IEEE CS Security and Privacy Workshops

Slender PUF Protocol: A lightweight, robust, and secure authentication by substring matching

Mehrdad Majzoobi[†,1], Masoud Rostami[†,2], Farinaz Koushanfar[†,3], Dan S. Wallach[‡,4], Srinivas Devadas[§,5]

[†] *Electrical and Computer Engineering Department, Rice University*
6100 Main Street, Houston, TX 77005 United States
[1] mehrdad.majzoobi@rice.edu, [2] mrostami@rice.edu, [3] farinaz@rice.edu

[‡] *Computer Science Department, Rice University*
6100 Main Street, Houston, TX 77005 United States
[4] dwallach@cs.rice.edu

[§] *Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology*
77 Massachusetts Avenue, Cambridge, MA 02142 United States
[5] devadas@mit.edu

***Abstract*—We introduce Slender PUF protocol, an efficient and secure method to authenticate the responses generated from a Strong Physical Unclonable Function (PUF). The new method is lightweight, and suitable for energy constrained platforms such as ultra-low power embedded systems for use in identification and authentication applications. The proposed protocol does not follow the classic paradigm of exposing the full PUF responses (or a transformation of the full string of responses) on the communication channel. Instead, random subsets of the responses are revealed and sent for authentication. The response patterns are used for authenticating the prover device with a very high probability. We perform a thorough analysis of the method's resiliency to various attacks which guides adjustment of our protocol parameters for an efficient and secure implementation. We demonstrate that Slender PUF protocol, if carefully designed, will be resilient against all known machine learning attacks. In addition, it has the great advantage of an inbuilt PUF error tolerance. Thus, Slender PUF protocol is lightweight and does not require costly additional error correction, fuzzy extractors, and hash modules suggested in most previously known PUF-based robust authentication techniques. The low overhead and practicality of the protocol are confirmed by a set of hardware implementation and evaluations.**

I. Introduction

Semiconductor devices and systems should be able to preserve the privacy of customers, provide identification, and be resilient against security attacks.

Classic security paradigms rely on a stored secret binary key and cryptographic algorithms. Secret keys are stored in on-chip non-volatile memory (NVM). However, on-chip NVM storage is prone to invasive physical attacks (e.g., probing) and non-invasive imaging attacks (e.g., SEM). Moreover, classic cryptographic algorithms are resource-intensive for many low power applications.

Physical unclonable functions (PUFs) have been proposed [1] to provide the desired integrity and robust resiliency against security attacks, while keeping the implementation costs at the minimum for authentication applications.

Silicon PUFs make it possible to have secrets that are physically bound to the hardware [2]. Silicon PUFs use the unclonable intrinsic process variability of silicon devices to provide a unique mapping from a set of digital inputs (*challenges*) to a set of digital outputs (*responses*). The imperfections and uncertainties in the fabrication technology make cloning of a hardware circuit with the exact same device characteristics impossible, hence the term unclonable. Moreover, PUFs can be designed to make it prohibitively hard to simulate, emulate, or predict their behavior [2]. Excellent surveys of various PUF designs can be found in [3]–[5].

Strong PUFs are a subclass of PUFs which naturally enjoy the unclonability of a physical-disorder-based medium. In addition, a Strong PUF has the property that the number of its possible challenge-response pairs (CRPs) has an exponential relationship with respect to the number of its physical components. This huge space of possible CRPs hinders attacks based on pre-recording and replaying previously used CRPs. A secure Strong PUF should be resilient against machine learning attacks which aim at predicting the PUF response to random

DOI 10.1109/SPW.2012.30

new challenges by studying a set of known PUF CRPs. Strong PUFs have been used in several applications of identification and authentication of hardware systems [3].

Since the number of actual physical components of a Strong PUF is finite, a compact polynomial-order model of the CRP relationships can be built. In particular, a trusted IP owner with physical access to the device (e.g., the original manufacturer) can build such a compact model by measuring the PUF components; physical access can be permanently disabled before deployment to avoid direct compact modeling. Such compact models can be treated as a *secret* which can be used by a trusted verifier to authenticate the prover's PUF. To date, a number of machine learning attacks have been launched against Strong PUFs [6]. Several such machine learning attacks were carried out by recording and analyzing a finite set of CRPs. Such attacks were possible because the CRPs leak structural information about the PUF and compact models. Our protocol aims at thwarting this class of attack.

In this paper, we introduce Slender PUF protocol, a lightweight secure and robust authentication protocol based on a Strong PUF. The protocol enables a prover with physical access to the PUF to authenticate itself to a verifier. It is assumed that verifier knows the secret relationship between the PUF challenge-response pairs through a pre-trained compact PUF model. The protocol leaks minimal amount of information about secret PUF parameters on the shared communication channel. The protocol is devised such that the verifier and the prover jointly generate the challenges to the PUF. This joint challenge generation is done such that neither a dishonest prover nor a dishonest verifier can soley control the challenges used for authentication. While none of the authenticating parties can solely control the challenges, the resulting challenge values are publicly known.

Using the set of known challenges, the prover's PUF generates a string of responses of a certain length. Instead of revealing the whole response string to the verifier, the prover randomly selects a response substring of fixed size and shares it with the verifier. The randomly selected index pointing to the location of the substring is kept as a secret at the prover side. An honest verifier with access to the PUF secret model could search and match the received substring to its estimated PUF response sequence and find the secret index. The authentication is successful if the prover's response substring matches at some location in the verifier's estimated response string within a predefined threshold.

To ensure security, we also provide a thorough discussion of attacks and protocol parameter adjustments. We demonstrate that the main advantage of keeping the index secret is enabling resiliency against machine learning attacks. The protocol also achieves robustness against inherent noise in PUF response bits, without costly traditional error correction modules. Note that recent work has used pattern matching for correcting errors while generating secret keys from a PUF [7]. However, the number of generated secret keys were limited. To the best of our knowledge, no application of pattern matching for authentication of Strong PUFs has been proposed thus far.

We demonstrate that our protocol can be implemented with a few simple modules, excluding the need for expensive cryptographic hashing and classic error correction techniques that have been suggested in earlier literature for achieving security. In brief, the main contributions of our paper are as follows:

- Introduction of Slender PUF protocol, lightweight and secure Strong PUF based device authentication based on pattern matching. The method is very suitable for ultra-low power and embedded devices.
- The protocol automatically provides robustness against inherent noise in the PUF response string, without requiring externally added and costly traditional error correction modules.
- We perform a thorough analysis of the protocol's resiliency to various attacks which guides adjustment of the protocol parameters. In particular, we demonstrate that our method, if carefully designed, will be resilient against all known Strong PUF machine learning attacks.
- The lightweight nature, security, and practicality of the new protocol are confirmed by a set of hardware implementation and evaluations.

