A Challenge Obfuscation Method for Thwarting Model Building Attacks on PUFs

Yansong Gao[1,2], Damith C. Ranasinghe[2], Gefei Li[2], Said F. Al-Sarawi[1], Omid Kavehei[3], and Derek Abbott[1]

[1] School of Electrical and Electronic Engineering,
The University of Adelaide, SA 5005, Australia,
{yansong.gao, said.alsarawi, derek.abbott}@adelaide.edu.au,
[2] Auto-ID Labs, School of Computer Science,
The University of Adelaide, SA 5005, Australia,
damith.ranasinghe@adelaide.edu.au, gefei.li@student.adelaide.edu.au,
[3] Functional Materials and Microsystems Research Group,
School of Electrical and Computer Engineering,
Royal Melbourne Institute of Technology, Victoria 3001, Australia,
omid.kavehei@rmit.edu.au

Abstract. Physical Unclonable Functions (PUFs), as novel lightweight hardware security primitives, provide a higher level security with lower power and area overhead in comparison with traditional cryptographic solutions. However, it has been demonstrated that PUFs are vulnerable to model building attacks, especially those using linear additive functions such as Arbiter PUF (APUF) and k-sum PUF as building units. Nevertheless, both APUFs and k-sum PUFs are highly desirable security primitives, especially for authentication, because they are capable of producing a huge number of challenge response pairs (CRPs) and can be easily integrated into silicon. In this paper, we actually rely on the demonstrated vulnerability of PUFs to model building attacks as well as the relative ease with which this can be achieved to develop a new parameter-based authentication protocol based on obfuscating challenges sent to PUFs and their subsequent recovery. We show, using statistical analysis and model building attacks using published approaches, that constructing a model using machine learning techniques are infeasible when our proposed method is employed. Finally, we also demonstrate that our challenge obfuscation and recovery method can be successfully used for secure key exchange between two parties.

Keywords: Physical Uncloanble Function, obfuscation, model building attacks, hardware security, authentication.

1 Introduction

Traditional digital keys are stored in a non-volatile memory (NVM) for cryptographic applications. However, it has been shown that the digital keys in NVM are vulnerable to invasive physical attacks. Complicated tamper sensing and temper-proofing mechanisms has to be implemented in hardware to secure digital keys in NVM, consequently increasing the area and power overhead of the device as well as limiting the use of these anti-tampering methods for resource-constrained devices.

The growing new area of PUFs is receiving increased attention because PUFs, especially circuit or silicon based PUFs, offer a simple alternative to storing digital keys in NVM with a small hardware footprint and without the need for tamper-sensing mechanisms for extracting secret key information from a complex physical system. Notably that PUFs are easy to build but practically impossible to duplicate because they rely on uncontrollable physical parameter variations that occur during hardware device manufacturing. More importantly, secrecy of a PUF is derived from the inherent complicated physical system instead of storing information in NVM memories and thus enabling a lightweight hardware security primitive.

When a PUF is stimulated by a challenge (input), C, a corresponding response (output), R, will be generated and determined by $f(R)$, where $f()$ is a physical function that is unique to each device. Given the same challenge, C, different PUF instantiations built upon the same design will respond with a different response, R. The challenge, C, and its corresponding response, R, are commonly referred to as a Challenge Response Pair (CRP). A set of CRPs can be treated as a fingerprint of a PUF and therefore a PUF integrated device or object. Therefore, it is favourable that a PUF has an exponential number of CRPs as in the Arbiter PUF (APUF) [8, 20] and k-sum PUF [25, 3] to enable distinguishing a unique device among a large device population.

PUFs can be used for authentication and cryptographic key generation as well as realising more complex cryptographic protocols such as oblivious transfer (OT), bit commitment (BC), key exchange (KE) [20, 11, 23, 26, 18]. Furthermore, because of a PUFs ability to bind to a physical entity, PUFs are increasingly used to offer protection against identity theft, cloning of devices, and counterfeiting of goods.

