Overview of PUFs and Emerging Chaotic Structures

David G. Tindall, Aubrey N. Beal, Tommy Morris
Dept. of Electrical & Computer Engineering
The University of Alabama in Huntsville
Huntsville, AL, USA
dt0085@uah.edu, aubrey.beal@uah.edu, tommy.morris@uah.edu

2024 IEEE Physical Assurance and Inspection of Electronics (PAINE) | 979-8-3315-4225-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/PAINE62042.2024.10792741

***Abstract*—We present an abbreviated survey of common Physically Unclonable Function (PUF) structures, and highlight two emerging chaotic structures, targeted towards application-driven practitioners who need reliable generation of device specific values. Interestingly, chaotic PUF structures output entropic values over successive iterations. We also outline three distinct PUF use cases and discuss well-suited electronic structures for each case. We discuss security in limited cases and summarize suitable implementation targets, aspects affecting reliability, and notional generation rates. Altogether, this work summarizes key concepts about PUF structures in an application-driven context to aid both designers and users of these devices in the broader security community.**

***Index Terms*—Physically Unclonable Functions (PUFs), Chaotic circuits, Hardware Security, Entropy Sources.**

I. Introduction

It has been more than twenty years since Gassend et al. presented the idea of an in-circuit physical random function [1], which they coined a PUF and now is known as a physically uncloneable function. This approach differed from prior work using physically-separate three-dimensional micro-structures excited by coherent radiation. They noted that statistical manufacturing differences in integrated circuits (ICs) were well-documented, and though viewed as a problem, could also be viewed as an opportunity - *a means to generate values useful for cryptographic operations*. Later, the concept of a controlled PUF that improved security described additional elements needed for real applications [2]. Importantly, Gassend combined both concepts and identified the need to ensure the reliable regeneration of PUF values [3]. The first reference in both papers is Ross Anderson's "Security Engineering: A Guide to Building Dependable Distributed Systems" [4] which demonstrates that these initial researchers approached the considered security applications from a system view.

Researchers and designers spent tremendous time and energy on PUFs over subsequent years. The community still struggles to produce circuits that are 1) reliable over temperature and age and 2) maintain the secrecy of dynamically generated PUF values for various use cases. This difficulty arises partly due to the continuous discovery of new attacks to extract PUF values. Additionally, researchers have shown the importance of characterizing PUF responses in terms of min-entropy; to accurately calculate the guess-ability of PUF values. In recent years, researchers have proposed new PUF structures based on chaotic dynamics. Though reliability of these PUF designs should be questioned, they raise interesting areas for future research, especially because these structures have the ability to iteratively generate entropy.

Our contribution is to compare and contrast in-circuit PUF structures, including emerging structures based on chaos. In Section II, we provide insights about PUF use cases and discuss how different PUF structures are particularly applicable to specific use cases. Next, in Section III, we highlight and compare several well-known PUF structures including arbiters, ring oscillators, static random access memory (SRAM) cells, butterfly architectures, and emerging chaotic structures. Section IV introduces new terminology for entropic sources, inspired by differences in chaotic PUFs, which generate entropic values over successive iterations. Finally, we propose future work in Section V and conclude in Section VI.

II. Use Cases

Before different PUF structures are introduced, it is important for practitioners to understand what security function an employed PUF is targeting. We consider three distinct use cases to illustrate the suitability of different structures for various applications. In this section three use cases are described: Device Authentication, Intellectual Property Protection, and Random Number Generation.

A. Device Authentication

The first silicon-based PUF in literature focused on the device authentication use case [2] [3]. Figure 1a shows a model which takes an input bitstring, known as a challenge, and outputs a device-unique value, known as a response. This model can provide a mechanism to authenticate a remote device over an insecure communication channel. Prior to deploying the device, a table of challenge-response pairs are stored on a secure server for use over the period of deployment. To verify that a device is authentic, a user (or server) will present a challenge value to the remote device. This device in turn will regenerate a unique response value that the user (or server) knows. The authentication operation can be performed over an untrusted communication channel as long as the Challenge Response Pairs (CRPs) are used only once. These early researchers proposed a Controlled PUF [2] for use in device authentication which introduced additional

Distribution Statement A. Approved for public release: distribution is unlimited PR Number: PR2024206

