IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 10 May 2013; revised 9 October 2013; accepted 22 December 2013.
Date of publication 15 January 2014; date of current version 7 May 2014.

Digital Object Identifier 10.1109/TETC.2014.2300635

Robust and Reverse-Engineering Resilient PUF Authentication and Key-Exchange by Substring Matching

MASOUD ROSTAMI[1], MEHRDAD MAJZOOBI[1], FARINAZ KOUSHANFAR[1],
DAN S. WALLACH[2], AND SRINIVAS DEVADAS[3] (Fellow, IEEE)

[1]Department of Electrical and Computer Engineering, Rice University, Houston, TX 77005, USA
[2]Department of Computer Science, Rice University, Houston, TX 77005, USA
[3]Department of Electrical and Computer Engineering, Massachusetts Institute of Technology, Cambridge, MA 02142, USA
CORRESPONDING AUTHOR: M. ROSTAMI (masoud@rice.edu)

This work was supported by the Army Research Office YIP Award under Grant R17450, the Office of Naval Research YIP Award under Grant R16480, the Semiconductor Research Corporation Award under Grant Task 1836.039, and NSF Career Award under Grant NSF-0644289.

ABSTRACT This paper proposes novel robust and low-overhead physical unclonable function (PUF) authentication and key exchange protocols that are resilient against reverse-engineering attacks. The protocols are executed between a party with access to a physical PUF (prover) and a trusted party who has access to the PUF compact model (verifier). The proposed protocols do not follow the classic paradigm of exposing the full PUF responses or a transformation of them. Instead, random subsets of the PUF response strings are sent to the verifier so the exact position of the subset is obfuscated for the third-party channel observers. Authentication of the responses at the verifier side is done by matching the substring to the available full response string; the index of the matching point is the actual obfuscated secret (or key) and not the response substring itself. We perform a thorough analysis of resiliency of the protocols against various adversarial acts, including machine learning and statistical attacks. The attack analysis guides us in tuning the parameters of the protocol for an efficient and secure implementation. The low overhead and practicality of the protocols are evaluated and confirmed by hardware implementation.

INDEX TERMS Physical unclonable functions, hardware security, security protocols.

I. INTRODUCTION

Classic security paradigms rely on a stored digital secret key and cryptographic algorithms. Secret keys are stored in an 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., by scanning electron microscopes). Moreover, correct implementation of security algorithms based on a pre-distributed secret key requires Password-Authenticated Key Exchange (PAKE) protocols. These protocols are provably secure; however, they require costly exponentiation operations [1], [2]. Therefore, they are not suitable for many low power resource-intensive applications.

Physical unclonable functions (PUFs) have been proposed [3] to provide a desired level of security with low implementation overhead. One type of PUF is based on silicon, and is designed to bind secrets to silicon hardware [4]. 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 must be designed to make it prohibitively hard to simulate, emulate, or predict their behavior [4]. Excellent surveys of various PUF designs can be found in [5]–[8].

Strong PUFs are a class of PUFs which have the property that the number of their possible challenge-response pairs (CRPs) has an exponential relationship with respect to the number of their physical components. This huge space of possible CRPs hinders attacks based on pre-recording and replaying previously used CRPs. However, physical components of a Strong PUF are finite. Therefore, given access to these components, a compact polynomial-order model of the CRP relationships can be built.

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 direct responses. Such compact models can be treated as a *secret* which can be used by a trusted Verifier to authenticate the Prover's PUF. (It should be noted that the physical access to these components should be permanently disabled before field deployment to avoid direct compact modeling.) An unfortunate fact is that third party observers may also be able to model the PUF based on a finite number of CRPs exchanged on the communication channel as it has been done before, see for e.g., [9]. This type of PUF *modeling* by untrusted third parties is also called the *machine learning* or *reverse engineering* attack as it harms the PUF security. Such attacks were possible because the challenge and response strings leak structural information about the PUF and compact models.

In this paper, we propose secure, low overhead, and robust authentication and key exchange protocols for the Strong PUFs that thwart the machine learning attack. The protocols enable a Prover with physical access to the PUF to authenticate itself to a trusted Verifier. It is assumed that the trusted Verifier has access to the secret compact PUF model. The protocol leaks minimal amount of information about secret PUF parameters on the communication channel. This is because the secret is the *index* of a response substring which is randomly selected from the full response string. The Prover also adds random padding strings to the beginning and end of the substring, where the indices of the padded bits is also a part of the secret. Only the substring is sent on the channel. Since the indices are not correlated with the substring content in any ways, the secret itself is never exposed on the communication channel. The Verifier, with access to the full string, can perform a substring matching and find the secret index. The matched strings may not be the same, but as long as they are within a small distance of each other (defined by a threshold), the matching is successful. Therefore, the method is inherently robust to the noise in the PUF responses eliminating the need for costly error correction or fuzzy extraction.

The protocol is devised such that the Verifier and the Prover jointly generate the challenges to the PUF. The challenges are generated in a way that neither a dishonest Prover nor a dishonest Verifier can solely control the challenges used for authentication. While none of the authenticating parties can solely control the challenges, the resulting challenge values are publicly known. The authentication protocol, described above, can also be leveraged to implement a low-power and secure key-exchange algorithm. The Prover only needs to select a random password and then encode it as a set of secret indices that was used in the authentication protocol.

We provide a thorough discussion of the complexity and effectiveness of attacks on proposed protocols. The protocols are designed to achieve robustness against inherent noise in PUF response bits, without costly traditional error correction modules. We demonstrate that our protocols can be implemented with a few simple modules on the Prover-side. Therefore, we do not need expensive cryptographic hashing and classic error correction techniques that have been suggested in earlier literature for achieving security. Note that recent work has used pattern matching for correcting errors while generating secret keys from a PUF [10]. However, unlike our protocol, the number of generated secret keys were limited. In addition, a higher level of protection against machine learning attacks can be achieved by our proposed protocols. To the best of our knowledge, no application of string matching for either authentication and key exchange based on Strong PUFs have been proposed before our work.

An earlier version of this work was published in [11]. Our previous work only discussed the application of PUFs for robust and attack-resilient authentication and did not propose a key exchange protocol based on PUFs. The proposed authentication protocol in [11] achieves a lower level of security than the proposed protocol in this paper. This is because we also add random padding to the PUF substring which generates a larger number of secret indices.

