Cloud FPGA Security with RO-Based Primitives

Shanquan Tian, Andrew Krzywosz, Ilias Giechaskiel, and Jakub Szefer

19th International Conference on Field-Programmable Technology (FPT),
Maui, Hawaii, United States of America, Dec. 2020

DOI:10.1109/ICFPT51103.2020.00029

Cloud FPGA Security with RO-Based Primitives

Shanquan Tian
Yale University
New Haven, CT, USA
shanquan.tian@yale.edu

Andrew Krzywosz
Yale University
New Haven, CT, USA
andrew.krzywosz@yale.edu

Ilias Giechaskiel
Independent Researcher
London, United Kingdom
ilias@giechaskiel.com

Jakub Szefer
Yale University
New Haven, CT, USA
jakub.szefer@yale.edu

***Abstract*—Physical Unclonable Functions (PUFs) and True Random Number Generators (TRNGs) are common primitives that can increase the security of user logic on FPGAs. They are typically constructed using Ring Oscillators (ROs). However, PUF and TRNG primitives are not currently available on Cloud FPGAs as some commercial Cloud FPGA providers prohibit deploying ROs implemented using Lookup Tables (LUTs). To aid in bringing RO-based PUFs and TRNGs to commercial Cloud FPGAs, this work implements and evaluates PUFs and TRNGs built using ROs that incorporate latches and flip-flops. The primitives are tested on Amazon's commercial F1 Cloud FPGAs. The designs are the first constructive uses of ROs in Cloud FPGAs and are available under an open-source license.**

***Index Terms*—Cloud FPGAs, Ring Oscillators, PUFs, TRNGs**

I. Introduction

With the emergence of Cloud FPGA offerings, users can easily deploy their FPGA-based designs to the cloud on demand. Unfortunately, deploying FPGA-based designs in the cloud poses new challenges. Thus, most existing Cloud FPGA research has primarily focused on attacks, e.g., [1]–[3], and countermeasures to these attacks, e.g., [5], [6].

Meanwhile, this paper addresses an orthogonal issue: providing a good source of randomness for cryptography, and allowing users to fingerprint FPGAs for authentication and reliability purposes. To this end, this paper introduces and evaluates Physical Unclonable Function (PUF) and True Random Number Generator (TRNG) primitives based on novel types of Ring Oscillators (ROs) for use in Cloud FPGAs. As some commercial Cloud FPGA providers block ROs implemented using Lookup Tables (LUTs), this work instead leverages ROs consisting of latches and flip-flops [1], [9].

The presented PUFs achieve good inter- and intra-device Hamming Distances (HDs), and can reliably distinguish between dozens of different Cloud FPGA chips. Moreover, our PUF fingerprints are accurate even when the temperature of the FPGA changes. Our designs only use few resources, and work well on several locations of the FPGA die.

The presented TRNGs pass all the National Institute of Standards and Technology (NIST) randomness tests. They can produce random bits for use both within the FPGA and by the host software. The bandwidth is limited only by the number of ROs used or the current PCIe interface implementation.

We have made the code used in this work available under an open-source license at https://caslab.csl.yale.edu/code/cloud-ro-primitives/.

II. Background

This section provides the relevant background on ROs (Section II-A), PUFs (Section II-B), and TRNGs (Section II-C).

A. Ring Oscillators

Traditional ROs implemented using LUTs are detected and prohibited by some cloud providers, such as Amazon Web Services (AWS), preventing LUT-based RO PUFs and TRNGs from being possible on the cloud. However, alternative ROs replace one of the LUT stages with a latch or a flip-flop [1], [9], as shown in Figure 1. We use these types of ROs for the first time to develop PUFs and TRNGs in Cloud FPGAs.

B. Physical Unclonable Functions

RO-based PUFs exploit the RO frequency sensitivity on random manufacturing variations to create a unique fingerprint of otherwise-identical chips [10]. These fingerprints can be used for IP protection and reliability purposes, in the absence of digital IDs that are currently inaccessible in Cloud FPGAs.

