A Noise Bifurcation Architecture for Linear Additive Physical Functions

Meng-Day (Mandel) Yu[*+] **Ingrid Verbauwhede**[+] **Srinivas Devadas**[†]
David M'Raïhi[*]

{myu, david}@verayo.com
[*]Verayo, Inc.
San Jose, CA, USA

ingrid.verbauwhede@esat.kuleuven.be
[+]COSIC, KU Leuven
Leuven, Belgium

devadas@mit.edu
[†]MIT
Cambridge, MA, USA

Abstract— Physical Unclonable Functions (PUFs) allow a silicon device to be authenticated based on its manufacturing variations using challenge/response evaluations. Popular realizations use linear additive functions as building blocks. Security is scaled up using non-linear mixing (e.g., adding XORs). Because the responses are physically derived and thus noisy, the resulting explosion in noise impacts both the adversary (which is desirable) as well as the verifier (which is undesirable). We present the first architecture for linear additive physical functions where the noise seen by the adversary and the noise seen by the verifier are *bifurcated* by using a randomized decimation technique and a novel response recovery method at an authentication verification server. We allow the adversary's noise $\eta_a \rightarrow 0.50$ while keeping the verifier's noise η_v constant, using a parameter-based authentication modality that does not require explicit challenge/response pair storage at the server. We present supporting data using 28*n*m FPGA PUF noise results as well as machine learning attack results. We demonstrate that our architecture can also withstand recent side-channel attacks that filter the noise (to clean up training challenge/response labels) prior to machine learning.

Keywords- Architecture, Authentication, Machine Learning, Physical Security, Side Channel

I. INTRODUCTION

Physical Unclonable Functions (PUFs) allow a silicon device to be authenticated using a challenge/response evaluation of its manufacturing variations [4]. PUF building blocks are often linear and additive functions. To achieve security against machine learning attacks [5,7,9,14], building blocks are often composed in a non-linear fashion, as in the XOR Arbiter PUF [17] and Lightweight Secure PUF [10]. More recently, machine learning attacks coupled with side channel information [1,2,8,15,16] have been developed. Current proposed methods to address these attacks include increasing amount of non-linear mixing (e.g., adding XORs [17]) at PUF output. While this increases machine learning attack complexity for the adversary, it also increases authentication noise for the verifier. It is, therefore, desirable to derive and analyze a PUF construction that induces detrimental learning noise as seen by the adversary while *simultaneously* limiting authentication noise as seen by the verifier. It will be advantageous if such a PUF construction also addresses some of the recently published side channel leaks that aid machine learning via noise filtering and reduction.

II. MACHINE LEARNING AND NOISE

A. *Learning With Noise*

In our context, the purpose of machine learning is to predict the response to a previously unseen (and randomly chosen) challenge. The machine learning-equipped adversary obtains a set of labels $\{x, f(x)\}$, where x is a challenge (e.g., 64 bits or 128 bits), and $f(x)$ is a single bit response. The adversary attempts to build a predictor $g(x)$ during the *training phase*. The quality of the predictor would obviously depend on the amount of noise in (or the accuracy of) the labels. During the *attack phase*, the adversary would like to see a small classification (prediction) error ε_c, which is the probability that $g(x) \neq f(x)$ for a previously unseen and randomly chosen challenge x.

A relationship between machine learning complexity and noise of input labels was established in Kearns' seminal work [6] on the Statistical Query model of machine learning, where machine learning labels that are used to train $g(.)$ are subject to random noise bit corruption characterized by η, $0 \leq \eta \leq 0.5$. A value of $\eta = 0$ means that none of the labels, and specifically $\{f(x)\}$, are corrupted. An $\eta = 0.5$ means that $\{f(x)\}$ values are completely corrupted by random noise. As suggested in [6], learning complexity, e.g., learning time, is polynomial to inverse of noise, which is expressed follows:

$$LC(\eta) \propto \left(\frac{1}{1-2\eta}\right)^e \qquad 0 \leq \eta \leq 0.5, e \geq 1 \quad (1)$$

While Eqn. 1 seems to indicate that increasing η will increase the adversary's learning difficulty, increasing η also requires the verifier to relax the threshold for authentication, implying that the adversary does not need to learn with the same accuracy. It would, therefore, seem useful to develop a PUF architecture that can *bifurcate* noise η into two aspects:

i. η_a, average noise seen by the *adversary*; and
ii. η_v, average noise seen by the *verifier*.

Ideally, we want to keep η_v constant while allowing security scaling by making η_a approach 0.5 (at 0.5, adversary cannot distinguish PUF output bits from random). None of the papers published to date using linear additive physical building blocks have attempted this, for it seems obvious that the adversary and the verifier *should* see the same noise. We use a randomized decimation approach, where out of every D bits generated by a PUF, D-1 bits are randomly thrown away inside the device, and only 1 bit is sent outside the device. The adversary does not know the precise challenge x associated with the post-decimated response bit, and therefore has to make *imperfect inferences* to form the challenge/response labels $\{x, f(x)\}$ which are needed as input for the machine learning training phase. Our architecture allows for a novel response recovery method such that the noise seen by the verifier η_v remains constant as a function of the decimation factor D, while the adversary's noise η_a asymptotically approaches 0.5.