In brief, the main new contributions of our work are as follows:

- We introduce and analyze two lightweight and secure protocols based on substring-matching of PUF response strings to perform authentication and session key exchange.
- The protocols automatically provide robustness against inherent noise in the PUF response string, without requiring externally added and costly traditional error correction modules or fuzzy extraction.
- We perform a thorough analysis of the resiliency of protocols against a host of attacks.
- Our analyses provide guidelines for setting the protocol parameters for a robust and low-overhead operation.
- 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. Authentication and key exchange protocols are described in Section IV. The parameters of our protocols and their security against multiple attacks are investigated in Section V. The trade-offs in choosing the parameters of the protocols are explored in Section VI. Hardware implementation and performance evaluations are presented in Section VII. Section VIII concludes the paper. If the reader is familiar with PUF circuits and its related literature, he can now jump to Section IV.

II. BACKGROUND ON STRONG PUFs

In this section, without loss of generality, we introduce a popular instance of Strong PUF 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.

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. The other type is called *Strong PUF* [12]. Strong PUFs are built based on the unclonable 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.

In this work, we use a Strong PUF implementation called "delay-based arbiter PUF" introduced in [13]. In this PUF, the delay difference between two parallel paths is compared. The paths are built identically to make their nominal delays equal by design. However, the delay of fabricated paths on chips will be different due to process variations, see Fig. 1. A step input simultaneously triggers the two paths. At the end of the two parallel (racing) paths, an arbiter (typically a D-Flip Flop) is used to convert the analog difference between the paths to a digital value. The arbiter output becomes one if the signal arrives at its first input earlier than the second one, otherwise, it stays at zero. 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).

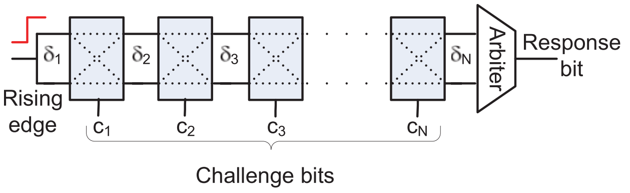

FIGURE 1. An arbiter linear PUF block with *N* challenges and one response bit. The arbiter converts the analog delay difference between the two paths to a digital value.

The PUF only consists of linear addition and subtraction of delay elements. Therefore, the behavior of the PUF in Fig. 1 can be modeled by the following linear inequality [14]:

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

where δ_j is the differential segment delay and ρ_j is related to the input challenge that controls the switch selectors by the following relation,

$$\rho_i = \bigoplus_{x=i,i+1,\ldots,N} C_x = C_i \oplus C_{i+1} \oplus \ldots \oplus C_N. \tag{2}$$

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:

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

where $\Delta = [\delta_1, \delta_2, \ldots, \delta_{N+1}]$ is the delay parameter vector, $\Phi = [(-1)^{\rho_1}, (-1)^{\rho_2}, \ldots, (-1)^{\rho_N}, 1] = [\varphi_1, \varphi_2, \ldots, \varphi_{N+1}]$ 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 to each other.

B. LINEAR ARBITER PUF STATISTICAL PROPERTIES

In this subsection, the statistical properties of a linear arbiter PUF are reviewed. It has been demonstrated in [15] 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 δs are independent and identically distributed Gaussian variables, 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. 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(HD(C_0, C_1)).[1] For example, in the trivial cases HD(C_0, C_1)=0, i.e. C_0=C_1, then $Prob\{r_0 \neq r_1\} = 0$. s the Hamming distances between the input challenge vector becomes larger, the probability of having different PUF response bits increases.

The second property leaks information about the PUF response sequence which would help in breaking the PUF security by pattern matching. Ideally, PUFs are expected to have a property called strict avalanche criterion. Any flip in the challenge bits of a PUF with avalanche criterion should cause the response bits to flip with probability of 50%. Any deviation from this criterion reduces the security of the system built based on these PUFs. o achieve this criterion, it has been proposed [15], [16] to mix the responses from the arbiter PUFs with XOR logic. In the next subsection, we review this subclass of PUFs.

C. XOR-MIXED ARBITER PUFs

Fig. 2 [15] shows a two-stage XOR-mixed arbiter PUF. In the figure, note that the challenge sequence in the second stage is applied in the reverse order. The order is flipped to help achieve the avalanche criterion. As more independent PUF

[1] The Hamming distance between challenges C_x and C_y is defined as $\text{HD}(C_x, C_y) = \sum_{i=1}^{N} \mid C_x[i] - C_y[i] \mid /N$ where $C_x[i], C_y[i] \in \{-1, 1\}$.

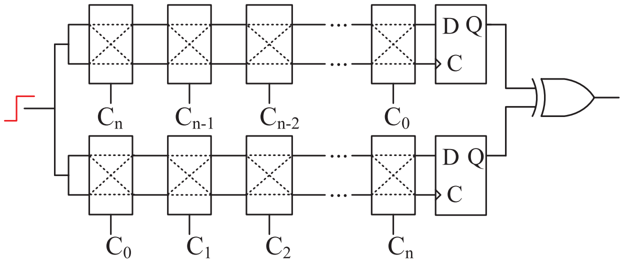

FIGURE 2. Two independent linear arbiter PUFs are XOR-mixed in order to implement an arbiter PUF with better statistical properties. The challenge sequence in the second stage is applied in the reverse order to help achieve this property.

response bits are mixed, the probability that output is flipped when one input bit changes, comes closer to the ideal probability of 0.5.

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. However, there is a cap on the number of stages that can be actually used in practice. This is due to the fact that XOR-mixing causes error accumulation of PUF responses. For instance, for a single PUF response bit error of 5%, the probability of error for a 4-XOR-mixed PUF is 19% [15]. The protocols proposed in this paper allows higher level of security without increasing the number of XOR stages.

In the rest of the paper, we build our protocols based on the assumption that the PUF at hand is a linear XOR-mixed arbiter PUF with near ideal statistical properties. We argue that our protocols are applicable to any Strong PUF which follows the statistical properties discussed in this Section.

III. RELATED WORK

PUFs have been subject to modeling attacks. The basis for contemporary PUF modeling attacks is collecting a set of CRPs, 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 new challenges 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 linear arbiter PUFs and feed-forward arbiter PUFs [9], [14], [15], [17], [18]. More comprehensive analysis and description of PUF security requirements to protect against modeling attacks were presented in [19]–[21]. In recent years, there have been an ongoing effort to model and protect PUFs against side channel attacks such as power analysis [22] and fault injection [23].