However, if an adversary can eavesdrop on CRPs or have access to a physical device for a short period to measure CRPs, it has been shown that an adversary armed with only thousands of CRPs from, for example a highly desirable PUF architecture with exponential CRPs such as an APUF or a k-sum PUF built on linear additive blocks, and with access to a typical modern laptop computer can build a model of a PUF in less than a few seconds. The model building attack vulnerability threatens conventional PUF based authentication protocols, as shown in [20], and other cryptographic applications based on PUFs [17].

To enhance the security level of a PUF, or in other words to increase the complexity of the task faced by an adversary to perform a model building attack, there are several solutions. One solution is to add nonlinearity, such as XOR-ing the responses of several PUF instantiations to generate a single bit response. Another solution is to alter the PUF architecture, for example, as in Feed Forward Arbiter PUFs [8, 13] and lightweight secure PUFs [12]. Unfortunately, increasing the complexity of model building attacks through the integration of more non-linear elements also significantly reduces the reliability of the PUF [8, 17]. In addition, it has been demonstrated that such a solution only provides a modest advantage as the PUFs can still be broken using machine learning based model building attacks [17, 18].

Unlike the previous approaches, two recent proposals [16, 24] have demonstrated a highly innovative alternative that effectively confuses the adversary by hiding the direct relationship between C and R. In these proposals, an adversary cannot obtain the exact pair of C and R, however, the server (or verifier) is still able to discover the exact pair of C and R to successfully authenticate a device (or prover) because the server can take advantage of the PUF's vulnerability to model building attacks to verify the response from the device [14, 16, 24]. In general, both of these approaches [16, 24] post-process, i.e. decimate, the response generated on device and subsequently only expose the post-processed response to an adversary.

In contrast, our approach focuses on challenge obfuscation instead of post-processing PUF responses as done in [16, 24]. We demonstrate that exposing an obfuscated PUF challenge through random elimination of challenge bits to an adversary and subsequent pre-processing of the challenge on the device to generate a full challenge before it is used to stimulate the physical PUF instance can *also* provide resiliency to model building attacks. Further, the challenge recovery method is both different and more simpler compared to recovering the response where the length of the decimated response has to expanded to ensure successful authentication. Additionally, we also demonstrate that given a set of *obfuscated* challenge and response pairs, an adversary cannot mount a model building attack and also demonstrate how our proposed approach can be used to secure key exchange. We summarize the contributions from our study below:

- We propose a challenge obfuscation and a recovery method to thwart model building attacks on PUFs by first taking advantage of the ability to rapidly build a parametrised model of a physical PUF.
- We analyse the challenge obfuscation and recovery method to demonstrate that a server (verifier) can successfully recover a challenge. That is a server can identify the exact full length challenge used in the device, where this exact full length challenge is obfuscated from both the attacker and the server, because the server is able to exploit a securely stored parametrised model of the physical PUF on the device.
- We illustrate a modified parameter-based authentication protocol and then further show that our modified authentication protocol can be used for securing key exchange.
- We perform statistical analysis to illustrate the exponentially increasing workload that an attacker will face to build a model of a PUF after implementing the proposed challenge obfuscation and recovery method. Further, we also perform model building attacks methodologies successfully employed against PUFs to demonstrate the enhanced security provided by our approach.

The remainder of the paper is organized as follows: Section 2 introduces related work; Section 3 presents the proposed challenge obfuscation method used on the device and challenge recovery method used on the server. In addition, the modified parameter-based authentication protocol and the affiliated key exchange protocol is given in this section; Section 4 provides statistical analysis and model building attack employed to evaluate the enhanced security of a PUF after implementing the challenge obfuscation and recovery method; and then Section 5 concludes this paper followed by an acknowledgment.

2 Related Work

Over the years, a number of PUF structures have been proposed, built and analyzed. These include *time delay based* PUFs such as the Arbiter PUF [1, 2, 7] (APUF), Feed-Forward APUF [8], Ring-Oscillator PUF [20] (RO-PUF), and Glitch PUF [21]; *Memory-based* PUFs leveraging device mismatch such as SRAM PUF [4, 5], Latch PUF [19], Flip-flop PUF [10, 22], Butterfly PUF [6]. A comprehensive review of different PUF architectures can be found in [15, 3].