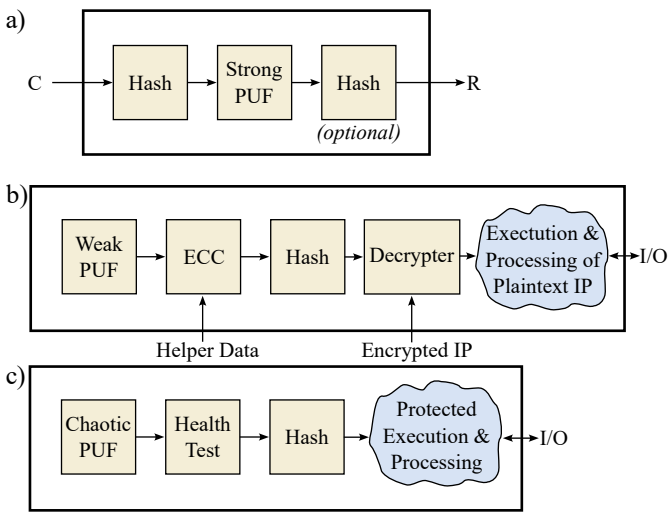

Fig. 1. a) Functional block diagram of a device authentication use case b) Functional block diagram of an IP protection use case c) Functional block diagram of a random number generation use case

elements aimed to increase reliability and security, the later by hashing to defend against modeling attacks.

Note that Figure 1a indicates the use of a strong PUF. Herder et al. [5] state that a PUF can be viewed as a black box receiving a challenge c and returning a response r = f(c). They explain that "the fundamental difference between weak and strong PUFs is the domain of f(.)." [5] Weak PUFs only process a few challenges while inputs to Strong PUFs become exponentially large. Strong PUFs are intrinsically suited for the authentication use case because the challenge-response space is so large it prohibits an adversary from enumerating all the possible pairs.

B. Intellectual Property (IP) Protection

We define IP Protection as encrypting off-chip firmware and/or software as well as preventing chip cloning. For many years, microcontrollers or Field Programmable Gate Arrays (FPGAs) have used fuse-based mechanisms or embedded nonvolatile memory to store cryptographic keys on-chip. These cryptographic keys decrypt soft-IP stored off-chip via symmetric-key cryptography. Gassend et al. described a Physically Obfuscated Key (POK) as a means to prevent potential IP theft where an adversary could simply make a mask of a chip to clone it. IP theft is thwarted in this example because the weak PUF (since it always receives a single, hardwired challenge) will vary in output across devices. The real manufacturer can program the fuse-based key split of each chip to produce the correct decryption key when XORed with the PUF value generated by the device. A cloned chip will produce a different PUF value and thus an incorrect decryption key.

The POK presented was the first hint of using PUFs for cryptographic symmetric key generation. Though anti-cloning could be argued as a separate use case, in essence the IP protection Gassend et al. propose relies on the ability to deny an adversary from decrypting software in the embedded ROM. Generating a symmetric secret key is the more general use case that is of interest in this paper. Note that no error correction was incorporated in the POK and, again, a fixed challenge was hardwired because a strong PUF was used in this design.

In 2007, Suh and Devadas [6] presented a model for generating a cryptographic key which included syndrome data, or Error Correction Code (ECC) helper data, that is stored off-chip along with encrypted IP firmware or IP software as shown in Figure 1b. Suh and Devadas also introduced the concept of an enrollment phase to produce the syndrome followed by an operational phase where the system applies error correction to ensure reliable key generation. When using symmetric key encryption, such as AES, a single bit error would be catastrophic to the required operation. Also note that a weak PUF suffices for this application since only one encryption key is needed. Hashing in this model is a mechanism to transform a large non-uniform distribution of bits into a more uniform distribution of the proper key length. This dynamically generated encryption key is then used to decrypt IP that is stored externally to the device.

C. Random Number Generation

Gassend et al. originally noted that PUFs were Physical Random Functions, so using PUFs as Random Number Generators (RNGs) is an obvious use case. RNGs have numerous applications in security systems beyond key generation and device authentication. Random nonce values are used to establish secure sessions that provide forward secrecy. Applications that output encrypted data often need random padding. Some cryptographic implementations use random values to mask intermediate values for side-channel protection.

These one-time uses of random values differ from the prior use cases presented. Most significantly, the ephemeral values do not need to be error-corrected since there is no need to ensure they are regenerable; however, there are other considerations for this use case. Ideally the PUF in this use case should continue to generate new random values. Past applications have used pseudo-random functions that are seeded with a unique, typically confidential, initial value that deterministically generate a pseudo-random series. But in this use case, the rate at which entropy can be generated becomes of interest.

