IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received February 26, 2021, accepted March 11, 2021, date of publication March 17, 2021, date of current version March 26, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3066948

Hybrid Boolean Networks as Physically Unclonable Functions

NOELOIKEAU CHARLOT[1], DANIEL CANADAY[2], ANDREW POMERANCE[2], AND DANIEL J. GAUTHIER[1], (Member, IEEE)

[1]Department of Physics, The Ohio State University, Columbus, OH 43202, USA
[2]Potomac Research, LLC, Alexandria, VA 22314, USA

Corresponding author: Noeloikeau Charlot (charlot.5@osu.edu)

This work was supported in part by the Department of the Army under Award W31P4Q-19-C-0014, in part by (for NC and DJG) Asymmetric Technologies, LLC, through the project Resilient and Enhanced Security UAS Flight Control supported by the Ohio Federal Research Network. An international patent was filed based on this work under PCT/US2020/027072 for Systems and methods using hybrid Boolean networks as physically unclonable functions.

ABSTRACT We introduce a Physically Unclonable Function (PUF) based on an ultra-fast chaotic network known as a Hybrid Boolean Network (HBN) implemented on a field programmable gate array. The network, consisting of N coupled asynchronous logic gates displaying dynamics on the sub-nanosecond time scale, acts as a 'digital fingerprint' by amplifying small manufacturing variations during a period of transient chaos. In contrast to other PUF designs, we use both N-bits per challenge *and* obtain N-bits per response by considering challenges to be initial states of the N-node network and responses to be states captured during the subsequent chaotic transient. We find that the presence of chaos amplifies the frozen-in randomness due to manufacturing differences and that the extractable entropy is approximately 50% of the maximum of $N2^N$ bits. We obtain PUF uniqueness and reliability metrics $\mu_{inter} = 0.40 \pm 0.01$ and $\mu_{intra} = 0.05 \pm 0.00$, respectively, for an $N = 256$ network. These metrics correspond to an expected Hamming distance of 102.4 bits *per response*. Moreover, a simple cherry-picking scheme that discards noisy bits yields $\mu_{intra} < 0.01$ while still retaining $\sim$ 200 bits/response (corresponding to a Hamming distance of $\sim$ 80 bits/response). In addition to characterizing the uniqueness and reliability, we demonstrate super-exponential scaling in the entropy up to $N = 512$ and demonstrate that PUFmeter, a recent PUF analysis tool, is unable to model our PUF. Finally, we characterize the temperature variation of the HBN-PUF and propose future improvements.

INDEX TERMS Chaos, physically unclonable function (PUF), field programmable gate array (FPGA), autonomous Boolean network (ABN), hybrid Boolean network (HBN).

I. INTRODUCTION

Physically unclonable functions (PUFs) are an emerging technology that extract randomness, or entropy, from uncontrollable manufacturing variations in the physical structure of identically produced devices [1], [2]. PUFs use this entropy to reliably generate a 'digital fingerprint' - a unique sequence of 0's and 1's known as a bitstream - that is produced by the device but never stored [3]. In practice, PUFs are often circuits embedded in other devices that reliably map an input (or *challenge*) to an output (or *response*) in a way that is unique to a particular copy (or *instance*) of the device.

For example, the start-up behavior of static random-access memory (SRAM) produces an identifying bit pattern suitable for use as a PUF [4]. Ideally, this identifying behavior cannot be reproduced (or cloned), either because it is physically impossible to recreate the same conditions in another device, or because it is mathematically impossible to accurately predict the PUF's behavior. In summary, we highlight three practical properties of PUFs:

- *Uniqueness*: Responses from different instances to the same challenge are different enough to distinctly identify each instance;
- *Reliability*: Responses from an individual instance to the same challenge are similar enough to consistently identify that instance;
- *Unclonability*: The challenge-response pairs (CRPs) of an individual instance cannot be: (1) physically replicated by another instance, or (2) inferred from knowledge of the device manufacturing process or previously revealed CRPs.

The associate editor coordinating the review of this manuscript and approving it for publication was Sedat Akleylek.

In early work, PUFs were constructed using complex optical scattering devices or custom fabricated silicon chips [5]. More recently, there is an industry trend toward using reprogrammable devices such as field-programmable gate arrays (FPGAs) for PUF-based IP protection. For example, IntrinsicID offers the commercially available 'butterfly PUF,' an SRAM-PUF embedded directly into some manufacturers higher-end FPGAs [6].

However, SRAM-PUFs, such as the butterfly PUF, are 'weak' in the sense that there are relatively few CRPs obtainable per device (in this case resulting from the static initialization of each memory cell at power-up) [2]. As a result, their use for authentication purposes are limited because an attacker can clone the device by obtaining the full set of CRPs in a short amount of time. 'Strong' PUFs, on the other hand, contain a relatively large number of independent CRPs, making attempts to extract or predict all of them a difficult or impossible task [7]. Moreover, the design and practical implementation of strong FPGA-based PUFs remains an open problem [1].

Modern PUF proposals have also started to explore chaotic dynamics as an additional source of entropy [8]–[10]. Chaos is characterized by an exponential divergence between initially similar trajectories. As discussed in more detail below, this behavior can be used by a PUF to amplify the entropy available from the small physical variations inherent in any manufacturing process. Moreover, we hypothesize that chaos provides resilience to machine learning due to the existence of 'fractal basin boundaries' [11], which is a phenomenon in chaotic systems in which dividing lines between different behaviors have a fractal structure. In the standard interpretation, this means that an infinitesimal change in the initial conditions of the system does not yield a smooth change in the asymptotic behavior of the system; instead, the system may evolve to a disjoint attracting set. We hypothesize that a chaotic PUF has a similar behavior with respect to the system parameters, such that an infinitesimal change does not yield a smooth change in the measured response. Hence, even marginal uncertainty in the system parameters changes the entire class of possible outcomes, likely confusing attempts at prediction.

Finally, many PUFs incorporate asynchronous (unclocked and analog-like) logic into their design [1], [2]. Asynchronous logic can require fewer resources (time, area and power) than conventional synchronous circuits governed by a global clock. Morevoer, compared to synchronous designs, asynchronous designs are much more sensitive to manufacturing variations. This is because clocked operations are stabilized by waiting an entire clock period before the next operation, so that any variations in, *e.g.*, rise time or signal propagation time are eliminated. On the other hand, dynamical properties of even simple unclocked systems such as the frequency of a ring oscillator depend sensitively on variations in rise and fall times. In general, combinatorial loops can be designed that operate at the maximum frequency allowed by the hardware, where the dynamics are most sensitive to manufacturing variations. Thus, asynchronous PUFs are useful as compact, low-power cryptography primitives.

A. THIS WORK

In this paper, we propose a design for a strong, chaos-enhanced, asynchronous PUF and demonstrate its implementation on an FPGA. Our PUF is based on a network of coupled, unclocked logic gates known as an autonomous Boolean network (ABN) combined with a clocked digital control and readout layer, forming what we call a hybrid Boolean network (HBN, HBN-PUF). The HBN-PUF can be incorporated into existing FPGA designs without specialized hardware, having a resource count proportional to the number of nodes in the network N. The unique properties of the HBN-PUF compared to existing strong PUF proposals are:

- The HBN-PUF produces N (or potentially more) response bits per N-bit per challenge. Thus, extracting secrets of a given length requires $\sim 1/N$ the number of queries, which translates into time, storage and network traffic efficiency. Moreover, the additional bits per response can be used for error correction and improving environmental resilience, and the multi-dimensional response space and possible fractal basin boundaries will likely frustrate machine learning attacks.
- Unlike many conventional PUFs, such as delay-line PUFs [2], the HBN-PUF does not require carefully constructed circuit paths with specified delay characteristics; rather, automatic placement of circuit elements by standard vendor-supplied compilation and synthesis tools yield usable HBN-PUFs.
- The ABN part of the HBN-PUF exhibits picosecond-scale asynchronous transient-chaotic dynamics. Because of these ultra-fast dynamics, response readout occurs in less than 10 ns, which has important practical applications because the number of CRPs required for strong industrial-scale enrollment can be obtained in a short time.

The paper is organized as follows. Our proposed HBN-PUF design is described in Sec. II, with a discussion of the circuit and data collection process in Sec. II-A and the physical origins of PUF behavior in Sec. II-B. Section III is devoted to experimentally characterizing the HBN-PUF behavior by measuring its uniqueness and reliability (III-A), entropy scaling (III-B), resilience to machine learning (III-C), and temperature variation (III-D). Section IV concludes with a brief discussion and future work. Supporting materials are given in the Appendix, including the hardware description language code that instantiates our design.

