Custom Comput. Mach. (Short Paper)*, May 2010, pp. 135–138.

[13] Q. Tang, B. Kim, Y. Lao, K. Parhi, and C. Kim, "True random number generator circuits based on single- and multi-phase beat frequency detection," in *Proc. IEEE Custom Integr. Circuits Conf.*, Sep. 2014, pp. 1–4.

[14] G. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proc. 44th ACM/IEEE Des. Autom. Conf.*, Jun. 2007, pp. 9–14.

[15] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Proc. 17th ACM Conf. Comput. Commun. Security*, 2010, pp. 237–249.

[16] S. Shen and M. Carugi, "Standardizing the Internet of Things in an evolutionary way," in *Proc. ITU Kaleidoscope Acad. Conf.: Living in a Converged World - Impossible Without Standards?*, Jun. 2014, pp. 249–254.

[17] M. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. Grieco, G. Boggia, and M. Dohler, "Standardized protocol stack for the Internet of (important) Things," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 3, pp. 1389–1406, 3rd Quarter 2013.

[18] D. Uckelmann, M. Harrison, and F. Michahelles, *Architecting the Internet of Things*, 1st ed. New York, NY, USA: Springer, 2011.

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

[20] (2014, Feb.). Xilinx UltraScale MPSoC Architecture. Xilinx Inc. [Online]. Available: http://www.xilinx.com

[21] C. Wang, X. Li, and X.-H. Zhou, "CRAIS: A crossbar-based interconnection scheme on FPGA for big data," *J. Comput. Sci. Technol.*, vol. 30, no. 1, pp. 84–96, 2015.

[22] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1998.

[23] P. Bomel, J. Crenne, L. Ye, J. Diguet, and G. Gogniat, "Ultra-fast downloading of partial bitstreams through ethernet," in *Proc. 22nd Int. Conf. Archit. Comput. Syst.*, 2009, pp. 72–83.

[24] S. Gören, Y. Turk, O. Ozkurt, and A. Yildiz, H. F. Ugurdag, "Achieving modular dynamic partial reconfiguration with a difference-based flow," in *Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 2013, pp. 270–270.

[25] H. Li and Z. Friggstad, "An efficient architecture for the AES mix columns operation," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2005, vol. 5, pp. 4637–4640.

[26] J. Daemen and V. Rijmen, *The Design of Rijndael: AES-the Advanced Encryption Standard*. New York, NY, USA: Springer Science & Business Media, 2002.

[27] D. Mukhopadhyay, "An improved fault based attack of the advanced encryption standard," in *Proc. 2nd Int. Conf. Cryptol. Africa: Progress Cryptol.*, 2009, pp. 421–434.

[28] M. Tunstal, D. Mukhopadhyay, and S. Ali, "Differential fault analysis of the advanced encryption standard using a single fault," in *Proc. 5th IFIP WG 11.2 Int. Conf. Inf. Security Theory Practice. Security Privacy Mobile Devices Wireless Commun.*, 2011, pp. 224–233.

[29] K. Xiao and M. Tehranipoor, "BISA: Built-in self-authentication for preventing hardware Trojan insertion," in *Proc. IEEE Int. Symp. Hardware-Oriented Security TRUST*, 2013, pp. 45–50.

[30] S. S. Ali, R. S. Chakraborty, D. Mukhopadhyay, and S. Bhunia, "Multi-level attacks: An emerging security concern for cryptographic hardware," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2011, pp. 1–4.

[31] B. Sunar, W. Martin, and D. Stinson, "A provably Secure true random number generator with built-In tolerance to active attacks," *IEEE Trans. Comput.*, vol. 56, no. 1, pp. 109–119, Jan. 2007.

