An FPGA Implementation of 4×4 Arbiter PUF

Can Aknesil
Royal Institute of Technology (KTH)
Stockholm, Sweden
aknesil@kth.se

Elena Dubrova
Royal Institute of Technology (KTH)
Stockholm, Sweden
dubrova@kth.se

***Abstract*—The need of protecting data and bitstreams increases in computation environments such as FPGA as a Service (FaaS). Physically Unclonable Functions (PUFs) have been proposed as a solution to this problem. In this paper, we present an implementation of Arbiter PUF with 4 × 4 switch blocks in Xilinx Series 7 FPGA, perform its statistical analysis, and compare it to other Arbiter PUF variants. We show that the presented implementation utilizes five times less area than 2 × 2 Arbiter PUF-based implementations. It is suitable for many real-world applications, including identification, authentication, key provisioning, and random number generation.**

I. Introduction

Field-Programmable Gate Arrays (FPGAs) are widely used for fast data processing in cloud computing environment [1]–[6]. They are faster than Central Processing Units (CPUs) and at the same time more flexible than Application-Specific Integrated Circuits (ASICs), due to their reconfigurability.

In recent years, many companies, e.g. Amazon [1], IBM [2], Intel [3], Huawei [4], have started to offer FaaS. The users get a possibility to deploy their workloads on the physical FPGAs owned by the cloud provider. However, this raises a number of security and privacy concerns [7], [8], e.g. the user of FaaS may be willing to protect confidential information (personal data, medical records, etc.), or proprietary algorithms, e.g. Deep Neural Network (DNN) models, or bitstream, from the hardware owner.

Therefore, there is a need to assure data security in cloud services, including FaaS. This is achieved by data encryption, which relies on a secret key that should be securely stored. To assure security, this secret key should not be accessed by any unauthorized party including the hardware owner, and the manufacturer of the FPGA. To address this problem, advanced key storage methods based on PUFs have been proposed in FaaS setting [7], [9].

PUFs exploit random manufacturing process variations of electronic devices such as ASICs and FPGAs to generate device-specific keys that cannot be cloned [10]. The keys are generated only when required and do not remain stored on-chip. This provides a higher resistance to physical attacks. Furthermore, they cannot be cloned because it is nearly impossible to fabricate an electronic device with the same manufacturing imperfections.

An interesting type of Arbiter Physically Unclonable Function (APUF) bases on 4×4 switch blocks, has been presented in [11]. The new APUF is claimed to provide better resistance against modeling attacks (which aim to replicate the functionality of the target PUF) while keeping hardware overhead and computation time low. In this paper, we present an implementation of the 4×4 APUF in FPGAs.

The main contributions of this paper are:

1) An FPGA implementation of APUF with 4×4 switch blocks.
2) Statistical analysis of the implementation with respect to uniformity, reliability, and uniqueness.
3) Hardware overhead comparison to other APUF variants.

II. Background

A. The 4×4 APUF Design

Fig. 1 shows a block diagram of an n-stage APUF with 4×4 switch blocks. Every challenge c_i configures switch block i by selecting four different paths out of 16 possible paths, $i = 1, \dots, n$. The arbiter determines the outcome of racing of the four selected paths. There are $4! = 24$ possible orderings. So, an n-stage 4×4 APUF induces a 24-valued function of type $M^n \to M$, $M := \{0, 1, \dots, 23\}$. The function is defined by the delays of the $16n$ input-output connections of the n switch blocks.