II. PROPOSED HBN-PUF DESIGN

Our proposed design is shown in Fig. 1, which consists of a network of N coupled 'nodes' and a clocked digital readout and control layer, forming an HBN. Each node is a combinatorial logic circuit that takes as input the outputs

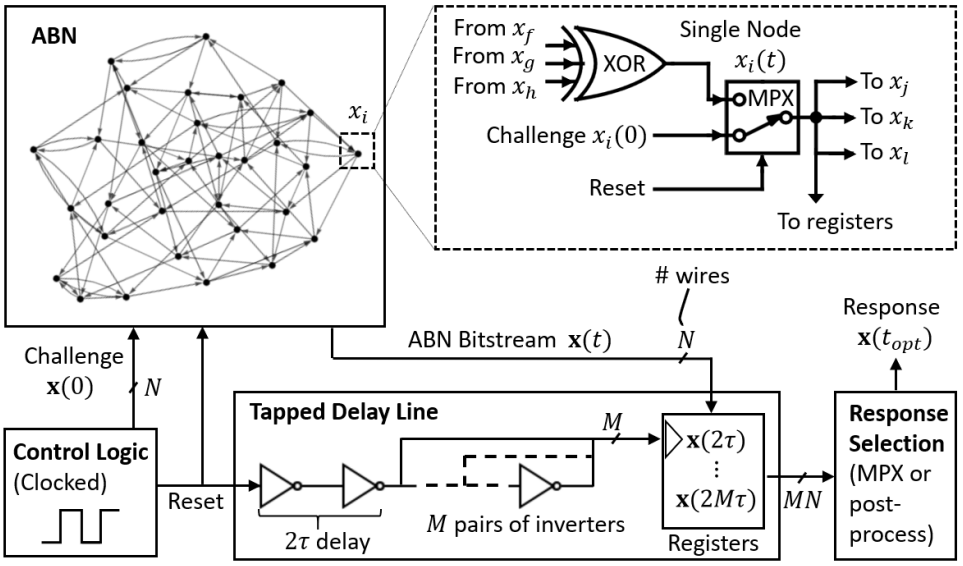

FIGURE 1. Proposed HBN-PUF design. The studied XOR-ABN topology is shown in the upper left, and the logic of an individual node in the upper right. Shown in the bottom are the clocked logic used to apply the challenge and the tapped delay line used to select the response. The specific connections between nodes (i.e, the identity of nodes $f - l$) are governed by the topology of the network; shown is an $N = 32$ regular random graph of degree 3.

of other nodes in the network and a global reset signal; we refer to the output of this circuit as the 'state' of each node in the network. When the reset signal is high, the state of the node is the corresponding bit in the challenge string, and the state of the entire network is exactly equal to the challenge string $\mathbf{C}$. When the reset signal is low, the state of the node is given by the XOR of the states of its input nodes and the entire network is a large recurrent combinatorial loop that evolves in time without a clock (*i.e.*, autonomously). Typical clocked digital logic circuits constrain voltages to be near logic high or low most of the time so that the output voltages of a gate is near logic high or low. In contrast, the individual semiconductor devices in an ABN act as highly nonlinear input-output devices with analog (but Boolean-like) dynamics, and the voltages take on a continuous range of values between logic-high and -low. During this time, the digital readout layer captures a Booleanized representation of the true analog network state in discrete time intervals. A single state at the optimal time of measurement is then selected as the response $\mathbf{R}$.

In contrast to other PUF designs, we stress that the challenge and response are both N-bit strings, specifying the network's initial condition and Booleanized state in a chaotic transient, respectively. Thus, there are N response bits for each of 2^N challenges. Hence, the number of extractable bits from the HBN-PUF may scale super-exponentially as $N2^N$, yielding a strong PUF.

A. DESIGN SPECIFICS AND DATA COLLECTION

For the specific HBN considered in this work, each node takes exactly 3 inputs, and the combinatorial function is the 3-input XOR, as shown in the upper right of Fig. 1. Both of these design choices are flexible. The XOR function is chosen because it is maximally sensitive to its inputs, and the output is balanced between high and low; the overall bias of the response can be controlled by replacing the XOR with a Boolean function that has more or fewer high outputs. Three inputs were chosen in order to fit within a Cyclone V logic element; more or fewer inputs can be used to match the layout to other FPGA architecture details. Moreover, structure can be applied to the network (such as ring topologies [12]) to fine tune statistical and performance properties of the resulting response. These aspects will be explored in follow up papers, but in this work each node's XOR gate takes the output of three nodes (f, g, h in Fig. 1), randomly chosen without replacement from among the $N-1$ other nodes, and in turn its multiplexer feeds the XOR gate of three other nodes (j, k, l in Fig. 1). When the clocked reset signal is low, the multiplexer passes the node's XOR gate. When the reset signal is high, the node's multiplexer holds the initial condition, which is given by a corresponding bit of the challenge. In this way, the analog state of all nodes in the network $\mathbf{x}(t) = \{x_i(t)\}_{i=1}^N \in [0, 1]^N$ are initially held fixed to the digital N-bit challenge string $\mathbf{C}$, described mathematically as

$$\mathbf{x}(0) = \mathbf{C} \in \{0, 1\}^N. \tag{1}$$

The HBN stabilizes to the initial condition nearly instantaneously, but we hold it there for several ~ 100 MHz clock cycles of holding Reset high. The dynamics are then enabled by setting the Reset signal low, causing each multiplexer to pass the output of the autonomous XOR gate that feeds it. The network then evolves continuously in time and each XOR gate updates asynchronously based on the analog voltage of its neighbors.

During this time, the HBN dynamics are measured by sending the Reset signal down M pairs of inverter gates

(*i.e.*, a delay line). An associated register is triggered after the delayed Reset signal passes over a given pair of inverters. Each register Booleanizes the analog state of the HBN at that time and stores it digitally. This results in a sequence of N-bit Boolean state vectors in memory recording the bitstream produced by the network $\{\mathbf{x}(2\tau), \mathbf{x}(4\tau), \ldots, \mathbf{x}(2M\tau)\} \in \{0, 1\}^{NM}$.

Here, $\tau \sim 0.25$ ns is the mean delay time of a single inverter-gate, which is similar to the timescale of the XOR gate and multiplexer operations. Thus, the bitstream is sampled at a similar rate as the HBN dynamics, in roughly $2\tau \sim 0.5$ ns intervals. However, like all logic elements, each delay is subject to manufacturing variation, and so the sampling rate is not completely uniform. This also contributes to the manufacturing variation that gives rise to PUF behavior. Moreover, by using pairs of inverter gates rather than a clock source, the delay through the delay line varies with temperature and voltage in a similar way to the dynamical timescale of the nodes in the network. Thus, the delay line imparts some robustness to environmental variation.

The response $\mathbf{R}$ is selected from among this bitstream as a single state of the network at an optimal point in time $\mathbf{x}(t_{opt})$ during the chaotic transient

$$\mathbf{R} = \theta(\mathbf{x}(t_{opt})) \in \{0, 1\}^N, \tag{2}$$

where $\theta : [0, 1]^N \to \{0, 1\}^N$ is an element-wise thresholding operation, corresponding to the Booleanization of the real-valued $\mathbf{x}(t)$ performed by the registers. The details of determining t_{opt} are discussed in Sec. III.

B. HBN DYNAMICS AND PUF BEHAVIOR

If each logic gate in an HBN were synchronously updated by a global clock, it would execute the digital Boolean XOR function exactly, and node states would take on discrete values 0 or 1 at each clock cycle. In this mode, the state at each discrete time step would be exactly determined by the N-bit Boolean state at the previous time step, and the entire network would act as a pseudo-random number generator. However, because the logic gates are unclocked, their inputs can change at the same time that they are transitioning between logic high and low. As a result, nodes have the potential to take on intermediate logic values (analog voltages) [13]. Thus, the dynamics of nodes are better described by continuous differential equations that model the rise and fall times resulting from the finite capacitances and resistances in the devices, and not by discrete Boolean dynamics. Moreover, the state at a specific time is not given by the states of its inputs at the current time, but rather by time-delayed versions, due to the finite speed at which signals propagate along interconnects. Taken together, this causes the asynchronous XOR gate to behave as a highly nonlinear input-output device that multiplies signal edges, which quickly causes the dynamics to reach the maximum switching frequency allowed by the hardware [14], [15].

Under these conditions, the network dynamics become highly sensitive to amplitude fluctuations about the intermediate voltage value. Here, small perturbations to the voltage at the XOR gate, such as those due to manufacturing variation, noise, and differences in initial conditions, will cause the time at which the node switches between logic high and low to vary, resulting in previously similar waveforms diverging. As a result, ABNs consisting of XOR gates can exhibit chaos even in small networks [16]. When combined with a digital readout and control layer to form an HBN, they have been used as ultra-fast true-random-number generators (TRNGs) capable of a 12.8 Gbit/s entropy rate [12].