NIST has a series of publications related to RNGs. In Special Publication 800-90B, "Recommendation for the Entropy Sources Used for Random Bit Generation", Turan et al. [7] provided the system model where the digital noise source could be a chaotic PUF as shown in Figure 1c. Note that for security applications, RNG implementations should incorporate some type of health monitoring to make sure the source continues to generate random values. It is also advisable that the implementation includes an optional conditioning function, shown as a hash function in this example, to ensure random looking values are produced even in the case of failure.

D. Comparing and Contrasting

For the device authentication use case, Herder et al. [5] pointed out that PUF authentication can incorporate "an allowable error threshold" to eliminate the need for error correction. For reliability, the model shown in Figure 1a must incorporate error correction since hashed PUF output with a single bit error will change all of the response bits with a 50% probablity according to the strict avalanche criterion. Herder et al. [5] stated, "It should also be noted that the security models for weak and strong PUFs differ. The output of a weak PUF must be kept private, while a strong PUF's responses do not have the same restriction." Rather than attributing privacy to whether a PUF is strong or weak, the authors assert this is really a function of the use case.

Comparing and contrasting these diagrams is useful for practitioners to understand security implications. First, each model defines a boundary. Notionally that boundary provides some security; it could be a secure enclosure or it may only delineate the bounds of an integrated circuit. Of particular interest are differences among use cases regarding interfaces that cross the security boundary. The interface exposes attack surfaces that may be exploitable to leak information about the values being generated by a PUF.

III. PUF Structures

This section presents well-known PUF structures that a practitioner should understand. We describe the mechanisms to produce unique values or entropic values and how they are affected by manufacturing variations. We first present arbiter PUFs, followed by ring oscillators PUFs. Process variations manifest as time delays in both of these structures. Next, we describe Static Random Access Memory (SRAM) PUFs which, when powered but not initialized, assume a state based on the imbalanced threshold voltages of internal gates. We describe the butterfly PUF which is an FPGA implementation, primarily because this structure exposes specific implementation problems that practitioners should know. Finally, we introduce two chaotic PUFs that are far more complex than the others. The authors believe these additional chaotic structures warrant further study to understand their reliability and security.

A. Arbiter PUFs

The arbiter PUF structure, introduced in 2002 by Gassend et al., was the first PUF to be completely implemented with complementary metal-oxide-semiconductor (CMOS) technology. Figure 2 shows an arbiter PUF from Suh and Devadas [8]. In this PUF circuit, a pulse is injected into two symmetric paths where a race condition will result in either a 0 or a 1. Each path has a series of multiplexers which allow the pulse to switch paths depending on a common control bit. If the control bit is 0 the path simply passes to the next stage; if 1, the paths are swapped. The series of control bits compose the challenge value. With each additional pair of multiplexers and associated control bit, the possible path grows exponentially.

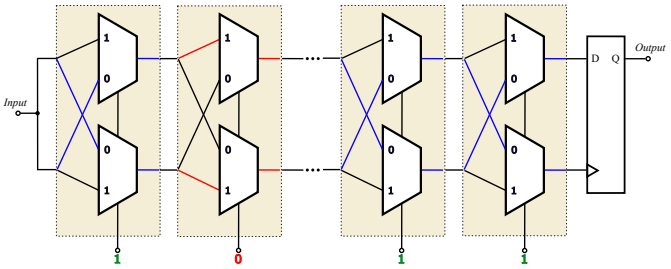

Fig. 2. Arbiter PUF Circuit

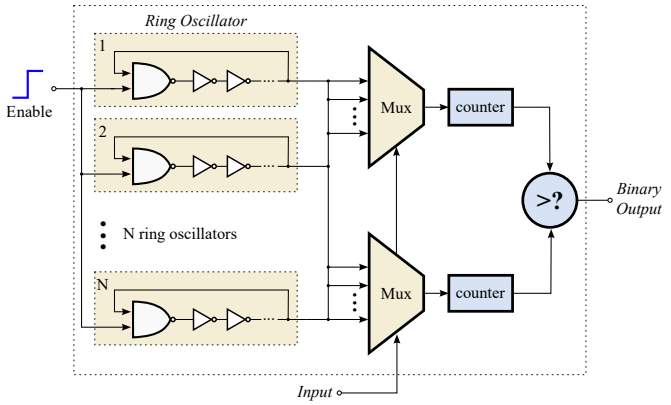

Fig. 3. Ring Oscillator PUF Circuit

Since the layout of multiplexer circuits are the same, in theory, the challenge control bits should have no effect on the resulting output, but in practice each mux has a different propagation rate which means some are faster than others. For this circuit, a 128 bit challenge value will result in a single response bit. To generate n response bits, this structure can be replicated n times, which can each take the same challenge value and execute in parallel. Alternately the same circuit could be evaluated n times, where the challenge value changes each time. Circuitry, such as a linear feedback shift register, could take the initial challenge value and produce a pseudo random series of challenge values. This, of course, is a classic time-space trade.