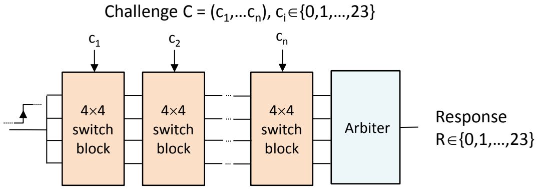
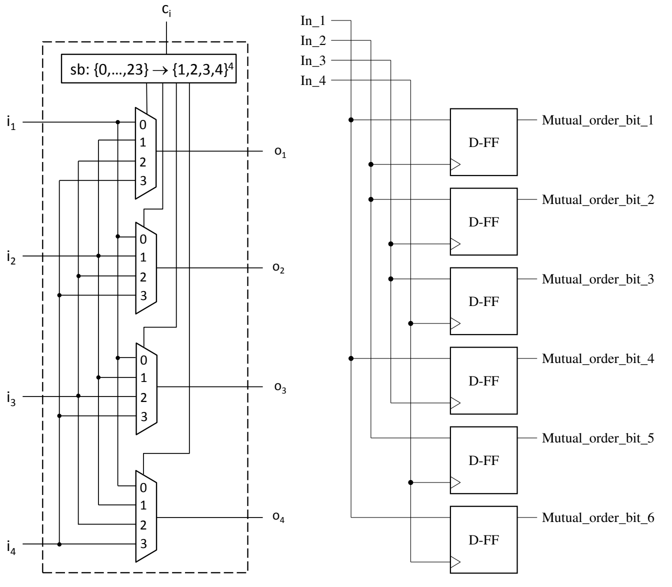


Fig. 1: An block diagram on an n-stage 4×4 APUF.

Every 4×4 switch block has four inputs, i_1, i_2, i_3, i_4, four outputs, o_1, o_2, o_3, o_4, and one control input, $c_k \in M$, for $k \in \{1, 2, \dots, n\}$. The operation of a switch block is defined by a function $sb : M \to S^4$ which maps each value of c_k into a permutation of the set $S := \{1, 2, 3, 4\}$. This permutation defines the connections between input and output. For example, permutation $(2, 3, 4, 1)$ means connections are (i_1, o_2), (i_2, o_3), (i_3, o_4) and (i_4, o_1). Any of 24! possible permutations of the set M can be used for this mapping.

B. Other PUF Designs

Many PUF designs aiming resistance to the modeling attacks have been proposed, including Feed-Forward Arbiter PUF [10], Non-Linear Arbiter APUF [10], XOR-PUF [10],



(a) 4×4 switch block

(b) Arbiter

Fig. 2: 4×4 APUF hardware diagrams

CRC-APUF [12], Lightweight Secure PUF [13], Reconfigurable Optical PUF [14], Phase Change Memory (PCM) based Reconfigurable PUF [14], Intrinsically reconfigurable D-RAM based PUF (D-PUF) [15], R^{3}PUF [16], Interpose PUF [17], MPUF [18], Majority Vote XOR-PUF [19], FF-XOR-PUF [20], FPGA implementation of a challenge pre-processing structure APUF [21].

III. 4x4 APUF FPGA Implementation

Our FPGA implementation for 4×4 APUF constitutes of several switch blocks, and an arbiter.

Inputs and outputs of a switch block can be mapped in 24 (4!) different configurations specified by the 5-bit challenge representing numbers in the interval $[0, 23]$.

The arbiter compares every 6 pairs of the outputs belonging to the last switch block ($\binom{4}{2} = 6$), and generates a 6-bit response representing the order of every pair. Later on, outside hardware, this 6-bit response is transformed into an integer in $[0, 23]$ that represents a particular order of 4 paths.

A. Switch Blocks

The hardware design of a switch block is shown in Fig. 2a.

A switch block includes 4 4×1 multiplexers, each producing one of the 4 outputs. A challenge translation module (function sb) is implemented to translate the 5-bit challenge into 8 bits: 2-bit selectors for every multiplexer. One of the possible permutations among 24! is selected and hard-coded into hardware. The multiplexers and the translation module are implemented with 6-input Look-Up Tables (6-LUTs) on the FPGA.

1) Equality of Paths: In theory, in our implementation, the length of all 16 paths in a switch block should be identical so that only factor causing the delays to differ must be the manufacturing differences. However, when it comes to implementing a design on an FPGA, there are many other factors that can cause the delays to differ; such as, the compiler decisions on the placement and routing, internal implementation of cells and interconnections. These additional factors can dominate the manufacturing differences and decrease uniqueness across chips.

In this paper, we implemented and evaluated two different kinds of placement for switch block and the arbiter cells: default placement performed by the design tool, and manual placement. The manual placement is performed to make the paths within and between switch blocks, and paths between the last switch block and the arbiter as symmetrical as possible. Only 4 out of 8 6-LUTs of a switch block (belonging to the multiplexers) are manually placed.

The results, however, are better with the default placement. Accordingly, we only present default placement results.