The remainder of the paper is organized as follows. Section II provides a background on Strong PUFs. In Section III, related literature is discussed and the new aspects of our work are highlighted. In Section IV, the Slender PUF methodology is introduced and explained in detail. The parameters of our method and its security against multiple attacks are investigated in Section V. Hardware implementation and performance evaluations are presented in Section VI. Section VII concludes the paper.

II. Background

In this section, we provide a brief background on Strong PUFs using the implementation of an instance of Strong PUFs known as arbiter PUF or delay-based

PUF. Desired statistical properties of a Strong PUF are briefly reviewed, and XOR mixing of arbiter PUFs to improve the statistical properties is discussed.

Note the proposed protocol can work with any Strong PUF that satisfies the requirements discussed in this section. We use arbiter PUF to demonstrate the protocol only due to its widespread use and ease of implementation.

A. Strong PUFs and their implementation

There are a number of different PUF types, each with a set of unique properties and applications. For example, *Weak PUFs*, also known as *Physically Obfuscated Keys (POKs)* are commonly used for key generation applications. Following the taxonomy in [8], the class of PUF that is of interest in this paper is called a *Strong PUF*. Strong PUFs are built based on the unclonability and disorder in the physical device features, with very many challenge-response pairs. The size of the CRP space is an exponential function of the number of underlying components. Strong PUFs have the property that they are prohibitively hard to clone; a complete enumeration of all their CRPs is intractable. To be secure, they should be resilient to machine learning and prediction attacks. Important applications of Strong PUFs include identification, authentication, and key establishment [8].

A well-known implementation of a Strong PUF is the delay-based arbiter PUF introduced in [9] which we also use in this paper. In this PUF structure, the delay difference between two parallel paths is compared. The paths are built identically to make their nominal delays equal by construction, but due to process variations, one of the paths has slightly longer or shorter delay in practice. The architecture of the delay-based PUF is illustrated in Fig. 1. A step input simultaneously triggers the two paths. At the end of the two parallel (racing) paths, an arbiter (typically implemented with a D-Flip Flop) is used to convert the analog difference between the path to a digital value. The arbiter output becomes one if the signal arrives at its first input earlier than the second one, i.e., the path ending at the first arbiter input has a shorter delay. The arbiter output stays zero otherwise. The two paths are divided into several smaller sub-paths by inserting path swapping switches. Each set of inputs to the switches acts as a challenge set (denoted by C_{i}).

This type of PUF falls into the class of linear Strong PUFs, since the PUF only consists of linear addition and subtraction of delay elements. The behavior of the PUF in Fig. 1 can be modeled by the following linear

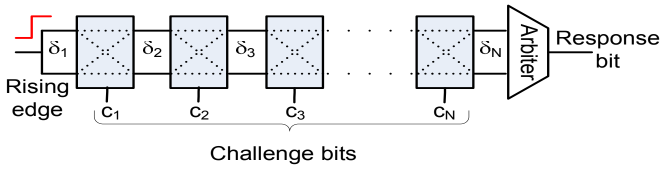

Fig. 1. An arbiter linear PUF block with N challenges and one response bit.

inequality [10]:

$$\sum_{j=1}^{N}(-1)^{\rho_j}\delta_j + \delta_{N+1} \underset{r=1}{\overset{r=0}{\lessgtr}} 0, \tag{1}$$

where ρ_j is related to the input challenge that controls the switch selectors by the following relation,

$$\rho_i = \bigoplus_{x=i,i+1,...,N} c_x = c_i \oplus c_{i+1} \oplus ... \oplus c_N. \tag{2}$$

According to Inequality 1, if the difference between the sum of delays on the top and bottom paths is greater than zero, then the response will be '1'; the response is '0' otherwise. To simplify the notations, Inequality 1, can be rewritten as:

$$r = Sign(\mathbf{\Delta}.\mathbf{\Phi}), \tag{3}$$

where $\Delta = [\delta_1, \delta_2, ..., \delta_{N+1}]$ is the delay parameter vector, $\mathbf{\Phi} = [(-1)^{\rho_1}, (-1)^{\rho_2}, ..., (-1)^{\rho_N}, 1] = [\varphi_1, \varphi_2, ..., \varphi_{N+1}]$ is the transformed challenge vector in which $\varphi_i \in \{-1, 1\}$, '**.**' is the scalar product operation, r is the response bit, and $Sign$ is the sign function. We will refer to $\mathbf{C}$ as the input challenge vector in the remainder of the paper. Note that the parameters φ, ρ, and c are related.

B. Linear Arbiter PUF statistical properties

Before describing the protocol, we briefly review the statistical properties of a linear arbiter PUF circuit. It has been demonstrated in [11] that when the delay parameters $\delta \in \Delta$ come from identical symmetric distributions with zero mean (in particular it is safe to assume that the δ's are i.i.d Gaussian, i.e., $\delta \in N(0, \sigma)$), then the following statistical properties hold for a linear arbiter PUF:

- The output response bits are equally likely over the entire space of challenges, i.e, $Prob\{r = -1\} = Prob\{r = 1\} = 0.5$. Half of the challenges map to $r = -1$ and the other half maps to $r = 1$.

- The responses to similar challenges are similar. Hamming distance is used as the measure of similarity. In other words, the probability that the responses r_0 and r_1 to two input challenge vectors C_0 and C_1 are different is a monotonically increasing function of the Hamming distance between the input challenges, i.e., $Prob\{r_0 \neq r_1\}=f(\text{HD}(C_0,C_1))$. For example, In the trivial cases HD(C_0,C_1)=0, i.e. C_0=C_1, then $Prob\{r_0 \neq r_1\} = 0$.

The Hamming distance between challenges C_x and C_y is defined as $HD(C_x, C_y) = \sum_{i=1}^{N} \mid C_x[i] - C_y[i] \mid /N$ where $C_x[i], C_y[i] \in \{-1, 1\}$. The boxplot in Fig. 2 shows the distribution of output transition probabilities obtained for 50 synthetic PUF instances as a function of the input challenge Hamming distances. As the Hamming distances between the input challenge vector becomes larger, the probability of having different PUF response bits increases.