B. *Example PUF Noise at 28nm*

In Table I, we show average environmental noise η_e in a 4-XOR PUF 28*n*m FPGA implementation subject to a wide operating temperature range spanning -65°C to 125°C ambient. To the best of our knowledge, PUF implementation results below 40*n*m have not yet been published. The implementation is linear additive 4-XOR using 64-stage k-sum RO architecture in [18], in a 28 nm Xilinx Artix-7 FPGA.

TABLE I. *NOISE VS. TEMPERATURE, 4-XOR (REFERENCE @ 25°C)*

Temperature	-65°C	-45°C	25°C	85°C	125°C
η_e	0.217	0.195	0.046	0.241	0.317

Now let's consider the un-bifurcated case, where $\eta_a = \eta_v = \eta_e$. While a PUF's environmental noise η_e varies depending on implementation specifics, process node, and operating environmental specifications, the data above shows that the number of XORs cannot be scaled without severely impacting authentication noise, especially as the environmental difference between the provisioning (reference) condition and a regeneration (authentication) condition grows. 8-XOR PUFs were not successfully attacked in [14] but attacks proposed in [8] and [16] may require a move to 12, 16 or more XORs. At 85°C, a 4-XOR 28nm implementation has a noise of $\eta_e = 0.24$. At 12-XOR, the same implementation would have a noise level close to the reliability limiting "wall" at 0.50. In our architecture, we bifurcate η and allow η_v to stay constant as security is scaled up.

III. RELATED WORK

A. Physical Unclonable Function

Gassend et al. in [4] coined the term Physical Unclonable Function, and presented the first silicon PUF with integrated measurement circuitry. Many other silicon realizations of PUFs have been proposed; of these, the Arbiter PUF [4], Lightweight PUF [10], Ring-Oscillator-based [17] k-sum PUF [18], and Slender PUF [11] use linear additive building blocks.

It has been observed by [3,12] that storage requirements corresponding to CRPs in PUF authentication may be reduced if the verifier instead stores a secret model for the PUF, by which it can emulate and predict arbitrary responses of the PUF. The server no longer needs to maintain an explicit challenge/response database that grows with the number of authentication events, but instead maintains a *constant-size* database per device. Our architecture is confined to this usage modality (herein referred to as *parameter-based authentication*), so that the server can predict the response to an arbitrary challenge and the device and server can perform an exchange of challenge seed for each authentication.

B. Machine Learning and Related Side Channel Attacks

Machine learning attacks on linear additive PUFs were proposed in [7,9] and certain constructs broken in [14]. Last year, machine learning attacks coupled with side channel information [1,2,8,15,16] became an active area of research. Our architecture addresses certain side-channel attacks, specifically power and fault injection attacks that require *repeated* measurements.

IV. RESPONSE DECIMATION ARCHITECTURE

A. Background

Arbiter PUF. Arbiter PUFs were introduced in [4], shown in Figure 1. We shall refer to it as "basic" Arbiter PUF when we want to distinguish it from more complex constructs to be discussed later. PUF output is derived from a race condition formed by successive delay stages each comprising a crossbar switch formed using two 2:1 multiplexers. A challenge bit x^i, $0 \leq i < k$, for each of the k stages determines whether parallel path or cross path is active. Collectively, k challenge bits determine which path is chosen to create the transition at the top input to the arbiter latch, and which is used to create the bottom input. The difference comparison between the two delays determines whether the PUF produces a "1" output bit or a "0" output bit. The layout of the design has to be symmetrically matched such that random manufacturing variation would affect the response.

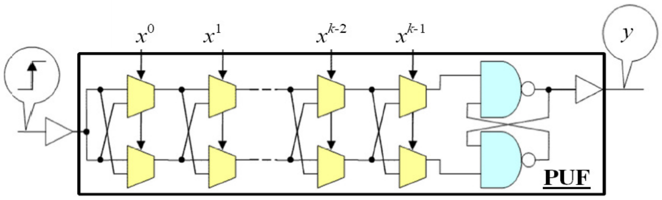

Figure 1: Basic Arbiter PUF Structure

An n-way XOR Arbiter PUF is constructed using n copies of structure above.

k-sum PUF. In the k-sum PUF [18], each of the k stages above is replaced with a pair of identically manufactured ring oscillators. Delay propagation is mimicked by using digital logic to add an appropriate delay value for each stage depending on the challenge bit associated with that stage. From a machine learning attack standpoint, the k-sum PUF and Arbiter PUF can be reduced to the same topology.

Parameter-based Authentication. A basic k-stage Arbiter PUF can be modeled with $k + 1$ parameters [4,7], with k parameters in the machine learning algorithm representing difference delays in each stage, and one parameter representing a systematic DC offset. During provisioning, instead of a server obtaining and storing CRPs explicitly for future use, the server instead obtains enough CRPs to model a basic PUF.[1] This can be repeated n times for an n-XOR PUF. Instead of storing the explicit CRPs of the combined n-XOR PUF, the server stores the $n \cdot (k + 1)$ parameter values that constitute a PUF parameter "snapshot" (p_{ss}); this allows the server, acting as the verifier, to synthesize *any* CRP during a later authentication event. For the k-sum implementation, k delay difference counter values need to be stored for each basic k-sum PUF, and there are n of these.