B. Arbiter

The hardware design of an arbiter is shown in Fig. 2b.

The arbiter module includes 6 D-flip-flops. The D-input and the clock input of every one of them are connected to one of the input pairs it compares. If the rising edge sent to switch blocks comes to the D-input first, the output is 1, otherwise, 0. In the rest of the paper, the output of each of these flip-flops is called a "mutual order bit".

We are not translating mutual order bits into a 5-bit number that represents the order of the 4 paths on the hardware; instead, we are directly taking mutual order bits from the hardware and doing the translation on computer when necessary. This decision enables us to detect illegal responses (mentioned below) and, accordingly, prevent disturbance of the statistical analysis.

1) Arbiter placement: Default and manual placement were implemented and evaluated for arbiter flip-flops. Default placement results were better. Accordingly, we only present default placement results.

C. Testbed

- FPGA target board: 2 CW305 Artix FPGA Target boards [22] with Xilinx Artix-7 XC7A100T FPGA
- Capture board: CW1173 ChipWhisperer-Lite [22]
- Hardware design tool: Xilinx Vivado v2019.2.1 (64-bit)
- Software: ChipWhisperer 5.0 [22]

D. Further Details on Implementation

The primary objective of this particular implementation is to evaluate the statistical properties of the proposed PUF on FPGAs. Countermeasures against potential hardware attacks, such as power and side-channel analysis, are left to future work.

IV. Data Collection & Analysis

Data is collected as challenge-response pairs. A predefined set of challenges was applied to 4×4 APUFs implemented on unique FPGA chips. In this paper, this process is called an "experiment". Also each experiment is performed multiple times to perform reliability analysis.

24-stage APUF is the main focus during data collection and analysis due to limited resources.[1] A randomly chosen set of challenges was applied since considering all possible challenges is not realizable.

A. Format of Responses

All of the analysis is performed on individual responses, rather than a concatenation of multiple responses. In a real-world application, it may be necessary to combine multiple responses of an APUF to achieve a larger response (such as a 128-bit response) for utility purposes.

B. Influence of Arbiter Paths

During our experiments, an anomaly in the responses was discovered: Delay differences within the path pairs, leading from the last switch block to each flip-flop in the arbiter, cause some of the mutual order bits to be inconsistent.[2] This sometimes results in responses that cannot be translated into the permutation information. These responses are called "illegal" in this paper. And others are called "legal". Illegal responses are analyzed in more detail in [23].

C. Uniformity Analysis

In uniformity analysis, the response distribution is evaluated using data collected from one experiment. The challenge set in the experiment consists of 13824 challenges picked randomly in a uniform way.[3]

We looked at the average Hamming weights of 6 mutual order bits separately, as well as together. In the theoretical case, where responses are perfectly uniform, average Hamming weights of each mutual order bit should be 0.5, and the overall Hamming weight of responses should be 3.

We translated mutual order bits into integers in the interval $[0, 23]$, and looked at their distribution.[4]

D. Reliability Analysis

In theory, an ideal APUF should generate the same responses when the same challenge is applied over and over again. However, our experiments show that APUF responses are not always deterministic. In other words, the application of the same challenge over and over again does not always produce the same response.

In this paper, we define the reliability of a challenge to whether it always generates the same response or not. Challenges that always generate the same response are called "reliable challenges", others "unreliable challenges".

Reliability analysis is performed by running one experiment 407 times. The challenge set in the experiment consists of 13824 challenges picked randomly in a uniform way (the same set used in uniformity analysis).

We looked at the distribution of "number of responses" per every challenge across multiple applications of the same experiment. In the ideal case, where all the challenges are reliable, the ideal distribution is $(13824, 0, 0, \dots)$, where there are 407 entries. The n^{th} entry is the number of challenges that generate n different responses over 407 repeated experiments, $n = 1, 2, \dots, 407$. The first entry is the number of challenges that generate one single response during all 407 experiments. Its percentage over the sum of all the entries gives the percentage of reliable challenges.

We also looked at the probabilities of a randomly selected challenge to produce its n^{th} likely response for $n = 1, 2, \dots$. These probabilities are calculated by analyzing response histograms for individual challenges.