B. Ring Oscillator PUFs

A ring oscillator is comprised of an odd number of inverters to generate a free-running square wave. Due to process variations, multiple instances of ring oscillator circuits which have identical components and layouts will operate at different frequencies. By selecting a pair of ring oscillators and comparing their frequencies, an entropic bit can be produced. Figure 3 from Suh and Devadas shows a circuit where if the ring selected by the upper mux is faster than the ring selected by the lower mux determines whether a 1 or 0 is produced. Counters are used to add up pulses over a fixed period of time and these values are compared. Suh and Devadas [6] introduced this Ring Oscillator PUF (RO PUF) design in 2007.

At first, one may believe the number of bits of entropy that the given n RO PUFs could produce is simply based on the

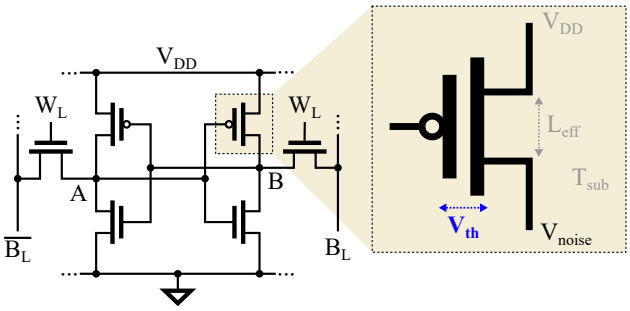

Fig. 4. SRAM cell with threshold voltage determining PUF State

number of combinations of the n items taken two at a time or $n(n–1)/2$. But note that in this approach rings are reused when combined in a different pairing so they are not independent. Suh and Devadas explained that if oscillator A ran faster than oscillator B and B in turn ran faster than C, clearly A runs faster than C and this shows a correlation between pairings. However, they showed by viewing the set of RO PUFs as a randomly ordered list of varying frequencies (fast to slow), the independent entropy can be calculated as $log_2(n!)$, since the likelihood of different orderings is equal.

C. Static Random Access Memory (SRAM) PUFs

A Static Random Access Memory (SRAM) cell is composed of two cross-coupled inverters. It is volatile memory that needs power to maintain state. Two access transistors are used to set the state of the cell and provide opposite values. An SRAM PUF relies on the fact that when uninitialized, a mismatch between threshold voltages determine which state the cell takes. In 2007, Holcomb et al. [9] presented the use of SRAM PUFs to generate device-unique RFID tags. Figure 4 is from Holcomb et al.'s 2009 [10]paper discussing the power-up state of SRAM PUFs.

When the circuit in Figure 4 is unpowered, A and B are both zero, therefore, at power-up AB is 00 and this is an unstable state. Writing to the cell will allow AB to assume valid states of 01 or 10, and as Holcomb et al. [10] noted AB will never hold the value 11, but after power and before the cell is programmed, the circuit will be driven to a stable state of either 01 or 10, again depending on the threshold voltages of the coupled inverters. The authors also pointed out data collected from real devices showed an unequal distribution of states, and the SRAMs they studied tended to settle to 1 over 0. Though not uniform, the number of bits generated was directly proportional to the number of SRAM cells used.

D. Butterfly PUF

In 2008, Kumar et al. [11] introduced a PUF structure implementable in FPGA fabric called a butterfly PUF. The authors used two cross-coupled latches in a novel manner to replicate SRAM PUF behavior as shown in Figure 5. They noted that it is not possible to create the combinatorial loops in FPGA fabric, but by holding the CLK signal high they produced asynchronous behavior. The preset of latch 1 and

