# PUFSec: Protecting Physical Unclonable Functions Using Hardware Isolation-based System Security Techniques

Mengmei Ye*, Mehrdad Zaker Shahrak**, and Sheng Wei*

*Department of Computer Science and Engineering, University of Nebraska-Lincoln
Email: mye@huskers.unl.edu, shengwei@unl.edu
**School of Computing, Informatics, and Decision Systems Engineering, Arizona State University
Email: mzakersh@asu.edu

***Abstract*—This paper aims to address the security challenges on physical unclonable functions (PUFs) raised by modeling attacks and denial of service (DoS) attacks. We develop a hardware isolation-based secure architecture extension, namely PUFSec, to protect the target PUF from security compromises without modifying the internal PUF design. PUFSec achieves the security protection by physically isolating the PUF hardware and data from the attack surfaces accessible by the adversaries. Furthermore, we deploy strictly enforced security policies within PUFSec, which authenticate the incoming PUF challenges and prevent attackers from collecting sufficient PUF responses to issue modeling attacks or interfering with the PUF workflow to launch DoS attacks. We implement our PUFSec framework on a Xilinx SoC equipped with ARM processor. Our experimental results on the real hardware prove the enhanced security and the low performance and power overhead brought by PUFSec.**

## I. Introduction

Physical unclonable function (PUF) is a hardware implementation of a secure one way function leveraging the intrinsic random and unique hardware features [1]. In the past decade, PUF has become a very popular research area in the hardware security community. Researchers have developed a variety of PUFs, such as RO PUF [1][2], arbiter PUF [3], and SRAM PUF [4], and adopted them in various security applications, such as key generation and exchange [5] , intellectual property protection [4], and hardware root of trust [6].

Despite the strong security feature and the low overhead provided by many PUF designs, PUF as a security sensitive device is exposed and vulnerable to a variety of threat models. **First,** PUF is vulnerable to the attacks that attempt to reverse engineer and uncover its internal structure [7][8][9]. One of the most powerful threats in this category is PUF modeling attack, where the attacker intentionally collects a large set of challenge response pairs (CRPs) and employs machine learning techniques to build a software model and generate correct responses to arbitrary challenges [8][9]. **Second,** similar to other system resources that provide critical services, PUF is also vulnerable to denial of service (DoS) attacks, which attempt to maliciously stress or modify the PUF services and compromise the usability of PUF and its applications.

978-1-5386-1421-1/17/$31.00 © 2017 IEEE

To the best of our knowledge, there have been no intensive study on mitigating DoS attacks targeting PUFs. Instead, the community has been dedicated to developing modeling-resilient PUFs to defeat PUF modeling attacks. The existing approaches mainly follow the direction of *active design*, which aims to complicate the modeling attack by increasing the randomness and reducing the correlation between challenges and responses, such as XOR-mixed PUF [2] and feed-forward PUF [10]. The active design-based security schemes provide strong and effective protection to the PUF. However, one must develop a specific protection mechanism for each specific type of PUF, as the security mechanism is an integral part of the PUF. It limits the flexibility in terms of adapting to new security attacks, as well as introduces a high cost in the PUF deployment. Also, the inline security protection often complicates the PUF design and results in a significantly increased overhead in performance, such as the response delay.

We develop a hardware isolation-based secure architecture extension to effectively protect the PUF from not only modeling attacks but also DoS attacks. Our key idea is to place the PUF in a bus-level hardware-isolated secure world on the target system, where the adversaries from the non-secure world does not have access even if they have compromised the target system and taken control over the kernel. Moreover, the access to the secure world is strictly controlled by a *secure monitor*, which examines the validity of the requests and executes the secure/non-secure mode switching. Different from the existing PUF protection methods, PUFSec is a *passive protection* mechanism that is independent of specific PUF designs and implementations. In other words, it treats the PUF as a blackbox and does not require any modification to its internal design while deploying the security protection. Furthermore, PUFSec introduces minimum performance and power overhead, as it does not require complicated cryptographic or hashing operations.

