On the Problems of Realizing Reliable and Efficient Ring Oscillator PUFs on FPGAs

Alexander Wild
Horst Görtz Institute for IT-Security
Ruhr-Universität Bochum, Germany

Georg T. Becker
Horst Görtz Institute for IT-Security
Ruhr-Universität Bochum, Germany

Tim Güneysu
University of Bremen, Germany

***Abstract*—Physical Unclonable Functions (PUFs) are a promising way to securely generate and store keys by using the inherent process variations of each chip as a source of randomness. One of the most promising PUFs for FPGAs is the Ring-Oscillator (RO) PUF. In this paper we take a closer look at RO PUFs and their open challenges. Starting from a reference design for a Spartan-6 FPGA based on PUFKY, we show how the RO design can be optimized by taking full advantage of the available resources, reducing the RO area by nearly 50%. Furthermore, we analyze the observed structural bias of the RO PUFs and show how the entropy of the RO PUF can be improved by taking the FPGA structure into account when extracting the PUF response bits. However, we also point out a very important problem of FPGA based RO PUFs that has not gained the needed attention: counter failures. We show that the frequency counter is a very crucial element in RO PUF design that itself is very susceptible to process variations. While the counters might work properly on most devices, in some they fail to count correctly. For example, in one experiment only one out of 22 FPGAs failed to count correctly. Our results therefore show that the correct functioning of the frequency counter is not only design-dependent, but also depends highly on process variations, i.e., on the individual FPGA. We argue that solving this issue is non-trivial, since the internal details of the FPGA are secret and hence circuit-level simulations of an FPGA design are not possible. However, the large security implications of such failures make it inevitable that this problem is solved before RO PUFs on FPGAs can be used in practice.**

***Keywords*—Physical Unclonable Functions (PUFs), Ring-Oscillator, FPGA**

I. Introduction

Storing and generating keys securely in embedded devices is a challenging problem, especially if no dedicated secure non-volatile memory is available. Physical Unclonable Functions (PUFs) are a promising way to generate and store cryptographic keys without the need of non-volatile memory. The main idea behind PUFs is that each chip is slightly different due to process variations. PUFs amplify these differences to generate and store cryptographic keys. Many different PUFs have been proposed, e.g., delay based PUFs [1], capacitive coatings [2], butterfly circuits [3], [4], sense-amplifiers [5], write collisions in block memories [6], or initial bit configurations in asynchronous memories [7], [8]. Most of these PUFs have been proposed for ASIC designs and cannot easily be ported to FPGAs. One of the most popular PUFs for FPGAs is the Ring-Oscillator (RO) PUF [1]. It is based on the observation that the frequency of a RO heavily depends on process variations. In the classic RO PUF, two ROs are compared to generate a single bit. By using multiple of these RO pairs a device-specific key can be generated. The disadvantage of this method is that a large number of ROs are needed to generate a key. To reduce this overhead, Yin and Qu proposed not to compare two ROs to generate one bit, but to sort n ROs according to their frequencies to generate up to $log(n!)$ bit-entropy [9]. This greatly reduces the number of needed ROs. However, to build a secure and reliable key generator, a PUF on its own is not enough since the PUF responses are not completely reliable and also do not have full bit-entropy. A so-called Fuzzy Extractor is therefore needed to generate secure and reliable keys from noisy PUF responses [10]. One of the few examples of a fully functional PUF-based key generator was introduced at CHES 2012 [11]. This construct, called PUFKY, uses 848 ROs and 2052 bits of helper data to generate a 128 bit full-entropy key.

While PUFKY shows great promise as a secure key generation architecture, some open questions remain. For one, the area requirements are quite large and hence optimizations are needed. Furthermore, strong structural bias was observed in PUFKY that was countered using a normalization step. In this paper, we therefore made an exhaustive analysis of the PUFKY design and searched for possible optimization. The main findings and contributions of our work can be summarized as follows:

- We introduce an optimized RO design that can save up to 50% of area compared to previous designs.
- We show that in practice RO PUFs are susceptible to counter failures. This phenomenon has very large security implications for RO PUFs but has not been discussed in this context before.
- We study these counter failures and show that they are device as well as environmental dependent. This makes it extremely challenging to verify RO PUFs on FPGAs and raises questions about their reliability.
- We furthermore show that unrelated IP cores can greatly reduce the entropy of the PUF once they are activated. This is even true for IP cores that are placed in opposite corners of the FPGA.