Based on past research and the discussions above, we identify three sources of entropy in XOR-HBNs related to PUF behavior:

1) Frozen-in heterogeneity (manufacturing differences),
2) Thermal and charge fluctuations (noise), and
3) Deterministic chaos (unpredictability and nonlinear amplification of timing differences)

Each source of entropy produces variations in the bitstream generated by the digital readout layer of the clocked portion of the network. However, each source has a separate physical origin as discussed in the rest of this section.

Frozen-in heterogeneity is due to small variations in the physical properties of the wiring and logic elements and it is this source of entropy that forms the primary basis of PUF behavior. Slight physical differences between nodes and wires - such as node input impedence, switching rate, and signal propagation time - alter the time at which the analog voltage of individual nodes cross the logic threshold for nominally identical inputs. The effect of these manufacturing variations are more pronounced at the ultra-fast time scale of the dynamics, which become distinctly correlated with the unique physical characteristics of an individual device. Such correlations produce the identifying information used to distinguish different FPGAs programmed with the same HBN design. They are quantified by the uniqueness parameter μ_{inter} (Appendix C, (9)).

Thermal and charge fluctuations are sources of time-dependent stochastic behavior (often referred to as 'noise'), which reduce the reliability of the PUF. Noise perturbs the amplitude of the logic gates in the asynchronous portion of the network and changes the times at which nodes cross the threshold separating logic high from logic low. If a transition is near the time at which the readout logic registers the node state, small variations in the threshold crossing time can change a registered zero to a one or vice versa. This alters the bitstream of a single device under repeated measurement, introducing unreliability quantified by μ_{intra} (Appendix C, (8)).

Chaotic systems have a positive entropy rate separate from noise and manufacturing variations, which serves to amplify both of these sources of entropy. The entropy attributed to chaos is due to the finite precision of physical measurements and the exponential sensitivity of chaotic systems to initial conditions. Any physical measurement of initial conditions has a necessarily limited precision, and so two trajectories measured to have the same initial conditions will diverge due

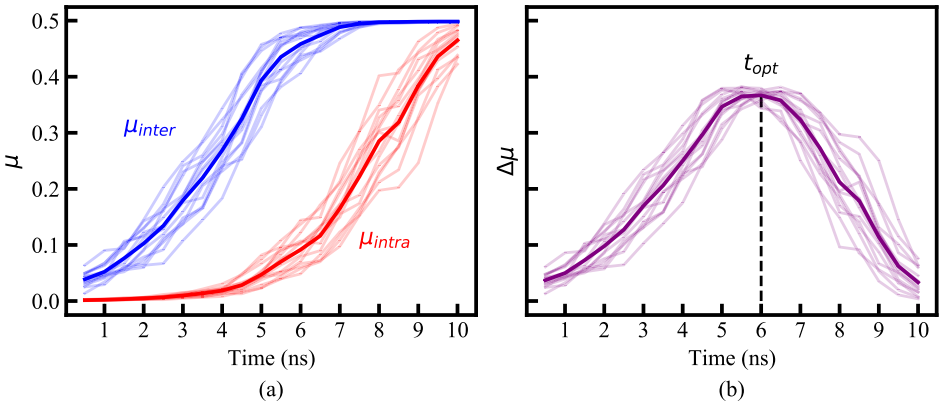

FIGURE 2. (a). μ_{inter} (blue) and μ_{intra} (red) vs time. Light lines correspond to the metrics calculated on a single PUF class (*i.e.*, choice of random network and placement on an FPGA) for an $N = 256$ node HBN-PUF, and the dark lines correspond to the average values calculated over all PUF classes (*i.e.*, the average expected behavior for an $N = 256$ random, 3-input HBN-PUF). (b) $\Delta\mu_{intra}$ vs time, with same definitions for light and dark lines. The highlighted t_{opt} is the time at which the average $\Delta\mu$ is maximum. We note, however, that there is some variation in the specific t_{opt} for each PUF class.

to the unmeasurable differences in the true initial state of each system. Chaos thereby magnifies any small differences in the applied challenge over time, acting as a nonlinear amplifier of the other sources of entropy and contributing to the unclonability property.

These three sources of entropy are visible in Fig. 2(a), which is a plot of μ_{inter} and μ_{intra} vs. measurement time. Frozen-in heterogeneity is illustrated by the separation between μ_{inter} and μ_{intra} at very short measurement times, noise is illustrated by the fact that μ_{intra} is non-zero, and the effect of chaos is illustrated by the fact that both measures grow exponentially until saturating at 0.5. In the next section, we discuss finding t_{opt} that balances these competing effects.

III. ABN-PUF PERFORMANCE STATISTICS

To be an effective PUF, the entropy rate due to the frozen-in heterogeneity of the HBN must be greater than the noise-induced entropy rate. This is captured by the metric

$$\Delta\mu(t) := \mu_{inter}(t) - \mu_{intra}(t), \tag{3}$$

which is plotted vs. time for $N = 256$ in Fig. 2(b). There is an optimal time of measurement t_{opt} for which the network has coupled sufficiently to manufacturing variations to act as a unique identifier ($\mu_{inter} \sim 1/2$), while remaining unperturbed enough by noise to be reliable ($\mu_{intra} \sim 0$), defined by

$$t_{opt} := \underset{t\in[2\tau,2M\tau]}{\arg\max}\ \Delta\mu(t). \tag{4}$$

All future statistics are calculated from the network state at this time. In practice, we find $t_{opt} \sim 2-8$ ns for the networks studied, with slowly increasing t_{opt} with network size N. Note that t_{opt} is calculated exactly once over an entire PUF class and represents a characteristic timescale of the HBN dynamics. Further, we do not observe significant variation in t_{opt} or $\Delta\mu$ due to differences in the layout of the network or delay line, as demonstrated in Fig. 2 and described in Appendix A.

In the remainder of this section, we study the performance statistics of the proposed HBN-PUF, including its reliability and uniqueness (III-A), entropy (III-B), resilience to machine learning (III-C), and temperature variation (III-D). Corresponding definitions and experimental procedures are elaborated in Appendices A-G.

A. RELIABILITY AND UNIQUENESS

Reliability and uniqueness are standard means of gauging PUF performance [1]. The average fraction of dissimilar bits between responses of different PUFs to a given challenge is ideally 0.5 (random). It is known as 'uniqueness' and described by μ_{inter}. Likewise, the average fraction of dissimilar bits between responses of a fixed PUF to a given challenge, known as 'reliability' (μ_{intra}), is ideally 0 (no error). To gauge these measures, we study the pairwise difference between HBN-PUF responses to various challenges; see Appendix C for details.

Figure 3 shows the number of unique challenge bitstrings yielding response pairs differing on average by the given fraction (bottom axis) or number of bits (top axis) for three different PUF sizes, $N = 64$, 256, and 1024. Two histograms are plotted, where the differences are calculated with respect to the same chip (red) and with respect to other chips (blue). Eight different chips were used to estimate μ_{inter}. There appears clear separation of intra- and inter-device distributions, indicating vanishing false-positive rate for authentication using both network sizes, especially as N increases. This means that our PUF is well-suited to authentication. Furthermore, fewer challenges ($\sim 1/N$) are required for authentication than with single-bit PUFs since the HBN-PUF produces N-bit responses.

In practice, we find that μ_{intra} is driven by a relatively small, fixed subset of nodes (where the subset depends on the chip and the response). We hypothesize that these nodes are in a metastable state at the measurement time t_{opt}, and that a cherry picking error correction scheme [17] that removes these error-prone bits from the response can be

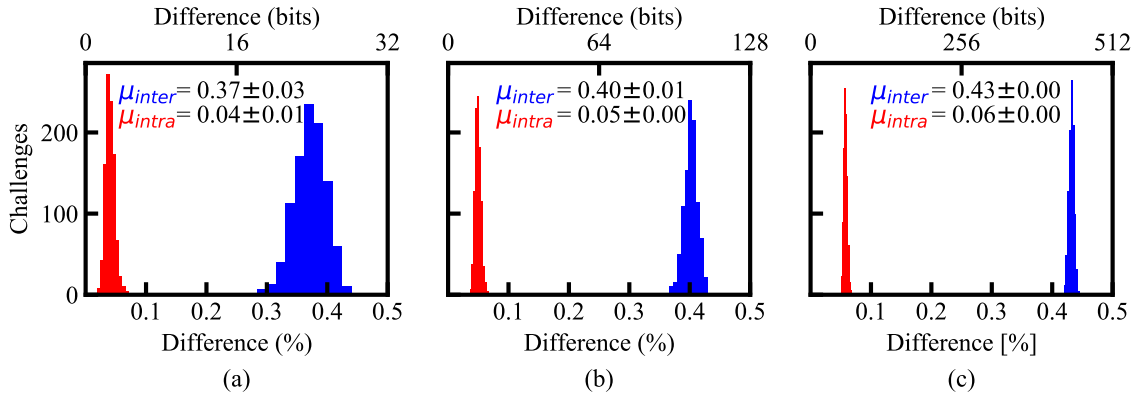

FIGURE 3. Challenge-response histograms for network sizes (a) $N = 64$, (b) $N = 256$, (c) $N = 1024$. We plot (8) and (9) (see Appendix C), the means of which are μ_{intra} and μ_{inter}, respectively.

highly effective. This is illustrated in III-D and will be studied more extensively in future work.