PUFSec is realizable with the existing hardware isolation primitives as part of the security features provided by system

processors, such as ARM TrustZone [11] and Intel SGX [12]. Such a CPU platform with security features is widely available to PUFs, because (1) ARM and Intel are the two most commonly used processors in the mobile and PC domains, respectively; and (2) Even though PUF is a low-level silicon device, most of its use cases are on a system-level platform where either ARM or Intel processors are available, such as most of the PUF applications presented in the hardware security community [13][14]. On the other hand, PUFSec does not rely on any specific hardware isolation primitive or platform. In fact, the existing hardware isolation techniques only provide the low level isolation without defining a full-fledged security framework to support upper-level security applications. PUFSec fills this gap by employing a strict access control scheme to restrict unauthorized or excessive service requests to the target PUF, which prevents against both modeling and DoS attacks.

Note that PUFSec is not a PUF design or implementation. Instead, it is a lightweight and non-intrusive security framework that can protect the numerous PUFs developed by the hardware security community. By developing PUFSec, we aim to deliver a "divide and conquer" security paradigm to the PUF community, which extracts the security protection task from the PUF design process. Benefiting from such a paradigm, PUF designers can solely focus on effective PUF designs and leave the security protection layer to a security extension like PUFSec, which employs the state-of-the-art system security techniques to achieve a strong protection.

## II. Background: PUF Threat Models and Protection Mechanisms

### A. PUF Threat Models

As shown in Fig. 1, a typical PUF workflow is the blue (solid) arrow from the PUF to the verifier, where the verifier obtains the responses from PUF based on its chosen challenges and compares the responses with the ground truth CRP database. In the following subsections, we elaborate how modeling attacks and DoS attacks may maliciously compromise this workflow and break the security of PUF.

*1) PUF Modeling Attacks:* Most of the security attacks on PUF focus on reverse engineering the internal structure of the PUF, so that one can clone and emulate the PUF behavior using a software model [8][9]. Since it is very expensive if not impossible to physically reverse engineer and clone a PUF, most of the existing methods have focused on software model building attacks in the following ways: (1) Collect a sufficiently large set of CRPs and employ machine learning techniques to build a software model for the PUF [9] (i.e., *Attack M-1* in Fig. 1); or (2) Conduct side channel analysis (SCA) on the PUF hardware to uncover the PUF design and build a software emulator to generate the same PUF responses [8] (i.e., *Attack M-2* in Fig. 1).

*2) PUF DoS Attacks:* PUF DoS attacks attempt to interfere with the normal PUF functionality so that the verifier either does not obtain any response from the PUF or obtain erroneous responses. In either case, the target PUF will fail to provide its service (i.e., generating the correct response upon receiving a challenge) and thus causing a DoS attack. We consider two types of PUF DoS attacks.

- *Traditional DoS Attack*, shown as *Attack D-1* in Fig. 1, where the attacker attempts to issue many CRP requests and occupy all the capacity of the PUF. Consequently, a legitimate PUF challenge from the verifier may not be accepted or processed in a timely manner. Attack D-1 is different from Attacks M-1 and M-2 in that D-1 does not require CRP modeling.
- *Data Falsifying-based DoS Attack*, shown as *Attack D-2* in Fig. 1, where the attacker compromises and modifies either the PUF hardware or the memory that stores the PUF response data. Both of the cases will result in erroneous PUF responses compared to the pre-registered ground truth CRPs maintained by the verifier. Consequently, the authentication task will likely to fail, causing a false negative situation.

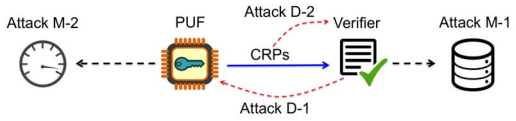


Fig. 1. Workflow of PUF threat models.

