The Monte Carlo PUF

Vladimir Rožić, Bohan Yang, Jo Vliegen, Nele Mentens and Ingrid Verbauwhede
COSIC, KU Leuven
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
E-mail: firstname.lastname@esat.kuleuven.be

***Abstract*—Physically unclonable functions are used for IP protection, hardware authentication and supply chain security. While many PUF constructions have been put forward in the past decade, only few of them are applicable to FPGA platforms. Strict constraints on the placement and routing are the main disadvantages of the existing PUFs on FPGAs, because they place a high effort on the designer. In this paper we propose a new delay-based PUF construction called Monte Carlo PUF, that does not require low-level placement and routing control. This construction relies on the on-chip Monte Carlo method that is applied for measuring the delays of logic elements in order to extract a unique device fingerprint. The proposed construction allows a trade-off between the evaluation time and the error rate. The Monte Carlo PUF is implemented and evaluated on Xilinx Spartan-6 FPGAs.**

I. Introduction

Modern communication systems face numerous security challenges related to entity and data authentication, as well as data confidentiality and integrity. These challenges can only be addressed using cryptography, which requires methods for secure key generation and storage. In addition, supply-chain security issues are gaining more importance due to an increasing number of counterfeiting incidents and recycled integrated circuits (ICs) [1]. For FPGA applications, IP licensing and management is handled using bitstream encryption. In this case, device-specific keys are necessary in order to prevent cloning the IP cores. These problems can be mitigated using physically unclonable functions (PUFs) [2].

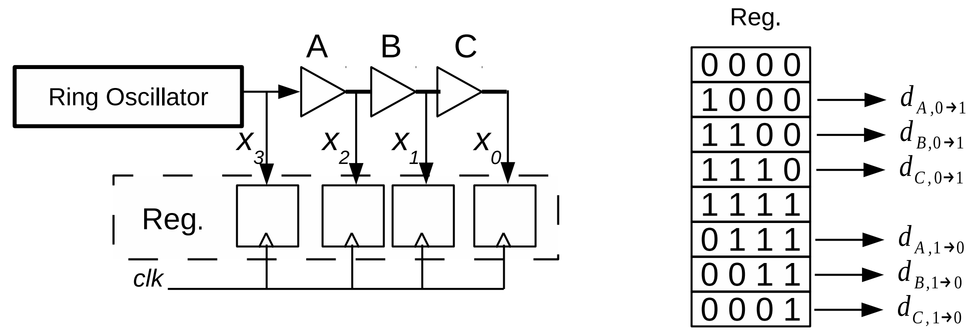
PUFs are security primitives that rely on the manufacturing variations of transistor and wire parameters, to produce unique IC identifiers. The security of these primitives relies on their uniqueness – the ability of the design to produce different identifiers on different devices based on identical circuits with identical layouts. In addition to uniqueness, a very important property of PUFs is their reliability – the ability to produce the same identifier on the same device after multiple measurements across a range of operating conditions. PUF constructs can be classified into two categories: *strong PUFs* where the number of identifier bits scales exponentially with circuit area, and *weak PUFs* where the number of PUF bits scales subexponentially (usually linearly) with area.

This work was supported in part by the Research Council KU Leuven: C16/15/058. In addition, this work is supported in part by the Flemish Government through G.0130.13N and FWO G.0876.14N, the Hercules Foundation AKUL/11/19, and through the Horizon 2020 research and innovation programme under grant agreement No 644052 HECTOR and Cathedral ERC Advanced Grant 695305. B. Yang is supported by the Scholarship from China Scholarship Council (No.201206210295).