B. EXPONENTIAL SCALING OF ENTROPY WITH NETWORK SIZE

Entropy is of central importance in determining the cryptographic and security properties of a PUF [18]. The HBN-PUF, with its multiple bits per response, presents unique challenges to entropy estimation that will be discussed in future work, but in this section we apply previously reported entropy estimation techniques adapted to the HBN-PUF. A PUF can be idealized as a table that gives the response corresponding to a given challenge (called the 'CRP table' below). For most strong PUFs, the number of challenges (*i.e.*, the number of rows in the CRP table) grows as 2^N, and each response is a single bit so the CRP table for a given PUF realization can be described by a binary string of length 2^N. For the HBN-PUF, on the other hand, each row in the CRP table is itself an N-bit string so the entire CRP table is described by an $N2^N$-bit string. Estimating the distribution of binary strings of length $N2^N$ is infeasible even for relatively small N; however, we can apply entropy estimates from the PUF literature that make assumptions about this distribution–H_{min}, H_{joint}, and H_{CTW} (see Appendices D-F). We do not report the values of H_{CTW} below because in nearly all cases it produces full entropy and is never below H_{min} or H_{joint}.

The most basic measure is the minimum entropy H_{min}, which assumes no correlations between bits and responses and serves as a median. The joint entropy H_{joint} does not assume independence, but does assume that all correlations are pairwise and that no other higher-order correlations exist. Finally, the context-tree weighted entropy H_{CTW} serves as an upper bound by generating a minimum-length compressed binary string encoding the CRP behavior. We plot the first two of these quantities as a function of N in Fig. 4 and Table 1, observing that $H_{joint} \leq H_{min}$, which is true by definition.

Table 1 records the entropy and entropy density, ρ_{min} or ρ_{joint}, defined as the fraction of the observed entropy to the maximum possible entropy $N2^N$. We see that the entropy density for our median estimate H_{min} hovers around 0.6,

TABLE 1. Entropies $H_{joint} \leq H_{min}$ and entropy densities $\rho_i \sim H_i/(N2^N)$ for $N = 4 - 512$. Only H_{min} is estimated for $N > 8$.

N	H_{min}	ρ_{min}	H_{joint}	ρ_{joint}
4	17.74 ± 0.27	0.32	8.04 ± 0.59	0.14
5	80.69 ± 0.89	0.54	25.63 ± 1.74	0.17
6	197.26 ± 1.78	0.53	58.10 ± 3.70	0.16
7	479.40 ± 3.88	0.54	154.53 ± 9.01	0.18
8	1155.88 ± 9.29	0.57	398.78 ± 14.22	0.20
16	$(6.17 \pm 0.12) \times 10^5$	0.59		
32	$(8.18 \pm 0.08) \times 10^{10}$	0.60		
64	$(6.95 \pm 0.09) \times 10^{20}$	0.59		
128	$(2.56 \pm 0.02) \times 10^{40}$	0.59		
256	$(1.77 \pm 0.01) \times 10^{79}$	0.60		
512	$(4.06 \pm 0.00) \times 10^{156}$	0.59		

suggesting that the number of extractable bits is roughly $N2^N/2$ and hence that the min entropy scales super-exponentially with network size. Note however that there are theoretical bounds to the maximum entropy of PUFs and indeed any physical system, with arguments to be made that the entropy must be bounded polynomially by its size, such as the number of atoms [7]. What our measurements show is that in the range $N = 4 - 8$, for which entropy measures are calculated exactly over all possible CRPs, we observe super-exponential scaling with N. Outside this region, the entropy is computationally infeasible to calculate, and the reported values are extrapolations from limited measurements - which may not reflect the true entropy bounds of the system.

The inset to Fig. 4 illustrates the distribution of these entropy measures over 80 PUF classes for the exactly calculable network sizes $N = 4-8$ (see Appendix A). We observe that there is significant variation in the entropy estimates at very small PUF sizes, and that the joint entropy estimate in this region is approximately $15 - 20\%$ of full entropy. Note, however, that the joint entropy density increases and tightens as N increases. We expect it to approach ρ_{min} for larger networks.

We expect ρ_{joint} to approach ρ_{min} for two reasons. Firstly, larger networks ($N > 16$) consistently exhibit chaos, while small ABNs ($N \leq 8$) may enter non-chaotic periodic regimes [13] that induce pair-wise correlations. Secondly, there exist certain challenge strings that are steady-state fixed points. For the odd-input XOR functions used in this work, the all-zero

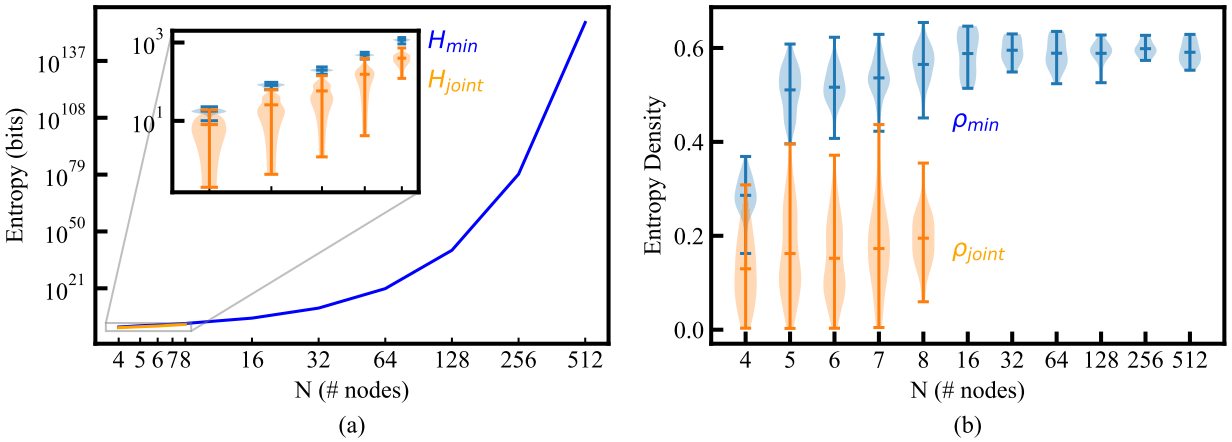

FIGURE 4. Entropy (a) measures and (b) densities of HBN-PUF classes as a function of network size N. Violin plots are over the distribution of classes, and solid lines indicate an average over classes.

and all-one challenge strings are fixed points; this can be seen since the output of the 3-XOR is zero or one if all its inputs are zero or one. (In the case of an even number of inputs, the all-one challenge is not a fixed point.) These trivial fixed points are filtered by our analysis, but there may exist other fixed points based on the details of the network wiring diagram that would need to be searched for via Boolean satisfiability algorithms which is not done in this work. We expect the density of these fixed points to go to zero as $N \to$ inf, but a non-negligible fraction of the challenge space at the industrially-irrelevant network sizes shown in the inset may be steady-state fixed points that reduces the entropy. We see some evidence of this in the observed tightening of both entropy distributions with increasing N, and by the super-exponential growth of the extrapolated H_{min} curve at larger sizes (see Appendix D).

Investigating these hypotheses and developing other means of estimating the entropy from limited samples for large networks is the subject of future work, as the exponential growth of the challenge space prevents full exploration even in principle.

C. MACHINE LEARNING ATTACK WITH PUFmeter

PUFmeter [19] is a recently designed machine learning platform used to assess the security of a PUF. It attempts to learn the challenge-response behavior of a given PUF using probably-approximately-correct learning, and indicates whether a PUF's behavior can be learned and hence is susceptible to various attacks without actually performing specific attacks. The theory behind PUFmeter is based upon single-bit responses. For this reason, we use PUFmeter to assess the security of an individual bit of our responses to an attack, as well as the XOR of our entire response string. These results are presented in Table 2.