### B. PUF Protection Mechanisms

There are several PUF protection schemes that have been proposed to prevent modeling attacks. One category of approaches focuses on improving the PUF *internally*, i.e., maximizing the randomness of PUF and minimizing the correlation between challenges and responses, so that the attackers have to collect an extreme large number of CRPs for the modeling attacks to succeed [2][10].

The second category of approaches focuses on protecting the PUF *externally* by obfuscating the CRPs or employing a security protocol to prevent the attackers from gaining illegal access to the CRP data. For example, controlled PUF design [1] applies random hashing to the PUF challenges and responses to hide the actual CRPs from the attackers. The Aegis secure processor [14] and the OASIS instruction set extension [13] achieve access control to PUF using encryption. Slender PUF [15] minimizes the exposure of PUF responses to the attackers by conducting substring matching. Yu et al. [16] design the lockdown PUF with strict mutual agreements between the PUF and the verifier, which requires both parties to determine the PUF challenges and thus ensures a two-way authentication.

The existing PUF security mechanisms provide strong protections against modeling attacks. However, they either require modifications to the existing PUF architecture (i.e., the first category) or deploy computation-intensive security scheme (i.e., the second category). Also, to the best of our knowledge, there has been no intensive study on preventing PUF DoS attacks. With PUFSec, we aim to address both threats from a new angle by using hardware isolation.

## III. PUFSec Framework

In order to prevent modeling attacks and DoS attacks, we develop PUFSec, which is a hardware isolation-based

secure architecture extension to protect PUF. As shown in Fig. 2, we divide the system components into two physically isolated zones, namely the normal world and the secure world, using bus-level hardware isolation primitives, such as ARM TrustZone [11]. The hardware isolation ensures that normal world cannot access secure world resources due to the physical boundary at the bus level, making the secure world a trusted execution environment to process security sensitive tasks. At system runtime, the secure monitor residing in the secure world controls the context switching between the two worlds to coordinate the execution of secure and non-secure tasks, and it ensures that there is no unintended sharing of resources between the two worlds. In the PUFSec design, we deploy the target PUF in the secure world so that it gains the isolation protection from the hardware level. Our secure world and normal world designs allow legal accesses but block the illegal ones to the target PUF, as presented in the next subsections.

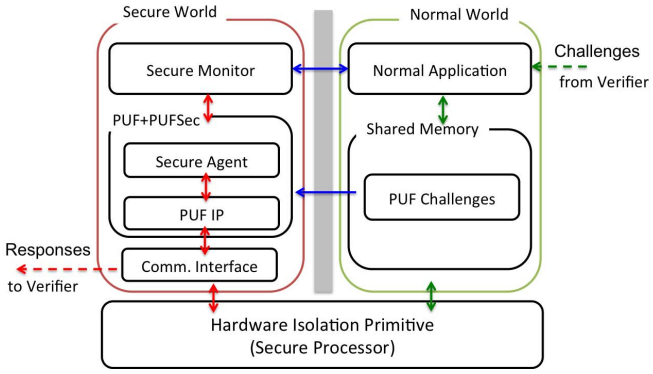


Fig. 2. PUFSec framework.

### A. PUFSec Normal World Design

The normal world of PUFSec framework consists of a normal application and a shared memory, as shown in Fig. 2. The normal application is the representative of the verifier, which requests PUF responses to chosen challenges for authentication purposes. The shared memory is accessible by both the normal world and the secure world, and it serves as the communication medium between the two worlds by storing the requested challenges from the authorized verifier. With the shared memory, the normal application can provide inputs to the secure world by issuing memory store instructions.

### B. PUFSec Secure World Design

The secure world of PUFSec consists of two critical security components that enable the protection of the target PUF, namely the secure monitor and the secure agent. The secure monitor controls the mode switching between the secure world and the normal world, and conducts such switches by issuing secure monitor calls (SMC) [11] either after the normal application loads the challenges to the shared memory or after the PUF generates the responses. In addition, the communication interface of the system, as a peripheral device, also resides in the secure world to deliver the PUF responses to the verifier without exposure to the attacker.