The examples of PUFs that builds upon linear additive blocks include APUF [1, 9, 20] and k-sum PUF [25, 3]. They have one key desirable feature that generates exponential number of CPRs. However, the shortcomings is that they have been shown to be vulnerable to model building attacks using machine learning techniques. From the model building attacks perspective, both architectures have the same topology, therefore in this paper, the APUF is considered to demonstrate the proposed method, nonetheless this does not limit its applications to other PUF structures.

Since our work is based on exploiting the vulnerability of APUFs to model building attacks, we will first summarise work in the area of building parametrised models of APUFs. Then we will briefly introduce the concept of parameter-based authentication we have employed in our work and highlight two recent proposals that use parameter-based authentication and relies on the vulnerability of APUFs to model building attacks to construct authentication approaches resilient to model building attacks using a response post-processing (decimation and sub-string padding of the response) and recovery technique.

2.1 Modelling Arbiter-PUFs

The APUF consists of k stages in sequence, each stage is composed of two 2-input multiplexers shown in Fig. 1, or any other architectures that have two signal paths. To generate a response bit, a signal is applied to the first stage input, while the challenge C is used to determine the signal path to the next stage. The two electrical signals simultaneously race through each multiplexer path (top and bottom paths) in parallel. At the end of the APUF architecture, an arbiter, which can be implemented by a latch, can be used to determine whether the top or bottom signal arrives first and hence outputs a logic ‘0’ or ‘1’ accordingly.

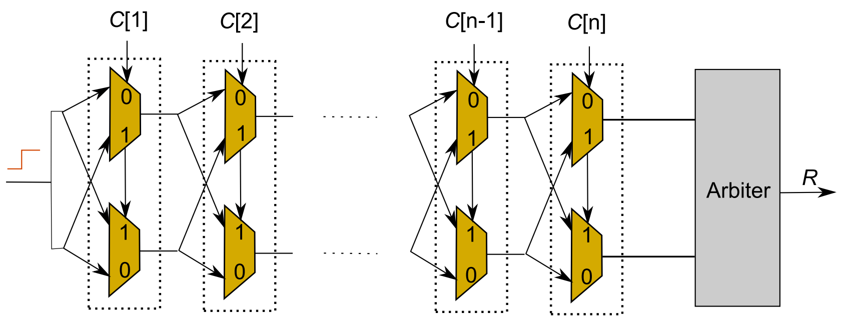


Fig. 1. An arbiter PUF (APUF) circuit.

It has been shown that an APUF can be modelled via a linear additive model since a response bit is generated based on the summation of each time delay segment in each stage (two 2-input multiplexers) depending on the challenge C, where C is made up of $(c_1, c_2, ...c_k)$ [8,17,18]. The final delay difference $\triangle$ between the two signals can be expressed as:

$$\triangle = \boldsymbol{\omega}^T \boldsymbol{\Phi}, \tag{1}$$

where $\boldsymbol{\omega}$ and $\boldsymbol{\Phi}$ are the delay determined vector and the parity vector, respectively, of dimension $k+1$ as a function of C. We denote $\sigma_i^{1/0}$ as the delay in stage i for the crossed $(c_i = 1)$ and uncrossed $(c_i = 0)$ signal path through the multiplexers, respectively. Hence σ_i^1 is the delay of stage i when $c_i = 1$, while σ_i^0 is the delay of stage i when $c_i = 0$. Then

$$\boldsymbol{\omega} = (\omega^1, \omega^2, ...\omega^k, \omega^{k+1})^T, \tag{2}$$

where $\omega^1 = \frac{\sigma_1^0 - \sigma_1^1}{2}$, $\omega^i = \frac{\sigma_{i-1}^0 + \sigma_{i-1}^1 + \sigma_i^0 - \sigma_i^1}{2}$ for all $i = 2, ..., k$ and $\omega^{k+1} = \frac{\sigma_k^0 + \sigma_k^1}{2}$. Furthermore,

$$\boldsymbol{\Phi}(\boldsymbol{C}) = (\Phi^1(\boldsymbol{C}), ..., \Phi^k(\boldsymbol{C}), 1)^T, \tag{3}$$

where $\Phi^j(\boldsymbol{C}) = \Pi_{\mathrm{i=j}}^{\mathrm{k}}(1 - 2\mathrm{c_i})$ for $j = 1, ..., k$.