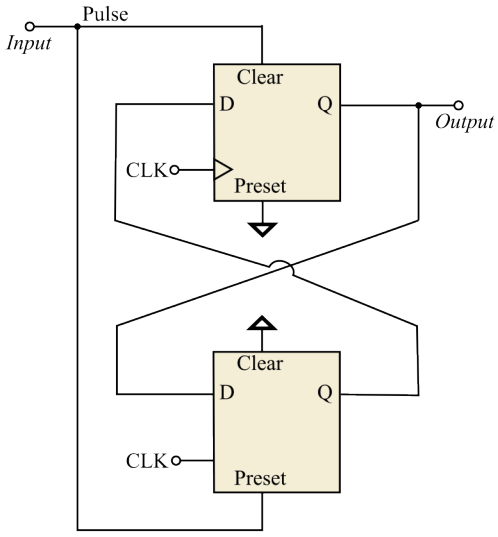

Fig. 5. Butterfly PUF: Cross-coupled Latches

clear of latch 2 were always set low. When the external excite-signal goes high, the crossed-coupled latches are forced into an unstable state. The authors stressed that the interconnections between the latches must be as symmetrical as possible, because as soon as the excite-signal goes low, the final state of the circuit is dependent on the delay of the connecting wires.

E. Tent-map PUF

In 2015, Cohen [12] published a novel circuit to be used as a PUF. This PUF design implements a one-dimensional (1-D), chaotic tent-map. Cohen cited as foundational Rosin's et al [13] work with autonomous boolean networks which realize asynchronous logic gates in FPGA fabric with cleverly arranged delay lines. The tent-map, is defined as

$$f[x] = \begin{cases} rx, & 0 \leq x \leq \frac{1}{2} \\ r - rx, & \frac{1}{2} < x \leq 1 \end{cases} \quad (1)$$

for $0 \leq r \leq 2$ and $0 \leq x \leq 1$. It forms a 1-D iterated map that, though deterministic, produces chaotic behavior when r is greater than one [14]. The Liapunov exponent, λ, is the rate at which entropy is generated by iterations of a map. For the tent-map λ = ln(r).

The ingenuity of Cohen's design was his leveraging un-clocked, digital delay lines to form 1-D iterated maps that output varying pulse widths [13] where the pulse-width at time t represents the current map iterate. The circuit shown in Figure 6 consists of a folding function f and a gain function g as well as a time delay element to ensure the circuit only processes one pulse at a time. An input pulse of initial width $w0$ is injected in the circuit at $t = 0$. The resulting output pulses with varying widths. This output pulse train can be quantized to produce a bitstring.

F. Hybrid Boolean Network (HBN) PUF

In contrast to low-dimensional iterated maps, high-dimensional chaotic structures may also be employed as PUFs.

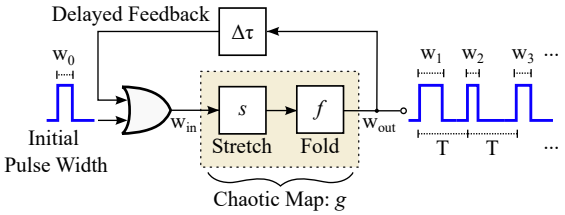

Fig. 6. Tent-map PUF

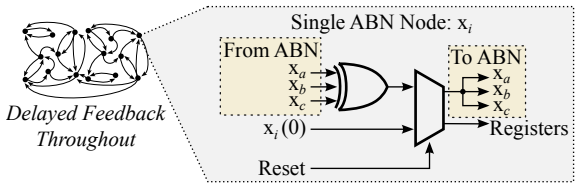

Fig. 7. Hybrid Boolean Network (HBN) PUF

In 2021, Charlot et al. [15] presented a strong PUF design suited for device authentication taking N challenge bits and producing N response bits. The core of this PUF is an Autonomous Boolean Network (ABN) made of unclocked logic gates in an FPGA. The ABN consists of N nodes where each node takes input from three neighbors and XORs them as shown in Figure 7. The number of neighbor connections is configurable, but limited to the maximum number of Look-Up Table (LUT) inputs available in a targeted FPGA. Each node has an associated multiplexer controlling whether it takes the challenge input bit or the result of XORed neighboring nodes. At the start of execution, a challenge bit is held while the network stabilizes. When the multiplexers are switched, values are allowed to propagate through the network asynchronously.

The authors coin this PUF element a Hybrid Boolean Network (HBN) because synchronous logic is also used to input challenges to the ABN as well as latch responses. Rather than simply using the system clock to select when to register the N node responses, the designers incorporated M pairs of inverters which are driven by the system clock to generate M registration periods. For this implementation, sampling rates are determined by inverter pairs and will vary in a manner similar to the nodes in the ABN with temperature and voltage, thus the design is interpreted to have some intrinsic reliability. This work also provides guidance about the optimal time period selection for registering a network response, but the authors acknowledge future work is needed to improve reliability.