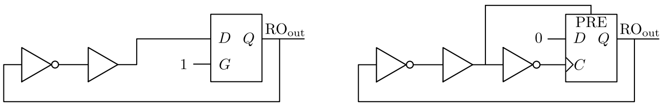
More precisely, an RO PUF consists of n ROs, whose frequencies are sampled and compared through counters. Out of the possible $n \cdot (n-1)/2$ pairs, typically only $m = O(n)$ are used to create an m-bit fingerprint. The number of bits in which two PUF responses differ (*Hamming Distance*, or HD) should be small when sampled from the same FPGA (intra-device HD), and large for different FPGAs (inter-device HD).

So far, almost all RO-based PUFs have been implemented with LUTs, and large-scale analyses of RO PUFs have been very limited [4], [16]. However, Wild *et al.* have implemented latch-based RO PUFs as a way of reducing resource utilization [15], [16]. Nevertheless, their RO design is different from the one used in this work, and they only evaluated it on local FPGAs, not on Cloud FPGAs. By contrast, we collect PUF data on dozens of FPGAs, implement the first RO-based PUFs on the cloud, and evaluate the first-ever flip-flop-based RO PUF, whether locally or on the cloud.

C. True Random Number Generators

TRNGs leverage unique device characteristics to produce unpredictable random sequences of bits for key generation and encryption, among others, without having to rely on potentially untrusted software for their source of randomness. Conversely, TRNGs on FPGAs can alternatively be used as a trusted hardware-based entropy generator to supplement the randomness that is available to host software.

This research was supported by NSF grant 1901901.



(a) Latch-based RO (b) Flip-flop-based RO

Fig. 1. The (a) latch-based and (b) flip-flop-based ring oscillators used in this work. Figures adapted from [1].

RO-based TRNGs exploit the jitter behavior of signal transitions in the analog output of an RO [11]. The uncertainty in the exact timing of a rising or falling edge is influenced by environmental and manufacturing factors, making samples within this transition period good candidates for generating random numbers. Due to the difficulties associated with directly sampling an RO within its jitter period, an XOR tree of n ROs is sampled at a fixed rate [11]. Unlike prior work which has exclusively used LUT-based ROs, our TRNGs leverage latch-based and flip-flop-based ROs for the first time.

III. RO PUF Design

Our PUF design introduces the idea of *redundant* ROs, which allows us to evaluate the quality of each RO pair by pre-testing the PUF design on a smaller number of Cloud FPGA instances, and discarding "bad" RO pairs, which decrease the effectiveness (uniqueness and reliability) of the PUF response.

Our PUFs consist of 512 ROs, with each RO only used in one comparison pair. The 256 resulting pairs then generate 256 bits for pre-testing with 40 FPGA instances. After the pre-testing phase, 128 "good" bits (i.e., RO pairs) that generate high entropy are chosen to be included in the final PUF responses, with the other 128 bits ignored entirely. As we show in Section V-A when testing with 160 instances, the selection of high-entropy bits decreases the intra-device HD of the PUF response, while increasing the inter-device HD. In other words, our approach significantly improves the reliability and uniqueness of the RO PUFs by finding the RO pairs that are stable within an FPGA, but differ among FPGAs.

Figure 2 shows our *RO PUF* module, which consists of $n = 512$ ROs with two multiplexers (MUXes) and two RO counters for comparing the RO pairs. The $m = n/2 = 256$ RO pairs use adjacent ROs to eliminate systematic variations [7], [8], and each RO is only used once. The RO outputs drive the counters, which are sampled on a system clock timer set by the software. To minimize noise, when an RO pair is sampled, the other ROs are disabled. In pre-processing, $m = 256$ bits are sent back each time, and are used to identify the 128 good RO pairs. In later experiments with more (and different) FPGAs, the same 128 good RO pairs that were identified in the pre-processing stage are used to generate the PUF fingerprints.

To evaluate the effect of temperature on the quality of the proposed RO PUFs, we further add an *RO Sensor and RO Heaters* module to our design. This module allows us to increase and observe the FPGA temperature, collecting PUF fingerprints at different thermal states of the fabric, proving the stability of our design across environmental conditions that might arise in a data center. The RO sensor and heaters are also used in testing the TRNG design, which is discussed separately in the next section.