In Table 2, κ is the minimum number of Boolean variables usable by PUFmeter to predict the response to a given challenge. Because $\kappa = 0$, PUFmeter is unable to model the behavior of the HBN-PUF. The noise upper bound, average sensitivity, and noise sensitivity are used to gauge the theoretical bounds for the types of attacks that are expected to be possible. From these results, PUFmeter indicates that an $N = 16$ HBN-PUF may be susceptible to a Fourier-based attack.

TABLE 2. PUFmeter machine-learning attack on an N = 16 node HBN-PUF with responses taken after 6 pairs of inverter gates, using PUFmeter parameters $\delta = 0.01$ and $\epsilon = 0.05$ governing the probability thresholds for the analysis. Abbreviations Noise Upper Bound (UB), Average Sensitivity (AS), and Noise Sensitivity (NS). The result $\kappa = 0$ indicates a failure of PUFmeter to model our PUF.

Response Bit	UB	AS	NS	κ
XOR	0.468	0.298	0.249	0
0th	0.469	0.316	0.246	0

Summarizing, the observed super-exponential entropy scaling, the presence of chaotic nonlinear dynamics, and the failure of PUFmeter to model our PUF suggests that the behavior of the HBN-PUF may be resilient to machine learning attack. We have attempted machine learning attacks, including deep learning-based methods and model-based attacks, which have also failed and will be described in future publications. Further study is required to explicitly rule out any given attack, such as Fourier-based attacks and side-channel attacks. In such cases, instantiating multiple HBN-PUFs on the chip may obscure the power supply draw or the EM radiation emitted due to the chaotic transients of nearby networks.

D. CHERRY PICKING AND TEMPERATURE VARIATION

A simple method of reducing errors is to mask out unreliable bits on a *per challenge and per device* basis, an approach known as cherry picking [17]. That is, at enrollment, each PUF is queried multiple times (100 in this case) and any bits that vary are discarded; the bit mask used to discard bits is stored as helper data for reconstructing the PUF response at query time. Fig. 5(a) shows the number of bits retained by this procedure (termed 'stable bits') as a function of measurement time. As can be seen, for measurement times up to about 7 ns, more than half of the 256 bits are stable at an error rate of less than 1%.

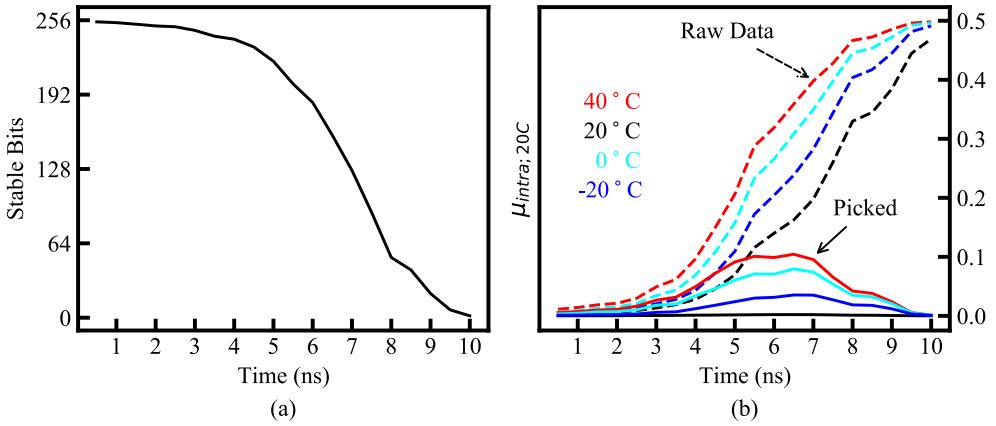

FIGURE 5. (a) The number of cherry picked stable bits vs. time for an $N = 256$ network. Stable bits are those that have less than 1% error rate. (b) μ_{intra} calculated with respect to a room temperature enrollment (20 ◦C) vs. measurement time for the same network when queried at different temperatures. Dashed lines correspond to μ_{intra} calculated without cherry picking, and solid lines are with cherry picking.

We illustrate the usefulness of this cherry picking approach when querying the HBN-PUF at different temperatures, which is an important practical concern when comparing PUFs in different environmental conditions or over long operating times [20]. A single $N = 256$ HBN-PUF on a single chip was enrolled at room temperature (20 °C), and μ_{intra} was calculated with respect to this enrollment at three additional temperatures (−20°C, 0 °C, and 40 °C, see Appendix G). This μ_{intra}; 20 °C is plotted vs. measurement time in Fig. 5(b) in dashed lines, and compared to a control of a second collection at 20 °C (black). We see that indeed there is an increased error rate compared to the control. It is significant in the case of raw data; however, the cherry picking procedure (solid lines) does significantly reduce the error due to temperature variation.

The HBN-PUF has some degree of environmental stability due to the use of the delay line for triggering the capture of the network state. Because the delay line is based on the same digital logic building blocks as the rest of the ABN, it is likely affected by temperature and voltage effects (*e.g.*, changing rise, fall, and signal propagation times) in a way similar to the rest of the ABN. Thus, if the entire network sped up or slowed down, the delay line would speed up or slow down in a commensurate way. Contrast this with, *e.g.*, an external, temperature-stabilized clock signal. Early designs using a clock signal rather than delayed reset showed μ_{intra} close to 50% for small temperature changes, but the delay line design is much more robust.

Strategies to reduce environmental variation, as well as experiments to test voltage sensitivity and aging effects, are future work for the HBN-PUF. Referring to Fig. 5, we see that there is a trade-space between entropy/response (*i.e.*, shorter measurement time corresponds to less entropy), error rate, bits/response, and effect temperature range that can be optimized over for specific applications. This observation suggests that we can trade some of those bits for error correction ability to reduce errors to a level needed for key exchange because of the large number of bits available per response. Moreover, the temperature effects *do not* appreciably change the overall behavior of the PUF. That is, there exists a t_{opt} (that is constant for a PUF class over temperature) corresponding to $\Delta\mu \sim 0.35$ at any given temperature; it is changes to the specific bitstream, not differences in qualitative behavior, that drives these errors. As a result, a temperature-aware enrollment protocol, in which the HBN-PUF is enrolled at multiple temperatures may be applicable [21].

IV. CONCLUSION AND FUTURE WORK

In summary, we present a novel HBN-PUF design that maps the challenge-response mechanism of the PUF onto the full state-space of a chaotic dynamical system (the HBN). The HBN-PUF represents an improvement in the state-of-the-art for strong PUFs several ways. First, to our knowledge, the HBN-PUF is the only strong PUF proposal that produces multiple bits per response, thus reducing time, network, and storage resources for authentication and key exchange. This will also likely frustrate machine learning attacks, as illustrated by our tests with PUFmeter, because the attacker will need to guess an N-dimensional Boolean vector instead of a one-dimensional one. Second, the HBN-PUF is fast: response readout occurs in less than 10ns, which combined with the multiple bits per response, means that Gbps key generation rates are easily achievable. Finally, the HBN-PUF is relatively insensitive to placement on the FPGA chip and resource usage scales linearly with the size of the PUF. As a result, $N = 1024$ or larger HBN-PUFs are easily realizable within resource constraints on modern low-end FPGAs (Cyclone V), but could produce upwards of 2^{1024} independent cryptographic keys at a rate of 100 Gbps. This is fast enough so that, for instance, modern communications networks could be one-time pad encrypted with HBN-PUF output, but with such a large CRP space that it would take many lifetimes of the universe to exhaust the entropy.

The HBN-PUF has many attractive properties that suggest that it could be a true, machine-learning resistant and practical strong PUF. However, there remain substantial questions to

be addressed in future work. The most obvious is further environmental testing and development of error mitigation strategies that are applicable to the HBN-PUF. On a more theoretical level, the multiple bits per response stress existing entropy estimation methods and will require new techniques to more accurately lower-bound the actual extractable entropy. Moreover, we need to test and confirm the hypothesis that HBN-PUFs are in fact chaotic to prove the security properties of the HBN-PUF. We have developed models of HBN-PUFs that can reproduce the behavior described here, which will appear in a follow up study, and we will use these models to execute model-based attacks to demonstrate machine learning resistance. In addition to this theoretical work, a study of the effects of the network layout (*e.g*, random vs. ring vs. other possible topologies) and detailed placement of the HBN-PUF elements in terms of the optimal measurement time and entropy per response will also appear in follow up work.

APPENDIX.

A. EXPERIMENTAL PROCEDURE

The HBN-PUF is created by coding our design using the hardware description language Verilog (code in Appendix H) using the Quartus CAD software, which compiles our code with automatic placement and routing chosen by its optimization procedure. We then program $N_{chips} = 8$ separate DE10-Nano SOCs hosting Cyclone V 5CSEBA6U23I7 FPGAs with the same *.sof* file. This ensures each FPGA instantiates an identical copy of our PUF in both layout and design, meaning the only variations of instances within a PUF class are due to variations in the manufacturing of the FPGAs.