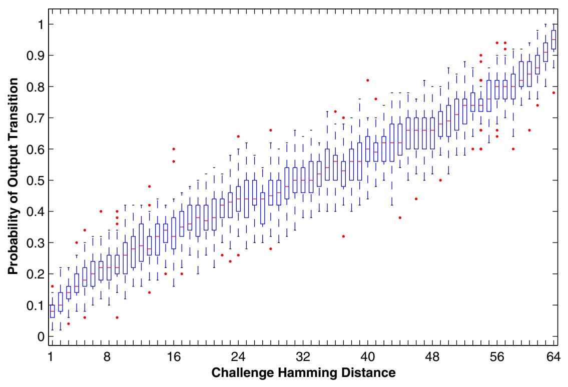

Fig. 2. The probability of output response transition as a function of the input challenge pair Hamming distance for a population of 50 linear arbiter PUFs.

Ideally it is expected that any flip in the challenge bits will cause half of the response bits to flip. This is known as the strict avalanche criterion. Any deviation from the avalanche criterion reduces the security of the system built based on these PUFs. Later in this paper, we discuss the direct implication of the PUF avalanche criterion in the proposed protocol's security.

Thus, it is desirable to have the curves in Fig. 2 to be as close as possible to the probability of 0.5. To improve the statistical properties of a linear PUF, the output of independent PUFs are XOR-mixed together as shown in Fig. 3 [11]. Fig. 4 shows the probability of output flipping versus the Hamming distance between two challenge sequences for 2, 4, and 8 independent XOR-mixed arbiter PUFs.

The figure shows that this probability is very close to the ideal 0.5 probability when just four independent PUF output bits are mixed by an XOR. As more independent PUF response bits are mixed, the curve moves closer to the ideal case; however, this would linearly increase the probability of error in the mixed output bit. For instance, for a single PUF response bit error of 5%, the probability of error for a 4-XOR-mixed PUF is reported to be 19% [11].

In addition to achieving the avalanche criterion, the XOR-mixed arbiter PUF requires a significantly larger set of challenge response pairs to successfully train the PUF model for a given target accuracy level. (A 6-XOR arbiter PUF is currently out of reach for state-of-the-art modeling attacks.)

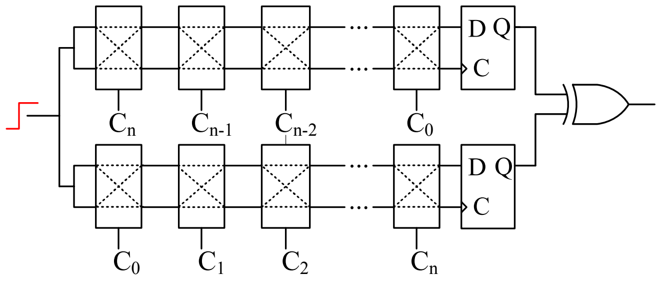

Fig. 3. Two independent linear arbiter PUFs are XOR-mixed in order to implement an arbiter PUF with better statistical properties.

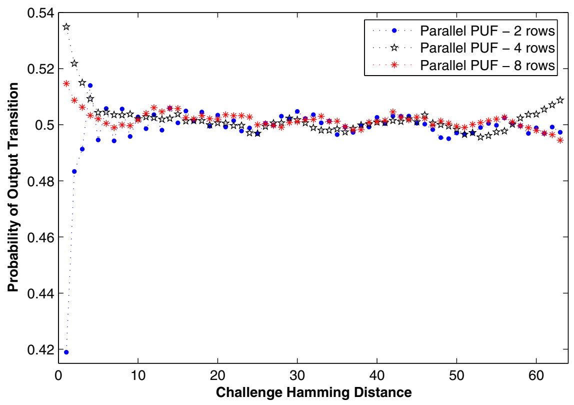

Fig. 4. The probability of the arbiter PUF output flipping versus the Hamming distance between two challenge sequences for 2, 4, and 8 independent XOR-mixed PUFs [11]

In the rest of the paper, we build our Slender PUF protocol based on the assumption that the PUF at hand is a linear XOR-mixed arbiter PUF with near ideal statistical properties as depicted in Fig. 4. We argue that our Slender PUF protocol is applicable to any lightweight Strong PUF which follows the statistical properties discussed in this Section. Furthermore, as mentioned earlier the protocol can be applied to any Strong PUF that satisfies the avalanche criterion and has an exponentially large space of challenge responses. Another example of a Strong PUF is introduced in [21] which uses CMOS transistor leakage currents instead of delays and a sense amplifier instead of arbiter to implement a linear Strong PUF structure.

III. Related work

PUFs have been subject to modeling attacks that breach their security and break any protocols built upon them. The basis for contemporary PUF modeling attacks is collecting a set of CRPs by an adversary, and then building a numerical or an algorithmic model from the collected data. For the attack to be successful, the models should be able to correctly predict the PUF response to any new challenge with a high probability. Previous work on PUF modeling (reverse-engineering) used various machine learning techniques to attack both implementation and simulations of a number of different PUF families, including the realizations and simulations of linear arbiter PUFs and feed-forward arbiter PUFs [6], [10]–[13].

The use of XORs for mixing the responses from the arbiter PUFs to safeguard them against attacks was pursued in [14]. More comprehensive analysis and description of PUF security requirements to ensure their protection against modeling attacks were presented in [15], [16]. The latest reported attacks on PUFs with k levels of XORs at their output were able to model up to $k = 5$ (after a year of running their algorithms on supercomputers) [6]. This was assuming that the full string of CRPs was known to the attacker. At the time of this publication, to the best of our knowledge, no stronger attacks on k-level XOR arbiter PUFs have been reported. We also note that, the results for $k = 5$ in [6] are for "synthetic" PUFs, not for a silicon realization of a PUF.

In this paper, we use 4-XOR arbiter PUFs in our design. Note that a drawback of XOR'ing is the increase in response noise (error); this needs to be carefully considered.

Extracting secret keys from PUF responses has been explored in previous work, including [2], [12], [17]–[19]. Since cryptographic keys need to be stable, error correction is used for stabilizing inherently noisy PUF response bits. The classic method for stabilizing noisy PUF bits (and noisy biometrics) is error correction which is done by using helper bits or *syndrome* [20].