We note that there needs to be a secure provisioning mechanism to extract p_{ss} from a device, after which the extraction mechanism needs to be disabled (e.g., via a fuse). The server securely stores p_{ss} instead of explicit CRPs. During authentication verification, the server synthesizes responses from p_{ss} and compares against responses from the device.

B. Goal

Consider an n-XOR PUF, shown in Figure 2. A challenge value x_i is applied as input to produce a single bit value y_i. For illustrative purposes, we assume $|x_i| = n \cdot k$ ($| \, . \, |$ is the size operator, expressed in bits). For an n-XOR PUF, each of n basic PUFs produces a 1-bit response that is bit-wise XOR'ed to produce the composite single bit y_i. The resulting

[1] The output of a basic PUF, prior to the application of any XOR mixing, can be learned to an accuracy of greater than 99% according to data in [15].

challenge/response pair x_i, y_i constitutes a single label in the training set $\{x, f(x)\}$. In practice, to generate multiple response bits, $x_{0\ldots r-1}$[2] is derived from a seed value c_{seed} that serves as the initial value for a digital circuit, e.g., LFSR-based circuit (not shown), to produce $y_{0\ldots r-1}$, where r is the response length.

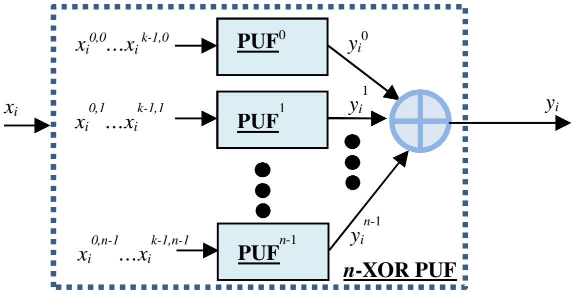
$x_i^{0,0}\ldots x_i^{k-1,0}$
PUF^0
y_i^0
x_i
$x_i^{0,1}\ldots x_i^{k-1,1}$
PUF^1
y_i^1
y_i
$x_i^{0,n-1}\ldots x_i^{k-1,n-1}$
PUF^{n-1}
y_i^{n-1}
n-XOR PUF

Figure 2: *n*-XOR PUF

Now let's analyze how applying the *n*-XOR changes the authentication noise. Let η' be the probability of response bit mismatch for the basic PUF building block (no XORs). For *n*-XOR, we get the *correct* result when all *n* outputs do not mismatch (between provisioning and a later authentication query), or when an *even* number of bits mismatch. The probability of each basic PUF being incorrect is η'. The probability of the post-*n*-XOR result being incorrect is:

$$\eta(n,\eta') = 1 - \sum_{i=0}^{\left\lfloor \frac{n}{2} \right\rfloor} \binom{n}{2i} \eta'^{(2i)} (1-\eta')^{n-2i} \qquad (2)$$

Observe that the verifier sees a noise level according to Eqn. (2), and the adversary is impacted, under the Statistical Query model of machine learning, based on Eqn. (1). If both parties see the same increase in η (noise) as a result of increasing n (XORs) to scale up security, both parties get impacted negatively. There is an asymptotic wall at $\eta = 0.5$, where PUF responses are indistinguishable from random for the adversary; for the verifier, it is impossible to authenticate. Our goal is to impact the adversary and the verifier asymmetrically, allowing $\eta_a \rightarrow 0.50$ while keeping η_v constant.[3]

C. Response Decimation Architecture

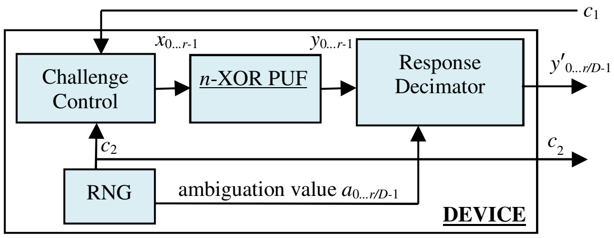

Figure 3: Response Decimation Architecture

We add Response Decimation as shown in Figure 3. An *n*-XOR PUF is augmented with challenge control logic, response decimation logic as well as a true random number generator (RNG). RNG can be derived from PUF noise (e.g., Ring Oscillator PUF least significant bit count values in certain implementations), but is shown as a separate block for clarity.

The challenge control logic mixes subchallenge seeds c_2 and c_1, i.e., c_{seed} is split into two parts so no one party controls x. c_2 is generated using the device's RNG and c_1 by the verifier's RNG. In one possible configuration, c_1 has $n \cdot k/2$ bits and c_2 has $n \cdot k/2$ bits, e.g., $k = 64$, $n = 6$.[4] During an authentication event, the server issues c_1 using its own RNG; the device returns c_2 and y'. We note that the adversary is capable of passively observing all transactions, and also adaptively issuing different values of c_1 to obtain the resulting c_2 and y', with the goal of fooling the verifier in a future authentication event.

In the Response Decimator module, randomized response decimation is performed with a down-mixing factor of *D*. As a result, the adversary no longer knows the precise challenge associated with each post-decimated response bit to launch a machine learning attack and has to make imperfect inferences to create the machine learning training labels. Next, we shall detail *response decimation* performed by the device as well as *response recovery* by the server.