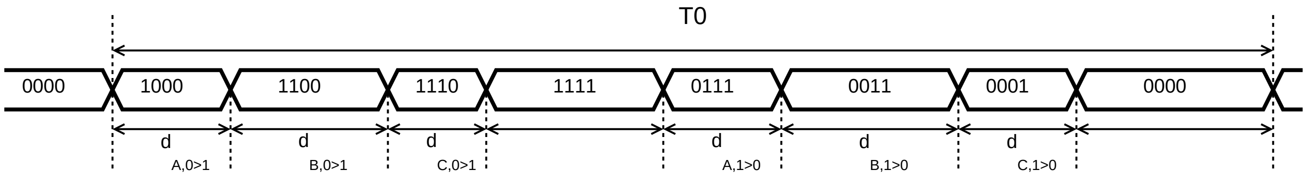
Many PUF constructions have been presented in literature in recent years but not all of them are suitable for FPGA implementations. The concept of a PUF was first introduced in [2] while the first silicon PUF was presented in [3]. Memory-based PUFs extract identifier bits from the power-up states of metastable elements. Examples are the SRAM-based PUF [4], latch-based PUF [5], flip-flop based PUF [6], buskeeper PUF [7] and butterfly PUF [8]. Some memory-based PUFs are not suitable for FPGA implementations because they rely on specialized ASIC cells – this is the case with the buskeeper PUF. SRAM blocks and flip-flops are available on all FPGAs but their states are usually reset after power-up which makes them unsuitable for generating device identifiers. The butterfly PUF [8] is suitable for any FPGA but it requires dedicated routing to achieve the balanced structure.

Delay-based PUFs exploit the timing variations of logic circuits. Some well known examples are the arbiter PUF [9], [10], ring-oscillator (RO) PUF [11], [12] and glitch PUF [13]. The main disadvantage of delay-based PUFs on FPGA is the increased design effort. The RO PUF requires identical layouts of the used ring-oscillators, and the arbiter PUF relies on balancing dual signal paths.

In this paper we present a novel PUF construction called the *Monte Carlo PUF*. This structure is a delay-based PUF. Unlike the existing delay-PUF constructions which rely on racing the signals along the different paths, the proposed construction relies on measuring the delays of the logic elements and then comparing them to generate a PUF response. The Monte Carlo PUF consists of an on-chip measurement setup that applies the Monte Carlo methodology to determine and compare delays of individual logic elements. This PUF falls into the category of weak PUFs because the number of extracted bits is proportional to the number of logic elements used for the delay measurement. The Monte Carlo PUF is suitable for both ASIC and FPGA implementations, but in this paper we focus on a Xilinx FPGA implementation using CARRY4 primitives [14]. These primitives are typically used for the synthesis of high-speed adders and multipliers. The advantage of using CARRY4 primitives in the Monte Carlo PUF is that these hardware modules can be cascaded using dedicated routing paths, which doesn't require additional effort of the designer. Half of the slices available on the Xilinx Spartan-6 FPGA contains CARRY4 primitives, and similar primitives exist on FPGAs from other families and vendors [15]. Several PUFs that utilize CARRY4 elements have been presented in the previous years [16], [17].



(a) Hardware setup and possible pattern values.



(b) Timing of signal $x_3x_2x_1x_0$.

Fig. 1: Setup configuration for measuring buffer delays using the Monte Carlo methodology.

This paper is organized as follows: Section II provides a brief overview of the PUF quality metrics. In Section III we present the concept of the Monte Carlo PUF and discuss the operating principle and the generic architecture. Section IV explores the implementation of the Monte Carlo PUF on a Xilinx Spartan-6 FPGA. In that section we present the results of the initial experimental analysis of the CARRY4 elements. In Section V we analyze the uniqueness and reliability of the implemented PUF and explore the trade-off between the error rate and evaluation time. We conclude the paper and discuss future work in Section VI.

II. Background

The quality of PUFs is usually measured using three main quality metrics:

- Uniqueness
- Reliability
- Randomness

Uniqueness is usually evaluated in terms of between-class Hamming distance. This distance is computed by collecting responses of many PUF instances and computing their average Hamming distance. Ideally, this distance should be approximately half of the width of the response.

Reliability is assessed in terms of the bit-error rate. The bit-error rate is obtained by evaluating the same PUF instance many times and computing the relative average Hamming distance. An ideal PUF has an error rate of 0 %. In practice, the error rates of up to 5 % are tolerable because they can be corrected using the error-correcting codes.

Randomness of a PUF is evaluated in terms of entropy. Min-entropy is the most conservative measure of unpredictability and it is defined as the minimal amount of information that is obtained from any observation of a random variable. It is computed as:

$$H_\infty(X) = -log_2(Pr(x_{max}))\,, \tag{1}$$

where x_{max} is the most likely outcome of variable X.