[32] A. Cherkaoui, V. Fischer, A. Aubert, and L. Fesquet, "A self-timed ring based true random number generator," in *Proc. IEEE 19th Int. Symp. Asynchronous Circuits Syst.*, May 2013, pp. 99–106.
[33] M. Dichtl and J. Golić, "High-speed true random number generation with logic gates only," in *Proc. 9th Int. Workshop Cryptographic Hardware Embedded Syst.*, 2007, vol. 4727, pp. 45–62.
[34] H. Martin, T. Korak, E. S. Millan, and M. Hutter, "Fault attacks on STRNGs: Impact of glitches, temperature, and underpowering on randomness," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 2, pp. 266–277, Feb. 2015.
[35] R. S. Chakraborty, I. Saha, A. Palchaudhuri, and G. K. Naik, "Hardware Trojan insertion by direct modification of FPGA configuration bitstream," *IEEE Des. Test Comput.*, vol. 30, no. 2, pp. 45–54, Apr. 2013.
[36] J. Katz and Y. Lindell, *Introduction to Modern Cryptography (Chapman & Hall/CRC Cryptography and Network Security Series)*. Boca Raton, FL, USA: CRC Press, 2007.
[37] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," DTIC Document, Fort Belvoir, VA, USA, Tech. Rep. SP 80022 Rev. 1a, 2001.

Anju P. Johnson received the BTech degree in electronics and communication engineering from Cochin University of Science and Technology (CUSAT) in 2010 and the MTech degree in VLSI Design from Amrita University in 2012. She has been a PhD research fellow and a senior project officer in the Department of Computer Science and Engineering, IIT Kharagpur, since 2012. Before Joining IIT Kharagpur, she was a faculty (Adhoc) in the Department of Electronics and Communication Engineering, NIT Calicut. Her research interests include design and analysis of hardware Trojan Horse and VLSI system design. She is a student member of the IEEE.

Rajat Subhra Chakraborty received the BE (Hons.) degree in electronics and telecommunication engineering from Jadavpur University, India, in 2005, and the PhD degree in computer engineering from Case Western Reserve University, Cleveland, OH. He has been an assistant professor in the Computer Science and Engineering Department, IIT Kharagpur, since 2010. His professional experience includes a stint as CAD software engineer at National Semiconductor, and a graduate internship at AMD Headquarters at Santa Clara (California). His research interests include hardware security; including design methodology for hardware IP/IC protection; Hardware Trojan detection and prevention through design and testing, attacks on hardware implementation of cryptographic algorithms, and reversible watermarking for digital content protection. He has close to 50 publications in international journals and conferences of repute. He has delivered keynote talks and tutorials at several international conferences and workshops, and has rendered his service as a reviewer and program committee member for multiple international conferences and journals. He is the co-author of four book chapters, one published book: *Reversible Digital Watermarking: Theory and Practice* (Morgan Claypool, USA), and one forthcoming book: *Hardware Security* (CRC Press, Boca Raton, FL). He is one of the recipients of the IBM Faculty Award for 2012, and a "Royal Academy of Engineering (United Kingdom) Fellowship" in 2014. He holds one US patent, and two more international patents and three Indian patents have been filed based on his research work. He is a member of the IEEE.

Debdeep Mukhopadhyay received the BTech degree from the Department of Electrical Engineering, IIT Kharagpur, Kharagpur, India, and the MS and PhD degrees in computer science and engineering from IIT Kharagpur. He was an assistant professor with the Department of Computer Science and Engineering, IIT Madras, Chennai, India, and is currently an associate professor with the Department of Computer Science and Engineering, IIT Kharagpur. His research interests include cryptography, VLSI of cryptographic algorithms, and side channel analysis. He received the Indian Semiconductor Association Techno Inventor Award for best Ph.D. thesis in 2010, the Indian National Science Academy Young Scientist Award in 2010, the Indian National Academy of Engineers Young Engineer Award in 2010, the Associate of Indian Academy of Science in 2011, the Outstanding Young Faculty Fellowship in 2011, and the IUSSTF Fellowship in 2012. He is a member of the IEEE.