D. Response Decimation with Pre-expand (RDP)

Let's suppose we have a PUF where probability of a response bit mismatch is η (ref: Table I). Now let r'' be PUF response size in bits (the reason for the "double prime" notation will become clear soon), and τ be the authentication threshold expressed as a percentage. A PUF authentication fails if more than $r'' \cdot \tau$ bits mismatch, computed as described in [7]. The *false negative* (*fn*) probability is given by:

$$fn(r'',\tau,\eta) = \sum_{i=\lceil r''\cdot\tau \rceil}^{r''} \binom{r''}{i} \eta^i (1-\eta)^{(r''-i)} \qquad (3)$$

Correspondingly, the *false positive* (*fp*) probability is:

$$fp(r'',\tau,\beta) = \sum_{i=0}^{\lfloor r''\cdot\tau \rfloor} \binom{r''}{i} \beta^i (1-\beta)^{(r''-i)} \qquad (4)$$

(Note: β is the PUF bias value that can be made close to an ideal value of 0.5 in modern PUF designs [19].) From this, one observes that given the same τ, η and β, so long as we target a response size r'' and pre-compensate for decimation effects, we can preserve some or all of the *fn* and *fp* probabilities.

Suppose we decimate the response bits by a factor of D = 2, as shown in the strike-throughs (e.g., θ1) in Figure 4. We divide y into non-overlapping pairs of bits. For each 2 bits of response in y, the device randomly eliminates 1 of 2 bits, thus only outputting a single bit to the server. On the surface, authentication appears to have become problematic, since the server has a ½ chance of correctly guessing actual challenge used for post-decimated response bit. The server does not know whether the incoming response bit corresponds to the first or second challenge in the pair. Fortunately, due to the parameter-based authentication modality, the server has access to a PUF parameter snapshot p_{ss} for the device that allows the server to synthesize via emulation *any* challenge/response pair without having explicitly collected and stored those pairs during a device provisioning process. The server can apply both possible challenges to computationally evaluate using p_{ss}

[2] $x_{0\ldots r-1} = x_0 \| x_1 \| \ldots \| x_{r-1}$, and similarly for other variables.

[3] We note that adding a cryptographic hash to a basic PUF's raw output bits would indeed help thwart learning attacks ($\eta_a \rightarrow 0.50$), but the PUF's physical noise would explode through the hash, causing $\eta_v \rightarrow 0.50$ as well.

[4] In this example, c_2 = 192 bits. In general, we want c_2 to be large enough to prevent challenge collisions inside the device. This prevents the adversary from undoing the randomized response decimation, thwarts recently-published power and fault injection side channel attacks that rely on repeated measurements, and reduces the capabilities of an adaptive chosen challenge adversary into a passive one.

the two resulting output bits. Assuming that PUF outputs are unbiased and challenges are uncorrelated, there is ¼ chance that both possible challenges produce a "0" response bit, same for a "1" response bit. This brings us to the key observation. There is a ¼ + ¼ = ½ chance that incoming (post-2x-decimated) bit can actually be verified by the server. The server authenticates for the cases that "11" or "00" are produced by the 2 possible challenges. This corresponds to an average of ½ of incoming bits, and the server ignores the rest. This is less reliable given the same authentication threshold τ, since we decimated by 2 at the device and we authenticate on only half of those bits at the server. We can, however, pre-expand y by 4x to compensate, to recapture the "lost" *fn* and *fp*. Therefore, to get a reliability associated with a $r'' = 256$-bit response, we set $|y| = r = r'' \cdot 4 = 1024$.[5] In general, for a RDP scheme, we can recapture *fn* and *fp* for the average case with an expansion factor of

$$E = D \text{ x } 2^{D-1} \tag{5}$$

So:

$r\ = |y| =$ # of pre-decimated bits generated by PUF

$r' = |y'| = r/D =$ # of post-decimated bits

$r'' = r/E =$ average # of bits authenticated, determines *fp*, *fn*

As shown by Eqn. (3) and Eqn. (4), if τ, η, and β are fixed, authentication reliability in term of both *fp* and *fn* becomes a function of r'' which, in the case of decimation, we can pre-expand using Eqn. (5) by setting $r = r''{\cdot}E$. Intuitively, for an $r = 1024$-bit (pre-decimated) PUF response, if the server authenticates on a subset comprising of $r'' = 256$ bits that are unambiguous, the authentication reliability should be that of a 256-bit response.