For each network size N, we instantiate $N_{classes} = N_{graph}N_{loc}$ different HBN-PUF classes, where each class corresponds to a particular network topology randomly drawn from the set of possible regular graphs of degree 3 (N_{graph} draws) and/or a particular location of the PUF on the chip (N_{loc} PUFs per random graph). These draws are performed using custom Python scripts and the numpy.random module and written to the indicated positions in the Verilog file in Appendix H.

In order to reduce the dependence on random seeds in the CAD's optimization procedures for the experiments presented here, we fix the locations of the nodes in the network to specific logic elements on the chip (which are randomly chosen from within a grid) but nothing about the HBN-PUF's behavior requires detailed control of node placement. For $N \leq 16$, $N_{loc} = 16$; else, $N_{loc} = 3$. For all sizes, $N_{graph} = 5$. We create one *.sof* file per random graph and place N_{loc} PUFs at different locations (each with the same graph layout) on the chip in order to populate the distribution of HBN-PUFs. We find that the variation due to location is comparable to the variation due to graph layout, and so treat these on an equal footing; this yields a total of $N_{classes} = 80$ different HBN-PUF classes for $N \leq 16$ and 15 HBN-PUF classes for $N > 16$.

The Cyclone V chips that we use have an integrated hard processor running Linux. We therefore use Altera's Avalon interface to make the PUF accessible to the Linux system and collect CRPs using custom C code that presents $N_{challenges}$ to each PUF via this interface to set the initial state of a given HBN. The HBN is held at a challenge for several 200 MHz clock cycles due to synchronous controller logic and to stabilize the dynamics of the autonomous nodes. The network is then released and evolves for a short time during the transient phase, and the state is registered at a given delay time by choosing the length of the delay line via a multiplexer. The response is transferred and the PUF is reset to the same challenge. The entire process is repeated $N_{repeats} = 100$ times before moving to the next challenge, so that the total number of applied challenges to each HBN is equal to $N_{challenges} \times N_{repeats}$.

Peculiar to the XOR function, there are two steady-state fixed points corresponding to when the network is all 0 or all 1. These fixed points are discarded from the challenge space as they have no entropy, however they can be used to identify 'glitchy' PUF classes. That is, since the HBN-PUF violates most commonly accepted design rules (in particular the guidance against large combinatorial loops), occasionally the Quartus software produces glitchy designs. If a given PUF class does not produce all-ones or all-zeros as the response to an all-one or all-zero challenge, we discard the PUF class from consideration. This occurs approximately 10% of the time. All metrics are calculated using the valid challenges, $N_{vc} = 2^N - 2$. For $N < 16$, $N_{challenges} = N_{vc}$. For $N \geq 16$, $N_{challenges} = 1000$ unique and randomly selected valid challenges. In all cases, $N_{repeats} = 100$.

These parameters are used for all experimental data collection unless otherwise noted.

B. FORMAL CHALLENGE-RESPONSE DEFINITIONS

Let $P \in \mathbb{P}$ be a particular PUF instance P belonging to the set of all PUF instances $\mathbb{P}$ of a particular PUF class. The response $\mathbf{R}$ is a random variable $\mathbf{R} : \mathbb{S}_P \rightarrow \{0,1\}^N$ mapping from the set of all possible physical states $\mathbb{S}_P$ of PUF instance P to the set of all binary strings of length N, denoted $\{0,1\}^N$. Specifically, the response takes as input a particular state $S_{P,C} \in \mathbb{S}_P$ of PUF instance P resulting from challenge $\mathbf{C} \in \{0,1\}^N$.

We characterize the reliability and uniqueness of $\mathbb{P}$ by studying the distributions of $\mathbf{R}$ for various P and $\mathbf{C}$. That is, we study how our design performs as a PUF by comparing responses from individual and different instances on a per-challenge basis using the metrics defined in the next appendix.

C. INTRA- AND INTER-DEVICE STATISTICS DEFINITIONS

The degree to which two binary strings are different is given by the Hamming distance:

$$D(\mathbf{A}, \mathbf{B}) = \sum_{i=1}^{N} \mathbf{A}(\mathbf{i}) \oplus \mathbf{B}(\mathbf{i}), \tag{5}$$

where $\mathbf{A}$ and $\mathbf{B}$ are the two binary strings to compare, of length N, $\mathbf{A}(\mathbf{i})$ and $\mathbf{B}(\mathbf{i})$ refer to the i-th bits of $\mathbf{A}$ and $\mathbf{B}$, respectively, and $\oplus$ is the XOR function. For random strings, the Hamming distance is on average $N/2$. Moreover, it is convenient to normalize the Hamming distance by N: $d(\mathbf{A}, \mathbf{B}) = D(\mathbf{A}, \mathbf{B})/N$. For random strings $\mathbf{A}$ and $\mathbf{B}$, $d(\mathbf{A}, \mathbf{B}) = 1/2$.

Consider two different responses from the same challenge string $\mathbf{C}_c$. These responses may result from applying the same challenge string to the same PUF instance (indexed by p) two different times (indexed by r for repetition), $\mathbf{R}_c^{p,r}$ and $\mathbf{R}_c^{p,r'}$, or they may result from applying the challenge exactly once to two different PUF instances, $\mathbf{R}_c^{p,r}$ and $\mathbf{R}_c^{p',r}$. Repeated application used to gauge reliability: a single PUF instance should ideally produce identical responses when presented with the same challenge (*i.e.*, $d(\mathbf{R}_i^{p,r}, \mathbf{R}_i^{p,r'}) = 0$ for all p, r, and r'). Applying the same challenge to different PUF instances is used to gauge uniqueness: two different PUF instances should give responses to the same challenge which, when compared, appear random and uncorrelated. In terms of Hamming distances, $d(\mathbf{R}_i^{p,r}, \mathbf{R}_i^{p',r}) \approx 1/2$ (although this does not capture correlations in bits).

For clarity we summarize these indices:

- $c \in [0, N_{challenges})$: Distinct challenge;
- $r, r' \in [0, N_{repeats})$: Separate applications of distinct challenge;
- $p, p' \in [0, N_{chips})$: Separate PUF instances.

If we take each response to be an N-bit string, then the fraction of dissimilar bits between the two responses is denoted as

$$\mathfrak{R}(c, p, r, r') = d(\mathbf{R}_c^{p,r}, \mathbf{R}_c^{p,r'}), \tag{6}$$

$$\mathfrak{U}(c, p, p', r) = d(\mathbf{R}_c^{p,r}, \mathbf{R}_c^{p',r}). \tag{7}$$

Above, $\mathfrak{R}$ (mnemonic 'reliability') is the intra-device fractional Hamming distance between responses for the fixed PUF instance p resulting from applications r and r' of challenge c. Likewise, $\mathfrak{U}$ (mnemonic 'uniqueness') is the inter-device fractional Hamming distance between responses of PUF instances p and p' resulting from the fixed application r of challenge c.

To obtain distributions of these distances on a per-challenge basis, we average over the pairwise combinations used to construct them, and then further average over the remaining indices to obtain mean measures of reliability μ_{intra} and uniqueness μ_{inter}. Specifically, if we let $\langle\cdot\rangle_{a,b}$ indicate the average of a quantity over indices a, b, then

$$\mathfrak{r}(c) = \langle \mathfrak{R}(c, p, r, r') \rangle_{r,r',p}, \tag{8}$$

$$\mathfrak{u}(c) = \langle \mathfrak{U}(c, p, p', r) \rangle_{p,p',r}. \tag{9}$$

We record a time series of N-bit strings representing the time evolution of the network, so that the metrics introduced above exist at every measurement time. If we wish to measure the reliability on a per-chip basis, we simply do not average over p in (8).

TABLE 3. An illustration of response-bit ordering for $N = 3$, where there are $3 \times 6 = 18$ total bits.