E. Uniqueness Analysis

The essence of APUF is that responses to a particular set of challenges should be unique for every chip. Uniqueness analysis analyzes the degree of uniqueness.

Uniqueness analysis is performed by running one experiment 407 times on 2 different FPGA chips with the same part number. The challenge set in the experiment consists of 13824 challenges picked randomly in a uniform way (the same set used in uniformity analysis). Later on, analyses are performed on every 407 pairs of experiments (2 experiments each performed on a different chip in a pair). Finally, the results for every pair are averaged.

We are defining the theoretically ideal case as follows: responses are perfectly random, illegal responses are assumed not to occur, and responses are perfectly reliable. In this ideal case, the probability for each legal response to occur is $\frac{1}{24}$. The probability of responses to the same challenge from 2 APUFs implemented on different chips to be different, also the expected number of challenges that produce different results on 2 different chips is $\frac{23}{24} = 95.8333\%$. During the analysis, the actual number of challenges that produced different results on 2 different chips are compared with this ideal value.

There is one complication due to unreliable challenges. When responses from different chips differ, it is difficult to understand whether the cause is unreliability or manufacturing differences. Furthermore, as our experiments show, unreliable challenge sets are highly different for every chip. As a solution we also calculate uniqueness by excluding the union of unreliable challenges.

[1] ChipWhisperer toolchain only supports 128-bit to communicate the challenge. This selection was to keep the communication with the FPGA simple, at the same time, having large enough number of challenges to make brute force attacks infeasible.

[2] This concept can be demonstrated with an example: Let response bits (a, b, c, d, e, f) represent the comparison between output pairs of the last switch block, respectively, (o_1, o_2), (o_2, o_3), (o_3, o_4), (o_1, o_3), (o_2, o_4), and (o_1, o_4). If p_1, the path leading to o_1, is faster than p_2, and p_2 is faster than p_3, then, p_1 is concluded to be faster than p_3. So, responses, where $a = b \neq d$, are illegal. Yet, during experiments some of the responses we got were illegal. This phenomenon was not analyzed in [11].

[3] The size of the challenge set, 13824, and number of repetitions of experiments, 407, are chosen considering the time required to collect enough challenge-response pairs to accurately perform mentioned analyses.

[4] During the translation, we ignored illegal responses because their influence was negligible.

V. Results

A. Uniformity Analysis Results

The number of challenges in the experiment was 13758 and 66 of them were illegal. The percentage of legal responses is 99.52%.

Following results are calculated by ignoring the presence of illegal responses, because their influence is negligible.

Response distribution is shown in Fig. 3.

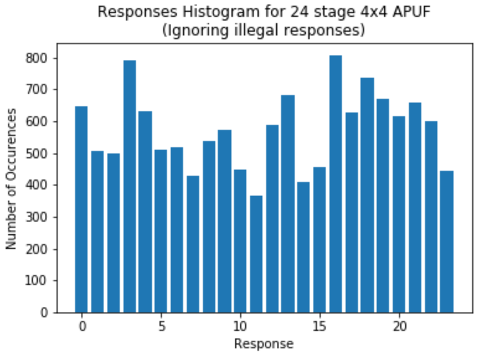


Fig. 3: Response distribution of 24-stage 4×4 APUF.

1) Distribution of Mutual Order Bits: Average of every mutual order bit, from bit 1 to 6: 0.5579, 0.4862, 0.5207, 0.5097, 0.4331, 0.4901.

The sum of these numbers, in other words, the average Hamming weight of mutual order bits together is 2.9978.

The Hamming weight distribution of responses (in mutual order bits format) is shown in Fig. 4.

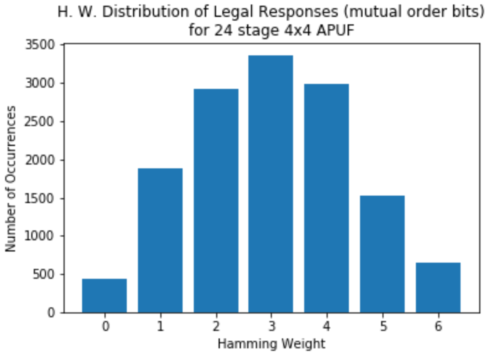


Fig. 4: Hamming weight distribution of responses (in mutual order bit format) for 24-stage 4×4 APUF.