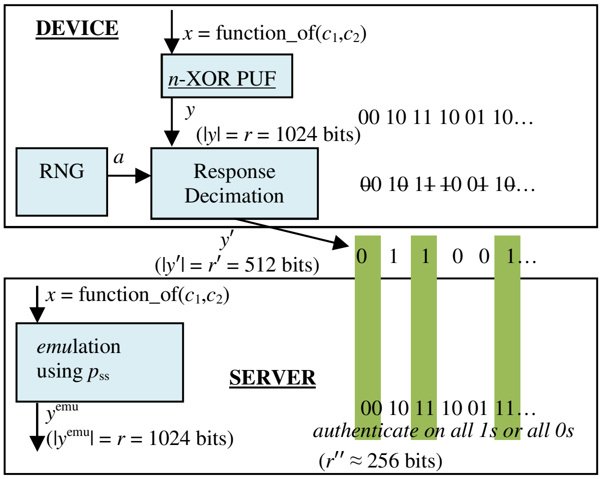

Figure 4: Response Decimation Example

More generally, the RDP algorithm comprises of the steps in Figure 5 (*MV* = manufacturing variations). The transaction begins with a *challenge seed exchange*, with the server and device each generating one half of the challenge seed (steps i. and ii.). Next, *response decimation* is performed on the device (steps iii. through v.). In step v.,

$$y'_j = y_{j\cdot D+a_j},\ \ 0 \le j < r/D,$$

$$a_j \in \{0,1,\ldots,D\text{-}1\} \text{ randomly selected for each } j$$

The ambiguation value a_j (from the device's RNG) determines which bits are decimated, or equivalently, which bit is kept and sent out. Note that the a_j values are kept secret on the device while the challenge seed values c_1 and c_2 are known to the server as well as the adversary. Next, *response recovery* is performed on the server (steps vi. through ix.). In step viii., the server derives a mask identifying the locations of the "good" bits for authentication (and equivalently, which bits to ignore):

$$m_j \ = \ all\_ones_j \,|\, all\_zeros_j$$

$$all\_ones_j \ = \ \ y^{\text{emu}}_{j\cdot D+0} \,\&\, y^{\text{emu}}_{j\cdot D+1} \,\&\, \ldots \,\&\, y^{\text{emu}}_{j\cdot D+D\text{-}1}$$

$$all\_zeros_j \ = \sim\!(y^{\text{emu}}_{j\cdot D+0} \,|\, \ y^{\text{emu}}_{j\cdot D+1} \,|\, \ \ldots \,|\, \ y^{\text{emu}}_{j\cdot D+D\text{-}1})$$

$$0 \le \ j < r/D$$

Here, & is the bit-wise AND operator (detects all 1s), | is the bitwise OR operator (detects all 0s), and ~ is logical inversion. In step ix., a masked threshold comparison is performed. The device is authentic if (^ is bitwise XOR operator, **wt** is the Hamming weight operator counting the number of ones):

$$\mathbf{wt}((y'_{0\ldots r/D\text{-}1} \wedge all\_ones_{0\ldots r/D\text{-}1}) \,\&\, m_{0\ldots r/D\text{-}1}) \le \ \tau \cdot \mathbf{wt}(m_{0\ldots r/D\text{-}1})$$

Response Decimation with Pre-expansion (RDP)

i.	c_1	← Server	
ii.	c_2	← Device	
iii.	$x_{0\ldots r\text{-}1}$	= function_of(c_1 , c_2)	(on Device)
iv.	$y_{0\ldots r\text{-}1}$	= function_of(MV, $x_{0\ldots r\text{-}1}$)	(on Device)
v.	$y'_{0\ldots r/D\text{-}1}$	← function_of($y_{0\ldots r\text{-}1}$,D,$a_{0\ldots r/D\text{-}1}$)	(on Device)
vi.	$x_{0\ldots r\text{-}1}$	= function_of(c_1 , c_2)	(on Server)
vii.	$y^{\text{emu}}_{0\ldots r\text{-}1}$	= function_of(p_{ss}, $x_{0\ldots r\text{-}1}$)	(on Server)
viii.	$m_{0\ldots r/D\text{-}1}$	= function_of($y^{\text{emu}}_{0\ldots r\text{-}1}$)	(on Server)
ix.	Authenticate on "good" bits of y'		(on Server)

Figure 5: Response Decimation Procedure

V. Evaluations

A. Inferring CRPs from Decimated Bits

A machine learning algorithm requires challenge/response input labels $\{x,\ f(x)\}$ during the *training phase*. Its classification error during the *attack phase* is highly dependent on the accuracy (noise level) of these labels. Due to randomized decimation, the adversary no longer knows the precise challenge x associated with each post-decimated response bit, and has to *infer* the best $\{x,\ f(x)\}$. In this section, we analyze different CRP inference strategies based on the goal of minimizing CRP uncertainty (the adversary's noise), and provide a mathematical proof.

Let's consider the 2x decimation case. Eqn. (1) would suggest that the strategy that produces lowest noise is best. For the 2x decimation case, there are *only* two possibilities.

Single-Challenge-Guess Strategy. The adversary can guess, among all the candidate challenges, a single challenge. As an example, the adversary can form $\{x,\ f(x)\}$ using $\{x_{2j},\ y'_j\}$. Assuming unbiased response bits and uncorrelated challenges, given an incoming response bit, the adversary has a $1/D$ chance of guessing the correct challenge, but in the case where the wrong challenge was guessed there is a ½ chance that the CRP derived is still correct. Thus, the probability that an accurate CRP is derived is $1/D \cdot 1 + (D\text{-}1)/D \cdot ½$. The CRP uncertainty is:

$$\varepsilon_{\text{u}}(D) = 1 - \ (D{+}1)/2D = \ (D - 1)/2D \tag{6}$$

Full-Response-Replication Strategy. The adversary replicates the incoming response bit for both challenges (and in general

[5] Alternately, one can consider response expansion and pad random noise at random locations. In fact, that's what RDP is doing, except that it is done in a manner where the verifier, in possession of p_{ss}, knows which post-decimated bits to ignore, thereby creating the bifurcated noise effect.