The secure agent serves as a secure gateway to the PUF, which enforces a security policy, namely *PUFSec Access Control*. This security policy ensures that PUFSec only accepts legitimate CRP requests, and it blocks illegal requests before they are processed by the system. To prevent modeling and DoS attacks, the secure agent categorizes the verifiers into three lists:

- *White List*, which contains benign verifiers that are not suspected to be adversaries. The secure agent will *grant* PUF accesses to these verifiers.
- *Black List*, which contains verifiers that are confirmed to be malicious based on the application-specific security policy, e.g., verifiers that have been requesting an excessive number of CRPs with a high frequency. The secure agent will *decline* PUF accesses to these verifiers.
- *Gray List*, which contains verifiers that are highly suspicious to be attackers based on their recent behavior, e.g., an increasing number of PUF requests beyond the normal demand. The secure agent will *penalize* these suspicious verifiers by adding intentional delays to the responses.

The secure agent monitors the request growth rate in terms of the number of CRP requests at runtime and moves the verifier IDs among the three lists to accomplish the access control. The request growth rate for a specific verifier is based on two factors: (1) current CRPs, which are the CRPs that the verifier requests in the current session (i.e., a pre-defined period of time); and (2) total CRPs, which are all the CRPs that the verifier has requested since the launch of the system application. We calculate the request growth rate following Equation (1) and update it upon receiving each request from the normal application. In particular, for verifiers with total number of CRPs less than a small start-up threshold ($SUT$) and thus holding a large growth rate according to Equation (1), we grant the access to prevent the verifiers from being moved to black or gray lists.

$$Request\ Growth\ Rate = \frac{\#\ Current\ CRPs}{\#\ Total\ CRPs} \quad (1)$$

Pseudocode 1 describes the detailed procedure of invoking PUF with PUFSec. We define a white list threshold ($WLT$) as the maximum request growth rate allowed for white-list verifiers and a gray-list threshold ($GLT$) for that of the gray-list verifiers, which are used to determine whether a verifier should be moved from one list to another. Our key idea is that the verifier should be penalized once it is "downgraded" from the white list to the gray list. In that case, we hold the requests into a waiting queue and begin serving them only if the number of pending requests reaches a threshold $C$ set by the secure agent. In the event that a verifier is "downgraded" from the gray list to the black list, we immediately dismiss its currently pending requests and block its future requests. It is worth noting that the threshold values (i.e., $SUT$, $WLT$, $GLT$, and $C$) are assigned according to the expected total number of CRPs from the target PUF applications.

With the enforced access control policy, the secure agent is capable of differentiating the malicious verifiers from benign

**Pseudocode 1** PUFSec Workflow.

```
Normal application issues SMC.
Secure monitor switches state from normal to secure.
if Verifier is not in black list or gray list then
    if Request history is less than SUT then
        Invoke PUF IP.
    else if Growth rate is less than WLT then
        Invoke PUF IP.
    else
        Move request into waiting queue.
        Place verifier ID into gray list.
        if Waiting queue size reaches C then
            Invoke PUF IP for requests in waiting queue.
        end if
    end if
else if Verifier is in gray list then
    if Growth rate is less than GLT then
        Invoke PUF IP.
    else
        Place verifier ID into black list.
    end if
end if
Send PUF responses to verifier through secure interface.
Secure agent issues SMC.
Secure monitor switches state from secure to normal.
```

verifiers. Malicious verifiers who request large volume of CRPs frequently (i.e., Attacks D-1, M-1, and M-2) will be trapped in the gray/black list and get penalized/blocked to avoid further damage to the system. The benign verifier will stay in the white list and obtain PUF responses via the secure communication channel within the isolated secure world. In this way, the PUF responses will be immune to data falsifying attack (i.e., Attack D-2).

### C. PUFSec Implementation

We implement PUFSec on a Xilinx ZC702 board with Zynq-7000 All Programmable SoC, which supports ARM TrustZone features [11]. Fig. 3 shows the overall architecture of the processing system (PS) and programmable logic (PL) on the board that are connected via an AXI interconnect.