In the following section a quick introduction into the Spartan-6 FPGA structure is given before the reference PUFKY design is explained and analyzed in Section III. Based on these findings, three optimized RO designs are presented in

This work has been partially supported by European Union's Horizon 2020 and Deutsche Forschungsgemeinschaft (SAFEcrypto:644729, Ubicrypt:GRK1817/1). We also like to thank R. Maes, A. Van Herrewege, and I. Verbauwhede for providing their code of PUFKY.

Section IV. This section also includes a very detailed analysis of the counter failure phenomena. Finally, the implications of our results are discussed in the conclusion.

II. Xilinx FPGA Structure

In our work, Spartan-6 FPGAs from Xilinx are used to build a RO-based PUF. In the following, a low level description of this device family is given which will help to understand the structural bias of the PUFKY implementation and the RO design optimizations proposed in Section IV. From a global point of view, the device is organized as a grid of interconnected Configurable Logic Blocks (CLBs). A CLB can be further subdivided into two logical components called slices. There are three different slice types in a Spartan-6, SliceX, SliceM, and SliceL. Each CLB consists of two slice types, where one slice is always a SliceX and the other is either a SliceM or SliceL. A SliceX is made of four Look-up Tables (LUTs) to implement Boolean logic and eight Flip-Flops (FFs) to store intermediate values. A SliceL and SliceM extend the basic functionality with a carry chain and the ability to use the LUTs as dedicated memory elements.

LUTs of a Spartan-6 can be either configured as single 6-to-1 LUT or as two 5-to-1 LUT. Furthermore, it is assumed that a Spartan-6 LUT consists of a multiplexer tree with two output pins called O_5 and O_6. Due to the internal structure of a LUT, both outputs do not share any LUT internal resources and are hence capable in calculating independent outputs based on the same input signals. Figure 1 shows the logical structure of a Spartan-6 LUT [12].

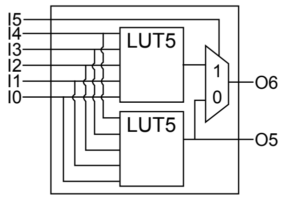


Fig. 1: The logical structure of a Spartan-6 LUT consisting of two 5-to-1 LUTs and two output pins.

The signal routing in an FPGA is performed by routing elements called switch matrices which are interconnected to form a mesh structure. Each CLB is hard wired to a switch matrix. By configuring the switch matrices, the CLBs can be logically interconnected and signal values can be transmitted between CLBs. Such a logical connection, which in general involves multiple switch matrices, is called a route. The internal structure of the switch matrices used in Spartan-6 is not publicly available. Such switches can be realized with tri-state buffers or pass-through transistors and typically a mixture of both techniques can be found in FPGAs [13]. The switch matrices in Spartan-6 are connected in a very structured way. Switch matrices that are directly neighboring are connected by soscalled *single wires*. Switch matrices that are two or four positions apart are connected via *double* and *quad wires* respectively. A switch matrix can also be used to bounce signals from the same CLB, which is called local routing.

For RO PUFs it is important that all ROs are identical and hence have identical routes. To fix the RO structure, Xilinx offers the opportunity to use Hard Macros or directed routing constraints (DIRT) to fix specific routes.

III. Example RO Architecture: PUFKY

As case study for our work, we analyzed the code of PUFKY which was published at CHES 2012 [11]. PUFKY is based on two major components: The RO PUF architecture, which is used as an entropy source, and the post processing that consists of a Fuzzy Extractor for the error correction and entropy extraction. This work only focuses on the RO PUF architecture, hence a brief description of this component is given in the following. The RO PUF architecture (depicted in Figure 2) consists of a RO grid, multiplexer, counter, a normalization step, a Lehmer-Gray encoder and an entropy compression component.

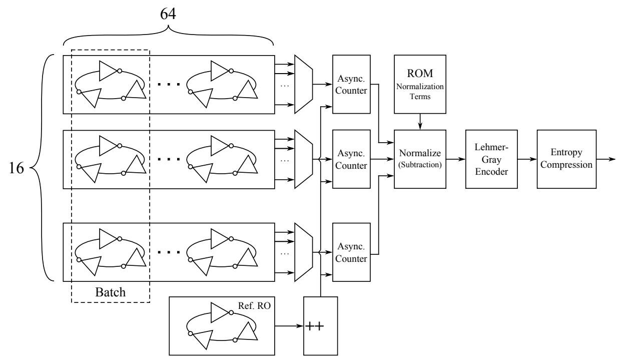


Fig. 2: The RO PUF architecture of PUFKY. Note that this representation swapped the columns and rows of the RO grid.