D challenges). As an example, the adversary forms $\{x, f(x)\}$ using $\{x_{2j}, y'_j\} \cup \{x_{2j+1}, y'_j\}$.

Now we derive the CRP uncertainty. First, we introduce two new concepts and give examples to aid understanding. Let $h = \mathbf{wt}(x_{2j...2j+D-1})$ be the Hamming weight of a grouping of *D pre*-decimated bits. Let's define ■ as the "pseudo-majority" case surviving after decimation, and ▪ as the "pseudo-minority" case surviving. For *D* = 3, if the pre-decimated bits are 110, ■ refers to a post-decimated bit of "1", and ▪ refers to a post decimated bit of "0". So for *D* = 3, we have Pr(■| *h* = 2) = ⅔ and Pr(▪| *h* = 2) = ⅓. If there is an equal number of 1s and 0s (*D* is even), we define Pr(■| *h* = *D*/2) = Pr(▪| *h* = *D*/2) = ½.

Now, let's compute the probability that the adversary makes a *correct CRP inference* (herein denoted as *CInfr*) when a *post*-decimated response bit is replicated *D* times and inferred as the response to all *D* candidate challenges. In the 110 case, if the adversary observes a "1", the adversary would infer a replicated response of 111 to map to the three challenges. Here, Pr(*CInfr* | ■ , *h* = 2)= ⅔. Similarly, Pr(*CInfr* | ▪ , *h* = 2) = ⅓. More generally,

Pr(■ | *h, D*) = max(*h*/*D*, (*D*-*h*)/*D*) (7)
Pr(▪ | *h, D*) = min(*h*/*D*, (*D*-*h*)/*D*) (8)
Pr(*CInfr* | ■, *h, D*) = max(*h*/*D*, (*D*-*h*)/*D*) (9)
Pr(*CInfr* | ▪, *h, D*) = min(*h*/*D*, (*D*-*h*)/*D*) (10)

This gives the joint probabilities:

Pr(*CInfr*, ■ | *h, D*) = max(*h*/*D*, (*D*-*h*)/*D*)^2 (11)
Pr(*CInfr*, ▪ | *h, D*) = min(*h*/*D*, (*D*-*h*)/*D*)^2 (12)

Note that the two probabilities immediately above are conditioned on *h*. If for each one we compute the marginal probability over $0 \leq h \leq D$ and sum them, we arrive at Pr(*CInfr* | *D*). The CRP uncertainty is 1 - Pr(*CInfr* | *D*) :

$$\varepsilon'_{\mathrm{u}}(D) = 1 - \sum_{h=0}^{D} \left[\frac{\binom{D}{h}}{2^D}\right]\left[\left(\frac{h}{D}\right)^2 + \left(\frac{D-h}{D}\right)^2\right] \qquad (13)$$

Proof of Equivalence. Now we prove that the single-challenge-guess strategy has same CRP uncertainty, i.e., contributes same noise level to adversary, as the full-response-replication strategy. From the binomial theorem:

$$(x+a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k} \qquad (14)$$

Setting $a = 1$ and taking first and second derivatives with respect to *x*, and substituting variables, we have:

$$\sum_{h=0}^{D} \binom{D}{h} \cdot h = D \cdot 2^{D-1} \qquad (15)$$

$$\sum_{h=0}^{D} \binom{D}{h} \cdot h^2 = (D + D^2) \cdot 2^{D-2} \qquad (16)$$

Thus, Eqn. (13) is equivalent to:

$$\begin{aligned} \varepsilon'_{\mathrm{u}}(D) &= 1 - \frac{1}{D^2 \cdot 2^D} \sum_{h=0}^{D} \binom{D}{h} [2h^2 + D^2 - 2hD] \\ &= 1 - \frac{1}{D^2 \cdot 2^D} (2(D+D^2) \cdot 2^{D-2} + D^2 \cdot 2^D - 2D^2 \cdot 2^{D-1}) \\ &= 1 - (D+1)/2D = \varepsilon_{\mathrm{u}}(D) \qquad \square \end{aligned}$$

For *D* = 2, the single-challenge-guess and full-response-replication are the only two possible strategies. In the proof, we showed that both have an equivalent CRP uncertainty and contribute the same amount of noise to the adversary (i.e., $\varepsilon'_{\mathrm{u}}(D) = \varepsilon_{\mathrm{u}}(D)$), and this noise scales with *D*. However, for *D* = 3, there are additional possible strategies. The adversary can use a *partial-response-replication strategy*, taking the incoming bit and replicating for only two of the three challenges (or in general, up to *D*-1 challenges). We can extend the proof to show that a partial-response-replication strategy can do no better than the prior strategies (observe that Eqn. (9) and Eqn. (10) holds regardless of the number of times the incoming bit is replicated to infer CRPs, encompassing the single, full, as well as partial replication cases).