III. The Monte Carlo PUF

A. HW setup for Monte-Carlo measurements

The Monte Carlo method is an algorithm that can be used for numerical integration, simulations of physical systems and measurements of physical quantities. It uses samples from a uniform distribution to obtain the numerical approximations of the measured quantities.

An example of a Monte-Carlo-based experiment for measuring delays of logic elements is illustrated in Figure 1. The measurement setup consists of a free-running, possibly noisy, ring oscillator, three cascaded delay buffers (denoted by A, B and C), and a 4-bit register sampled by the system clock with period T_{clk}. T_0 is the period of the ring oscillator. For CMOS circuits, a $0 \rightarrow 1$ delay is different from a $1 \rightarrow 0$ delay because different transistors are used in the pull-up and the pull-down networks. Therefore, for each buffer there are two delay values denoted by $d_{A,0\rightarrow1}$,$d_{A,1\rightarrow0}$,$d_{B,0\rightarrow1}$,$d_{B,1\rightarrow0}$,$d_{C,0\rightarrow1}$,$d_{C,1\rightarrow0}$. We assume that the sum of all six delays is lower than T_0. This condition is easy to achieve by using a sufficiently slow ring oscillator. We further assume that the ring oscillator is not interlocked with the system clock. This requires some care in choice of the sampling frequency. We recommend to choose the ratio T_{clk}/T_0 that cannot be represented as a ratio of small integers. The uniformity of RO phase at sampling moments is ensured by this carefully chosen ratio and by the noise in the circuit.

The goal of the experiment is to determine buffer delays relative to the value T_0. The hardware setup for the experiment is shown in Figure 1a. The setup operates as follows: The ring

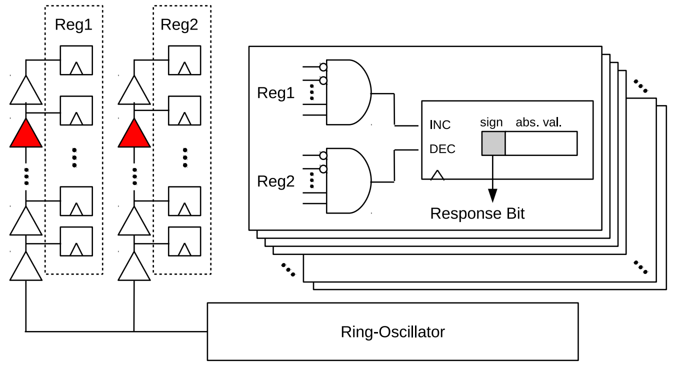


Fig. 2: The generic architecture of the Monte Carlo PUF.

oscillator is started and N consecutive samples of the 4-bit register are taken at times T_{clk}, $2T_{clk}, ...NT_{clk}$. These values are collected and counted. The right part of the Figure 1a shows the 8 possible values of the signal $x_3x_2x_1x_0$ that could be sampled. Signal $x_3x_2x_1x_0$ goes through these 8 values during each period T_0 as shown in Figure 1b. The measurement setup operates by sampling this signal many times and counting the appearances of each pattern. If the sampling moments are uniformly distributed across the phase of the ring oscillator, the counts of each pattern are proportional to delays of the elements. The delay corresponding to each pattern is indicated in Figure 1a.

Let N_{0011} denote the number of times the value 0011 is sampled. For each sample, the probability p_{0011} of sampling value 0011 is proportional to a delay of the buffer B.

$$p_{0011} = \frac{d_{B,1\to 0}}{T_0}\,. \tag{2}$$

The counter value N_{0011} after N trials is a random variable with binomial distribution.

$$Pr(N_{0011} = k) = \binom{N}{k} p_{0011}^k (1 - p_{0011})^{N-k}\,. \tag{3}$$

The mean of this distribution is:

$$E(N_{0011}) = N \cdot p_{0011} = N \cdot \frac{d_{B,1\to 0}}{T_0}\,. \tag{4}$$

Therefore, the delay of the buffer B can be approximated by:

$$d_{B,1\to 0} = T_0 \cdot \frac{N_{0011}}{N}\,. \tag{5}$$

Other delays can be computed in the same manner using the corresponding counter values.