Challenge	Node 1	Node 2	Node 3
001	x_1	x_2	x_3
010	x_4	x_5	x_6
011	x_7	x_8	x_9
100	x_{10}	x_{11}	x_{12}
101	x_{13}	x_{14}	x_{15}
110	x_{16}	x_{17}	x_{18}

Fig. 3 shows the histograms of (8) and (9) at time t_{opt}. We further summarize the reliability and uniqueness as single numbers by averaging (8) and (9) over challenges, *i.e.*,

$$\mu_{intra} = \langle \mathfrak{r}(c) \rangle_c, \tag{10}$$

$$\mu_{inter} = \langle \mathfrak{u}(c) \rangle_c. \tag{11}$$

D. MINIMUM ENTROPY

The min-entropy of a random variable X is defined as

$$H_{min}(X) = -\log(p_{max}(X)), \tag{12}$$

where $p_{max}(X)$ is the probability of the most likely outcome. If $X = (x_1, x_2, \ldots, x_n)$ is a vector of n independent random variables, then the min-entropy is

$$H_{min} = \sum_{i=1}^{n} -\log(p_{max}(x_i)). \tag{13}$$

In the case of a strong PUF with multiple challenges and a large response space, we need an ordering of the response bits in order to make sense of entropy calculations. A natural ordering is to define the response of the i-th node to the j-th challenge as x_{jN+i}, where the challenges are ordered lexicographically. This is illustrated in Table 3 for the simple case of $N = 3$. Here, there are only 6 challenges because we omit the all-0 and all-1 challenges as discussed in Appendix A.

Assuming independence of x_i, the min-entropy for the HBN-PUF can be readily calculated with (13) from empirical estimates of $p_{max}(x_i)$ [4], [22]. For each x_i, the estimate of $p_{max}(x_i)$ is simply the observed frequency of 0 or 1, which ever is larger. To put the entropy calculations into context, we also present them as a fraction of the optimal case. If all of the x_i were independent and completely unbiased, *i.e.*, each x_i were equally likely to be 0 or 1 (*i.e.*, $p(x_i) = 1/2$), then the min-entropy would be equal to N times the number of valid challenges N_{vc}. We therefore define the min-entropy density as

$$\rho_{min} = H_{min}/(NN_{vc}). \tag{14}$$

Due to the exponential scaling of the challenge space, we do not measure these values using all of the possible valid challenges for $N > 8$. This is because of the computing time required in both calculating the entropy measures and obtaining the full CRP space. For $N > 8$, we randomly choose challenges from a representative sample and multiply by the fraction of the unused space to obtain H_{min}.

In the next appendix, we study the full challenge space for low N.

E. JOINT ENTROPY

In the previous appendix, we assume hat x_i are independent, though this need not be the case. It is possible that some bits reveal information about others, reducing the entropy. Here we study these correlations between bit pairs, first by calculating the mutual information defined as

$$I(x_i, x_j) = \sum_{x_i, x_j} p(x_i, x_j) \log[\frac{p(x_i, x_j)}{p(x_i)p(x_j)}] \tag{15}$$

between all pairs of x_i, x_j. Unlike min-entropy, the mutual information is difficult to calculate for higher N, so we will restrict our attention to $N = 4 - 8$ and use the full valid challenge space.

An adversary can use knowledge of any structure in the mutual information to more effectively guess response bits, thereby reducing the available entropy. In particular, the entropy is reduced to [18]

$$H_{joint} = H_{min} - \sum_{i=0}^{n-1} I(x_i, x_{i+1}), \tag{16}$$

where the ordering of the bits is such that the penalty is as large as possible. Calculating the ordering of the bits to maximize the joint information penalty is effectively a traveling salesman problem, which we solve approximately with a 2-opt algorithm [23].

F. CONTEXT-TREE WEIGHTING TEST

In this appendix, we estimate the entropy through a string compression test. The results here should be understood as an upper-bound for the true entropy, especially for larger N. In particular, we consider the context tree weighting (CTW) algorithm [24].

The CTW algorithm takes a binary string called the context and forms an ensemble of models that predict subsequent bits in the string. It then losslessly compresses subsequent strings into a codeword using the prediction model. The size of the codeword is defined as the number of additional bits required to encode the PUF instance's challenge-response behavior. If the context contains information about a subsequent string, then the codeword will be of reduced size.

In the case of PUFs, the codeword length approaches the true entropy of the generating source in the limit of unbounded tree depth [25]. However, the required memory scales exponentially with tree depth, so it is not computationally feasible to consider an arbitrarily deep tree in the CTW algorithm. Instead, we vary the tree depth up to 20 to optimize the compression.

We perform a CTW compression as follows:

- We collect data for $N = 4 - 8$ HBN-PUFs with $N_{repeats} = 1$.
- We concatenate the resulting measurements for all but one PUF instances into a 1D string of length $(N_{chips} - 1)N_{vc}N$ to be used as context.
- We apply the CTW algorithm to compress the measurements from the last PUF with the context, using various tree depths to optimize the result.
- We repeat steps 2-3, omitting measurements from a different PUF instance, until all PUFs have been compressed.

The final entropy estimate is the average codeword length from all of the compression tests described above. If the behavior of the $N_{chips} - 1$ PUF instance can be used to predict the behavior of the unseen instance, then the PUFs do not have full entropy.

G. TEMPERATURE VARIATION

We calculate at each temperature the deviation of an HBN-PUF with respect to itself at 20 °C, a quantity which we denote μ_{intra}; 20 °C. This measure is equivalent to considering an individual chip as consisting of different instances - one for each temperature. It is calculated at each temperature by comparing responses to those generated at 20 °C, then averaging over all challenges. These plots are presented in Fig. 5 as a function of t, the number of inverter gates after which the response is registered. Each curve is a separate temperature.

H. HARDWARE DESCRIPTION LANGUAGE CODE

This Verilog code is used for synthesizing the HBN in Fig. 1.

```
//This module corresponds to the node zoom-in of Fig. 2.
module Node(reset, challenge, in1, in2, in3, out);

   input reset;
   input challenge;
   input in1;
   input in2;
   input in3;
   output out;

   wire node_clocked;
   wire node_asynchronous;

   assign node_asynchronous = in1 ^ in2 ^ in3;
   assign node_clocked = reset ? challenge : node_asynchronous;
   assign out = node_clocked;

endmodule


module HBN(

    clk,
  reset,
    challenge,
  delay_address,
    response,
  ready

);

parameter N = 256;     // HBN Size
```

```
parameter INDEX = 0; // Index on the chip (0..2 or 0..15 depending on N)
parameter MEASUREMENT_DELAY = 20;

localparam DELAY_ADDRESS_BITS = $clog2(MEASUREMENT_DELAY);

input            clk;
input            reset;
input [N-1:0]    challenge;
input [DELAY_ADDRESS_BITS:0]  delay_address;
output [N-1:0]   response;
output reg       ready;


wire [N-1:0] ring_state;
wire [N-1:0] ring [3];
wire         reset_buf /*synthesis keep*/;
wire         delayed_reset /*synthesis keep*/;


assign reset_buf = reset;

// create N nodes
genvar i;
generate
for (i=0; i<N; i=i+1) begin : generate_ring
   Node n (

      .reset(reset_buf),
     .reset_delay(delayed_reset),
     .challenge(challenge[i]),
     .in1(ring[0][i]),
     .in2(ring[1][i]),
     .in3(ring[2][i]),
     .out(ring_state[i]),
     .response(response[i])

  );
end
endgenerate


always @(posedge clk) begin
  if( reset ) ready <= 0;
  else ready <= ~delayed_reset;
end

AddressableDelayLine #(MEASUREMENT_DELAY) DLM (reset, delay_address, delayed_reset);

/// AUTO-GENERATED CODE TO DEFINE THE WIRING DIAGRAM
assign ring[0][0] = ring_state[5];       // 0th input of node 0
assign ring[1][0] = ring_state[37];      // 1st input of node 0
assign ring[2][0] = ring_state[131];     // 2nd input of node 0

/// ...and so-on...
```

This Verilog code is used for synthesizing the tapped-delay line in Fig. 1.

```
// This is the tapped delay line in Fig. 1. Reconstructing the time series requires resetting the PUF and incrementing the delay_address
module AddressableDelayLine(
   in,
   delay_address,
   out
);

parameter N = 5;     // # of pairs of inverters.
localparam DELAY_ADDRESS_BITS = $clog2(N);

input                          in;
input[DELAY_ADDRESS_BITS:0]    delay_address;
output                         out;

wire [2*N-1:0] delay /*synthesis keep*/;

assign delay[0] = in;
assign out = delay[2*delay_address-1];

genvar i;
    generate for (i=0; i<2*N-1; i=i+1) begin : generate_delays
       assign delay[i+1] = ~delay[i];
    end
    endgenerate

endmodule
```

REFERENCES

[1] R. Maes, *Physically Unclonable Functions: Concept and Constructions*. Berlin, Germany: Springer, 2013, ch. 2, pp. 25–48. [Online]. Available: http://link.springer.com/10.1007/978-3-642-41395-7_2

[2] T. McGrath, I. E. Bagci, Z. M. Wang, U. Roedig, and R. J. Young, "A PUF taxonomy," *Appl. Phys. Rev.*, vol. 6, no. 1, Mar. 2019, Art. no. 011303. [Online]. Available: http://aip.scitation.org/doi/10.1063/1.5079407