Extracting secret keys from PUF responses has been explored in previous work, including [4], [17], [24]–[26]. 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* [27], which has a high overhead.

In the context of challenge-response based authentication for Strong PUFs, sending the syndrome bits for correcting the errors before hashing was investigated [4]; 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. A side from sensitivity to PUF noise (because it satisfies the strict avalanche criterion), hashing and error correction has the drawback of high overhead in terms of area, delay, and power.

A newer information-theoretically secure Index-Based Syndrome (IBS) error correction coding for PUFs was introduced and realized in [26]. In [28], authors proposed the notion of public physically unclonable functions (PPUF) and proposed a public key-exchange protocol based on them.

All of the aforementioned methods incur a rather high overhead of error correction and/or hashing, which prohibits their usage in lightweight systems. An alternative efficient error correction method by pattern matching of responses was very recently proposed [10], which inspired the pattern matching method used in our protocols. However, their proposed protocol and application area was limited to secret key generation. Authors lightweight PUF authentication.

This paper introduces lightweight PUF authentication and key-exchange protocols based on string pattern matching and covert indices. Modeling attack against these protocols is thwarted by leaking very limited information from a PUF response string. The random indices used in the protocols are inherently independent of the response string content.

IV. AUTHENTICATION AND KEY EXCHANGE PROTOCOLS

In this section, the proposed authentication and key exchange protocols are introduced and explained in detail. The protocols are based on a Strong PUF with acceptable statistical properties, like the one shown in Fig. 2. The authentication protocol enables a Prover with physical access to the PUF to authenticate itself to a Verifier, and the key exchange protocol enables the Prover and the Verifier to securely exchange secret keys between each other.

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 response pairs. As long as the PUF challenge response pairs are obtained from the linear PUF, right before the XOR-mixing stage, building and training such a compact model is possible with a relatively small set of CRPs as demonstrated in [9], [14], [15], [17], and [18]. 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.

Unlike the original PUF challenge response pair identification and authentication methodologies, our protocols are 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 methods, an adversary cannot build a database of CRPs and use an entry in the database for authentication or key exchange. The next two subsections describe these protocols in details. The last subsection concludes the section with some notes about the PUF secret sharing process.

A. AUTHENTICATION PROTOCOL STEPS

Fig. 3 illustrates the steps of our authentication 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 [29]. 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., $Seed = \{Nonce_v \parallel Nonce_p\}$.

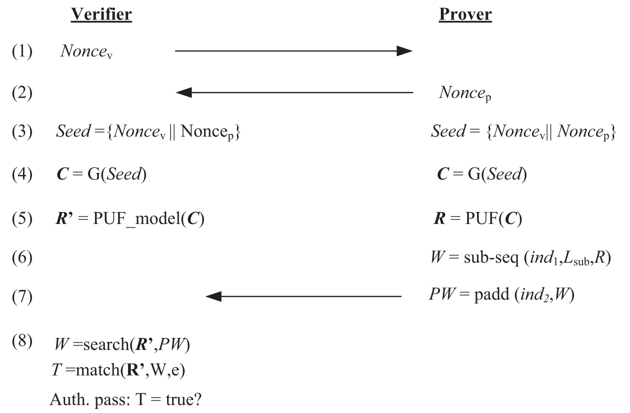
Verifier
Prover
(1) $Nonce_v$
(2) $Nonce_p$
(3) $Seed = \{Nonce_v \parallel Nonce_p\}$ $Seed = \{Nonce_v \parallel Nonce_p\}$
(4) $\boldsymbol{C} = \mathrm{G}(Seed)$ $\boldsymbol{C} = \mathrm{G}(Seed)$
(5) $\boldsymbol{R'} = \mathrm{PUF\_model}(\boldsymbol{C})$ $\boldsymbol{R} = \mathrm{PUF}(\boldsymbol{C})$
(6) $W = \text{sub-seq}\,(ind_1, L_{sub}, R)$
(7) $PW = \mathrm{padd}\,(ind_2, W)$
(8) $W = \mathrm{search}(\boldsymbol{R'}, PW)$
$T = \mathrm{match}(\mathbf{R'}, \mathrm{W}, \mathrm{e})$
Auth. pass: T = true?

FIGURE 3. **The 8 steps of PUF-based authentication protocol.**

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., $C = \mathrm{G}(Seed)$, 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., $R = \mathrm{PUF}(C)$. An honest Verifier with access to a secret compact model of the PUF (PUF_model) also estimates the PUF output stream; i.e., $R' = \mathrm{PUF\_model}(C)$.

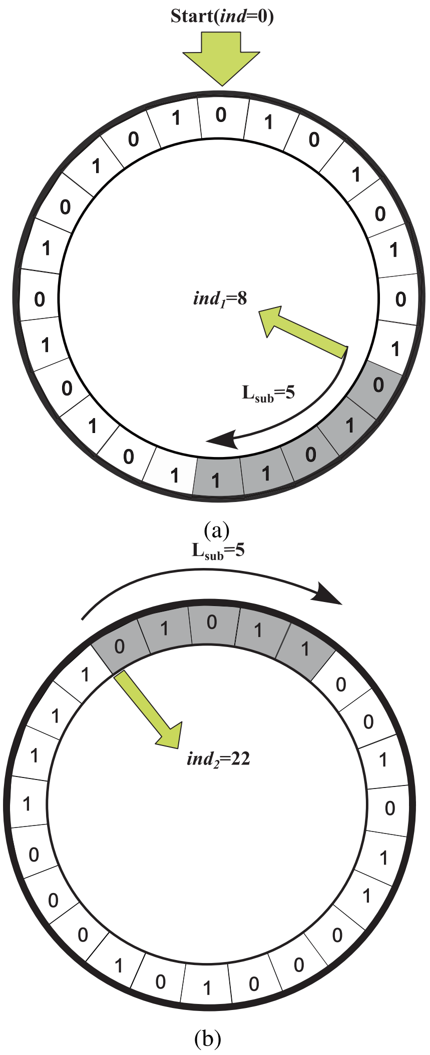

FIGURE 4. **The steps that are performed on the PUF response string by the Prover. Top: random selection of ind_1 and extraction of a substring with a predefined length. Bottom: circular padding the substring at a random location (ind_2) with random bits. In this toy example, $L = 24$, $L_{PW} = 24$, and $L_{sub} = 5$. Note the circular manner of extraction and padding. (a) Circular extraction of PUF substring. (b) Circular padding of substring.**

Let us assume that the full response bitstring is of length L. In Step 6, the Prover randomly chooses an index (ind_1) that points to a location in the full response bitstring. This index points to the beginning of a substring (W) with a predefined length of L_{sub}. We use the full response string in a circular manner, so if the value $(ind_1 + L_{sub}) > L$, the remainder of the substring values are taken from the beginning of the full response bitstream. This operation has been illustrated in Fig. 4(a).