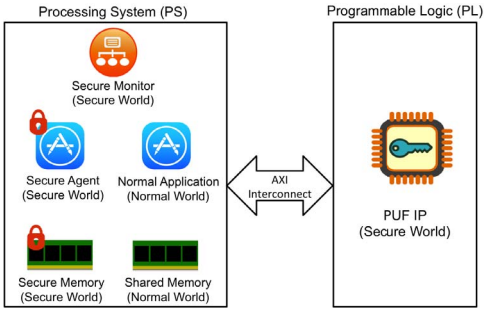


Fig. 3. System implementation of PUFSec.

**PL Implementation.** We implement an 8-bit RO PUF following the design by Suh et al. [2] and deploy it into the PL of the board to serve as the target PUF. Note that the implementation and evaluation of PUFSec do not depend on the implementation of the target PUF, and that is the reason why we adopt a simple PUF design in our study.

**PS Implementation.** In order to set up the security checking feature at the bus level, we set the non-secure (NS) bit to 0 on the AXI interconnect port connected to the PS by utilizing the Xilinx Vivado software. Then, we implement the secure monitor, the secure agent with access control, and the normal application according to the PUFSec framework. In the secure monitor, we implement an SMC handler using assembly code to detect, save, and restore the world states. We set the NS bit to 1 on part of the on-chip memory (OCM) addresses (i.e., 0x00020000 to 0x0002FFFF) to configure it to the normal world. Also, we keep the NS bit for the remaining memory space on the board to 0, which configures it to the secure world. For access control of the secure agent, we set $SUT$ as 500, $WLT$ as 20%, $GLT$ as 15%, and $C$ as 3.

## IV. Experimental Results

### A. Experimental Setup

We develop three test cases for performance and power evaluations, including the following:

- *Regular PUF*, which is the RO PUF we implement on the PL of the SoC (i.e., discussed in Section III-C) without any PUF protection mechanisms. We regard it as the baseline to evaluate the performance of the security enhanced PUFs;
- *Regular PUF + PUFSec*, in which we employ PUFSec to protect the regular PUF; and
- *Controlled PUF*, in which we hash the regular PUF responses using the DJB hash function [17].

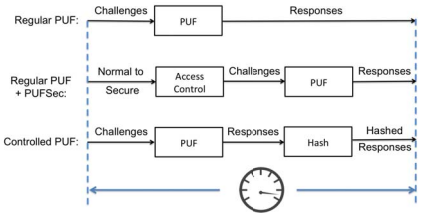


Fig. 4. Evaluation workflow.

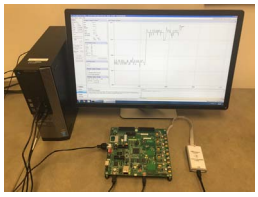

Fig. 5. Power setup.

Fig. 4 illustrates the workflow of performance and power/energy measurements in the three test cases. We capture the system times and power consumption between the the two time points (1) when the verifier feeds the challenge to the PUF, and (2) when the PUF responses are generated. We regard the difference between the two captured time stamps as the delay of the PUF. For power measurements, we follow the power measuring approach described in [18] using the TI Fusion tools. Fig. 5 shows the power experimental setup, in which we collect one power sample every 250 ms.

### B. Security Analysis

Based on the design and implementation of PUFSec, we analyze its security against the PUF threat models mentioned in Section II-A compared to the other types of PUF protection schemes, as shown in TABLE I.

TABLE I
SECURITY ANALYSIS OF PUFSEC.

| PUFs | Modeling | Denial of Service |
|---|---|---|
| Regular PUF [2] | NS | NS |
| Controlled PUF [1] | S | NS |
| XOR-mixed PUF [2] | S | NS |
| Lockdown PUF [16] | S | PS |
| Regular PUF + PUFSec | S | S |

S - Secure; NS - Non-Secure; PS - Partially Secure

The *XOR-mixed PUF* [2], as an internally protected PUF, succeeds to obfuscate the modeling attacks. However, without external security protection and access control, it is vulnerable to the *Attack D-1* and *Attack D-2* discussed in Section II-A.