Since error correction needs to be robust, secure, and efficient, it is important to consider limiting the amount of secret bit leakage through the disclosed syndrome bits. A generic secure key extraction framework based on biometric data and error correction was devised in [20]. A newer information-theoretically secure Index-Based Syndrome (IBS) error correction coding for PUFs was introduced and realized in [19]. All the aforementioned methods incur a rather high overhead of error correction logic, e.g., BCH, which prohibits their usage in lightweight systems. An alternative efficient error correction method by pattern matching of responses was very recently proposed [7]. We use this pattern matching idea in our work. In [7], a 4-XOR arbiter has been used which for real PUFs has not yet been broken. Their architecture also works with a higher than 4 XOR mixing. However the error correction performance would be reduced. Their proposed protocol and application area was limited to secret key generation.

In the context of challenge-response based authentication for Strong PUFs, sending the syndrome bits for correcting the errors before hashing was investigated [2]; the necessity for error correction was due to hashing the responses before sending them to avoid reverse engineering. Naturally, the inputs to the hash have to be stable to have a predictable response. The proposed error correction methods in this context are classic error correction and fuzzy extraction techniques. Aside from sensitivity to PUF noise (because it satisfies the strict avalanche criterion) hashing has the drawback of high overhead in terms of area, delay, and power.

This paper introduces Slender PUF protocol, the first lightweight PUF authentication scheme based on string pattern matching and covert indices. We demonstrate that the Slender PUF protocol is secure against the model building attacks. Modeling is thwarted by leaking very limited information about the index bit from a response string. The random index is inherently independent of the response string content.

IV. Slender PUF Protocol

In this section, the proposed protocol is introduced and explained in detail. The protocol is based on a Strong PUF with acceptable statistical properties, like the one shown in Fig. 3. The protocol enables a prover with physical access to the PUF to authenticate itself to a verifier. It is assumed that an honest verifier has access to a compact secret model of the relationship between

Strong PUF challenge-response pairs (CRPs). Such a model can be built by training a compact parametric model of the PUF on a set of direct challenge responses pairs. As long as the PUF challenge response pairs are obtained from the linear PUF, right after the arbiter, building and training such a compact model is possible with a relatively small set of CRPs as demonstrated in the previous literature [6], [10]–[13]. The physical access to the measurement points should be then permanently disabled before deployment, e.g., by burning irreversible fuses, so other entities cannot build the same models. Once this access point is blocked, any physical attack that involves de-packaging the chip will likely alter the shared secret.

The Slender PUF methodology is different from the original PUF challenge response pair identification and authentication methodology. The Slender PUF methodology is devised such that both prover and verifier jointly participate in producing the challenges. The joint challenge generation provides effective protection against a number of attacks. Unlike original PUF authentication methods, an adversary cannot build a database of CRPs and use an entry in the database for authentication.

In the next step of the protocol, the prover generates a set of Strong PUF responses corresponding to the jointly generated challenges. After that, the prover selects a random substring of responses from the response super-string, without revealing the location in the response stream and sends it to the verifier. The verifier, with access to the secret compact PUF model, can perform substring matching, within a predefined error threshold, and validate the responses with a very high probability. The prover gets authenticated if his submitted response substring matches at any location in the simulated response super-string on the verifier side.

A. Slender PUF protocol steps

Fig. 5 illustrates the steps of the Slender PUF protocol. Steps 1-4 of the protocol ensure joint generation of the challenges by the prover and the verifier. In Steps 1-2 the prover and the verifier each uses its own true random number generator (TRNG) unit to generate a nonce. Note that arbiter PUFs can also be used to implement a TRNG [21]. The prover and verifier generated nonces are denoted by $Nonce_p$ and $Nonce_v$ respectively. The nonces are exchanged between the parties, so both entities have access to $Nonce_p$ and $Nonce_v$. Step 3 generates a random seed by concatenating the individual nonces of the prover and the verifier; i.e., $Seed = \{Nonce_{\text{v}} \parallel Nonce_{\text{p}}\}$.

The generated $Seed$ is used by a pseudo-random number generator (PRNG) in Step 4. Both the prover and the verifier have a copy of this PRNG module. The PRNG output using the seed, i.e., $C = G(Seed)$, is then applied to the PUF as a challenge set (C). Note that in this way, neither the prover nor the verifier has full control over the PUF challenge stream. In Step 5, the prover applies the challenges to its physical PUF to obtain a response stream (R); i.e., $R = \text{PUF}(C)$. An honest verifier with access to a secret compact model of the PUF (PUF_model) also estimates the PUF output stream; i.e., $R' = \text{PUF\_model}(C)$.

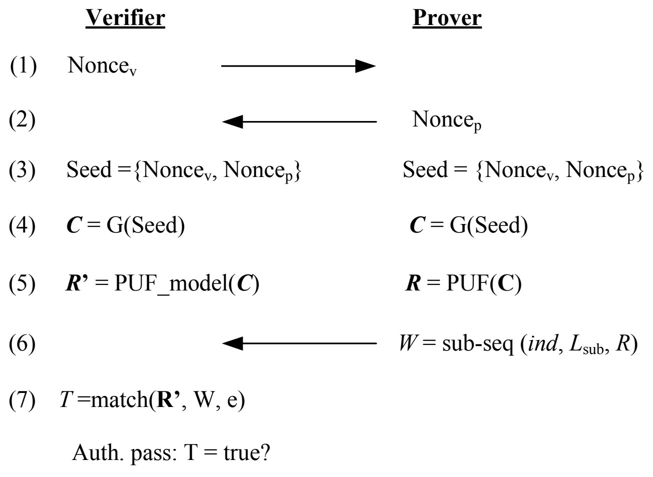
Verifier
Prover
(1) Nonce$_v$
(2) Nonce$_p$
(3) Seed ={Nonce$_v$, Nonce$_p$} Seed = {Nonce$_v$, Nonce$_p$}
(4) C = G(Seed) C = G(Seed)
(5) R' = PUF_model(C) R = PUF(C)
(6) W = sub-seq (ind, L_{sub}, R)
(7) T =match(R', W, e)
Auth. pass: T = true?

Fig. 5. The 7 steps of the SlenderPUF lightweight protocol.

Let us assume that the full response bitstring is of length L. In Step 6, the prover randomly chooses an index (ind) of bit-size $\log_2(L)$ that points to a location in the full response bitstring. The index is used to generate a substring W from the PUF output bitstream with a predefined length, denoted by L_{sub}. We use the full response string in a circular manner, so that if the value $(ind + L) > \log_2(L)$, the remainder of the substring values are taken from the beginning of the full response bitstream.