In step 7, the Prover circularly pads the substring W with random bits to create a bitstream PW of length L_{PW}. This bitstream is also referred to herein as "the padded substring". In this padding process, starting from a randomly chosen index (ind_2), the PUF substring from step 6 is inserted.

We pad the substring in a circular manner. Therefore, if the value $(ind_2 + L_{\text{sub}}) > L_{\text{PW}}$, the remainder of the PUF substring is inserted at the beginning of the padded stream. This operation is illustrated in Fig. 4(b).

In step 8, when an honest Verifier receives the padded substring (PW), he performs a circular maximum-sequence alignment against his simulated PUF output sequence (R') to determine which bits belong to PUF response string and which bits are generated randomly. The authentication is successful, only if the Hamming distance between the received and the simulated substrings is lower than a predefined threshold value. After this operation, the Verifier finds out the values of the two secret indices. However these values do not affect the authentication process.

In the proposed authentication, Prover does not reveal the whole response stream and the protocol leaks a minimal amount of information. The protocol is also lightweight and suitable for ultra-low power and embedded devices. Besides a Strong PUF, the Prover only needs to implement one TRNG and one PRNG. In addition to exchanging their respective session nonces, the Prover only needs to send a relatively short padded substring to the Verifier. Additionally, the protocol has the added benefit that the ranges of the respective secret indices (ind_1, ind_2) are flexible and can be tuned depending on the security requirements. The matching threshold can also be calculated to tolerate a predefined PUF error threshold.

B. SESSION KEY EXCHANGE PROTOCOL STEPS

It is possible to piggyback a session key exchange protocol on the authentication protocol of Fig. 3. The Prover can encode secret keys as the secret indices of authentication protocol (ind_1, ind_2). The Verifier can recover these secret indices at the end of a successful authentication. If the length of secret indices is not enough to encode the whole secret key, the authentication protocol may be repeated multiple times until the required number of secret bits is transmitted to the Verifier. We now describe this concept with an example.

If the length of PUF response string is 1024 bits, ind_1 is chosen from range of 0 to 1023. Therefore, we can encode 10 bits by using ind_1. If the length of the padded substring (L_{PW}) is 1024 bits, ind_2 is chosen from range of 0 to 1023. Therefore, 10 bits of secret key can be encoded by the ind_2. In this configuration, 20 bits overall can be exchanged between the parties with one run of the protocol. If the length of secret key is 120-bits, the protocol of Fig. 3 should be executed $\frac{120}{20} = 6$ times to transfer all of the secret key. This proposed protocol can securely exchange session keys with minimum overhead, while protecting against machine learning attacks and PUF response errors.

The key-exchange and authentication protocol can be followed up with a step to check whether the Verifier has received the correct indices. To do so, the Prover only needs to send the hashed values of the indices to the Verifier for verification.

C. 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 inputs to the XOR logic of arbiter PUF can be 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 before the XOR-mixing stage.

V. ANALYSIS OF ATTACKS

In this section, we quantify the resistance of the proposed protocols against different attacks by a malicious party (Prover or Verifier). Due to similarity of authentication and key exchange protocols, similar attacks analysis apply to both of them.

In the first subsection, we quantitatively analyze their resiliency to machine learning attacks. Second, we probabilistically investigate the odds of breaking the protocols 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 1 lists these parameters.

TABLE 1. **List of parameters.**

Parameter	Description
L_{n}	Length of nonce
L	Length of PUF response string
L_{sub}	Length of PUF response substring
L_{PW}	Length of padded substring
ind_1	Index to the beginning of substring, $0 \leq ind_1 < L$
ind_2	Index at which the PUF substring is inserted $0 \leq ind_2 < L_{\text{PW}}$
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. N_{min} depends on the PUF type and also the learning strategy. Based on theoretical considerations (dimension of the feature space, Vapnik-Chervonenkis dimension), it is suggested in [9] 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 $\epsilon = 10\%$. In the proposed protocol, the direct responses are not revealed and the attacker needs to correctly guess the secret indices to be able to discover L_{sub} challenge response pairs. ind_1 is a number between 0 and $L-1$ (L is the length of the original response string from which the substring is obtained), and ind_2 is a number between 0 to $L_{\mathrm{PW}} - 1$ (L_{PW} is the length of the padded substring).

Assuming the attacker tries to randomly guess the indices, he will be faced with $L \times L_{\mathrm{PW}}$ choices. For each *iter* choice, the attacker can build a PUF model (M_{iter}) by training it on the set of L_{sub} challenge response pairs using machine learning methods.

Now, the attacker could launch $L \times L_{\mathrm{PW}}$ 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 indices 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($L \times L_{\mathrm{PW}}$) number of attempts. However if $L_{sub} < N_{min}$, then the attacker needs to launch N_{min}/L_{sub} 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 $L \times L_{\mathrm{PW}}$ models based on the choice of indices value (ind_1 and ind_2), for N_{min}/L_{sub} rounds, the number of models will be of the following order:

$$(L \times L_{\mathrm{PW}})^{\frac{N_{min}}{L_{sub}}}. \tag{5}$$

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 that 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 Eq. 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.

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 Fig. 2 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 the experimental evaluation section, N_{min} is an order of magnitude larger for an XOR PUF than for a simple delay based PUF.

B. RANDOM GUESSING ATTACK

A legitimate Prover should be able to generate a padded 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}), the total length of the padded substring (L_{PW}), 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 authentication by a randomly guessing attacker, denoted P_{ADV}, would be:

$$P_{\mathrm{ADV}} = (L \cdot L_{\mathrm{PW}}) \times \sum_{i=L_{\mathrm{sub}}-th}^{L_{\mathrm{sub}}} \binom{L_{\mathrm{sub}}}{i} (\frac{1}{2})^i \cdot (\frac{1}{2})^{L_{\mathrm{sub}}-i}, \tag{6}$$

where L_{sub} and th are the length of the substring and the Hamming distance threshold, respectively. Eq. 6 is derived with this assumption that the adversary has $L \cdot L_{\mathrm{PW}}$ chances to match the simulated PUF response, and in each match, the probability of success is calculated using a binomial cumulative distribution function.