Here we can see that the $\boldsymbol{\omega}$ encodes the delays for the subcomponents in the APUF stages, while the $\boldsymbol{\Phi}$ is a parity vector as a function of $c_1, ..., c_k$. The delay difference, $\triangle$, is the inner product of $\boldsymbol{\omega}$ and $\boldsymbol{\Phi}$. If $\triangle$ is greater than 0, the response bit is '1', otherwise, the response bit is '0'. Then the task for an adversary is to find an estimate of $\boldsymbol{\omega}$ that mimics the actual delay vector $\boldsymbol{\omega}$ of a physical PUF structure, Notably that this estimate will be based on both the full knowledge of $\boldsymbol{\Phi}$ and the corresponding response.

2.2 Parameter-Based Authentication

Suppose we have an APUF which uses k challenge bits and produces n response bits. Let us assume $k = 64$ and $n = 128$ to simplify the description given in this paper. In practice, such a PUF structure can be implemented by the APUF instantiation shown in Fig. 1 by duplicating 128 APUFs in a circuit, where all of the 128 APUF instantiations share the same 64 bit challenge. In the following discussion, when we refer to n-bits APUF, we imply a PUF structure with n APUF instantiations such that an n-bits APUF produces a n bits response given a single challenge with k bits.

It has been demonstrated that the n-bits APUF could be easily modelled by using machine learning techniques. For instance, our own experimental results has shown that machine learning techniques can be successfully used to produce learnt models with prediction accuracies higher than 98% within 1 second for a specific 1-bit APUF—already higher than an APUF's reliability—after training with just 2000 CRPs. These results are consistent with published results in [18] and [17].

In parameter-based authentication, during the provisioning phase, the server stores $k+1$ parameters of a model for a 1-bit APUF instead of storing a large number of CRPs, where the model is built on training a small number of CRPs. If we are going to use a 128-bits ($n = 128$) APUF to generate a 128-bit response, we save $128 \times (k+1)$ parameters. The securely stored set of parameters then constitutes a "snapshot", p_{ss}, of a 128-bits APUF. If a challenge control block is implemented along with a PUF in hardware, the direct extraction of CRPs can be disabled (e.g., via a fuse) to avoid an adversary from extracting p_{ss} at a later time by gaining physical access to the device.

In the authentication phase, whenever a server (or verifier) needs to authenticate a device (or prover) onto which a PUF is already integrated, the server randomly generates a challenge and sends it to the device. Then the device applies the received challenge and gains a corresponding n-bit response. Subsequently, the device sends the generated response back to the server. Finally, the server emulates the response based on the stored parameterized model for the given challenge and compare the emulated response with the response received from the device. If they are matched, then the authenticity of the device is established, otherwise, the authenticity is rejected.

2.3 Mechanisms Resilient to Model Building Attacks

The concept of exploiting the machine learning based model building vulnerability of additive delay model based PUFs, such as APUFs, to in fact increase the resilience of a PUF implementation to model building attacks was very recently demonstrated in [16, 24].

In the first approach [16] a subset string of a PUF response is randomly selected and is padded with a randomly generated string to ensure the padded string is of identical length with a response expected from a PUF. The device then sends the post-processed response to the server which subsequently determines the randomly selected subset string to decide the authenticity of the device. While the authors demonstrate using statistical analysis the infeasible task faced by an attacker to build a model of a PUF, a model building attack is not conducted to evaluate the security of the proposed approach. In the second approach [24] a randomized decimation technique is used to randomly eliminate bits from a response generated by a device. Subsequently, a response recovery method is used by the server on receipt of a response to recover the original response generated by the device for authentication.