A ring oscillator is an asynchronous device element realized by an odd number of inverters connected to a cycle. The oscillation frequency of each RO depends on the routing parasitics as well as the transistor transition speeds. Therefore, the RO frequency depends on the design, e.g. the number of inverter stages, but also on process variations. Hence, ROs with identical layouts will not have the same frequency once they are fabricated. This process variation dependency is used in the RO PUF to generate cryptographic keys. To extract the hidden device-specific information from multiple ROs, PUFKY uses a resource-friendly approach based on an idea proposed in [9]. In this concept a grid of ROs (64×16) is generated, with each column sharing a counter. The frequencies of a full row (i.e. a batch of 16 ROs) are used to generate a multi-bit response which is in general the encoded order of the RO frequencies. There are several ways to represent an order of elements in binary format. For PUFKY, a Lehmer-Gray encoder is used which produces a 49 bit output. Please note that the maximum entropy of ordering 16 elements is $log_2(16!) = 44.3$ and therefore the 49 bit response does not have full bit-entropy. To increase the bit-entropy, a simple entropy compression is performed by XORing the bits with the least entropy. The result of this compression is a 40 bit or 42 bit value with a maximum bit-entropy of 0.9795 or 0.9878 respectively. In general, ROs are highly affected by environmental conditions like temperature and voltage. The time period the ROs oscillate is determined by a reference RO. In this way, environmental changes affect the reference RO as well as the PUF ROs which helps to stabilize the counter values.

In PUFKY [14], a very strong structural bias was observed and the authors of PUFKY proposed a normalization step to counter this structural bias. This normalization step is done by simply subtracting a normalization term from the RO frequency counters. In the following we will analyze the source of this structural bias and its implications in more detail. In Figure 7a, the frequency values of the RO grid used by the original PUFKY implementation are shown. One can clearly see the noted strong structural bias that follows a very regular pattern. A closer look at the implementation at a lower level reveals the reason for the bias: A RO in PUFKY is instantiated in half of a slice that is routed using the Xilinx tools. The ROs in a row are made of different slice types that are not routed equally. In particular, these slice types are internally different and have therefore different delay characteristics. Furthermore, an equivalent routing with different slice types is hard to achieve due to the non-public switch matrix structure and the different wire lengths between switch matrix and slice types. The different slice types and the routing are clearly the main sources of this structural bias.

In PUFKY, precomputed normalization terms are subtracted from the RO values to overcome this structural bias. In Figure 7b, the normalized frequencies are plotted and it seems that the normalization gets rid of the structural bias. However, not only the mean values but also the variances of the ROs have a structural bias. Figure 3 shows the variances of the 64 batches after normalization. Please note that structural variance also reduces the entropy of the sorted values. This is due to the fact that ROs with a high variance are more likely to either have very large or very low value while the ROs with low variance will be closer to the mean. But the normalization only changes the mean value of a distribution but not the variance and therefore a different strategy is needed to get rid of the structural variances. Fortunately, there is a rather simple solution to reduce the structural bias as well as structural variance. One simply has to build the batches over the same slice types and the same routing. In such a construction, the structural bias within a batch is minimal and hence the variance of each RO in a batch is very similar as well. For PUFKY this can be done by simply building the batches vertically instead of horizontally.

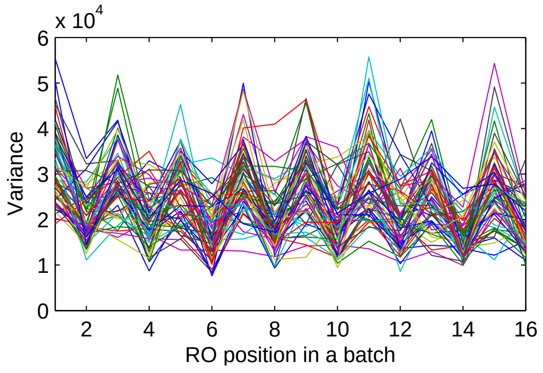


Fig. 3: Grid Variance of PUFKY after normalization.