B. Reliability Analysis Results

At the result of repeated applications of the same experiment, the distribution of the number of different responses for 2 FPGA chips is as follows:

# of responses	1	2	3	4	5	6	...
Chip 1	12835	957	22	10	0	0	...
Chip 2	12714	1063	30	16	1	0	...

The percentage of reliable challenges is as follows: 92.8458% (chip 1), 91.9704% (chip 2).

The probabilities of a randomly selected challenge to produce its n^{th} likely response are as follows, $n = 1, 2, \ldots$.

Responses	1st	2nd	3rd	4th	5th	...
Chip 1	99.0599%	0.9297%	0.0087%	0.0017%	0%	...
Chip 2	98.8828%	1.1015%	0.0133%	0.0023%	0%	...

On average for all chips, the probability of a randomly selected challenge to produce its most likely response is 98.9714%.

C. Uniqueness Analysis Results

Percentage of unreliable challenges (calculated using results of reliability analysis) for every FPGA chip is as follows: 7.1542% (chip 1), 8.0295% (chip 2).

Percentage of the union of unreliable challenges for 2 chips: 13.5923%

The percentage of challenges that produced different results on 2 different chips (unreliable challenges included) is 15.1520%.

The same percentage when the union of unreliable challenges from 2 chips are excluded is 9.3261%.

VI. Area Comparison

The area comparison of 4×4 APUF, implemented in this paper, to different variants of 2×2 APUFs is presented in Table I. Area ratio is calculated using 6-LUTs per response length.

The number of stages of 4×4 APUF is selected as 28 so that the number of possible challenges is equal or greater than 2^{128}.[5]

Compared to others, 4×4 APUF generates 5-bit response that represents a number in the range $[0, 23]$, rather than a single bit in the range $[0, 1]$. Since the interval $[0, 23]$ cannot be represented with an integer number of bits without redundancy, we are calculating the response length of 4×4 APUF as $\log_2 24 \cong 4.58$. The fact that 4×4 APUF generates 5-bit response is an advantage because to achieve the same amount of output, other APUFs should be run more than once or more than one instance of them should be implemented in parallel. It should also be underlined that critical path (path from the input stimuli, through all of the switch block, to the arbiter) of 4×4 APUF is shorter than 2×2 APUF, enabling response generation with a higher throughput.

VII. Discussion

In this section, we discuss whether an APUF with 4×4 switch blocks can be realized in FPGAs with a sufficient amount of manufacturing differences to be used in real-world applications.

First of all, our 4×4 APUF design extracts some amount of manufacturing differences. This means our design can be useful for some real-world applications.

[5] Keeping the number of challenge bits just above 128 would be wrong because challenge bits are redundant: 5-bit challenge of a switch block can only take values in the interval $[0, 23]$, rather than $[0, 31]$.

APUF type	Stages	Challenge length	Response length	6-LUTs	FFs	6-LUTs/response length	FFs/response length	Area ratio
2×2 APUF [10]	128	128	1	256	1	256	1	×5.24
8-XOR APUF [10]	128	128	1	2050	8	2050	8	×41.96
Interpose APUF [17]	128	128	1	514	2	514	2	×10.52
CRC-APUF [12]	128	128	1	320	1	320	1	×6.55
4×4 APUF	28	140	4.58	224	6	**48.86**	1.31	×1

TABLE I: Area comparison of 4×4 APUF to other various 2×2 APUF variants.

At first look, our design seems to have low uniqueness. Nevertheless, it should be kept in mind that our results are for responses of small length (4.58 bits). The overall uniqueness can be improved by combining multiple responses to create a larger one. Though, while doing so, non-ideal reliability should be taken into account.

A. *Our 4×4 APUF in the Real World*

We propose several methods to make better use of our 4×4 APUF implementation and give application examples to which these methods can be applied.