G. Comparing and Contrasting

It is worthwhile to compare and contrast these PUF structures.

1) Delay Based versus Threshold Voltage: As described before, both the arbiter PUF and RO PUF rely on random delay as the source of entropy, whereas in SRAM PUFs, the imbalanced voltage biases of interconnected transistors results in random states [5]. A practitioner should be cognizant of the time these circuits need to produce responses. The arbiter and RO PUFs presented, require much more time to generate values compared to the settle time of the SRAM PUF. Both chaotic PUFs presented also rely on random delay, but because the HBN PUF iterates N nodes in parallel, it quickly produces responses.

2) Weak versus Strong: Again the SRAM PUF is a weak PUF because one response is generated at power-up. Conversely, an arbiter PUF is a strong PUF, because by design it has a large challenge space producing responses. Herder et al. classified RO PUFs as weak, "since there are a limited number of 'challenge bits' that can configure the PUF's operation." [5] These challenge bits would involve the selection of different ring pairs. The authors believe the tent-map PUF is a weak PUF based on the limited challenge bits, but we noted that the width of the initial pulse is not the only parameter for this structure. The value of r is also a tunable parameter that could be part of the challenge space. It is difficult to provide more analysis since a specific implementation was not studied. Finally, as alluded to previously, we consider the HBN PUF a strong PUF when the number of nodes N is sufficiently large and would likely be 128 or 256 for such applications.

3) Potential multi-use of PUF Structures: The SRAM PUF is not a circuit designed specifically for generating random values compared to the other structures considered, rather standard SRAM memory cells are repurposed to capture one-time information at power up. After key generation the SRAM could be used as on-chip memory. If the tent-map PUF proves reliable for key generation, it could first generate an encryption key and then continue to run in the role as the digitial noise source for a random number generator. Note, as a random number generator, thermal noise would not be a detriment rather it would be an additional source of entropy. Likewise, a single instantiation of the HBN PUF could conceivably support device authentication in addition to Random Number Generation.

4) Candidates for FPGA Implementation: In 2009, Morozov et al. [16] provided a comparison of delay-based PUFs implemented in FPGA fabric. Their analysis showed that both the arbiter PUF and the butterfly PUF structures were not suitable designs for FPGA implementations. The reason neither of these PUFs could be realized is that the time delays caused by routing asymmetries were significantly greater than the propagation delays in the logic gates due to manufacturing variations. The authors also noted that RO PUFs do not suffer the same ill effects because symmetric connections are not required. However, practitioners should still be cognizant of the proportion of delay caused by routing when instantiating ring oscillators in an FPGA. A pair of poorly implemented RO PUFs, where the difference in their frequencies is not affected by process variation, would not vary across devices.

5) Reliability Design Principles: Herder et al. [5] explain that differential design techniques can be used to cancel out first-order environmental dependencies. This is an important principle that PUF designers should remember. Additionally, this paper notes the concept of conditioned timing, such as

the HBN PUF's use of clocked inverter pairs, to improve reliability.

6) Security: Some interesting comparisons can be made between the HBN PUF and arbiter PUF. As referenced earlier, researchers are investigating modeling and machine learning (ML) attacks that use a small number of known challenge-response pairs to predict responses of new challenge values. In 2019, Ganji et al. [17] introduced a tool called PUFmeter to examine if a PUF is susceptible to ML attacks. The authors of the HBN PUF reported that PUFmeter was unable to model its behavior [15].

7) Additional Thoughts on the HBN PUF: As stated earlier the HBN PUF is delay-based and implemented on an FPGA. Interestingly, Charlot et al. claimed that, "the HBN PUF does not require carefully constructed circuit paths with specified delay characteristics." [15] This claim warrants further investigation. It is unclear if the authors attributed the immunity to different time delays due to asymmetric routing among nodes simply because of the complexity of the network. The authors also suggested "additional bits per response can be used for error correction", but do not provide details on their error correction approach.