But not only the RO PUF design itself can lead to bias. It is known that surrounding logic can influence the behavior of the RO [14], [15]. There are different reasons why surrounding logic can impact the ROs. First of all, the surrounding active logic generates local heat so that the switching speed of near inverters is increased. A second reason can be cross talk of RO wires and the logic. A third reason is a slight voltage drop caused by surrounding logic at its transitions. An intuitive way to minimize the effect on the RO is an adequate spacing between the RO and the surrounding logic. However, the size of an adequate spacing, so that the logic has no impact on the RO behavior, and how big this impact can be, has not been analyzed yet. We therefore conducted an experiment to see how much impact an IP core that is placed on the same FPGA but without direct connection to the PUF can have. For this, we implemented a core that consists of XORs with looped output signals which produces a glitch core. Activating the core generates a high local temperature increase and high voltage drop. The glitch core is implemented in an area of 27×27 slices and can be activated with an external signal. The RO grid and the glitch core are placed on the opposite corners of the FPGA. When the glitch core was inactive, the RO PUF showed a similar behavior as the original design. However, once the glitch core was active, the frequencies of the RO changed drastically as can be seen in Figure 7d compared to Figure 7b. When the core is active, the frequencies of the ROs have a very visible slope towards the right. The security implication of this change can be seen in Figure 6c which shows the uniqueness of the PUF design before and after the glitch core was activated. Hence, even though the core was placed as far away as possible from the RO grid, the impact was large enough to greatly reduce the entropy of the RO PUF. This shows that there are no upper bounds on the required security margin and the entire FPGA design (including third party IP cores) has to considered during PUF analysis.

Due to the potential impact of surrounding logic, we inserted a spacial security margin around the RO grid of 6 slices in which no logic was placed. We then evaluated this PUF design using 22 Atlys boards from Digilent which are equipped with a Spartan-6 and expected a similar behavior on all boards. However, we observed something very curious: In one of these 22 boards, a few ROs returned a frequency count value of one. Figure 7c illustrates the RO frequency counts for this specific board. As one can see, the behavior of this board is clearly out of norm and it seems that the counter does not sample the oscillating RO output correctly. A result like this can have large implication on the security of the PUF based key generation. If multiple or all counters fail within a batch, the entropy is greatly reduced and weak keys are generated. This important phenomenon will be discussed in more detail in the next Section.

IV. Optimizes Ring Oscillator Designs

Implementing a large grid of ROs in an efficient and reliable way on FPGAs is a non-trivial task. In this section we introduce an optimized design strategy for implementing ROs on FPGAs with a Spartan-6 like architecture. Three different designs are introduced, each with different optimizations. These designs are then compared and analyzed with respect to the aforementioned counter failures.

The RO structure defined in the reference implementation of PUFKY uses half a slice per RO. However, there is room for improvement, since in such a design a lot of the available resources within a LUT are not used. In particular, a single LUT of a Spartan-6 is composed of two 5-input LUTs and one multiplexer as can be seen in Figure 1. If a '1' is applied to the multiplexer, the two output pins are completely independent and do not share any resources. Therefore, the delay characteristics of the two 5-to-1 LUTs are also independent. Thus, both output pins can be used to construct the RO. This way two RO elements (i.e. inverter, AND gate, or buffer) can be instantiated within a single Spartan-6 LUT. This idea is the main reason why our design achieves an area reduction of up to 50%. In addition to the use of both output pins, we also made optimizations in regard to the routing, which is very important for RO PUFs on FPGAs. A total of three different designs with different optimization goals and tradeoffs are tested.

- `OLocal` places all components of a RO into a single slice and only uses local routing resources, similar to the design proposed by Merli *et al.* [15].
- `OSingle` places half of its components into a slice of horizontally neighbored CLBs. In contrast to `OLocal`, this design uses single wires instead of local wires to form the RO loop. This routing aims to slow down the design with minimal area increase.
- `OSingle_slow` doubles the number of delay elements (LUTs) per RO and uses neighbored CLBs that are connected via single wires. The goal of this design is to drastically reduce the oscillation frequency.

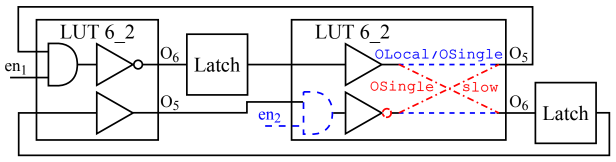


Fig. 4: Visualization of the proposed RO structure using both LUT output pins. Based on the used design, either one (i.e., `OSingle_slow`) or two ROs (i.e., `OLocal` or `OSingle`) are constructed from the given resources.