The prover then sends W to the verifier. In step 7, an honest verifier, with access to the compact secret PUF model, finds the secret index by searching and matching the received substring to its simulated PUF output sequence (R'). The authentication is successful, only if the Hamming distance between the received and the simulated substrings is lower than a predefined threshold value. In this way, prover does not reveal the whole response stream and the protocol leaks a minimal amount of information. This process is illustrated in Fig. 6.

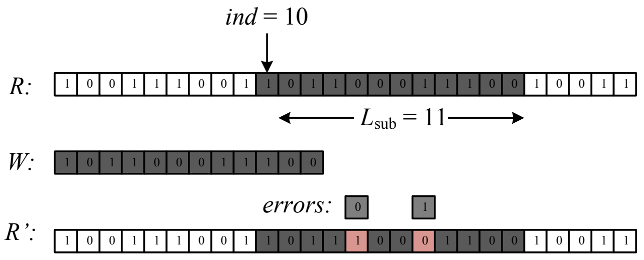

Fig. 6. Top: random selection of an index; Middle: extracting a substring of a predefined length; Bottom: the verifier matches the received substrings to its estimated PUF response stream.

The SlenderPUF protocol is lightweight and is suitable for ultra-low power and embedded devices. Besides a Strong PUF, the prover only needs to implement one TRNG and one PRNG. The information communicated between the parties is also minimal. In addition to exchanging their respective session nonces, the prover only needs to send a relatively short substring to the verifier.

B. Secret sharing

So far we assumed that the verifier possesses a model of the PUF and uses the model to authenticate the prover. The PUF in fact uses an e-fuse to protect the secret and prevent modeling attacks. The chip sets are handled by a trusted party before distributing to end users. The trusted party performs modeling on the PUF and disables the fuse before distribution. Anyone with access to the IC afterwards will not be able to model the PUF since the fuse is disabled. The trusted party can share the PUF models with other authorized trusted parties that want to authenticate the ICs.

The e-fuse mechanism is set up as follows. Before the e-fuse is disabled, the output of the arbiter prior to any XORs can be read and accessed from chip IO pins. This way, the verifier can obtain as many CRPs as needed to build an accurate model of the PUF. After the model is successfully trained, the trusted party and/or the verifier disables the e-fuse so that no one can obtain the "raw" PUF output.

V. Analysis of attacks

In this section, we quantify the resistance of the proposed protocol against different attacks by a malicious party (prover or verifier). First, we quantitatively analyze the resiliency of the method to machine learning and modeling attacks. Second, we probabilistically investigate the odds of authentication by random guessing. Third, we address the attack where a dishonest prover (verifier) attempts to control the PUF challenge pattern. Lastly, the effects of non-idealities of PUFs and PRNGs and their impact on protocol security are discussed.

Throughout our analysis in this section, we investigate the impact of various parameters on security and reliability of protocol operation. Table I shows the list of parameters.

TABLE I
List of design parameters

Parameter notation	Description
L	Length of PUF response string
L_{sub}	Length of PUF response substring
L_{n}	Length of the nonce
ind	Index value, $0 \leq ind < L$
N_{min}	Minimum number CRPs needed to train the PUF model with a misclassification rate of less than ϵ
k	Number of XORed PUF outputs
N	Number of PUF switch stages
th	Matching distance threshold
ϵ	PUF modeling misclassification rate
p_{err}	Probability of error in PUF responses

A. PUF modeling attack

In order to model a linear PUF with a given level of accuracy, it is sufficient to obtain a minimum number (N_{min}) of direct challenge response pairs (CRPs) from the PUF. Based on theoretical considerations (dimension of the feature space, Vapnik-Chervonenkis dimension), it is suggested in [6] that the minimal number of CRPs, N_{min}, that is necessary to model a N-stage delay based linear PUF with a misclassification rate of ϵ is given by:

$$N_{min} = O(\frac{N}{\epsilon}). \tag{4}$$

For example, a PUF model with 90% accuracy, has a misclassification rate of ϵ = 10%. In the proposed protocol, the direct responses are not revealed and the attacker needs to correctly guess the secret index to be able to discover L_{sub} challenge response pairs. The index is a number between 0 and $L - 1$ (L is the length of the original response string from which the substring is obtained). Assuming the attacker tries to randomly guess the index, then he is faced by L choices. For each *index* choice, the attacker can build a PUF model (M_{index}) by training it on the set of L_{sub} challenge response pairs using machine learning methods.

Now, the attacker could launch L rounds of authentication with the verifier and each time use one of his trained models instead of the actual PUF. If he correctly guesses the index and his model is accurate enough, one of his models will pass authentication. To build an accurate model as mentioned above, the attacker needs to obtain N_{min} correct challenge response pairs. If $L_{sub} > N_{min}$, then attacker can break the system with O(L) number of attempts. However if $L_{sub} < N_{min}$, then the attackers needs to launch N_{min}/L_{sub} multiple rounds of authentication to obtain at least N_{min} challenge response pairs. Under this scenario, the number of hypothetical PUF models will grow exponentially. Since for each round of authentication there are L models based on the choice of index value, for N_{min}/L_{sub} rounds, the number of models will be of the following order:

$$O(L^{\frac{N_{min}}{L_{sub}}}). \quad (5)$$

From the above equation, it seems intuitive to choose small values for L_{sub} to make the exponent bigger. However, small L_{sub} increases the success rate of random guessing attacks. The implications of small L_{sub} will be discussed in more detail in the next section.

The model the attacker is building has to be only more accurate than the specified threshold during the matching. For example, if we allow a 10% tolerance during the substring matching process, then it means that a PUF model that emulates the actual PUF responses with more than 90% accuracy will be able to pass authentication. Based on Equation 4, if we allow higher misclassification rate ϵ, then a smaller number of CRPs is needed to build an accurate enough model which passes the authentication.

For example, based on the numbers reported in [6], using 640 CRPs, an arbiter PUF of length 64 can be modeled with an accuracy of 95%. In this example we set the threshold to 5%, then to get an exponent equal to 10 from Equation 5, L_{sub} must be 64. In other words, the attacker needs to performs L^{10} operations to obtain 640 CRPs so that he can build a PUF model of 95% accuracy to pass the authentication. For L=1024, L^{10} will be a huge number. However we are faced with another problem. What if the PUF error rate (p_{err}) is higher than the maximum Hamming distance threshold (th)? Then we will have a lot of false negatives (i.e., the honest prover with access to the legitimate PUF will not be able to pass authentication due to noise in responses).