In all of these methods multiple responses are "combined" to produce a larger one. We are defining this combination as follows: Let R be the combination of $r_1, \dots r_m$, individual responses taken from the APUF.

$$R = 24^0 r_1 + \dots + 24^{m-1} r_m$$

We are performing the combination this way to prevent redundancy in the combined response.

a) Combining Responses of Randomly Selected Challenges: This is the most straightforward method. According to our results, the probability of a randomly selected challenge to produce different results on 2 different chips is 15.1520%. Let's call it p. Accordingly, the probability, u, of 2 different combined responses created from 2 different chips to be different can be expressed as follows.

$$u = 1 - (1 - p)^m \tag{1}$$

where m is the number of responses that are combined.

Let's generalize the case and call the same probability for n different chips $P(n)$, where $P(2) = u$. If we assume that number of PUFs is negligible compared to the possible number of combined responses ($n \ll 24^m$), $P(n)$ can be expressed as follows.

$$P(n) = u^{\binom{n}{2}} = u^{\frac{n(n-1)}{2}}, \quad n \geq 2 \tag{2}$$

If we combine equations 1 and 2, we can express P as a function of both n and m.

$$P(n, m) = \left[1 - (1 - p)^m\right]^{\frac{n(n-1)}{2}} \tag{3}$$

A contour diagram of P is shown in Fig. 5. Using equation 3, the sufficient number of response length for the required number of PUFs can be calculated. For example, if we want to uniquely identify 10000 PUFs with 99% probability, we need an at least 624-bit response.

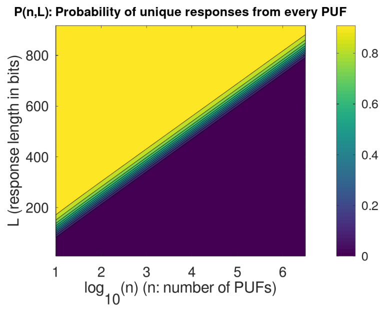


Fig. 5: Contour diagram for P (uniqueness probabilities).

The problem with this method is that unreliability of responses is not handled. If the same combined response is tried to be generated using the same challenges, the results will most probably differ, especially for higher m values. However, this unpredictability can be exploited by applications where True Random Number Generators (TRNGs) are necessary, such as key provisioning.

b) Detection of Reliable Challenges Beforehand: This method involves using only reliable challenges by detecting them beforehand. Detection can be performed either after fabrication before distribution or while runtime. This way, it will be possible to reproduce the same responses. This enables identification and authentication applications.

Equation 3 can also be used for this method with, this time, $p = 9.3261\%$. Need for larger responses (higher m) and extra cost for reliable challenge detection are traded with repeatability.

Better uniqueness would improve usage costs of our 4×4 APUF design by reducing the number of required responses to combine. However, there is a possibility that higher uniqueness can also cause lower reliability because when PUF delays are closer to each other, the results will be more sensitive to factors like noise.

VIII. Conclusion & Future Work

We implemented 4×4 APUF on FPGAs, performed statistical analysis and hardware overhead comparison to other APUF variants, and suggested methods for its real-life usage.

According to the analysis, the presented APUF implementation is suitable for many real-world applications, including identification, authentication, key provisioning, and random number generation. At the same time, it is five times more area efficient than the closest alternative. However, since uniqueness is lower than ideal, longer responses are needed to assure the desired level of security.

The presented APUF implementation is available at *https://github.com/canaknesil/4x4-apuf.*

Future work includes investigating possibilities of improving uniqueness by using alternative placement and routing, as well as performing security evaluation of 4×4 APUF. In particularly, it is necessary to evaluate the resistance of 4×4 APUF to Machine Learning-based modeling and side-channel attacks.

IX. Acknowledgement

This work was supported in part by the Vinnova Competence Center for Trustworthy Edge Computing Systems and Applications at KTH Royal Institute of Technology.

References

[1] (Nov. 24, 2020). "Amazon EC2 F1 instances," [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1/.

[2] (Nov. 24, 2020). "IBM cloudFPGA," [Online]. Available: https://www.zurich.ibm.com/cci/cloudFPGA/.

[3] (Nov. 24, 2020). "Intel DevCloud," [Online]. Available: https://software.intel.com/content/www/us/en/develop/tools/devcloud.html.

[4] (Nov. 24, 2020). "Huawei FPGA Accelerated Cloud Server," [Online]. Available: https://www.huaweicloud.com/en-us/product/facs.html.