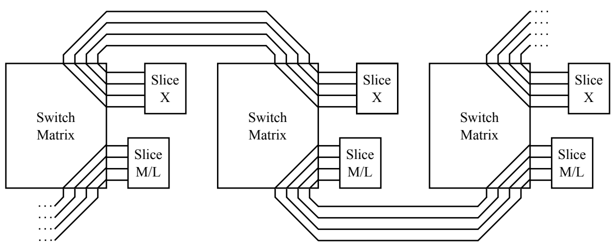


Fig. 5: The chained structure of `OSingle` and `OSingle_slow` to build the RO grid.

The new designs `OLocal` and `OSingle` utilize just two LUTs and two latches to form two ROs, while `OSingle_slow` consumes the same resources to form a single RO. Figure 4 visualizes the instantiation of the ROs. The dedicated routing ensures that there is no overlap between wires from vertically adjacent ROs so that no crosstalk effect is possible and the vertically adjacent ROs can be used in a batch. As noted before, `OLocal` and `OSingle` differ only in the placement of their elements. `OSingle` has moved half of its components to a horizontal neighbored CLB as depicted in Figure 5. The idea behind this design was to use the longer routing and hence additional switching stages in the FPGA to slow down the design without using additional LUTs. For a full resource utilization of a CLB, the single wires in both

TABLE I: A resource and frequency comparison of the given design structures to build a 64×16 RO grid. Note that the given mean frequencies are based on the active time-period of the counters which highly depends on the reference oscillator. The active time of a representative board was measured by an oscilloscope and set to $61\,\mu s$ for the calculations.

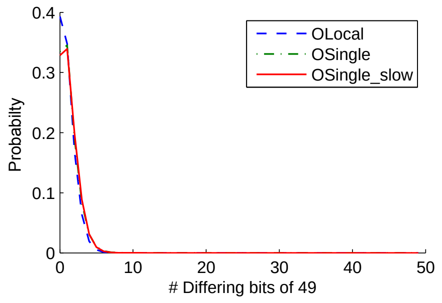
Design	RO / Slice	total Slices	used CLBs	Frequency
`Reference`	2	512	256	306.74 MHz
`OLocal`	4	256	128	433.96 MHz
`OSingle`	4	256	160	348.49 MHz
`OSingle_slow`	2	512	320	184.46 MHz

horizontal directions are used which results in a chain of ROs. Please note that a CLB consists of 2 slices and due to the shifted structure of the `OSingle` design, only one slice of the right and leftmost CLB is used as a RO. Hence, in each chain, two slices remain unused. Furthermore, in a Spartan-6 (XC6SLX45-2CSG324) FPGA a large routing line goes through the grid of CLBs after 2, 5 or 6 CLBs. If these routing lines are unused, this is not a problem as long as only ROs that cross the same routing lines are used to form a batch. However, if there are signals routed through these lines, there might be potential crosstalk effects. To be conservative, we chose to stop each RO chain before such routing lines even though it increased the area overhead by 16 CLBs. Therefore the number of used CLBs is higher for the `OSingle` although the same number of slices as in `OLocal` are used. The resulting resource consumption and RO frequencies of the different designs for a 64×16 RO grid can be found in Table I.

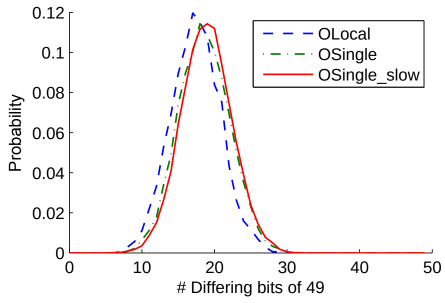
In the following, the different designs are analyzed in regard to their uniqueness, reliability, and counter failures using 22 Digilent Atlys boards equipped with Spartan-6 (XC6SLX45-2CSG324) FPGAs. For each of the three designs three different placements are used to study its impact:

- $P1$ does not have any security margin around the RO grid so that the synthesizer can do its best to place the multiplexers and counters.
- $P2$ has a security margin of 6 slices set around the RO grid.
- $P3$ is the extreme case where the RO grid and counter multiplexer combination are constrained to be placed in opposite corners of the FPGA.