Charlot et al. also noted given the number of network nodes $N >= 16$, the network "consistently exhibits chaos", however an ABN constructed with $N <= 8$ may produce periodic responses. When analyzing dynamical systems, fixed point analysis is an important tool to determine if a system will converge to a single value or a periodic pattern. When referencing a single XOR gate with two time-delayed feedback loops, Rosin [13] explained, "A Boolean fixed point in the feedback system corresponds to rows in the lookup table, where all entries have the same value and hence inputs and output can be the same." Rosin then eliminates the possible fixed point by altering the look-up table. Charlot et al. seemed to hypothesize that, given a sufficient length and interaction among nodes, this zero fixed point case does not occur. Though empirical results support their claim, the authors suggest that logic similar to Rosin's modified look-up table should be considered instead of simply an XOR.

IV. Entropic Sources

Charlot et al. described three sources of entropy in their HBN PUF: "1) Frozen-in heterogeneity (manufacturing differences), 2) Thermal and charge fluctuations (noise), and 3) Deterministic chaos (unpredictability and nonlinear amplification of timing differences)". The authors suggest that these sources might be generalized for all PUFs and are renamed here. In this section, we define and discuss three sources: instantiated entropy, execution entropy, and execution noise.

A. Instantiated Entropy

PUF designers intend to exploit the process variations introduced during manufacturing to generate, at run-time, values unique to an integrated circuit. These process/manufacturing differences are also called device heterogeneities. For the use of IP protection, i. e., generating a secret key, it is essential that the device heterogeneities reliably generate the same values. For example, each memory cell in an SRAM PUF should settle on the same value at startup since the bits will be used to derive a cryptographic key. In practice, however, this characteristic is not true for all SRAM cells. From the perspective of reliability, a practitioner does not want a probabilistic generation of memory values.

Imagine that, instead of a pair of cross-coupled inverters, each cell contains a weighted coin that is flipped at start-up. For PUF applications, practitioners want heavily weighted coins, ideally perfectly weighted. In fact, enrollment is the process of excluding fairly weighted coins. This may seem counterintuitive but one wants to be certain of the value the flipped coin will take. When generating the PUF value, uncertainty should be minimized; entropy should be minimized. Stated differently, across different devices or different instances, we want to maximize entropy, but for the generation of a PUF value in a single instance we want to mininize entropy(uncertainty) of the value it will produce.

The authors also contend that instantiated entropy is a better term than "Frozen-in heterogeneity", because all sources are dynamic over time. For example, a device's PUF response may change over time due to aging effects. The authors contend that "Frozen-in heterogeneity" can be misconstrued to imply reliable.

B. Execution Entropy

The tent-map PUF operates as an iterated map, thus discretely, and produces an entropic response each iteration. The HBN PUF operates in a continuous time regime and outputs entropic values that can be sampled at some rate. The authors want readers to understand these chaotic PUFs are fundamentally different from traditional PUF structures which perform an operation once to produce an entropic result. The authors suggest execution entropy captures this idea.

C. Execution Noise

PUF literature is full of examples of execution noise. PUF circuits fail to reliably regenerate values due to thermal or shot noise. Implementations may employ error correction methods to generate consistent values. Charlot et al. [15] show noise overwhelms their chaotic network over time. They show that within approximately 10 nanoseconds their circuit is stochastic and from a use case perspective is an ideal random number generator.

V. Future Work

The considered PUF structures need to be implemented to accurately compare their execution rates. The authors believe implementing these PUFs on a current FPGA would be valuable. In addition to demonstrating the rate these PUFs can generate comparable entropic results, the number of logic gates needed should also be captured. Note, as shown in this paper, additional circuitry is needed for certain use cases. For example, error correction code circuitry is required for the IP protection use case. Furthermore, error correction can affect

the rate; for example, if repetition error correction is used, the PUF must execute multiple times. Other design decisions, such as the number of inverters used in an RO PUF would affect rate. The work to implement and compare PUF structures would require navigation of trade-off spaces, and the authors assert that separate studies for different use cases should be performed.

Though implementations are not available to fairly measure execution rates, notional estimates of these PUF structures can be provided based on the number of gates and the number of times they are exercised. The SRAM PUF followed by the butterfly PUF have the fewest gates thus are the fastest. Arbiter PUFs have more stages than an RO PUF, but the RO PUF has to iterate many times to generate unique counts. Therefore, an RO PUF requires the longest time to execute. The HBN PUF is massively parallel and thus produces a result faster than an arbiter when implementing a challenge-response scheme. The tent-map we describe executes in a serial manner so it is slower than the HBN. A comparison of entropy generation rates for a tent-map versus an HBN implementation on a current FPGA would be useful.