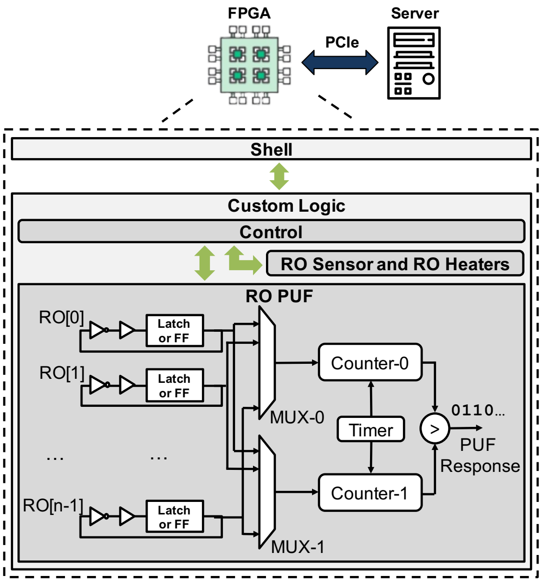


Fig. 2. Diagram of the RO PUF on AWS F1 instances. The controller communicates with the user's virtual machine running on the server over PCIe via the shell. The PUF consists of latch-based or flip-flop-based ROs, with each RO pair's response processed through multiplexers, counters, and a comparator. The RO-based sensor and heaters test the temperature stability of the PUF.

IV. RO TRNG Design

As explained in Section II-C, to extract entropy from phase jitter, our TRNG design (Figure 3) uses an XOR tree merging outputs from multiple ROs [11]. Our XOR tree combines the output of 16 latch-based or flip-flop-based ROs to produce a bit, and we use two independent XOR trees to capture 2 simultaneous bits of data at the FPGA clock rate of 125 MHz. The bits are passed into a 32-bit collector, which, upon becoming full, passes the value into a first-in, last-out (FILO) buffer, and resets itself to collect another set of 32 bits. Bits are generated at an effective rate of 250 MHz. This rate can be increased by adding more independent XOR trees, or increasing the sampling frequency, i.e., the FPGA clock speed.

Assuming the TRNG outputs are independent and identically distributed (i.i.d.), a *von Neumann extractor* can remove the TRNG bias [14]. The extractor works by splitting the sequence of TRNG outputs into consecutive non-overlapping bit pairs. If the two bits are identical (11 or 00), the extractor does not output a new random bit. Otherwise, it outputs the first of the two bits, i.e., a 10 outputs a 1 and a 01 outputs a 0. Denoting the probability that the TRNG outputs a 1 as p, the expected number of bits E that the extractor uses to produce an output satisfies the equation