The standard deviation of the distribution of N_{0011} is

$$\begin{aligned}\sigma(N_{0011}) &= \sqrt{N \cdot p_{0011}(1 - p_{0011})} = \\ &= \sqrt{N \cdot \frac{d_{B,1\to 0}}{T_0} \cdot \left(1 - \frac{d_{B,1\to 0}}{T_0}\right)}\,.\end{aligned} \tag{6}$$

The relative error of the approximation is proportional to the ratio $\sigma(N_{0011})/E(N_{0011})$.

$$\frac{\sigma(N_{0011})}{E(N_{0011})} = \sqrt{\frac{T_0 - d_{B,1\to 0}}{N \cdot d_{B,1\to 0}}}\,. \tag{7}$$

A higher number N of trials results in higher precision of the measurement procedure. We note that using a very slow RO results in many all-zero and all-ones values captured in the 4-bit register. These captured values are not used for computing the buffer delays. Therefore, increasing the period T_0 leads to a higher ratio of unused samples which reduces the measurement precision.

B. PUF architecture

The Monte Carlo PUF is based on comparing the delays of two logic elements with identical layouts and generating a single response bit as a result of the comparison. These delays are determined using the Monte Carlo methodology. The methodology can be used for comparing both $0 \to 1$ and $1 \to 0$ delays, potentially extracting two response bits from a singe pair. However, these response bits may be correlated.

The generic architecture of the Monte Carlo PUF is shown in Figure 2. The ring oscillator is used to produce a signal that is propagated through two delay lines. Two registers are used to capture the position of the signal edge and to update the counter values. As shown in Figure 1a, a specific pattern corresponds to each delay. A setup for comparing the $0 \to 1$ delays of the two highlighted buffers in Figure 2 is shown in the right part of the Figure. Every time a corresponding pattern is detected in $Reg1$, the counter value is incremented, if it is detected in $Reg2$ the counter value is decremented. If both registers contain this pattern, or if neither of them do, the counter value is not changed. The response bit value denotes the slower element, 1 if it is the left one, and 0 if it is the right one. This bit is the sign bit of the counter at the end of the measurement because the response depends only on whether the final counter value is positive or negative.

We note that, in order to ensure that the result depends on the process variations rather than the systematic variations, it is required that the two delay lines have identical layouts. However, there is no such requirement for individual buffers within the same delay line because the two buffers on the same delay line are never compared to produce a bit.

The design parameters of the Monte Carlo PUF are:

- T_{clk}–Sampling clock period. The value of this parameter doesn't affect the properties of PUF. A high-speed clock can be used in order to improve the evaluation time.
- T_0–Ring oscillator-period. The value of this parameter has to be chosen carefully. $T_0/2$ should be higher than the cumulative delay of the delay line in order to ensure the proper operation of the Monte Carlo PUF. However, high value of T_0 leads to reduced precision of the Monte Carlo measurements and the increased bit error rate.
- k– The number of buffers on the delay line.
- N – The number of Monte Carlo trials per evaluation. The value of this parameter is used to make trade-offs between the bit error rate and the evaluation time.