[5] (Nov. 24, 2020). "Microsoft Azure Machine Learning," [Online]. Available: https://azure.microsoft.com/en-us/services/machine-learning/.

[6] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," *IEEE Micro*, vol. 35, no. 3, pp. 10–22, 2015.

[7] H. Englund and N. Lindskog, "Secure acceleration on cloud-based FPGAs – FPGA enclaves," in *2020 IEEE International Parallel and Distributed Processing Symp. Workshops*, 2020, pp. 119–122.

[8] Y. Yu, M. Moraitis, and E. Dubrova, "Why deep learning makes it difficult to keep secrets in the FPGAs," in *Proc. of Workshop on Dynamic and Novel Advances in Machine Learning and Intelligent Cyber Security*, Dec. 2020.

[9] M. A. Will and R. K. L. Ko, "Secure FPGA as a Service — towards secure data processing by physicalizing the cloud," in *2017 IEEE Trustcom/BigDataSE/ICESS*, 2017, pp. 449–455.

[10] D. Lim, "Extracting secret keys from integrated circuits," M.S. thesis, Massachusetts Institute of Technology, 2004.

[11] E. Dubrova, "A reconfigurable arbiter PUF with 4 x 4 switch blocks," in *2018 IEEE 48th International Symp. on Multiple-Valued Logic*, May 2018, pp. 31–37.

[12] E. Dubrova, O. Näslund, B. Degen, A. Gawell, and Y. Yu, "CRC-PUF: A machine learning attack resistant lightweight PUF construction," in *2019 IEEE European Symp. on Security and Privacy Workshops*, Jun. 2019, pp. 264–271.

[13] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Lightweight secure PUFs," in *Proceedings of the 2008 IEEE/ACM International Conf. on computer-aided design*, IEEE Press, 2008, pp. 670–673.

[14] K. Kursawe, A. Sadeghi, D. Schellekens, B. Skoric, and P. Tuyls, "Reconfigurable physical unclonable functions - enabling technology for tamper-resistant storage," in *Proc. of IEEE International Work. on Hardware-Oriented Security and Trust*, July, pp. 22–29.

[15] S. Sutar, A. Raha, and V. Raghunathan, "D-PUF: An intrinsically reconfigurable DRAM PUF for device authentication in embedded systems," in *Proc. of the International Conf. on Compilers, Architectures and Synthesis for Embedded Systems*, New York, NY, USA: ACM, 2016, 12:1–12:10.

[16] Y. Gao and D. C. Ranasinghe, "R^3PUF: A highly reliable memristive device based reconfigurable PUF," ArXive Technical report, Tech. Rep., Feb. 2017.

[17] P. H. Nguyen, D. P. Sahoo, C. Jin, K. Mahmood, U. Rührmair, and M. van Dijk, *The interpose PUF: Secure PUF design against state-of-the-art machine learning attacks*, Cryptology ePrint Archive, Report 2018/350.

[18] D. P. Sahoo, D. Mukhopadhyay, R. S. Chakraborty, and P. H. Nguyen, "A multiplexer-based arbiter PUF composition with enhanced reliability and security," *IEEE Trans. on Computers*, vol. 67, no. 3, pp. 403–417, 2018.

[19] N. Wisiol and M. Margraf, "Why attackers lose: Design and security analysis of arbitrarily large XOR arbiter PUFs," *Journal of Cryptographic Engineering*, vol. 9, no. 3, pp. 221–230, 2019.

[20] S. V. S. Avvaru, Z. Zeng, and K. K. Parhi, "Homogeneous and heterogeneous feed-forward XOR physical unclonable functions," *IEEE Trans. on Information Forensics and Security*, vol. 15, pp. 2485–2498, 2020.

[21] W. Ge, S. Hu, J. Huang, B. Liu, and M. Zhu, "FPGA implementation of a challenge pre-processing structure arbiter PUF designed for machine learning attack resistance," *IEICE Electronics Express*, vol. 17, no. 2, 2020.

[22] (Apr. 1, 2020). "Chipwhisperer wiki," [Online]. Available: https://wiki.newae.com/Main_Page.

[23] C. Aknesil, "An FPGA implementation of arbiter PUF with 4x4 switch blocks," M.S. thesis, KTH Royal Institute of Technology, 2020.