$$E = 2 \cdot (2 \cdot p \cdot (1-p)) + (E+2) \cdot \left(p^2 + (1-p)^2\right)$$

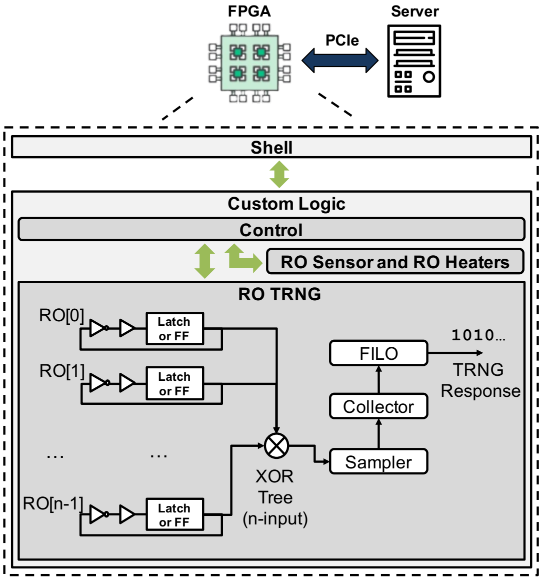


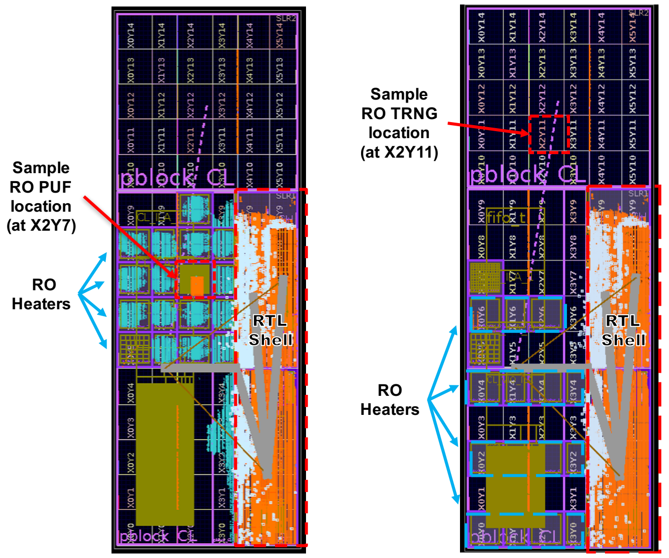
Fig. 3. Diagram of the RO TRNG on AWS F1 instances. The TRNG design exploits RO jitter through an XOR tree, which is sampled and aggregated into 32-bit values that are stored in a FILO buffer. The control logic and RO sensor and heaters are similar to the RO PUF design of Figure 2.

i.e., $E = 1/(p(1-p))$. When p is the ideal 50%, the yield $1/E = p(1-p)$ is 25%. For our RO TRNG, $p \approx 48\%$, resulting in a yield of 24.96%. As we further improve the randomness of the outputs by XORing four independent and debiased bitstreams together, the final bitstream yield of our TRNG is 6.24%, for a bandwidth of approximately 15.6 Mbps in its current implementation with two independent XOR trees sampled at 125 MHz.

V. Evaluation

This section evaluates our PUF (Section V-A) and TRNG (Section V-B) designs on Cloud FPGAs. All experiments are performed on AWS EC2 F1 instances in the North Virginia region, which use Xilinx Virtex UltraScale+ VU9P chips. Amazon's "shell" uses clock regions `X4Y0:X5Y9`, while by default we place the RO PUF on region `X2Y7` and the RO TRNG on `X2Y11` (on separate bitstreams), as shown in the sample floorplan of Figure 4. Sections V-A4 and V-B1 vary these locations for the PUF and TRNG respectively to show that the quality of the results does not fundamentally depend on the specific clock regions chosen.

Note that UltraScale+ FPGAs contain 8 Lookup Tables (LUTs) and 16 registers in each Configurable Logic Block (CLB). As a result, it is possible to fit four flip-flop- or latch-based ROs per CLB, each using 2 LUTs and 1 register. By contrast, only two traditional 3-stage LUT-based ROs can fully fit within each CLB, showing that our primitives result in better resource utilization compared to other designs.


Sample RO PUF location (at X2Y7)
RO Heaters
RTL Shell
pblock CL
Sample RO TRNG location (at X2Y11)
RO Heaters
RTL Shell
pblock CL

(a) Floorplan of RO PUF

(b) Floorplan of RO TRNG

Fig. 4. Sample floorplans of our (a) PUF and (b) TRNG designs for Cloud FPGAs, as realized on AWS F1 instances.

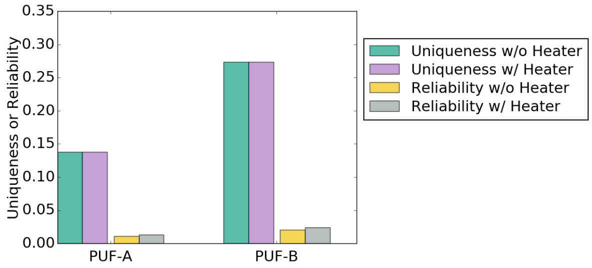


Fig. 5. Ignoring low-entropy bits (*PUF-B*) improves *Uniqueness* and maintains *Reliability* compared to the baseline *PUF-A*. Temperature increases due to RO heaters do not affect the PUF responses.

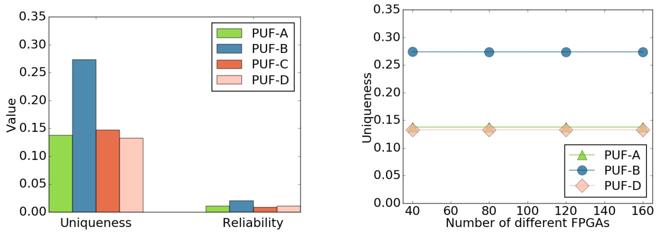
A. *Evaluation of RO PUF Design on AWS*

Our PUF generates 128-bit fingerprints out of 256 ROs pairs. We validate our design by calculating the *Uniqueness* (the average inter-device Hamming Distance (HD) over responses from different FPGAs), and the *Reliability* (the average intra-device HD over repeated measurements from the same FPGA). Ideal PUFs have *Uniqueness* values of 0.5, as their responses behave randomly. *Reliability* values are close to 0, as few bits differ when re-querying the PUF.