B. *Machine Learning: Baseline Results and Background*

We first replicated selected results from [14], and this comparison is shown in the "Baseline" columns in Table II. We used the most effective machine learning algorithm in [14] in modeling *n*-XOR 64-stage PUFs, namely RPROP [13]. An Intel i7 6-core, 12-thread CPU with 64 Gigabytes of memory was used. These attacks were performed on *synthetic* as opposed to *physical* PUFs. Based on Eqn. (1), noise *increases* learning complexity. This was demonstrated in [15], where even when majority voting was used to reduce noise, the researchers could not replicate breaking 6-XOR with 200k CRPs on a physical PUF and estimated that 700k CRPs are needed. All things being equal, our synthetic PUF (no noise) results serve as a bound for what machine learning might achieve on a physical PUF with noise.

C. *Machine Learning Results: Security-Enhanced*

TABLE II. MACHINE LEARNING: BASELINE VS. SECURITY-ENHANCED

	Baseline		***Security-Enhanced***	
n-XOR	***Results of [14]***	***Replicating [14]***	***500k CRPs***[6]	***1M CRPs***[6]
4-XOR	12k CRPs $\varepsilon_c \leq 1\%$ 4 min	12k CRPs $\varepsilon_c \leq 1\%$ 1 min	$\varepsilon_c = 8\%$ 5 hrs (*D* =2x)	-
5-XOR	80k CRPs $\varepsilon_c \leq 1\%$ 2 hrs	80k CRPs $\varepsilon_c \leq 1\%$ 10 min	$\varepsilon_c \approx 50\%$ 2 days (*D* = 2x)	$\varepsilon_c \approx 50\%$ 2 days (*D*= 2x)
6-XOR	200k CRPs $\varepsilon_c \leq 1\%$ 31 hrs	200k CRPs $\varepsilon_c \leq 1\%$ 5 hrs	$\varepsilon_c \approx 50\%$ 2 days (*D* = 2x)	$\varepsilon_c \approx 50\%$* 2 days (*D* = 2x)

* Latest results demonstrate 6-XOR not converging after 98 days, 16M CRPs

To derive the security-enhanced results, we made modifications and extended the RPROP algorithm. First, we added a CRP inference front end to optimally derive the CRPs using the full-response-replication strategy (cf. proof in V.A). We also modified the error signal to compute the classification error only for the known "good" response bit positions in the final computation for ε_c. For gradient search, the error feedback was similarly modified since only the all 1s and all 0s cases are significant. If either the error signal modification or the error feedback modification or both are removed, algorithm does not successfully converge, with a prediction rate of no better than 50% ± 1%.

With decimation applied, we show in Table II that for 5-XOR there is no successful machine learning convergence with up to 1M CRPs when a mere decimation = 2x is applied. We note that in comparison to [14], the same 5-XOR is learned to $\varepsilon_c \leq 1\%$ with considerably less than 1M CRPs, at 80k CRPs, in 2 hours.

Figure 6 highlights security gain for 5-XOR and 6-XOR in terms of increases in time and CRPs on a log-log scale, and yet the classification error remains near 50% with order of magnitude (10x to 100x) increases in attack resource. We

[6] 500k and 1M CRPs refer to the number response bits sent outside the device, visible to the adversary.

currently do not have data to see whether ε_c remains near 50% for 100x plus increases in run-time and CRP (i.e., we don't know the breaking points for 5-XOR and 6-XOR at D = 2).

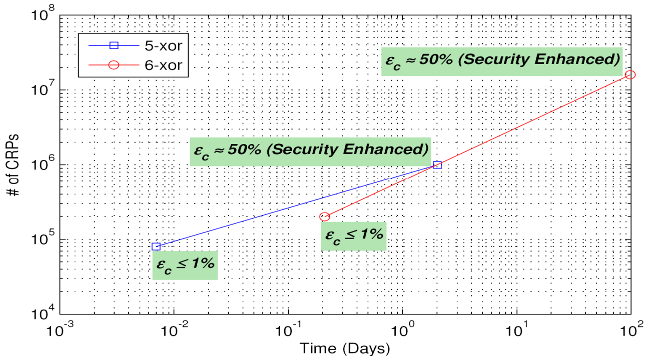

Figure 6: Example Security Increase

D. *Example of Bifurcated Noise at 28nm*

Figure 7 shows examples of bifurcated noise at D = 2, using 28nm data from Table I. We assume that the adversary attacks at the lowest noise level, which is at 25°C, where η_e = 0.046 for 4-XOR. Applying Eqn. (2), setting $\eta' = \eta_e$, we obtain 8-XOR and 12-XOR noise of 0.088 and 0.126. Verifier sees $\eta_v = \eta_e$. Adversary sees η_a based on *combined* effect of :

i. CRP uncertainty due to decimation $\varepsilon_u(D)$ (Eqn. (6))

ii. environmental noise η_e (e.g., Table I).

Specifically,

$$\eta_a(n) = 1 - [\varepsilon_u(D) \cdot \eta_e(n) + (1 - \varepsilon_u(D)) \cdot (1 - \eta_e(n))] \quad (11)$$

For example, at 4-XOR, the verifier sees a noise level of 0.046 (x-axis) while the adversary sees a noise level of 0.273 (y-axis). This advantage grows with D (not explicitly shown).