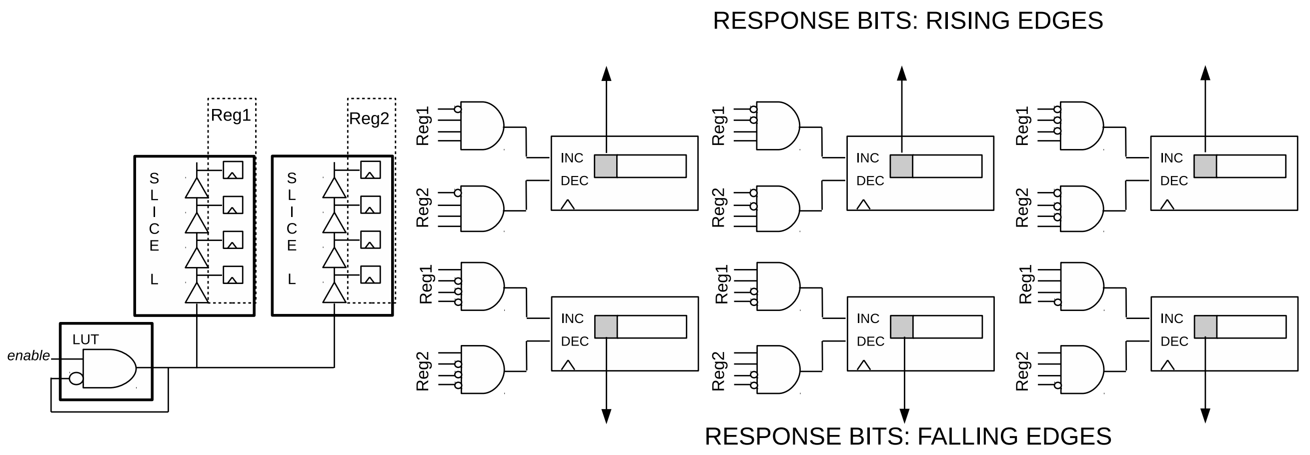


Fig. 3: The basic implementation of the Monte Carlo PUF.

IV. Implementation on Spartan-6

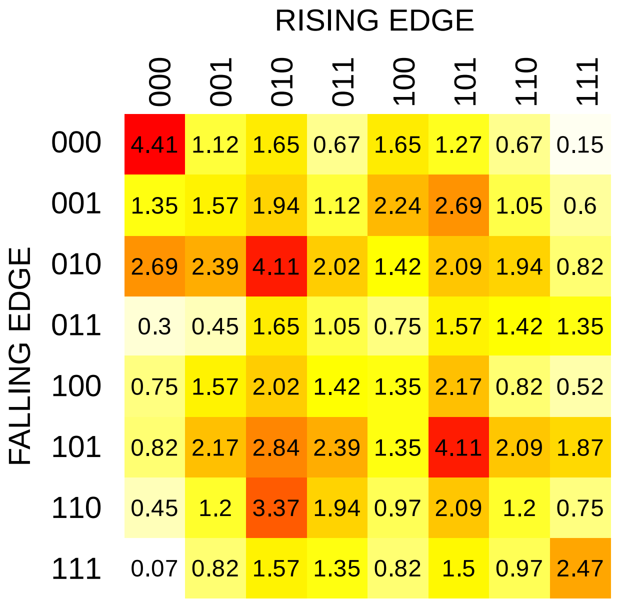
A. Basic Implementation

For our implementation platform we used a Xilinx Spartan-6 FPGA. This type of FPGA contains three types of slices called SliceX, SliceL and SliceM. The slices of type X don't contain the CARRY4 primitive. These slices comprise 50 % of all available slices. Slices of type L and M each comprise 25 % of the FPGA slices. Newer Xilinx FPGAs and SoCs such as Virtex6 [18] and ZYNQ-7000 [19] contain CARRY4 primitives on every slice. These slices contain the CARRY4 primitives and can be used for implementing the delay lines. We assume that slices of the same type have identical layouts. For this reason, in all of our implementations, the delay lines are implemented using slices of the same type (either both L or both M).

Figure 3 shows the basic implementation of the Monte Carlo PUF consisting of a ring oscillator implemented using a single look-up table (LUT) and two CARRY4 elements. Each CARRY4 primitive is configured to operate as a tapped delay line, with the output of each stage connected to a flip-flop inside the same slice. Using this implementation, six identifier bits can be extracted – three by comparing the delays of the $0 \rightarrow 1$ transitions and three by comparing the delays of the $1 \rightarrow 0$ transitions.

B. Initial Experiment Results

An initial study was performed on 1337 instances of basic Monte Carlo PUFs implemented on a single FPGA. Six identifier bits are extracted from each instance and the probabilities of all 64 combinations are computed. In order to examine the correlation between the bits extracted using the rising edge and those extracted using the falling edge, the results are represented in the form of the heat map as shown in Figure 4. It can be seen that the delays of the $0 \rightarrow 1$ transitions are highly correlated with the delays of the $1 \rightarrow 0$ transitions. If all six bits are used, the resulting identifier would have only 4.5 bits of min-entropy (0.75 per response bit). The same trend



	000	001	010	011	100	101	110	111
000	4.41	1.12	1.65	0.67	1.65	1.27	0.67	0.15
001	1.35	1.57	1.94	1.12	2.24	2.69	1.05	0.6
010	2.69	2.39	4.11	2.02	1.42	2.09	1.94	0.82
011	0.3	0.45	1.65	1.05	0.75	1.57	1.42	1.35
100	0.75	1.57	2.02	1.42	1.35	2.17	0.82	0.52
101	0.82	2.17	2.84	2.39	1.35	4.11	2.09	1.87
110	0.45	1.2	3.37	1.94	0.97	2.09	1.2	0.75
111	0.07	0.82	1.57	1.35	0.82	1.5	0.97	2.47

Fig. 4: Heat map showing the probabilities (%) of the six identifier bits extracted from carry chains – three bits using the rising edge and three using the falling edge.

is observed when the experiment was repeated on a different FPGA.