The *controlled PUF* [1], as an externally protected PUF using obfuscation, is immune to modeling attacks as the attackers can only access the obfuscated version of the PUF response, which is not useful for the machine learning model. However, controlled PUF cannot defend against DoS attack, as there are no access control mechanisms to prevent attackers from falsifying the PUF data or occupying the PUF capacity.

The *lockdown PUF* [16], as an externally protected PUF using authentication protocol, primarily focuses on preventing modeling attacks, and the authentication protocol guarantees the communication legitimacy between the PUF and the verifier. Therefore, it is secure to defend against modeling attacks. The authors of the lockdown PUF [16] mentioned that they require additional mechanisms, such as a timeout, to defeat the traditional DoS attacks (i.e., *Attack D-1*). However, the lockdown approach does not include a data protection mechanism to prevent data falsifying-based denial of service attack (i.e., *Attack D-2*).

Our PUFSec leverages the strict access control policy to limit the function calls from suspicious verifiers and decline the ones from malicious verifiers, which is effective in defending against the modeling attacks (i.e., Attacks M-1 and M-2). Moreover, PUFSec is able to prevent both types of DoS attacks. For *Attack D-1*, when the adversary hacks the normal verifier in the white list and issues a large number of PUF requests, the secure agent and PUF will refuse to respond to the requests according to the access control mechanism presented in Pseudocode 1. For *Attack D-2*, PUFSec ensures that the attackers do not have access to the PUF data, as it is stored in the secure world and physically protected by hardware isolation. Therefore, PUFSec is secure against both modeling and DoS attacks.

### C. Security Evaluation for DoS Attacks

Based on the security analysis presented in Section IV-B, in this subsection, we further quantitatively evaluate the effectiveness of the access control scheme (i.e., Pseudocode 1) against DoS attacks. Our evaluations are based on the difference of PUF response delays between a malicious verifier issuing DoS *Attack D-1* and a benign verifier. For *Attack D-2*, since it is blocked by the hardware isolation primitive, we do not evaluate its performance for security evaluation purposes. Fig. 6 shows our evaluation results. We observe that the benign verifier is able to obtain CRPs in a much faster speed than the malicious verifier. For example, when both verifiers send a 1000-CRP request to the PUF, the attacker verifier obtains the CRPs more than 74% later than the benign verifier. More importantly, the advantage of the benign verifier (or the disadvantage of the attacker verifier) would expand with the increase in the number of CRPs. Moreover, in a real attack scenario where the attacker is unaware of the security restrictions, PUFSec can eventually trap the attacker into the black list and deny all the requests.

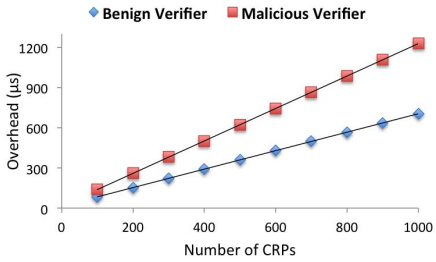


Fig. 6. PUF timing overhead for attacker and benign verifiers.

### D. Performance Overhead

To evaluate and compare the performance of PUFSec, controlled PUF, and regular PUF, we implement a system timer on the ZC702 board, which provides the system time at the resolution of microseconds. Fig. 7 shows the results of PUF response delays with different numbers of CRPs. Our PUFSec demonstrates slight performance overhead as compared to the regular PUF, as well as superior performance as compared to the controlled PUF [1].

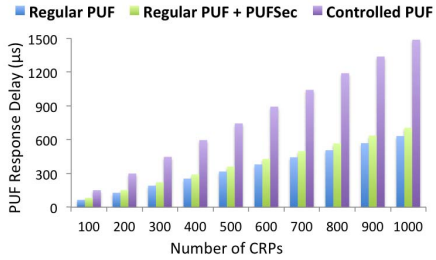


Fig. 7. PUFSec performance comparison with existing approaches.