To improve the security while maintaining reliable performance, N_{min} must be increased for a fixed ϵ and N. This requires a structural change to delay based PUF. In this paper, we use the XOR PUF circuit shown in Figure 3 for two reasons. First, to satisfy the avalanche criterion for the PUF. Second, to increase N_{min} for a fixed ϵ. Based on the results reported in [6], N_{min} is an order of magnitude larger for XOR PUF compared to a simple delay based PUF.

B. *Random guessing attack*

A legitimate prover should be able to generate a substring of PUF responses that successfully match a substring of the verifier's emulated response sequence. The legitimate prover must be authenticated by an honest verifier with a very high probability, even if the response substring contains some errors. Therefore, the protocol allows some tolerance during matching by setting a threshold on the Hamming distance of the source and target substrings.

Simultaneously, the probability of authenticating a dishonest prover should be extremely low. These conditions can be fulfilled by carefully selecting the Hamming distance threshold (th), the substring length (L_{sub}) and the original response string length (L) by our protocol.

A dishonest prover without access to the original PUF or its model, may resort to sending a substring of random bits. In this case, the probability of authenticating a randomly guessing attacker would be:

$$P_{\text{auth,guessing}} \leq L \times \sum_{i=L_{\text{sub}}-th}^{L_{\text{sub}}} \binom{L_{\text{sub}}}{i} \frac{1}{2}^{i} \cdot \frac{1}{2}^{L_{\text{sub}}-i}, \quad (6)$$

where L_{sub} and th are the length of the substring and the Hamming distance threshold, respectively. Note that Eq. 6 is a binomial cumulative distribution function. For an honest prover, the probability of being authenticated is:

$$P_{\text{auth,honest}} \simeq \sum_{i=L_{\text{sub}}-th}^{L_{\text{sub}}} \binom{L_{\text{sub}}}{i} (1-p_{\text{err}})^{i} \cdot p_{\text{err}}{}^{L_{\text{sub}}-i}, \quad (7)$$

where p_{err} is the probability of an error in a response bit. If L_{sub} is chosen to be a sufficiently large number, Eq. 6 will be close to zero and Eq. 7 will be close to one. Table II lists the probability of authenticating an honest prover and a dishonest randomly guessing prover for various L_{sub}, p_{err} and th.

As the results in Table II suggest, by setting L_{sub} = 512, and th = 154, the protocol guarantees reliable operation under 20% response error and zero probability of success for a random attacker. Meanwhile, assuming that with N_{min} = 5120 CRPs, the attacker can train the 4-XORed PUF model with more than 80% accuracy, then

TABLE II
THE PROBABILITY OF AUTHENTICATING AN HONEST PROVER AND AND A DISHONEST GUESSING PROVER FOR VARIOUS L_{SUB}, p_{err}, th AND L = 1024.

Substring length, L_{sub}	Hamming threshold, th	$p_{guessing}$ 50%	p_{err} 5%	p_{err} 10%	p_{err} 20%
128	33	$6 * 10^{-9}$	1	1	0.9332
256	76	10^{-11}	1	1	0.9999
512	154	0	1	1	0.9999

for L = 1024 the complexity of a model building attack will be $1024^{(5120/512)} = 10^{30}$. Note that N_{min} is chosen pessimistically and typical values based on the numbers reported in [6] are an order of magnitude larger.

C. Compromising the random seed

In Slender PUF protocol, the prover and the verifier jointly generate the random PRNG seed by concatenating the outputs of their individual nonces (generated by TRNGs); i.e., $seed = \{Nonce_{\text{v}} \parallel Nonce_{\text{p}}\}$. The stream of PRNG outputs after applying the seed is then used as the PUF challenge set. This way, neither the prover nor the verifier has full control over generating the PUF challenge stream.

If one of the parties can fully control the seed and challenge sequence, then the following attack scenario can happen. A dishonest verifier can manipulate an honest prover into revealing the secret information. If the same seed is used over and over during authentication rounds, then the generated response sequence (super-string) will always be the same. The response substrings now come from the same original response string. By collecting a large enough number of substrings and putting the pieces together, the original super-string can be reconstructed. Reconstruction will reveal L CRPs. By repeating these steps more CRPs can be revealed and the PUF can be ultimately modeled.

A dishonest prover (verifier) may intentionally keep his/her portion of the seed constant to reduce the entropy of seed. This way, the attacker can exert more control over the random challenges applied to the PUF. We argue that if the seed length is long enough this strategy will not be successful.

This attack leaves only half of the bits in the generated $Seed$ changing. For a seed of length $2L_{\text{n}}$-bits (two concatenated nonces of length L_{n}-bits), the chance that the same nonce appears twice is $\frac{1}{2^{L_{\text{n}}}}$). For example, for $L_{\text{n}} = |Nonce_{\text{v}}| = |Nonce_{\text{p}}| = 128$, the probability of being able to fully control the seed will be negligibly small.

Therefore, one could effectively guard against any kind of random seed compromise by increasing the nonce lengths. The only overhead of this approach is a twofold increase in the runtime of the TRNG.

D. Substring replay attack

A dishonest prover may mount an attack by recording the substrings associated with each used $Seed$. In this attack, a malicious prover records the response substrings sent by an honest prover to an honest verifier for a specific $Seed$. The recording may be performed by eavesdropping on the communication channel between the legitimate prover and verifier. A malicious party may even pre-record a set of response substrings to various random $Seed$s by posing as a legitimate verifier and exchanging nonces with the authentic prover.

After recording a sufficiently large number of $Seeds$ and their corresponding response substrings, the malicious party could attempt to impersonate an honest prover. This may done by repeatedly contacting the legitimate verifier for authentication and then matching the generated $Seed$s to its pre-recorded database. This attack could only happen if the $Seed$s collide. Selecting a sufficiently long $Seed$ that cannot be controlled by one party (Subsection V-B) would hinder this collision attack.

Passive eavesdropping is performed during the pre-recording phase, the chances that the whole $Seed$ collides will be $1/2^{L_{\text{n}}}$. The worst-case scenario is when an adversary impersonates a verifier and controls half of the seed which reduces the collision probability to $1/2^{L_{\text{n}}/2}$.

E. Exploiting non-idealities of PRNG and PUF

Thus far, we assumed that the outputs of PRNG and PUF are ideal and statistically unbiased. If this is not true, an attacker may resort to exploiting the statistical bias in a non-ideal PRNG or PUF to attack the system. Therefore, in this section we emphasize the importance of the PUF avalanche criterion for securing against this class of attacks.