Based on the results of this experiment, we decided to use only three bits per basic PUF instance to construct an identifier. Figure 5 shows the distribution of the 3-bit responses obtained using the $0 \rightarrow 1$ delay measurements. The distribution is not uniform as it fails the χ^2-test. This non-uniformity is most likely caused by the global variations within a single slice which cause correlations between the response bits. Based on the estimated probabilities, the response contains 2.5 bits of min-entropy (0.83 bits of min-entropy per response bit). Similar result was obtained when the experiment was repeated on another FPGA chip.

Further improvements in entropy per bit, could be made by

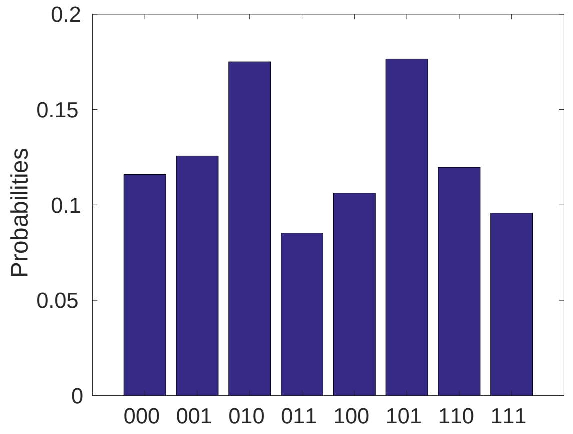


Fig. 5: The distribution of the 3-bit responses obtained from 1337 PUF instances on a single FPGA.

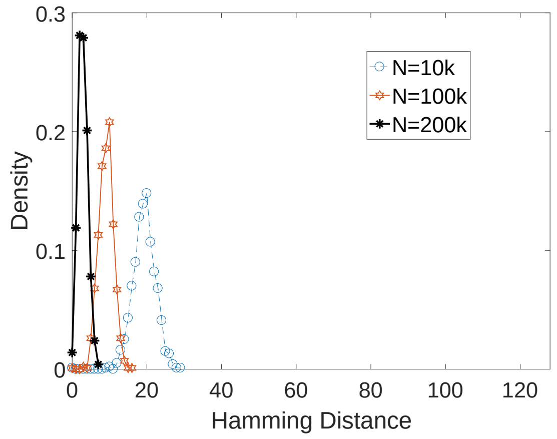


Fig. 6: Distribution of HD from 1000 evaluations of a single 128-bit PUF instance.

using only two least-correlated bits per CARRY4. Our experiments show that this results in only a minor improvement in entropy per bit, as all three combinations produce between 0.85 and 0.9 min-entropy per response bit. In the remainder of this paper, we are using the 3-bit version of the Monte Carlo PUF.

C. Discussion

Typical application of weak PUFs is for secure key generation and storage. For the purpose of evaluating uniqueness and reliability, we've implemented a 128 bit key generator. We note that in practice, more than 128 PUF bits are necessary to produce a 128 bit key. This is, in part, because the PUF response, doesn't have the full entropy – based on the empirical estimation a 128 bit PUF response contains less than 107 bits of min-entropy. In addition, some entropy may be lost due to the error-correcting procedure.

A 100 MHz clock signal generated by an on-board quartz oscillator was used for sampling. We note that the correct operation of the Monte Carlo PUF doesn't depend on the clock frequency, but only on the ratio T_{clk}/T_0. This leaves the opportunity for improving the evaluation time by using an asynchronous high-speed ring oscillator for sampling.