1) Improving Uniqueness while Maintaining Reliability: Previous RO PUF designs usually compare $m = n/2$ RO pairs out of n ROs and output $n/2$ bits, but our design eliminates low-entropy bits and instead produces $n/4 = 128$ bits. To evaluate the quality of our PUF design, we compare the *Uniqueness* and the *Reliability* of the baseline implementation (*PUF-A*), which uses all $n/2 = 256$ RO pair comparisons, to our improved 128-bit PUF, *PUF-B*. Figure 5 shows that the *Uniqueness* of *PUF-B* increases to ≈0.25 from ≈0.13 in *PUF-A*, while maintaining the *Reliability* at almost the same value. In addition, Figure 5 shows that the RO heaters do not

TABLE I
DIFFERENT RO PUF IMPLEMENTATIONS.

Design	**Explanation**	# ROs	# Bits
PUF-A	Baseline PUF using 512 RO pairs	512	256
PUF-B	As *A*, but ignores low-entropy ROs	512	128
PUF-C	As *B*, but physically removes ROs	256	128
PUF-D	As *B*, but ignores pairs $[1, 3, 5, \ldots]$	512	128



(a) *PUF-B* performs the best by increasing *Uniqueness*

(b) Uniqueness remains the same when testing on additional FPGAs

Fig. 6. Uniqueness and reliability for the four setups of Table I. (a) The best uniqueness is achieved when ignoring but not removing low-entropy bits (*PUF-B*). (b) Uniqueness remains constant when testing on additional FPGAs that were not used in pre-testing.

influence the *Uniqueness* and *Reliability* much, indicating that the RO PUFs are stable at different temperatures.

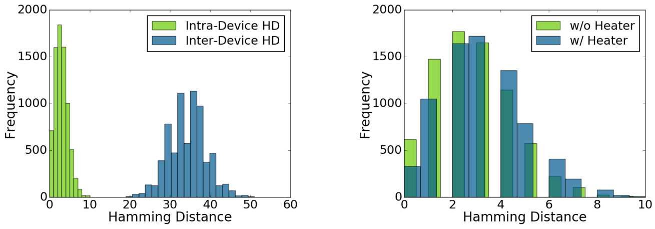
2) Choosing Low Entropy Bits: To show the benefits of the novel idea to ignore but not remove low-entropy RO pairs, we implement two additional types of PUFs: *PUF-C* which physically removes the "bad" ROs from the floorplan, and *PUF-D*, which instead ignores randomly selected RO pairs. Table I contains a summary of the four PUF designs. PUFs *A, B, D* utilize 669 slices, while *PUF-C* uses 359 slices.

Figure 6a shows that although the *Reliability* remains almost the same for all four PUF designs, the *Uniqueness* of *PUF-B* is much higher than that of the remaining three PUFs. In particular, *PUF-C* suggests that re-routed logic will still influence the entropy of the remaining RO pairs, i.e., the "good" bits from *PUF-B* no longer remain good in *PUF-C*.

Moreover, the locations of "good" bits remain stable across many FPGAs. Although 40 FPGAs were used in pre-testing for *PUF-B*, Figure 6b shows that the *Uniqueness* of *PUF-B* stays stable when applying the PUF with the same "good" bits to 160 different instances.

3) PUF Responses for Different FPGAs: By storing PUF responses in a database, users are able to infer (based on the HD) whether a given FPGA has been used before or not: as Figure 7 shows, the intra-device and inter-device HDs of $N = 40$ new F1 FPGA instances (measured 20 times) are clearly separated. Even if the RO heaters are turned on, the intra-device HD stays almost the same, and always under 10. By contrast, inter-device HD ranges from about 20–50, so a threshold of 15 can be used to separate PUF responses from the same device to those from different devices.

4) Different RO Types and Locations: As mentioned in Section I, we implement flip-flop-based RO PUFs in addition to latch-based RO PUFs. Figure 8 compares the two RO types on three different locations of the FPGA device. Both RO types work well on all three locations, but the *Uniqueness* of latch-based PUFs is generally higher than those with flip-flops.