### E. Power and Energy Consumption

Based on the experimental setup, we select VCCINT and VCCPINT rails on the ZC702 board to monitor PL and PS internal power consumption, respectively. In particular, we invoke each PUF with 10 million CRPs continuously in order to make the execution last longer and clearly capture the power differences in the measurements. Fig. 8 (a) shows the corresponding power consumption results for each approach. We observe that (1) during PUF execution, the power consumption in each case is very similar for both PS and PL; (2) after PUF execution, the power consumption immediately drops by a large amount; and (3) more importantly, different approaches would complete PUF execution at different times, due to the

[1]The controlled PUF for comparison is a simplified version compared to the one presented by Gassend et al. [1], where we applied the DJB hash function [17] with the hash length to be 1.

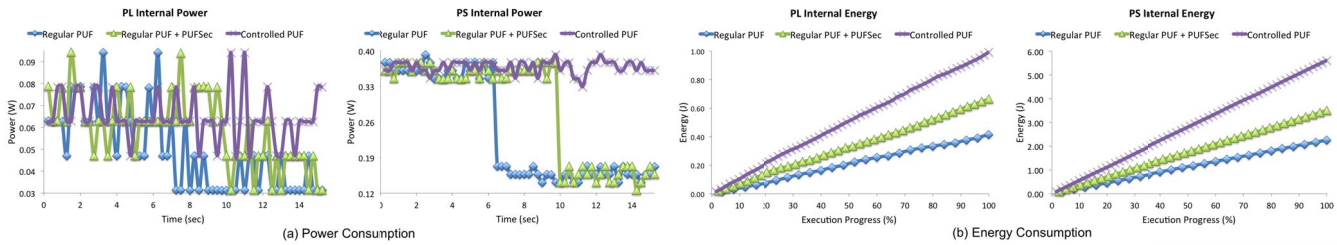


Fig. 8. PUFSec power and energy evaluation.

different timing overhead in the PUF protection mechanisms applied. For example, the completion times for regular PUF, PUFSec, and controlled PUF are around 7 sec, 9 sec, and 15 sec, respectively, which leads to considerably different energy consumption as shown in Fig. 8 (b). We observe that PUFSec consumes 33% (in PL) and 37% (in PS) less energy than the controlled PUF in the experiments with 10 million CRPs.

## V. Related Work of Hardware Isolation

Hardware isolation technology plays an essential role in system security benefiting from the security features in modern application processors, such as ARM TrustZone [11] and Intel SGX [12]. There have been many academic research efforts leveraging hardware isolation to protect various *software* applications , such as one-time password token [19], OS kernel [20], and software isolation [21]. More recently, hardware isolation has been adopted to protect security sensitive hardware components [22][23]. Our PUFSec work differentiates from the existing research efforts as it employs hardware isolation to defend against specific PUF attacks.

## VI. Conclusion

We for the first time developed a hardware isolation-based PUF protection mechanism, namely PUFSec, which defends PUFs against both modeling attacks and DoS attacks. PUFSec deploys the target PUF into an isolated secure environment and protects it using an access control policy. PUFSec is capable of restricting attackers from issuing large numbers of CRP requests for modeling or DoS attacks, as well as preventing data falsifying attempts made by attackers to generate false negatives in PUF-based authentication. We implemented PUFSec on a Xilinx programmable SoC. Our experiments on the real hardware proved the enhanced security and minimum performance and power overhead brought by PUFSec.

While accomplishing this work, we also realize the limitations of PUFSec, which only supports two types of PUFs: (1) PUFs that can be implemented as a peripheral device to the CPU, for which we can leverage bus-level isolation; and (2) SRAM PUFs, for which we can leverage either bus-level or memory-level isolation. We regard the applicability of PUFSec on the other types of PUFs as future work.