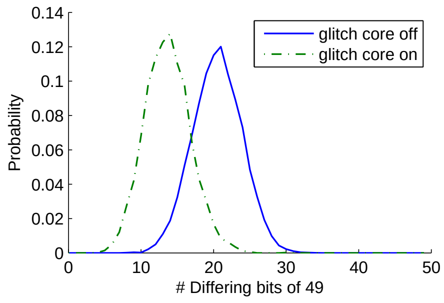
For the proposed RO structures, we calculated the reliability and uniqueness of the Lehmer-Gray output under nominal conditions as done in the PUFKY paper [11]. The analysis results can be seen in Figure 6. Note that we excluded the boards with counter failures from this evaluation. With respect to the reliability and uniqueness, the RO structure seems not to have a significant impact. We assume that the slight variations result from the small number of devices under test. As noted before, we observed that for some designs on some boards the counters failed to count. In this context we define a counter as failed when the number of sampled oscillations, i.e., the counter value after the evaluation phase has not reached a defined threshold. We set this threshold in our experiments to half of the average frequency of each design. Please note that most counter failures resulted in a value of 1 or 2. For each board, design, and placement combination, we measured the RO grid 100 times. The results of this analysis can be found in Table II.



(a) Reliability.



(b) Uniqueness



(c) Uniqueness with Glitch-Core

Fig. 6: Analysis of uniqueness and reliability of the new RO designs for position $P1$ each with 49-bit PUF responses after Lehmer-Gray encoding but without any entropy compression. 22 boards were used with 64 batches each. In (c) the uniqueness results for the glitch core are shown.

TABLE II: Summarized counter failures. At maximum, the counter for all 1024 ROs can fail.

ID	Implementations								
	`OLocal`			`OSingle`			`OSingle_slow`		
	$P1$	$P2$	$P3$	$P1$	$P2$	$P3$	$P1$	$P2$	$P3$
1	0	18-19	951-959	0	0	465-480	0	0	0
2	0	10-16	958-962	0	0	420-433	0-15	0	0
3	0	3	856-867	0	0	235-245	0	0	0
4	0	9-10	951-956	0	0	410-422	0	0	0
5	0	6-7	880-896	0	0	274-291	0	0	0
6	12-15	103-105	1023-1024	0	6-7	914-925	0	0	0
7	0	21-22	969-971	0	0	502-523	1003-1024	0	0
8	0-5	77-81	1019-1020	0	4	851-857	0	0	0
9	11-13	101-105	1024	0	5	912-917	0	0	0
10	6-7	83-88	1021-1022	0	4	873-882	0	0	0
11	1-2	78-82	1017	0	3	831-839	0	0	0
12	0	10-11	946-953	0	0	404-420	0	0	0
13	0	5	901-910	0	0	284-303	0	0	0
14	8	91-94	1022-1023	0	3-5	895-898	0	0	0
15	0	42	1000-1002	0	0	649-663	0-143	0	0
16	15-17	112-113	1024	0	11	943-947	0	0	0
17	5-6	94-96	1023	0	4-5	879-887	0	0	0
18	0	7-8	877-889	0	0	238-256	0	0	0
19	5	104-106	1024	0	7-8	905-911	0	0	0
20	0	2	757-767	0	0	156-163	0	0	0
21	9-12	108-111	1024	0	7	924-929	0	0	0
22	8-9	93-97	1023	0	5-8	900-905	0-320	0	0

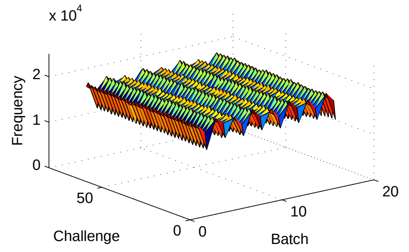
As expected, the `OLocal` implementation running with the highest frequency (433.96 MHz) had the most counter failures of all designs. On some boards some counters fail for position $P1$ while for others all counters work correctly. When the counters are moved further away in $P2$, the number of counter failures increases drastically. When the counters are placed in the opposite corner in $P3$ nearly all counters fail. For the `OSingle` implementation and position $P1$, not a single failure was observed. Hence, the smaller frequency of 348.49 MHz had the intended effect of a reduced counter failure rate. However, the trend that the counter failures increase if you place the counter further away from the RO grid also holds for this design: for $P2$ counter failures appeared, and failures increased for $P3$. In the `OSingle_slow` design the RO frequency is roughly halved compared to the `OSingle` implementation, with a frequency of 184.46 MHz. Therefore, one expects no counter failures for this design. It works as expected on most boards and no counter failures were observed for positions $P2$ and $P3$. However, on four boards we observe counter failures for position $P1$ with a very strange failure behavior. For board 15, multiple rows failed, which means that for certain challenges all counters refuse to work as illustrated in Figure 7f. Additionally, one outlier of the measurement is a counter value that is about 50% faster then the average. But this behavior could not be observed on each run. In other runs, the board worked like expected which is illustrated in Figure 7e.