Finally, analysis of error correction methods is warranted along with the security vulnerabilities they expose. Specifically, error-correction-code helper-data attacks which are not evaluated in this paper. Health monitoring methods for random number generation also merit investigation.

VI. Conclusion

This paper presents traditional PUF structures as well as emerging chaotic structures. We describe three use cases for PUFs: device authentication, IP protection, and random number generation. This paper shows why practitioners should first consider the use case when assessing the suitability of a PUF structure, given certain structures are inherently better for particular use cases. For example, the described HBN PUF naturally provides a large challenge-response space needed in device authentication. The authors also point out that these structures may be repurposed during system operation. SRAM may be used for key generation at startup and then provide normal memory functionality. Likewise, a HBN PUF can be used for device authentication at startup and then function as a true random number generator (TRNG).

Practitioners are concerned about three fundamental attributes of PUFs: entropy, reliability and security. This paper discusses entropy sources of the considered PUFs, time delay, and transistor bias. We also discuss reliability and highlight the use of differential logic to produce structures that are more resilient to environmental changes. The authors show that different use cases can expose different attack surfaces. Finally the authors call attention to the unique ability of chaotic PUFs to successively generate entropic values. Table 1 summarizes some of the features of PUF structures considered in this paper.

TABLE I
Comparison of PUFs

PUF	Use Case	Target	Reliability	Speed
Arbiter	Device Authentication	ASIC	Temperature Sensitive	Slow
Ring Oscillator	IP Protection	ASIC/ FPGA	Temperature Sensitive	Slowest
SRAM	IP Protection	ASIC	Aging Effects	Fastest
Butterfly	IP Protection	FPGA	Routing Imbalances	Fast
Tent-map	IP Protection	FPGA	Temperature Sensitive	Slow
Tent-map	RNG	FPGA	N/A	Slow
HBN	Device Authentication	FPGA	Temperature Sensitive	Fast
HBN	RNG	FPGA	N/A	Fast

References

[1] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Silicon physical random functions," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 148–160.

[2] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Controlled physical random functions," in *18th Annual Computer Security Applications Conference, 2002. Proceedings.*, 2002, pp. 149–160.

[3] B. Gassend, "Physical random functions," 2003.

[4] R. Anderson, *Security engineering: a guide to building dependable distributed systems*. John Wiley & Sons, 2020.

[5] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, "Physical unclonable functions and applications: A tutorial," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1126–1141, 2014.

[6] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proceedings of the 44th annual design automation conference*, 2007, pp. 9–14.

[7] M. S. Turan, E. Barker, J. Kelsey, K. A. McKay, M. L. Baish, M. Boyle *et al.*, "Recommendation for the entropy sources used for random bit generation," *NIST Special Publication*, vol. 800, no. 90B, p. 102, 2018.

[8] M.-D. Yu and S. Devadas, "Secure and robust error correction for physical unclonable functions," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 48–65, 2010.

[9] D. E. Holcomb, W. P. Burleson, K. Fu *et al.*, "Initial sram state as a fingerprint and source of true random numbers for rfid tags," in *Proceedings of the Conference on RFID Security*, vol. 7, no. 2, 2007, p. 01.

[10] D. E. Holcomb, W. P. Burleson, and K. Fu, "Power-up sram state as an identifying fingerprint and source of true random numbers," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1198–1210, 2009.

[11] S. S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls, "The butterfly puf protecting ip on every fpga," in *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*. IEEE, 2008, pp. 67–70.

[12] S. D. Cohen, "Structured scale dependence in the lyapunov exponent of a boolean chaotic map," *Physical Review E*, vol. 91, no. 4, p. 042917, 2015.

[13] D. P. Rosin, D. Rontani, D. J. Gauthier, and E. Schöll, "Excitability in autonomous boolean networks," *Europhysics Letters*, vol. 100, no. 3, p. 30003, 2012.

[14] S. H. Strogatz, *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

[15] N. Charlot, D. Canaday, A. Pomerance, and D. J. Gauthier, "Hybrid boolean networks as physically unclonable functions," *IEEE Access*, vol. 9, pp. 44 855–44 867, 2021.

[16] S. Morozov, A. Maiti, and P. Schaumont, "A comparative analysis of delay based puf implementations on fpga," *Cryptology ePrint Archive*, 2009.

[17] F. Ganji, D. Forte, and J.-P. Seifert, "Pufmeter a property testing tool for assessing the robustness of physically unclonable functions to machine learning attacks," *IEEE Access*, vol. 7, pp. 122 513–122 521, 2019.