For an honest Prover, the probability of being correctly authenticated, denoted by P_{Honest} is:

$$P_{\mathrm{Honest}} = \sum_{i=L_{\mathrm{sub}}-th}^{L_{\mathrm{sub}}} \binom{L_{\mathrm{sub}}}{i} (1 - p_{\mathrm{err}})^i \cdot {p_{\mathrm{err}}}^{L_{\mathrm{sub}}-i}, \tag{7}$$

where p_{err} is the probability of an error in a response bit.

If L_{sub} is chosen to be a sufficiently large number, P_{ADV} will be close to zero and P_{Honest} will be close to one.

C. COMPROMISING THE RANDOM SEED

In the protocols, the Prover and the Verifier jointly generate the random PRNG seed by concatenating the outputs of their individual nonces (generated by TRNGs); i.e., $seed = \{Nonce_{\mathrm{v}} \parallel Nonce_{\mathrm{p}}\}$. 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.

An adversary that poses as a 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.

An imposter 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 padded 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 be 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 $1/2^{L_\text{n}}$ and 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 guess a matching location for the substring. In other words, statistical bias in the responses will leak information about the values of secret indices.

Recall that an ideal Strong PUF should have the strict avalanche property [21]. 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. This probability can be almost achieved when at least more than two independent PUF output bits are mixed by an XOR. As more independent PUF response bits are mixed, the probability of a bit flip in the output due a one bit change in the input moves closer to the ideal case; however, this linearly increases the probability of error in the mixed output. 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 [21].

In our implementation, Linear feedback shift registers (LFSRs) are used as a lightweight PRNG. An ideal LFSR must have the maximum length sequence property [30]. 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 [30]. 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, 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 for the PRNG to be a cryptographically secure generator. The seed in the protocol is public and the only purpose of the PRNG is to generate sequences of independent random challenge vectors from the Prover and Verifier nonces.

F. MAN-IN-THE-MIDDLE ATTACK ON KEY EXCHANGE

Asymmetric cryptographic algorithms, such as RSA and Diffie–Hellman, are traditionally used to exchange secret keys. These asymmetric algorithms are susceptible to man-in-the-middle attacks [31]. Therefore, a certificate authority is necessary for a secure implementation of these algorithms. However, our proposed key exchange algorithm is not susceptible to man-in-the-middle attack and no certificate authority is required for implementation.

An attacker, who intercepts the padded PUF substring, does not know the PUF response string. Therefore, he does not know the value of secret indices and he cannot change the padded PUF substring to forge a specific key. An attacker, however, can possibly rotate the padded substring to add or subtract from the secret value of ind_2. Even in this case, the attacker does not know the new value of ind_2 and cannot act upon it to open a forged encrypted channel. Rotating the padded substring will only result in a denial of service attack which is already possible by jamming.

VI. TRADE-OFFS IN PROTOCOL PARAMETERS

In this section, the trade-offs in choosing the parameters of the protocols are explored by analyzing the PUF measurement data collected in the lab. False acceptance and false rejection probabilities depend on PUF error rates. There have been no comprehensive reports till this date on PUF response error rates (caused by variations in temperature and power supply conditions) nor any solid data on modeling error rates measured on real PUF challenge response pairs. The data reported in the related literature mainly come from synthetic (emulated) PUF results rather than actual reliable PUF measurements and tests.

A. EXPERIMENTAL SET UP

In this paper, we used the data we measured and collected across 10 Xilinx Virtex 5 (LX110) FPGAs at 9 accurately controlled operating condition (combination of different temperatures and power supply points). Each FPGA holds 16 PUFs and each PUF is tested using 64,000 random challenges.

Ideal PUF responses are obtained by challenging the PUF 128 times at the nominal condition (temperature = 35°C and $V_{DD} = 1$ V) and then taking a consensus of these responses. The error rate is now defined as the percentage deviation from the consensus response. For example if 10 bits from the 128 bits are ones and the rest are zeros, the deviation from the majority response, or the response error rate, is $(10/128) \times 100 = 7.8\%$. Table 2 shows the average deviation (taken over 64,000 challenge-response pairs) of these experiments from the ideal response at the nominal condition. As it can be seen from this table, the error rate is substantially higher in non-nominal conditions. The worst case scenario happens when the temperature is 5°C and the voltage is 0.95V. The table shows that 30°C degree change in temperature will have a bigger effect on the error rate than a 5% voltage change.

TABLE 2. **Average bit error rate of PUF in different voltage and temperature conditions in comparison with the ideal PUF output at nominal condition.**

V_{DD} \ Temperature	5^oC	35^oC	65^oC
0.95 V	8.4%	6.2%	7.1%
1.00 V	6.8%	3.1%	6.4%
1.05 V	7.2%	6.7%	7.9%

As mentioned earlier, the Verifier repeatedly tests the PUF in the factory to obtain a consensus of the PUF responses for an array of random challenges. The Verifier then uses the reliable response bits to build a PUF Model for himself. When the PUF is deployed in the field, the Prover challenges its own PUF and send the responses to the Verifier. The average error rate of the Prover response in different working conditions against the Verifier model is listed in Table 3. The listed errors are the compound of two types of error. The first type is the error in PUF output due to noise of environment as well as operating condition fluctuations. The second type is the inevitable modeling error of the Verifier PUF model. These error rates are tangibly higher than the error rates of Table 2. The worst error rate is recorded at 5°C temperature and voltage of 0.95V. This error rate is taken as the worst-case error rate between an honest Verifier and an honest Prover. We will use this error rate to estimate the false acceptance and false rejection probability of the authentication protocol.

TABLE 3. **Average bit error rate of the verifier PUF model against the PUF outputs in different voltage and temperature conditions. *: the worst-case scenario.**

V_{DD} \ Temperature	5^oC	35^oC	65^oC
0.95 V	13.2% (*)	10.5%	10.7%
1.00 V	8.9%	6.4%	8.9%
1.05 V	9.3%	10.2%	11.8%

B. MODELING ATTACK COMPLEXITY AND PROTOCOL PARAMETERS

As explained earlier, the attack complexity depends exponentially on the minimum required number of challenge response pairs (CRPs), i.e., N_{min}, to reach a modeling error rate of less than 'th', the matching threshold in the protocol. The matching threshold in the protocol is incorporated to create a tolerance for errors in the responses caused by modeling error as well as errors due to environment variations and noise.