If the PUF has poor statistical properties, then the attacker can predict patterns in the generated responses. The attacker can use these predicted patterns to more confidently find/guess a matching location for the substring. In other words, statistical bias in the responses will leak information about the location index of the response substring.

Recall that an ideal Strong PUF should have the strict avalanche property [16]. This property states that if one bit of the PUF's input challenges is flipped, the PUF output response should flip with a $\frac{1}{2}$ probability. If this property holds, the PUF output for two different challenges will be uncorrelated. Fig. 4 shows the probability of output flipping versus the Hamming distance between two challenge sequences for the Strong PUF proposed in [16]. It is desirable to make this probability as close as possible to $\frac{1}{2}$.

The figure shows that this probability is very close to the ideal number when at least four independent PUF output bits are mixed by an XOR. As more independent PUF response bits are mixed, the curve moves closer to the ideal case; however, this linearly increases the probability of error in the mixed output bit. For instance, for a single Strong PUF response bit error of 5%, the probability of error for 4-XOR mixing is reported to be 19% in [16].

In our implementation of Slender PUF protocol, Linear feedback shift registers (LFSRs) are used as a lightweight PRNG. An ideal LFSR must have the maximum length sequence property [22]. This property ensures that the autocorrelation function of the LFSR output stream is "impulsive", i.e., it is one at lag zero and is $\frac{-1}{N}$ for all other lags, where N is the LFSR sequences length. N should be a sufficiently large number, which renders the lagged autocorrelations very close to zero [22]. Therefore, if an LFSR generates a sequence of challenges to the PUF, the challenges are uncorrelated. In other words, for an ideal LFSR, it is highly unlikely that an attacker can find two challenges with a very small Hamming distance.

Even if the attacker finds two challenges with a small Hamming distance in the sequence, Fig. 4 shows that the output of our proposed PUF would be sufficiently uncorrelated to the Hamming distance of the input challenges. Therefore, a combination of PRNG and PUF with strict avalanche criteria would make this attack highly unlikely. It is worth noting that it is not required by any means the PRNG to be a cryptographically secure generator. The seed in the protocol is public and the only purpose of the PRNG is to automatically generate a sequence of challenge vectors. Simultaneously, it must not allow an attacker to completely control the challenges and thus the responses.

VI. Hardware implementation

In this section, we present an FPGA implementation of the proposed protocol for the prover side on Xilinx Virtex 5 XC5VLX110T FPGAs. Since there is a stricter power consumption requirement on the lightweight prover, we focus our evaluation on prover implementation overhead. The computation on the verifier side can run solely in software, however, the computation on the verifier may also be carried out in hardware with negligible overhead.

For the Slender PUF protocol, it is desirable to use a low overhead PUF implementation, such as the one introduced in [23]. If an ASIC or analog implementation of the PUF is required, the ultra-low power architecture in [21] is suitable for this protocol. A very low-power verifier implemented by a microcontroller such as MSP430 can easily challenge the PUF and run the subsequent steps of the protocol.

We use the implementation of the arbiter-based PUF in [24]. The arbiter-based PUF on FPGA is designed to have 64 input challenges. In total, 128 LUTs and one flip-flop are used to generate one bit of response. To achieve a higher throughput, multiple parallel PUFs can be implemented on the same FPGA.

There are various existing implementations for TRNGs on FPGAs [25], [26]. We use the architecture presented in [23] to implement a true random number generator. The TRNG architecture is shown in Figure 8. This TRNG operates by enforcing a metastable state on the flipflop through a closed loop feedback system. The TRNG core consumes 128 LUTs that are packed into 16 CLBs on Virtex 5. In fact, the TRNG core is identical to the arbiter-based PUF except that the switches act as tunable programmable delay lines. The core is incorporated inside a closed-loop feedback system. The core output is attached to a 12-bit counter (using 12 registers) which monitors the arbiter's metastability. If the arbiter operates in a purely metastable fashion, the output bits become equally likely ones and zeros. The counter basically measures and monitors deviations from this condition and generates a difference feedback signal to guide the system to return back to its metastable state. The counter output drives an encoding table of depth 2^{12} where each row contains a 128-bit word resulting in a 64KByte ROM. A table of size $2^{12} \times 8$-bits (=4KByte) implemented by a RAM block is used to gather and update statistics for online post processing.

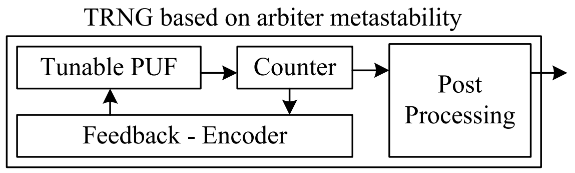

Fig. 7. True random number generation architecture based on flipflop metastability

The nonce size is set to 128 for both the prover and verifier. Each 128-bit nonce is fed into a 128-bit LFSR. The content of the two LFSRs are XORed to form the challenges to the PUF.

The pattern selection can be achieved by shifting the intended substring of the PUF responses into a FIFO. The shifting operation, however, begins only when needed. In other words, before running the PUF, the random index is generated by the TRNG. For example, in our implementation the response sequence has a length of 1024 which results in a 10-bit index. To generate a 10-bit random index, we have to run the TRNG 8×10 clock cycles according to Table III. Since we do not care about the response bits that are generated before and after the substring window, we do not need to even generate or store those bits. Therefore, the PUF has to only be challenged for the response bits in the substring. This significantly reduces the overall run time and the storage requirement on the FIFO. The FIFO size is accordingly equal to the length of the substring which is set to 256 in our implementation.

The propagation delay through the PUF and the TRNG core is equal to 61.06ns. PUF outputs can be generated at a maximum rate of 16Mbit/sec. Post-processing on the TRNG output bits can lower the throughput from 16Mbit/sec to 2Mbit/sec. Since the TRNG is only used to generate the nonce and the index, its throughput does not affect the overall system performance; the number of required true random bits is smaller than the PUF response bits.