## Acknowledgment

We would like to thank Nianhang Hu for his help with implementing the TrustZone framework. This work was supported in part by the University of Nebraska Foundation under the Layman Award 1024460.

## References

[1] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *CCS*, 2002, pp. 148–160.
[2] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *DAC*, 2007, pp. 9–14.
[3] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas, "Extracting secret keys from integrated circuits," *VLSI Systems*, vol. 13, no. 10, pp. 1200–1205, 2005.
[4] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in *CHES*, 2007, pp. 63–80.
[5] Z. S. Paral and S. Devadas, "Reliable and efficient PUF-based key generation using pattern matching," in *HOST*, 2011, pp. 128–133.
[6] S. Zhao, Z. Qianying, and Q. Yu, "Providing root of trust for ARM TrustZone using on chip SRAM," in *TrustED*, 2014, pp. 25–36.
[7] C. Helfmeier, C. Boit, D. Nedospasov, and J.-P. Seifert, "Cloning physically unclonable functions," in *HOST*, 2013, pp. 1–6.
[8] S. Wei, J. B. Wendt, A. Nahapetian, and M. Potkonjak, "Reverse engineering and prevention techniques for physical unclonable functions using side channels," in *DAC*, 2014, pp. 1–6.
[9] U. Rhrmair, F. Sehnke, J. Slter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *CCS*, 2010, pp. 237–249.
[10] J. W. Lee, D. Lim, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas, "A technique to build a secret key in integrated circuits for identification and authentication application," in *VLSl Circuits*, 2004, pp. 176–179.
[11] "ARM security technology: Building a secure system using TrustZone technology," 2009, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.prd29-genc-009492c/index.html.
[12] M. Hoekstra, R. Lal, P. Pappachan, C. Rozas, V. Phegade, and J. del Cuvillo, "Using innovative instructions to create trustworthy software solutions," in *HASP*, no. 11, 2013.
[13] E. Owusu, J. Guajardo, J. McCune, J. Newsome, A. Perrig, and A. Vasudevan, "OASIS: On achieving a sanctuary for integrity and secrecy on untrusted platforms," in *CCS*, 2013, pp. 13–24.
[14] G. E. Suh, C. W. O'Donnell, and S. Devadas, "Aegis: A single-chip secure processor," *IEEE Design & Test of Computers*, vol. 24, no. 6, pp. 570–580, 2007.
[15] M. Majzoobi, M. Rostami, F. Koushanfar, D. S. Wallach, and S. Devadas, "Slender PUF protocol: a lightweight, robust, and secure authentication by substring matching," in *SPW*, 2012, pp. 33–44.
[16] M.-D. Yu, M. Hiller, J. Delvaux, R. Sowell, S. Devadas, and I. Verbauwhede, "A lockdown technique to prevent machine learning on PUFs for lightweight authentication," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 3, pp. 146–159, 2016.
[17] A. Partow, "General purpose hash function algorithms," https://github.com/ArashPartow/hash.
[18] E. Srikanth, "Zynq-7000 AP SoC low power techniques part 2 - measuring ZC702 power using TI Fusion Power Designer," *Xilinx*, 2014.
[19] H. Sun, K. Sun, Y. Wang, and J. Jing, "TrustOTP: Transforming smartphones into secure one-time password tokens," in *CCS*, 2015, pp. 976–988.
[20] A. Azab, K. Swidowski, R. Bhutkar, J. Ma, W. Shen, R. Wang, and P. Ning, "SKEE: A lightweight secure kernel-level execution environment for arm," in *NDSS*, 2016.
[21] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *USENIX Security*, 2016, pp. 857–874.
[22] N. Hu, M. Ye, and S. Wei, "Surviving information leakage hardware trojan attacks using hardware isolation," *IEEE Transactions on Emerging Topics in Computing*, 2017.
[23] M. Ye, N. Hu, and S. Wei, "Lightweight secure sensing using hardware isolation," in *IEEE SENSORS*, 2016, pp. 1–3.