By relaxing the tolerance for errors in the protocol (i.e., increasing 'th'), we basically increase the probability of attack. In contrast, by lowering the tolerance for errors, the rate at which the authentication of a genuine PUF fails due to noisy responses increases. As a rule of thumb, the tolerance has to be set greater than the maximum response error rate to achieve sensible false rejection and false acceptance probabilities.

Once the tolerance level (th) is fixed to achieve the desired false rejection and false acceptance probabilities, N_{min} must be increased to hinder modeling attacks. However, N_{min} and th are inter-related for a given PUF structure. In other words, for a given fixed PUF structure, increasing th mandates that a less accurate model can pass the authentication, and that model can be trained with a smaller number of CRPs (smaller N_{min}). The only way to achieve a higher N_{min} for a fixed th is to change the PUF structure.

Earlier in the paper, we proposed using XOR PUFs instead of a single arbiter-based PUF in order to increase N_{min} for a fixed *th*. As reported previously in the related literature, XOR-ing the PUF outputs makes the machine learning more difficult and requires a larger CRP set for model building. The major problem with XORing the PUF outputs is error accumulation. For example, if the outputs of two arbiter-based PUFs are mixed with XORs, the XOR PUF response error rate will be about the sum of each individual arbiter-based PUF's errors. This means the error tolerance has to be doubled to have reliable operations. This observation of trade-off between N_{min} and *th*, led us to quantify this effect.

In order to quantify the trade-off between N_{min} and *th*, we first calculate the effective compound error rate of XOR-mixed PUF outputs for different operating conditions and different numbers of PUF stages. Tables 4–6 show the effective response error rate for 2-input, 3-input, 4-input XOR PUF respectively.

TABLE 4. 2-input XOR.

Temperature V_{DD}	5°C	35°C	65°C
0.95 V	24.7%	19.9%	20.3%
1.00 V	17.0%	12.4%	17.0%
1.05 V	17.7%	19.4%	22.2%

TABLE 5. 3-input XOR.

Temperature V_{DD}	5°C	35°C	65°C
0.95 V	34.6%	28.3%	28.8%
1.00 V	24.4%	18.0%	24.4%
1.05 V	25.4%	27.6%	31.4%

TABLE 6. 4-input XOR.

Temperature V_{DD}	5°C	35°C	65°C
0.95 V	43.2%	35.8%	36.4%
1.00 V	31.1%	23.2%	31.1%
1.05 V	32.3%	35.0%	39.6%

According to the above tables, the maximum error rates measured from the XOR PUF responses are 24.7%, 34.6%, and 43.2% for 2-input, 3-input, 4-input XOR-ed PUF, respectively. To guarantee reliable authentication at all operating conditions, the error tolerance of protocol (*th*) must be set above the maximum error rates. Now after deriving the PUF error rate, we would like to know how many challenge response pairs are required to train the PUF model and reach a modeling error rate that falls below the tolerance level. In other words, we need to know how many challenge/response pairs the adversary needs to collect in order to pass the authentication and break the system.

To answer this question, we trained and tested the PUF model on the data collected in the lab from real PUF implementations. We measured the modeling accuracy as a function of train/test set size for each PUF. The results in Fig. 5 show the modeling error using evolutionary strategy (ES) machine learning methods.

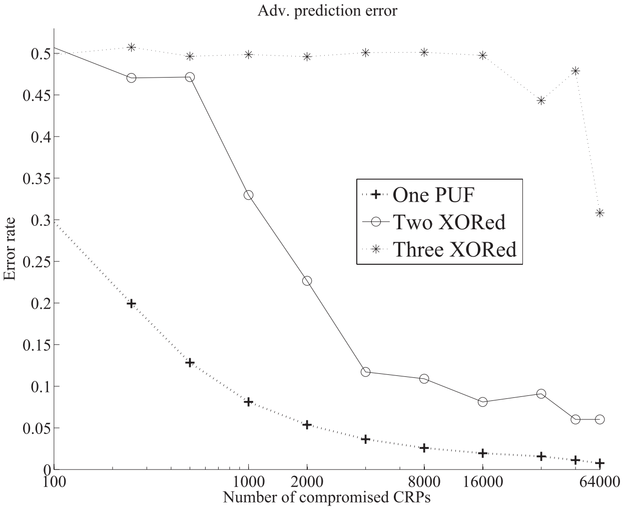

FIGURE 5. **The modeling error rate for arbiter-based PUF, and XOR PUFs with 2 and 3 outputs as a function of number of train/test CRPs.**

Based on the results in Fig. 5, the largest value of N_{min}, after taking into account the error threshold (*th*) derived earlier, is achieved by a 3 stages XORed-PUF. In other words, 64,000 CRPs must be collected to achieve a modeling error rate of less than 34.6%. Therefore, N_{min} = 64,000 for 3-stage XOR-ed PUF.

Table 7 shows the false rejection and false acceptance error rate of our protocol with the length of PUF response sequence and the length of additional pads fixed at 1028 and 512, respectively. False rejection rate is the rate in which the service to the truthful Prover is disrupted, it is calculated using Eq. 6: $1 - P_{\text{ADV}}$.

TABLE 7. False rejection and acceptance error probabilities for different protocol parameters.

L_{sub}	1250		
Error threshold	487	477	467
False rejection	0.2%	1%	5%
False acceptance	9e-10	0	0

The requirements on the false rejection rate are not usually as stringent as the requirements on the false acceptance rate, however, one should assume that a customer would deem a product impractical if the false rejection rate is higher than a threshold. In our protocol design, we tune the system parameter to achieve a false negative rate of 1%, while minimizing the false acceptance rate. Also, we take the worst-case error rate as the basis of our calculation of false acceptance and false rejection rates. The error rates that we report are the upper bound of what can be observed in the field by a customer/Prover.