Our results show that the problem of counter failures clearly depends on the used RO design and their oscillation frequency as well as on the placement of the counters themselves and the surrounding logic. But the most important observation is that process variations also have a very strong impact that should not be underestimated. The best example for this is the `OSingle_slow` design for position $P1$. Since the RO frequency of this design is only half of that of `OSingle` , one expects this design to be robust. However, in four boards nevertheless counters failed in position $P1$, while for position $P3$ no counters failed. The general conclusion to this experiment is that one can increase the reliability of RO PUFs with slower ROs, but it does not give any guarantees on the design functionality. Hence, without novel methods to test and simulate the counter behavior of RO PUFs on FPGAs, only an extensive large scale analysis among various environmental conditions could guarantee its correct functioning.

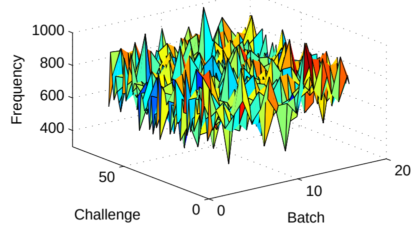
V. Conclusion and Open Research Problems

RO PUFs were proposed over a decade ago and many papers about this topic have been published. However, a lot of open research questions remain that we summarize briefly in this section with a focus on the new problems that arose as part of our conducted experiments. We have separated the open research questions according to the three main PUF properties: reliability, uniqueness, and implementation costs.

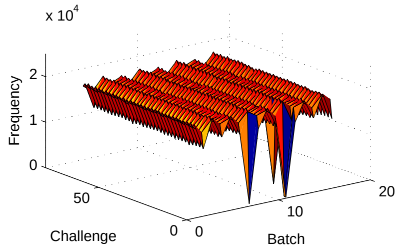
Reliability: Usually, the reliability of a PUF is either analyzed using experiments directly at the bit-level or a model based on the frequency changes of ROs in different environmental conditions is used to determine the reliability via simulations. However, such analyses do not take into consideration the counter failure problem that we observed in our experiments. These counter failures are considerably different compared to changes in the RO frequency and have big security implications. We argue that the problem of counter failures is maybe the hardest research problem to solve for FPGA-based RO PUFs: Usually, circuit-level simulations across a variety of process and environmental corners would be conducted to understand the source of the failures and to determine parameters that guarantee a correct functioning of the counters. However, due to the fact that accurate circuit level models of an FPGA are not publicly available, this is not possible for FPGA-based RO PUFs. Hence, the question of how to build reliable and efficient RO PUFs without counter failures is a very important but difficult open research problem.



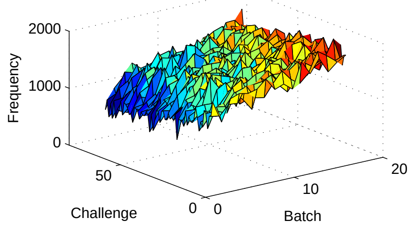
(a) Grid frequency of PUFKY.



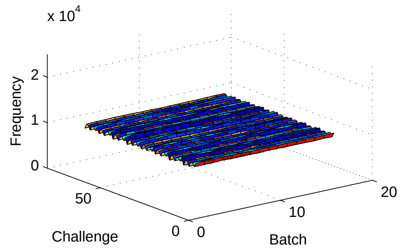
(b) Grid frequency of PUFKY after normalization.



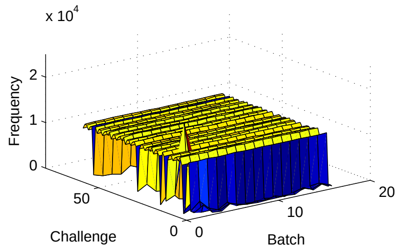
(c) One board failed in counting correctly.



(d) Grid frequency of PUFKY after normalization with noisy core in FPGA corner.



(e) Grid frequency of `OSingle_slow` $P1$ on board 15 run 1.



(f) Grid frequency of `OSingle_slow` $P1$ on board 15 run 2.

Fig. 7: Frequencies of RO grid.