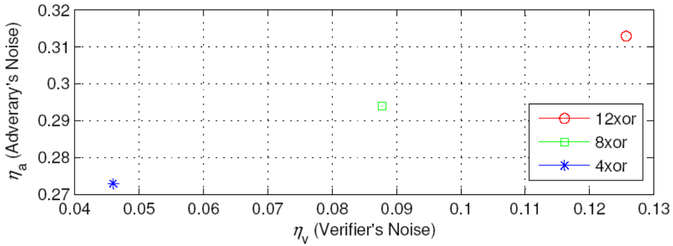

Figure 7: Examples of Bifurcated Noise @ 28nm, D = 2

E. *Machine Learning Attacks with Side Channel Information*

In 2013, machine learning attacks with side channel information became an active research topic. [15] performed majority voting from five measurements of a physical PUF to reduce noise. [1] looked at side channel noise, using repeated measurements to produce a reliability-aware algorithm. [2] used fault injection to attack Arbiter and RO k-sum PUFs. All these attacks required *repeated* measurements, which our architecture disallows. The adversary needs to find other means to reduce η_a.

VI. Conclusions

We presented the first noise bifurcation architecture for linear additive physical functions, allowing the adversary's noise $\eta_a \rightarrow 0.50$ while keeping the verifier's noise η_v constant; the former to confuse the adversary, the latter to ease reliable authentication. This approach provides an extra dimension for security scaling in the battle against emerging machine learning attacks. The architecture also thwarts emergent power and fault injection side channel attacks requiring repeated measurements. Future work includes using maximum likelihood methods to authenticate beyond the all 1s and 0s case, to reduce pre-expansion overhead.

References

[1] J. Delvaux, I. Verbauwhede, "Side Channel Modeling Attacks on 65nm Arbiter PUFs Exploiting CMOS Device Noise," *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, pp. 137–142.

[2] J. Delvaux, I. Verbauwhede, "Fault Injection Modeling Attacks on 65*n*m Arbiter and RO Sum PUFs via Environmental Changes," *IACR Cryptology ePrint*, 2013: 619 (2013).

[3] S. Devadas, "Non-networked RFID PUF Authentication," US Patent Application 12/623,045, 2008, US Patent 8,683,210, 2014.

[4] B. Gassend, D. Clarke, M. van Dijk, S. Devadas, "Silicon Physical Random Functions," *ACM Conference on Computer and Communication Security Conference (CCS)*, 2002.

[5] G. Hospodar, R. Maes, I. Verbauwhede, "Machine Learning Attacks on 65nm Arbiter PUFs: Accuracy Modeling Poses Strict Bounds on Usability," *IEEE International Workshop on Information Forensics and Security (WIFS)*, 2012.

[6] M. Kearns, "Efficient Noise-Tolerant Learning form Statistical Queries," *Journal of the ACM*, 1998, vol. 45(6), pp. 983-1006.

[7] D. Lim, "Extracting Secret Keys from Integrated Circuits," Master's thesis, MIT, 2004.

[8] A. Mahmoud, U. Rùhrmair, M. Majzoobi, F. Koushanfar, "Combined Modeling and Side Channel Attacks on Strong PUFs," *IACR Cryptology ePrint*, 2013: 632 (2013).

[9] M. Majzoobi, F. Koushanfar, M. Potkonjak, "Testing Techniques for Hardware Security," *IEEE International Test Conference (ITC)*, 2008.

[10] M. Majzoobi, F. Koushanfar, M. Potkonjak, "Lightweight Secure PUF," *International Conference on Computer Aided Design*, 2008.

[11] M. Majzoobi, M. Rostami, F. Koushanfar, D. Wallach, S. Devadas, "SlenderPUF: a Lightweight, Robust and Secure Strong PUF by Substring Matching," *IEEE International Workshop on Trustworthy Embedded Devices (TrustED)*, 2012.

[12] E. Oztiirk, G. Hammouri, B. Sunar, "Towards Robust Low Cost Authentication for Pervasive Devices," *International Conference on Pervasive Computing and Communications (PerCom)*, 2008.

[13] M. Reidmiller, H. Braun, "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm," *IEEE International Conference on Neural Networks*, 1993.

[14] U. Rùhrmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, J. Schmidhuber, "Modeling Attacks on Physical Unclonable Functions," *ACM Conference on Computer and Communication Security (CCS)*, 2010.

[15] U. Rùhrmair, J. Sölter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, S. Devadas, "PUF Modeling Attacks on Simulated and Silicon Data," *IEEE Transactions on Information Forensics and Security (TIFS)*, 2013, vol. 8(11), pp. 1876-1891.

[16] U. Rùhrmair, X. Xu, J. Sölter, A. Mahmoud, F. Koushanfar, W. Burleson, "Power and Timing Side Channels for PUFs and their Efficient Exploitation," *IACR Cryptology ePrint*, 2013: 851 (2013).

[17] G. Suh, S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," *Design Automation Conference (DAC)*, 2007, pp. 9–14.

[18] M. Yu, D. M'Raïhi, R. Sowell, S. Devadas, "Lightweight and Secure PUF Key Storage Using Limits of Machine Learning," *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2011, LNCS vol. 6917, pp. 358-373.

[19] M. Yu, R. Sowell, A. Singh, D. M'Raïhi, S. Devadas, "Performance Metrics and Empirical Results of a PUF Cryptographic Key Generation ASIC," *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2012.