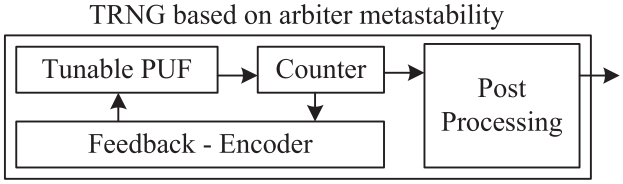

FIGURE 6. **True random number generation architecture based on flip-flop meta-stability.**

Table 7 shows that the desired false rejection rate of 1% with an acceptable false acceptance rate is achieved when $L_{sub} = 1250$ and the error threshold is 477/1250 = 38%. In this scenario, an adversary needs to perform $O((1300 \cdot 512)^{(64000/1250)}) \approx O(2^{988})$ machine learning attacks in order to break this system which makes the system secure against all computationally bounded adversaries.

At the end, it should be noted that the worst case bit error rate of our PUF implementation (13.2% in Table 3) is much higher than a recently reported bit error rate of arbiter PUFs [32] ($\approx 3-5\%$). The discrepancy might be explained by the fact that their implementation is based on a 65nm ASIC technology and ours is based on a Virtex 5 FPGA. Therefore, the reported security performance of our protocol has the potential to be further enhanced by a more custom implementation with a lower bit error rate.

VII. HARDWARE IMPLEMENTATION

In this section, we present an FPGA implementation of the proposed protocol for the Prover side on Xilinx Virtex 5 XC5VLX110T FPGAs. Fig. 7 summarizes the required resources on Prover and Verifier sides of the protocols. 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.

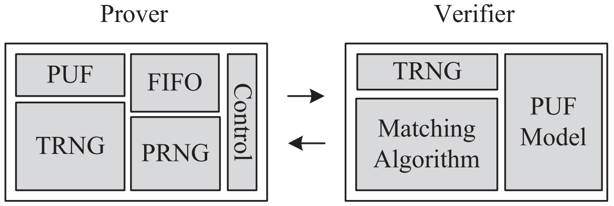

FIGURE 7. **Resource usage on prover and verifier sides.**

It is desirable to use a low overhead PUF implementation, such as the one introduced in [33]. If an ASIC or analog implementation of the PUF is required, the ultra-low power architecture in [29] is suitable for this protocol. A very low-power Verifier implemented by a microcontroller such as TI MSP430 can easily challenge the PUF and run the subsequent steps of the protocol.

We use the implementation of the arbiter-based PUF in [34]. 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 [35], [36]. We use the architecture presented in [33] to implement a true random number generator. One embodiment of the TRNG architecture is shown in Fig. 6. This TRNG operates by enforcing a meta-stable state on flip-flops through a closed loop feedback system. This TRNG has a Tunable PUF as its core that consumes 128 LUTs that are packed into 16 CLBs on Virtex 5. In fact, the PUF of the TRNG 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 meta-stability. If the arbiter operates in a purely meta-stable 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 meta-stable state. The counter output drives an encoding table of depth 2^{12} where Each row of encoding table 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.

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 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 indices, we can run TRNG before the start of the protocol and pre-record these values. Therefore, its throughput does not affect the overall system performance.

TABLE 8. **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	1250	0	0	N/A
2	LFSR	2	128	0	0	N/A
1	Control	12	9	0	0	N/A
Total		652	1400	4KB	64KB	N/A

The implementation overhead of our proposed authentication protocol is much less than traditional cryptographic modules. For example, robust hashing implementation of SHA-2 as implemented in [37] requires at least 1558 LUTs of a Virtex-II FPGA and it takes 490 clock cycles to evaluate. This overhead will occur on the top of the clock cycles required for PUF evaluation.

The overhead of our key exchange protocol should be compared against symmetric key-exchange algorithms not asymmetric key-exchange ones, since our protocol assumes that a secret PUF as a token has been pre-distributed between the Provers. Our key exchange protocol achieves desired level of security with minimal computational overhead. For example, AES-128 as implemented in [38] requires at least 738 LUTs of a Virtex-V FPGA, which is higher than the combined overhead of our authentication and key-exchange as listed in Table 8.

VIII. CONCLUSION

We have presented secure and low-overhead authentication and key exchange protocols based on PUFs. In the authentication protocol, the Prover reveals only a random subset of responses for authentication. The Verifier, which has access to a compact 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. Key-exchange protocol based on pattern matching was also proposed in this work. We demonstrated that carefully-designed protocols based on pattern-matching concept provides a much higher level of resiliency 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.

ACKNOWLEDGMENT

The authors would also like to thank the anonymous reviewers of this paper for their helpful comments and suggestions.

REFERENCES

[1] V. Boyko, P. MacKenzie, and S. Patel, "Provably secure password-authenticated key exchange using Diffie–Hellman," in *Advances in Cryptology*. New York, NY, USA: Springer-Verlag, 2000, pp. 156–171.

[2] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *Proc. 19th Int. Conf. Theory Appl. Cryptograph. Tech.*, 2000, pp. 139–155.

[3] P. S. Ravikanth, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, nos. 5589, pp. 2026–2030, 2002.

[4] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *Proc. Comput. Commun. Security Conf.*, 2002, pp. 148–160.

[5] U. Ruhrmair, S. Devadas, and F. Koushanfar, *Security Based on Physical Unclonability and Disorder*. New York, NY, USA: Springer-Verlag, 2011.

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

[7] 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. New York, NY, USA: Springer-Verlag, 2010.

[8] M. Rostami, J. B. Wendt, M. Potkonjak, and F. Koushanfar. (2014, Mar.). "Quo vadis, PUF." in *Design, Automation & Test in Europe*, to be published.

[9] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Proc. 17th ACM Conf. Comput. Commun. Security*, Oct. 2010, pp. 237–249.

[10] Z. Paral and S. Devadas, "Reliable and efficient PUF-based key generation using pattern matching," in *Proc. Int. Symp. Hardware-Oriented Security Trust*, 2011, pp. 128–133.

[11] M. Majzoobi, M. Rostami, F. Koushanfar, D. S. Wallach, and S. Devadas, "Slender PUF protocol: A lightweight, robust, and secure authentication by substring matching," in *Proc. IEEE Symp. Security Privacy Workshops*, May 2012, pp. 33–44.

[12] F. Koushanfar, *Hardware Metering: A Survey*. New York, NY, USA: Springer-Verlag, 2011.

[13] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Delay-based circuit authentication and applications," in *Proc. ACM Symp. Appl. Comput.*, 2003, pp. 294–301.