[3] J. Zhang, T. Q. Duong, A. Marshall, and R. Woods, "Key generation from wireless channels: A review," *IEEE Access*, vol. 4, pp. 614–626, 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7393435/

[4] D. E. Holcomb, W. P. Burleson, and K. Fu, "Power-up SRAM state as an identifying fingerprint and source of true random numbers," *IEEE Trans. Comput.*, vol. 58, no. 9, pp. 1198–1210, Sep. 2009. [Online]. Available: http://ieeexplore.ieee.org/document/4674345/

[5] R. Pappu, "Physical one-way functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, Sep. 2002. [Online]. Available: https://www.sciencemag.org/lookup/doi/10.1126/science.1074376

[6] S. S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls, "Extended abstract: The butterfly PUF protecting IP on every FPGA," in *Proc. IEEE Int. Workshop Hardw.-Oriented Secur. Trust*, Jun. 2008, pp. 67–70. [Online]. Available: http://ieeexplore.ieee.org/document/4559053/

[7] U. Rührmair, H. Busch, and S. Katzenbeisser, *Strong PUFs: Models, Constructions, and Security Proofs* (Information Security and Cryptography). Berlin, Germany: Springer, 2010, ch. 4, pp. 79–96. [Online]. Available: http://link.springer.com/10.1007/978-3-642-14452-3_4

[8] K. Gołofit and P. Wieczorek, "Chaos-based physical unclonable functions," *Appl. Sci.*, vol. 9, no. 5, p. 991, Mar. 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/5/991

[9] T. Tuncer, "The implementation of chaos-based PUF designs in field programmable gate array," *Nonlinear Dyn.*, vol. 86, no. 2, pp. 975–986, Oct. 2016. [Online]. Available: http://link.springer.com/10.1007/s11071-016-2938-3

[10] L. Chen, "A framework to enhance security of physically unclonable functions using chaotic circuits," *Phys. Lett. A*, vol. 382, no. 18, pp. 1195–1201, May 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S037596011830255X

[11] E. Ott, *Fractal Basin Boundaries*. Cambridge, U.K.: Cambridge Univ. Press, Aug. 1993, ch. 5.1, pp. 152–157.

[12] D. P. Rosin, D. Rontani, and D. J. Gauthier, "Ultrafast physical generation of random numbers using hybrid Boolean networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 87, no. 4, Apr. 2013, Art. no. 040902. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.87.040902

[13] D. P. Rosin, "Autonomous Boolean networks on electronic chips," in *Dynamics of Complex Autonomous Boolean Networks*. Cham, Switzerland: Springer, 2015, doi: 10.1007/978-3-319-13578-6_3.

[14] R. Zhang, H. L. D. S. D. Cavalcante, Z. Gao, D. J. Gauthier, J. E. S. Socolar, M. M. Adams, and D. P. Lathrop, "Boolean chaos," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 80, no. 4, Oct. 2009, Art. no. 045202. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.80.045202

[15] M. Ghil and A. Mullhaupt, "Boolean delay equations. II. Periodic and aperiodic solutions," *J. Stat. Phys.*, vol. 41, nos. 1–2, pp. 125–173, Oct. 1985. [Online]. Available: http://link.springer.com/10.1007/BF01020607

[16] H. L. D. D. S. Cavalcante, D. J. Gauthier, J. E. S. Socolar, and R. Zhang, "On the origin of chaos in autonomous Boolean networks," *Phil. Trans. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 368, no. 1911, pp. 495–513, Jan. 2010. [Online]. Available: https://royalsocietypublishing.org/doi/10.1098/rsta.2009.0235

[17] M. Hiller, M.-D. Yu, and G. Sigl, "Cherry-picking reliable PUF bits with differential sequence coding," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 9, pp. 2065–2076, Sep. 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7480444/

[18] R. Maes, *PUF-Based Key Generation*. Berlin, Germany: Springer, 2013, ch. 6, pp. 143–168. [Online]. Available: http://link.springer.com/10.1007/978-3-642-41395-7_6

[19] 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. 122513–122521, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8819883/

[20] S. K. Mathew, S. K. Satpathy, M. A. Anders, H. Kaul, S. K. Hsu, A. Agarwal, G. K. Chen, R. J. Parker, R. K. Krishnamurthy, and V. De, "16.2 a 0.19pJ/b PVT-variation-tolerant hybrid physically unclonable function circuit for 100% stable secure key generation in 22 nm CMOS," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 278–279. [Online]. Available: http://ieeexplore.ieee.org/document/6757433/

[21] Y. Gao, Y. Su, L. Xu, and D. C. Ranasinghe, "Lightweight (reverse) fuzzy extractor with multiple reference PUF responses," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 7, pp. 1887–1901, Jul. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8574914/

[22] P. Simons, E. van der Sluis, and V. van der Leest, "Buskeeper PUFs, a promising alternative to d flip-flop PUFs," in *Proc. IEEE Int. Symp. Hardw.-Oriented Secur. Trust*, Jun. 2012, pp. 7–12. [Online]. Available: http://ieeexplore.ieee.org/document/6224311/

[23] B. Chandra, H. Karloff, and C. Tovey, "New results on the old K-opt algorithm for the traveling salesman problem," *SIAM J. Comput.*, vol. 28, no. 6, pp. 1998–2029, Jan. 1999. [Online]. Available: http://epubs.siam.org/doi/10.1137/S0097539793251244

[24] F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens, "The context-tree weighting method: Basic properties," *IEEE Trans. Inf. Theory*, vol. 41, no. 3, pp. 653–664, May 1995.

[25] T. Ignatenko, G.-J. Schrijen, B. Skoric, P. Tuyls, and F. Willems, "Estimating the secrecy-rate of physical unclonable functions with the context-tree weighting method," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2006, pp. 499–503. [Online]. Available: http://ieeexplore.ieee.org/document/4036011/

NOELOIKEAU CHARLOT received the B.S. degree in physics and biological engineering from the University of Hawai'i at Manoa, Honolulu, HI, USA, in 2017, and the M.S. degree in physics from The Ohio State University, Columbus, OH, USA, in 2020, where he is currently pursuing the Ph.D. degree in physics.

He was a Research Assistant with the QuantInfo Laboratory and CYAN cybersecurity collaboration at OSU. He interned with the Gravitational Astrophysics Laboratory, NASA Goddard and the Dark Matter Detection Laboratory, UH Manoa. He is a prior McNair Scholar. His industrial experience includes bioreactor design and photonics with ProtaCulture, LLC. His research interests include network science, artificial intelligence, ultrafast electronics, and quantum gravity.

DANIEL CANADAY was born in Columbus, OH, USA, in 1991. He received the B.S. degree in physics and mathematics, and the M.S. and Ph.D. degrees in physics from Ohio State University, Columbus, in 2014, 2017, and 2019, respectively.

He is currently a Scientist with Potomac Research, LLC, Alexandria, VA, USA. His research is concerned with applied reservoir computing and the application of physical neural networks to cryptography.

ANDREW POMERANCE was born in Washington, DC, USA, in 1980. He received the B.S. and M.S. degrees in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2002, and the Ph.D. degree in physics from the University of Maryland, College Park, MD, USA, in 2009.

From 2009 to 2013, he was with Raytheon Applied Signal Technology, Tyson's Corner, VA, USA. He is currently the President of Potomac Research, LLC, Alexandria, VA, USA. His research is concerned with nonlinear dynamics with applications to machine learning and cryptography.

DANIEL J. GAUTHIER (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the University of Rochester, Rochester, NY, USA, in 1982, 1983, and 1989, respectively. His Ph.D. research on instabilities and chaos of laser beams propagating through nonlinear optical media was supervised by Prof. R.W. Boyd and supported in part through a University Research Initiative Fellowship. From 1989 to 1991, he developed the first CW two-photon optical laser as a Postdoctoral Research Associate under the mentorship of Prof. T.W. Mossberg at the University of Oregon. He joined the Faculty of Duke University, Durham, NC, USA, as an Assistant Professor of physics, in 1991. He was named as a Young Investigator of the U.S. Army Research Office, in 1992, and the National Science Foundation, in 1993. From 2011 to 2015, he was the Robert C. Richardson Professor of Physics at Duke. He was a Chair of the Duke Physics Department, from 2005 to 2011, and an Interim Chair, in spring 2015. He was a Founding Member of the Duke Fitzpatrick Institute for Photonics. He moved to The Ohio State University, in 2016. He is currently a Professor of physics and electrical and computer engineering with The Ohio State University. His research interests include reservoir computing, synchronization and control of the dynamics of complex networks in electronic and optical systems, quantum communication, and nonlinear quantum optics. He is a Fellow of the Optical Society of America and the American Physical Society.