Uniqueness: In this paper we analyzed the dominating source of structural bias and show how the impact of this structural bias can be reduced greatly by careful routing as well as sorting of the ROs. It is also important to note that our analysis shows that RO PUFs not only have a structural bias in the mean, but also in the variance. This has not been considered in this context before but it clearly needs to be taken into account when determining the entropy of a RO PUF design such as PUFKY. Furthermore, our results with the glitch core once more show that other IP cores can have great impact on the entropy of the RO PUF. How to determine this impact and how to make sure that other IP cores do not reduce the entropy of the RO PUF is an interesting and important open research question. One problem in analyzing the entropy of a PUF design is that to get meaningful data, a very large-scale analysis with many boards is needed. Even the largest conducted experiment, which consisted of 100 boards [14], is not really enough to accurately determine the entropy. In general, how to determine the exact entropy of a RO PUF response is another very important open research problem.

Implementation Costs: As PUFKY showed, to derive a cryptographic key from a RO PUF, a very large grid of ROs is needed. Hence, techniques to reduce the RO area are very interesting. In this paper we showed how the area of ROs can be greatly reduced by making full use of the available resources on a Spartan-6. Similar optimizations are likely possible for other FPGA architectures as well. However, an area-optimized RO design that suffers from counter failures is not helpful. The underlying research problem is therefore not to construct small RO PUFs, but rather to construct small RO PUFs that do not suffer from counter failures.

But the first step should be an in-depth analysis of the counter failure problem. Following this, new designs and design strategies need to be developed that can guarantee the proper functioning of the counters among all process corners and environmental conditions.

References

[1] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas. Silicon Physical Random Functions. In *CCS 2002*, pages 148–160. ACM, 2002.

[2] P. Tuyls, G. Schrijen, B. Skoric, J. van Geloven, N. Verhaegh, and R. Wolters. Read-Proof Hardware from Protective Coatings. In *CHES 2006*, volume 4249 of *LNCS*, pages 369–383. Springer, 2006.

[3] R. Maes, P. Tuyls, and I. Verbauwhede. Intrinsic PUFs from Flip-Flops on Reconfigurable Devices. In *WISSec 2008*, 2008.

[4] S.S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls. Extended Cbstract: The Butterfly PUF Protecting IP on every FPGA. In *HOST 2008*, pages 67–70. IEEE, 2008.

[5] M. Bhargava, C. Cakir, and K. Mai. Attack Resistant Sense Amplifier Based PUFs (SA-PUF) with Deterministic and Controllable Reliability of PUF Responses. In *HOST 2010*, pages 106 –111. IEEE, 2010.

[6] T. Güneysu. Using Data Contention in Dual-ported Memories for Security Applications. *Signal Processing Systems*, 67(1):15–29, 2012.

[7] J. Guajardo, S.S. Kumar, G.-J. Schrijen, and P. Tuyls. FPGA Intrinsic PUFs and Their Use for IP Protection. In *CHES 2007*, volume 4727 of *LNCS*, pages 63–80. Springer, 2007.

[8] J. Guajardo, S.S. Kumar, G.-J. Schrijen, and P. Tuyls. Physical Unclonable Functions and Public-Key Crypto for FPGA IP Protection. In *FPL 2007*, pages 189–195. IEEE, 2007.

[9] C.-E. Yin and G. Qu. LISA: Maximizing RO PUF's Secret Extraction. In *HOST 2010*, pages 100–105. IEEE, 2010.

[10] Y. Dodis, L. Reyzin, and A. Smith. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. In *Eurocrypt 2004*, volume 3027 of *LNCS*, pages 523–540. Springer, 2004.

[11] R. Maes, A. Van Herrewege, and I. Verbauwhede. PUFKY: A Fully Functional PUF-Based Cryptographic Key Generator. In *CHES 2012*, volume 7428 of *LNCS*, pages 302–319. Springer, 2012.

[12] Xilinx. *Spartan-6 Libraries Guide for HDL Designs*, October 2013.

[13] I. Kuon, R. Tessier, and J. Rose. FPGA Architecture: Survey and Challenges. *Foundations and Trends in Electronic Design Automation*, 2(2):135–253, 2007.

[14] A. Maiti, J. Casarona, L McHale, and P. Schaumont. A Large Scale Characterization of RO-PUF. In *HOST 2010*, pages 94–99. IEEE, 2010.

[15] D. Merli, F. Stumpf, and C. Eckert. Improving the Quality of Ring Oscillator PUFs on FPGAs. In *WESS 2010*. ACM, 2010.