Interlock of the oscillators is a possible threat to the functionality of the Monte Carlo PUF. The Monte Carlo measurement methodology relies on the uniformity of sampling. If the RO and the sampling oscillator are interlocked, the sampled phases are no longer uniformly distributed. This problem can be addressed by replacing a single-LUT RO with a structure with a period that changes in a pseudo-random manner. In addition, the counter values can be used to implement the on-the-fly testing procedure to detect problems at run time. These ideas will be explored in future work.

V. PUF Characterization

A 128-bit PUF instance is implemented using 43 basic 3-bit PUF structures. Hamming distances between two 128-bit PUFs across ten chip instances are used to evaluate reliability and uniqueness of the Monte Carlo PUF on Spartan-6.

A. Reliability

To evaluate the reliability of proposed PUF, output trails from a single 128-bit Monte Carlo PUF instance are collected. To test the influence of the number N of the measurements on the error rate, this Monte Carlo PUF is operated with three different parameter values: $N = 10\ 000$, $N = 100\ 000$ and $N = 200000$. For each value of N, 1000 output trails are recorded and analysed. One output trial is selected from these three sets of 1000 output trails and regarded as the reference trial. All Hamming distances between the reference output trail and all other trails are calculated. The possible values of these Hamming distances are within the range $[0, 128]$.

The probability densities of having any possible Hamming Distance are plotted in Figure 6. The bit error rates in average are 15.1% for $N = 10\ 000$, 7.1% for $N = 100\ 000$ and 2.26% for $N = 200\ 000$. As expected, an increase in the number N of Monte Carlo iterations will lead to a reduction in the bit error rate. In other words, the reliability can be improved by choosing a larger N. Evaluation time for $200K$ measurement using on-board 100 MHz clock is 2 ms. This evaluation time can be improved by using high-speed asynchronous ring oscillators for measurement.

B. Uniqueness

In order to evaluate the uniqueness, ten disjoint 128-bit PUF instances are implemented on ten different FPGA chips. There are, in total, 100 different output trails. Uniqueness can be demonstrated by examining the Hamming distances between any two output trails. All 4950 Hamming distances are measured and plotted on Figure 7. The mean of these Hamming distances is 63.98 (minimum 41, maximum 86). It means that a 128-bit Monte Carlo PUF instance will, in average, have half of its PUF bits different from another 128-bit Monte Carlo PUF instance.

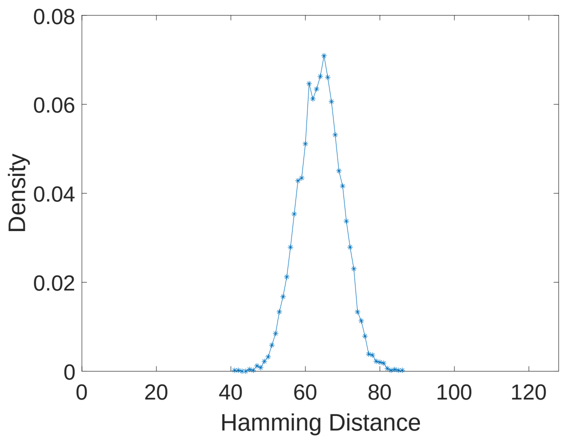


Fig. 7: Uniqueness.

TABLE I: Implementation Results.

Utilization for 3-bit response	38 slices (59 FFs + 90 LUTs)
Utilization for 128-bit response	1662 slices (2537 FFs + 3871 LUTs)
Evaluation time	2 ms (20000 cycles at 100MHz)
Bit error rate	2.26%

VI. Conclusions and Future Work

In this paper, we proposed a novel Monte-Carlo-based PUF which is suitable for FPGA applications. It relies on a measurement procedure that determines the delays of logic elements. A response bit is generated by comparing the measured delays of two logic elements. An implementation on a Xilinx Spartan-6 FPGA is presented and analyzed. The results show that the proposed PUF provides a high uniqueness and a low error rate. Results of the reference implementation are summarized in Table I.

In future work, we will investigate the PUF behavior under changing operating conditions. In addition, several hardware implementation aspects remain to be explored. In particular, a reduction of the evaluation time using asynchronous oscillators for sampling, and a reduction of the area by sharing resources (oscillators and counters) between different PUF instances, will be considered. PUF implementations on various FPGA families from different vendors remain to be explored.