(a) Intra- and inter-device HD

(b) Temperature effect on intra-HD

Fig. 7. *PUF-B* Hamming Distances (a) intra- and inter-device, and (b) with or without the RO heaters enabled.

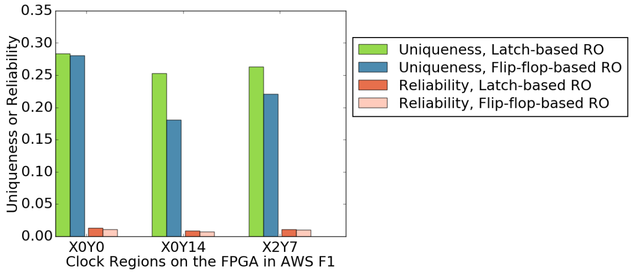


Fig. 8. The *Uniqueness* and *Reliability* of *PUF-B* using two types of ROs at three locations on the FPGA.

B. Evaluation of RO TRNG Design on AWS

To assess the proposed RO TRNG, the generated bit sequences are tested against the NIST SP 800-22 test suite, which leverages several common statistical tests for evaluating random number generators (though such tests cannot prove "true" randomness). Table II shows the test results for $N = 500$ sequences and $L = 400{,}000$ bits per sequence. Our TRNG passes all tests, which require a success rate of $488/500$ or $209/216$. As a result, our design is capable of producing sufficiently random sequences at a 96% confidence interval. Because some tests run multiple times, we have only included the minimum pass rate and p-value for each test, which was generated through a χ^2 test on the p-values obtained from individual sequences for each statistical test.

1) RO Types, Location, and Thermal Impact on TRNG Quality: We tested the RO TRNG design using both latch-based and flip-flop-based ROs on different FPGA locations, and with the RO heaters enabled to evaluate the impact of thermal changes on the TRNG quality. There was no statistically significant difference among the RO types (latch-based or flip-flop-based), location (different SLR dies), or temperature, with the results almost identical to Table II.

2) TRNG Bandwidth: As explained in Section IV, the TRNG has a yield of 6.25%, for a bandwidth of $15.6\,\mathrm{Mbps}$. The design utilizes 58 slices on the FPGA, which translates to a raw bandwidth on the FPGA of $7.8\,\mathrm{Mbps}$ per XOR tree,

TABLE II
NIST SP 800-22 RANDOMNESS TESTS.
† MINIMUM OUT OF ALL RUNS SHOWN. ‡ MINIMUM OCCURS IN 1/148 RUNS, AND THE OVERALL TEST PASSES.

Statistical Test	Result	p-value
Frequency	494/500	0.5955
BlockFrequency	498/500	0.0711
Runs	494/500	0.7981
LongestRuns	496/500	0.2702
MatrixRank	494/500	0.6038
FFT	494/500	0.3070
NonOverlappingTemplate†‡	487/500	0.7830
OverlappingTemplate	493/500	0.9140
MaurerUniversal	492/500	0.4154
LinearComplexity	496/500	0.8343
Serial1	497/500	0.1296
Serial2	494/500	0.5914
ApproximateEntropy	494/500	0.2649
CumulativeSums1	495/500	0.1815
CumulativeSums2	495/500	0.5261
RandomExcursions†	211/216	0.2299
RandomExcursionsVariant†	212/216	0.5729

or $0.27\,\mathrm{Mbps}$ per CLB slice. Bandwidth can be easily further improved by introducing additional XOR tree instances within the design, using more ROs to improve the randomness of each TRNG, or sampling at a higher frequency.

VI. Related Work

Most security-related research on Cloud FPGAs has focused on attacks such as crosstalk between long routing wires [1], covert channels between different physical dies of the same FPGA chip [2], and side-channel leakage within the same FPGA die [3]. Besides attacks in the multi-tenant setting, temperature-based temporal covert channels between consecutive users of the same FPGA are also possible [12].

Such attacks assume that adversaries can fingerprint the FPGA. Only one other work has introduced Cloud FPGA fingerprints through DRAM PUFs [13]. There are three drawbacks to that approach: first, it depends on loading and unloading two AWS FPGA Images (AFIs) to cause DRAM cells to decay. As a result, it is time-consuming and not runtime-accessible. Second, it fingerprints the external DRAM chips (which could be replaced), and is not intrinsic to the FPGA itself. And, third, it exploits a data remanence feature that Amazon could patch at any moment by scrubbing memory for bitstreams without a memory controller. By contrast, our work ties the PUF directly to the FPGA hardware, uses minimal resources, and can be integrated into any FPGA design.