TABLE III
IMPLEMENTATION OVERHEAD ON VIRTEX 5 FPGA

No.	Type	LUT	Registers	RAM blocks	ROM blocks	Clock Cycles
4	PUF	128	1	0	0	1
1	TRNG	128	12	4KB	64KB	8
1	FIFO	0	256	0	0	N/A
2	LFSR	2	128	0	0	N/A
1	Control	12	9	0	0	N/A
Total		652	278	4KB	64KB	N/A

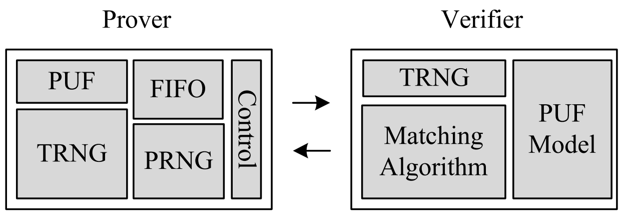

Fig. 8. Resource usage on prover and verifier sides

The implementation overhead of our proposed protocol is much less than traditional cryptographic modules. For example, robust hashing implementation of SHA-2 as listed in Table IV requires at least 5492 LUTs of a Virtex-II FPGA [27] and it takes 68 clock cycles to evaluate. This overhead will occur on the top of the clock cycles required for PUF evaluation.

Finally, note that most of the area overhead for the protocol implementation is coming from the TRNG. By using non-volatile memory storage, TRNG can also be avoided. A Slender PUF implementation without a TRNG generates $log_2(L)$ extra response bits and uses the extra bits as the index value. Also to generate the nonce, previously used and revealed substring response bits can re-write the nonces and be used for the next round of authentication. This is because for a statistically unbiased PUF, the responses follow random number properties. In this case, the contents of the non-volatile memory is publicly available and there will be no external access point to change or re-write the values to the memory. However, this implementation is vulnerable to invasive attacks that aim to alter the memory content.

TABLE IV
SHA-2 IMPLEMENTATION OVERHEAD AS REPORTED IN [27]

SHA-256	Freq. (MHz)	Clock Cycles	TP (Mbps)	Area (LUTs)
Basic	133.06	68	1009	5492
2x-unrolled	73.97	28	996.7	8128
4x-unrolled	40.83	23	908.9	11592

VII. CONCLUSIONS AND FUTURE DIRECTION

We presented Slender PUF, a lightweight and secure protocol to dependably authenticate the responses generated from a Strong PUF with minimal information leaked to adversaries. The prover in this protocol reveals only a random subset of responses for authentication. The verifier which has access to an approximate model of the PUF can search and match the received substring with the estimated PUF response string. The authentication is successful if a sufficiently close match is found. We demonstrated that a carefully-designed Slender PUF is resilient against all known machine learning attacks. The experimental results on FPGAs showed a significantly lower area and speed overhead compared to any protocol that potentially uses conventional cryptographic modules such as hashing. An even smaller footprint and power consumption can potentially be achieved by using analog leakage based PUFs, analog TRNGs, and low power micro-controllers.

VIII. Acknowledgment

This research is in part supported by Army Research Office (ARO) YIP award under grant No. R17450, Office of Naval Research (ONR) YIP award under grant No. R16480, and Semiconductor Research Corporation (SRC) award under grant task No. 1836.039.

References

[1] P. S. Ravikanth, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, pp. 2026–2030, 2002.

[2] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *Computer and Communication Security Conference (CCS)*, 2002, pp. 148–160.

[3] U. Ruhrmair, S. Devadas, and F. Koushanfar, *Security based on Physical Unclonability and Disorder*. Springer, 2011.

[4] F. Armknecht, R. Maes, A. Sadeghi, F.-X. Standaert, and C. Wachsmann, "A formalization of the security features of physical functions," in *IEEE Symposium on Security and Privacy*, 2011, pp. 397 –412.

[5] R. Maes and I. Verbauwhede, "Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions," in *Towards Hardware-Intrinsic Security*, A.-R. Sadeghi and D. Naccache, Eds. Springer, 2010.

[6] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *ACM Conference on Computer and Communications Security (CCS)*, 2010, pp. 237–249.

[7] Z. Paral and S. Devadas, "Reliable and efficient PUF-based key generation using pattern matching," in *International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2011, pp. 128 –133.

[8] F. Koushanfar, *Hardware Metering: A Survey.* Springer, 2011.

[9] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Delay-based circuit authentication and applications," in *Proceedings of the 2003 ACM symposium on Applied computing*, 2003, pp. 294–301.

[10] D. Lim, "Extracting Secret Keys from Integrated Circuits," Master's thesis, Massachusetts Institute of Technology, may 2004.

[11] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Testing techniques for hardware security," in *International Test Conference (ITC)*, 2008, pp. 1–10.

[12] B. Gassend, "Physical Random Functions," Master's thesis, Massachusetts Institute of Technology, jan 2003.

[13] E. Oztürk, G. Hammouri, and B. Sunar, "Towards robust low cost authentication for pervasive devices," in *Pervasive Computing and Communications (PerCom)*, 2008, pp. 170–178.

[14] G. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Design Automation Conference (DAC)*, 2007, pp. 9–14.

[15] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Lightweight secure PUF," in *International Conference on Computer Aided Design (ICCAD)*, 2008, pp. 670–673.

[16] ——, "Techniques for design and implementation of secure reconfigurable PUFs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 2, no. 1, 2009.

[17] C. Bösch, J. Guajardo, A. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient helper data key extractor on FPGAs," in *Cryptographic Hardware and Embedded Systems (CHES)*, 2008, pp. 181–197.

[18] R. Maes, P. Tuyls, and I. Verbauwhede, "Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs," in *Cryptographic Hardware and Embedded Systems (CHES)*, 2009, pp. 332–347.

[19] M.-D. M. Yu and S. Devadas, "Secure and robust error correction for physical unclonable functions," *IEEE Design and Test of Computers*, vol. 27, pp. 48–65, 2010.

[20] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: how to generate strong keys from biometrics and other noisy data," in *Advances in Cryptology - Eurocrypt*, 2004.

[21] M. Majzoobi, G. Ghiaasi, F. Koushanfar, and S. Nassif, "Ultra-low power current-based PUF," in *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 2071–2074.

[22] M. Baldi, F. Chiaraluce, N. Boujnah, and R. Garello, "On the autocorrelation properties of truncated maximum-length sequences and their effect on the power spectrum," *Signal Processing, IEEE Transactions on*, vol. 58, 2010.

[23] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA-based true random number generation using circuit metastability with adaptive feedback control," *Cryptographic Hardware and Embedded Systems–CHES 2011*, pp. 17–32, 2011.

[24] ——, "FPGA PUF using programmable delay lines," 2010, pp. 1 – 6.

[25] C. K. Koc, Ed., *Cryptographic Engineering*, 1st ed. Springer, Dec. 2008.

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

[27] R. McEvoy, F. Crowe, C. Murphy, and W. Marnane, "Optimisation of the SHA-2 family of hash functions on FPGAs," in *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on*. IEEE, 2006, pp. 6–pp.