The final goal of this work is to build up a complete PUF-based key generator including error correcting codes and privacy amplification (entropy compression), and to demonstrate the use of the Monte Carlo PUF in a security system.

References

[1] U. Guin, D. DiMase, and M. Tehranipoor, “Counterfeit Integrated Circuits: Detection, Avoidance, and the Challenges Ahead,” *J. Electronic Testing*, vol. 30, no. 1, pp. 9–23, 2014.

[2] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, “Physical One-Way Functions,” *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002. [Online]. Available: http://science.sciencemag.org/content/297/5589/2026

[3] B. Gassend, D. E. Clarke, M. van Dijk, and S. Devadas, “Silicon Physical Random Functions,” in *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, 2002, pp. 148–160. [Online]. Available: http://doi.acm.org/10.1145/586110.586132

[4] J. Guajardo, S. S. Kumar, G. J. Schrijen, and P. Tuyls, “FPGA Intrinsic PUFs and Their Use for IP Protection,” in *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, 2007, pp. 63–80.

[5] Y. Su, J. Holleman, and B. Otis, “A 1.6 pJ/bit 96% Stable Chip-ID Generating Circuit using Process Variations,” in *2007 IEEE International Solid-State Circuits Conference, ISSCC. Digest of Technical Papers*, Feb 2007, pp. 406–611.

[6] R. Maes, P. Tuyls, and I. Verbauwhede, “Intrinsic PUFs from Flip-flops on Reconfigurable Devices,” in *Workshop on Information and System Security*, 2008.

[7] P. Simons, E. van der Sluis, and V. van der Leest, “Buskeeper PUFs, a promising alternative to D Flip-Flop PUFs,” in *2012 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2012, San Francisco, CA, USA, June 3-4, 2012*, 2012, pp. 7–12.

[8] S. S. Kumar, J. Guajardo, R. Maes, G. J. Schrijen, and P. Tuyls, “The Butterfly PUF: Protecting IP on every FPGA,” in *IEEE International Workshop on Hardware-Oriented Security and Trust, HOST 2008, Anaheim, CA, USA, June 9, 2008. Proceedings*, 2008, pp. 67–70.

[9] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas, “Extracting secret keys from integrated circuits,” *IEEE Trans. VLSI Syst.*, vol. 13, no. 10, pp. 1200–1205, 2005.

[10] M. Majzoobi, F. Koushanfar, and M. Potkonjak, “Lightweight Secure PUFs,” in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD ’08. Piscataway, NJ, USA: IEEE Press, 2008, pp. 670–673.

[11] G. E. Suh and S. Devadas, “Physical Unclonable Functions for Device Authentication and Secret Key Generation,” in *Proceedings of the 44th Design Automation Conference, DAC 2007, San Diego, CA, USA, June 4-8, 2007*, 2007, pp. 9–14.

[12] A. Maiti, I. Kim, and P. Schaumont, “A robust physical unclonable function with enhanced challenge-response set,” *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 1, pp. 333–345, Feb 2012.

[13] D. Suzuki and K. Shimizu, “The Glitch PUF: A New Delay-PUF Architecture Exploiting Glitch Shapes,” in *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, 2010, pp. 366–382.

[14] *Spartan-6 FPGA Configuration Logic Block: User Guide*, Xilinx, Inc, 2 2010.

[15] *Cyclone V Device Handbook*, Altera Corporation, 12 2016.

[16] J. H. Anderson, “A PUF design for secure FPGA-based embedded systems,” in *Proceedings of the 15th Asia South Pacific Design Automation Conference, ASP-DAC 2010, Taipei, Taiwan, January 18-21, 2010*, 2010, pp. 1–6.

[17] S. Vyas, N. K. Dumpala, R. Tessier, and D. E. Holcomb, “Improving the efficiency of PUF-based key generation in FPGAs using variation-aware placement,” in *26th International Conference on Field Programmable Logic and Applications, FPL 2016, Lausanne, Switzerland, August 29 - September 2, 2016*, 2016, pp. 1–4.

[18] *Virtex-6 FPGA Configuration Logic Block: User Guide*, Xilinx, Inc, 2 2012.

[19] *7 Series FPGAs Configuration Logic Block: User Guide*, Xilinx, Inc, 9 2016.