[14] D. Lim, "Extracting secret keys from integrated circuits," M.S. thesis, Dept. Electr. Eng. Comput., Massachusetts Inst. Technol., Cambridge, MA, USA, 2004.

[15] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Testing techniques for hardware security," in *Proc. Int. Test Conf.*, 2008, pp. 1–10.

[16] G. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proc. 44th ACM/IEEE Des. Autom. Conf.*, Jun. 2007, pp. 9–14.

[17] B. Gassend, "Physical random functions," M.S. thesis, Dept. Electr. Eng. Comput., Massachusetts Inst. Technol., Cambridge, MA, USA, Jan. 2003.

[18] E. Oztürk, G. Hammouri, and B. Sunar, "Towards robust low cost authentication for pervasive devices," in *Proc. 6th Annu. IEEE Int. Pervas. Comput. Commun.*, Mar. 2008, pp. 170–178.

[19] U. Ruhrmair, J. Solter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, *et al.*, "PUF modeling attacks on simulated and silicon data," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 11, pp. 1876–1891, Nov. 2013.

[20] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Lightweight secure PUF," in *Proc. Int. Conf. Comput. Aided Des.*, 2008, pp. 670–673.

[21] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Techniques for design and implementation of secure reconfigurable PUFs," *ACM TRETS*, vol. 2, no. 1, pp. 1–33, 2009.

[22] A. Mahmoud, U. Ruhrmair, M. Majzoobi, and F. Koushanfar. (2013). *Combined Modeling and Side Channel Attacks on Strong PUFs* [Online]. Available: https://eprint.iacr.org/2013/632

[23] J. Delvaux and I. Verbauwhede. (2013). *Fault Injection Modeling Attacks on 65 nm Arbiter and RO sum PUFs via Environmental Changes* [Online]. Available: https://eprint.iacr.org/2013/619

[24] C. Bösch, J. Guajardo, A. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient helper data key extractor on FPGAs," in *Proc. 10th Int. Workshop Cryptograph. Hardw. Embedded Syst.*, 2008, pp. 181–197.

[25] R. Maes, P. Tuyls, and I. Verbauwhede, "Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs," in *Proc. 11th Int Workshop Cryptograph. Hardw. Embedded Syst.*, 2009, pp. 332–347.

[26] M.-D. M. Yu and S. Devadas, "Secure and robust error correction for physical unclonable functions," *IEEE Des. Test Comput.*, vol. 27, no. 1, pp. 48–65, Jan./Feb. 2010.

[27] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *Proc. Adv. Int. Theory Appl. Conf. Cryptol. Eurocrypt*, Interlaken, Switzerland, 2004, pp. 523–540.

[28] N. Beckmann and M. Potkonjak, "Hardware-based public-key cryptography with public physically unclonable functions," in *Information Hiding*. New York, NY, USA: Springer-Verlag, 2009, pp. 206–220.

[29] M. Majzoobi, G. Ghiaasi, F. Koushanfar, and S. Nassif, "Ultra-low power current-based PUF," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2011, pp. 2071–2074.

[30] 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," *IEEE Trans. Signal Process.*, vol. 58, no. 12, pp. 6284–6297, Dec. 2010.

[31] C. Paar, J. Pelzl, and B. Preneel, *Understanding Cryptography: A Textbook for Students and Practitioners*. New York, NY, USA: Springer-Verlag, 2010.

[32] S. Katzenbeisser, Ü. Kocabaş, V. Rožić, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann, "PUFs: Myth, fact or busted? A security evaluation of physically unclonable functions (PUFs) cast in silicon," in *Proc. 14th Int. Conf. Cryptograph. Hardw. Embedded Syst.*, 2012, pp. 283–301.

[33] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA-based true random number generation using circuit metastability with adaptive feedback control," in *Proc. Int. Cryptograph. Hardware Embedded Syst.*, 2011, pp. 17–32.

[34] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA PUF using programmable delay lines," in *Proc. IEEE Int. Workshop Inf. Forensics Security*, Dec. 2010, pp. 1–6.

[35] C. K. Koc, *Cryptographic Engineering*, 1st ed. New York, NY, USA: Springer-Verlag, 2008.

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

[37] M. Kim, J. Ryou, and S. Jun, ''Efficient hardware architecture of SHA-256 algorithm for trusted mobile computing,'' *Information Security and Cryptology*, Berlin, Germany: Springer-Verlag, 2009, pp. 240–252.

[38] S. Drimer, T. Guneysu, and C. Paar, ''DSPs, BRAMs, and a pinch of logic: Extended recipes for AES on FPGAs,'' *ACM Trans. Reconfigurable Technol. Syst.*, vol. 3, no. 1, pp. 1–3, 2010.

MASOUD ROSTAMI received the M.S. degree in electrical engineering from Rice University in 2010, where he is currently pursuing the Ph.D. degree in computer engineering. His research interests include double gate devices, hardware security, and security of implanted medical devices.

MEHRDAD MAJZOOBI received the M.Sc. and Ph.D. degrees in electrical and computer engineering from Rice University, Houston, TX, USA, in 2009 and 20013, respectively. He is currently a Chief Executive Officer with Mesh Motion, Inc., on the social implications of access management.

FARINAZ KOUSHANFAR received the Ph.D. degree in electrical engineering and computer science and the M.A. degree in statistics from the University of California Berkeley in 2005. She is currently an Associate Professor with the Department of Electrical and Computer Engineering, Rice University, Houston, TX, USA. Her research interests include adaptive and low power embedded systems design, hardware security, and design intellectual property protection. She is a recipient of several awards and honors, including the Presidential Early Career Award for Scientists and Engineers, the ACM SIGDA Outstanding New Faculty Award, the NAS Kavli Foundation Fellowship, and the Young Faculty (or CAREER) Awards from ARO, ONR, DARPA, and NSF.

DAN S. WALLACH is a Professor with the Department of Computer Science and a Rice Scholar with the Baker Institute for Public Policy, Rice University. He is a member of the Board of Directors of the USENIX Association.

SRINIVAS DEVADAS (F'99) is the Edwin Sibley Webster Professor of electrical engineering and computer science with the Massachusetts Institute of Technology, where has been on the faculty since 1988. He served as an Associate Head with the Department of Electrical Engineering and Computer Science from 2005 to 2011. His research interests include computer security, computer architecture, and computer-aided design. He has written numerous papers and books and received several best paper awards.