VII. Conclusion

This paper implemented the first PUFs and TRNGs using latch-based and flip-flop-based ROs on Cloud FPGAs, providing the first comprehensive large-scale analysis of any PUF or TRNG primitives on the cloud.

The RO PUFs proposed differentiated between dozens of separate Cloud FPGA instances, and provided a reliable way to identify FPGAs in the absence of digital IDs, which are currently inaccessible in today's Cloud FPGAs. Moreover, they remained robust even when the FPGAs were exposed to temperature variations in the data-center environment. Finally, the PUF designs benefited from a novel insight in their construction which ignored (but did not physically remove) unstable RO pairs from the PUF response.

The RO TRNG was similarly able to pass all of the NIST randomness tests, delivering high-bandwidth random bits for use in FPGA key generation and encryption applications, but also as a hardware-based entropy generator for use by host software. By open-sourcing our code, we hope to encourage further research into positive uses of ROs on Cloud FPGAs.

Acknowledgment

We would like to thank Obi Nnorom, Cesar Rodriguez, Michael McNamara and Wenjie Xiong for discussions and contributions to early versions of the code.

References

[1] I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "Measuring long wire leakage with ring oscillators in cloud FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2019.

[2] ——, "Reading between the dies: Cross-SLR covert channels on multi-tenant cloud FPGAs," in *IEEE International Conference on Computer Design (ICCD)*, 2019.

[3] O. Glamocanin, L. Coulon, F. Regazzoni, and M. Stojilović, "Are cloud FPGAs really vulnerable to power analysis attacks?" in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020.

[4] R. Hesselbarth, F. Wilde, C. Gu, and N. Hanley, "Large scale RO PUF analysis over slice type, evaluation time and temperature on 28nm Xilinx FPGAs," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2018.

[5] J. Krautter, D. R. E. Gnad, and M. B. Tahoori, "Mitigating electrical-level attacks towards secure multi-tenant FPGAs in the cloud," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 3, pp. 1–26, Sep. 2019.

[6] T. M. La, K. Matas, N. Grunchevski, K. D. Pham, and D. Koch, "FPGADefender: Malicious self-oscillator scanning for Xilinx UltraScale+ FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 13, no. 3, pp. 1–31, Sep. 2020.

[7] A. Maiti and P. Schaumont, "Improved ring oscillator PUF: An FPGA-friendly secure primitive," *Journal of Cryptology*, vol. 24, no. 2, pp. 375–397, Apr. 2011.

[8] D. Merli, F. Stumpf, and C. Eckert, "Improving the quality of ring oscillator PUFs on FPGAs," in *Workshop on Embedded Systems Security (WESS)*, 2010.

[9] T. Sugawara, K. Sakiyama, S. Nashimoto, D. Suzuki, and T. Nagatsuka, "Oscillator without a combinatorial loop and its threat to FPGA in data centre," *Electronics Letters*, vol. 15, no. 11, pp. 640–642, May 2019.

[10] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *ACM/IEEE Design Automation Conference (DAC)*, 2007.

[11] B. Sunar, W. J. Martin, and D. R. Stinson, "A provably secure true random number generator with built-in tolerance to active attacks," *IEEE Transactions on Computers (TC)*, vol. 56, no. 1, pp. 109–119, Jan. 2007.

[12] S. Tian and J. Szefer, "Temporal thermal covert channels in cloud FPGAs," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019.

[13] S. Tian, W. Xiong, I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "Fingerprinting cloud FPGA infrastructures," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2020.

[14] S. P. Vadhan, "Pseudorandomness," *Foundations and Trends in Theoretical Computer Science*, vol. 7, no. 1–3, pp. 1–336, 2012.

[15] A. Wild, G. T. Becker, and T. Güneysu, "On the problems of realizing reliable and efficient ring oscillator PUFs on FPGAs," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2016.

[16] ——, "A fair and comprehensive large-scale analysis of oscillation-based PUFs for FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2017.