Notably, both approaches requires increasing lengths of response bits, especially [24] which demands exponentially increasing length of response bits, to ensure the same authentication failure or success rates expected from a PUF before implementing the response post-processing strategies. In addition, the server has a more complex task of completely recovering a post-processed response. More interestingly, since a challenge response pair is never utilized more than once (nonce), these approaches are also reported to be resilient to side-channel attacks that filter the noise prior to machine learning through repeated measurements of CRPs.

In contrast, we consider an alternative perspective that is simpler by developing a challenge obfuscation and recovery method suitable for parameter-based authentication and key exchange and how that indeed it is infeasible to for a passive adversary to build a model of a PUF after the challenge obfuscation and recovery method are implemented.

3 Challenge Obfuscation and Recovery Method

Our goal is to obfuscate an adversary by implementing a challenge obfuscation method that prevents the adversary from obtaining a meaningful relationship between the observed challenge-response pairs, while the server can still successfully perform authentication by using a challenge recovery method. So in this section, firstly, we introduce our challenge obfuscation and the appropriately challenge control, subsequently, we provide the challenge recovery method to discover the full obfuscated challenge used in the device. Then we summarize the modified parameter-based authentication protocol and we further propose a secure key exchange protocol as an affiliate of the modified parameter-based authentication protocol. Table 1 describes the parameters used in our discussions.

Table 1. Description of parameters

Parameter	Description
k	Length of PUF challenge (stages of APUF)
n	Length of PUF response
m	Eliminated length of challenge from the server
τ	Response Hamming distance (HD) threshold
ϵ	Error prediction rate of a machine learning model
p	Number of *Patterns*
C'	Partial challenge of $k - m$ bits

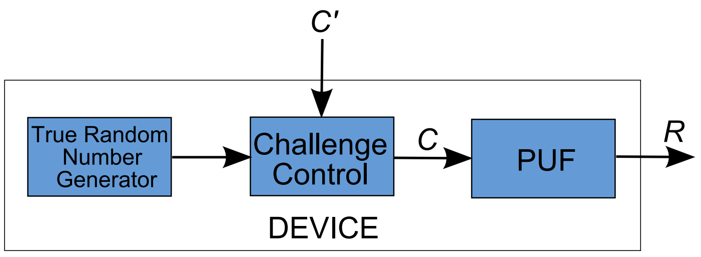
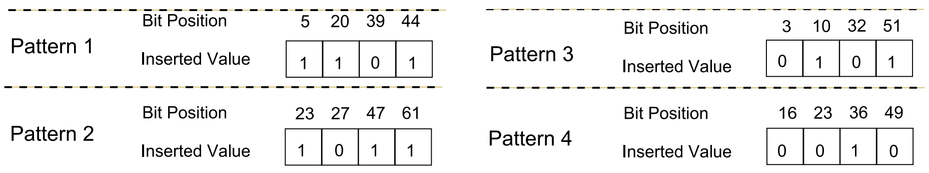


Fig. 2. A block diagram illustrating the challenge obfuscation method. Note that the true random number generator on the device can be implemented by using the least significant bit (LSB) of frequency from a ring oscillator (RO) or from utilizing the unstable output bits of an n-bit APUF.



Pattern 1	Bit Position	5	20	39	44
	Inserted Value	1	1	0	1
Pattern 2	Bit Position	23	27	47	61
	Inserted Value	1	0	1	1
Pattern 3	Bit Position	3	10	32	51
	Inserted Value	0	1	0	1
Pattern 4	Bit Position	16	23	36	49
	Inserted Value	0	0	1	0

Fig. 3. Challenge obfuscation example. Here we list 4 ($p = 4$) *Patterns* of positions and values that can be inserted into a partial challenge C' to produce a full length challenge C.

3.1 Challenge Obfuscation

Figure 2 shows a block diagram of our challenge obfuscation approach. Here a partial challenge, C', of $k - m$ bits are randomly selected by a server. The true random number generator determines which challenge obfuscation *Pattern* out of p possibilities, see the example shown in Fig. 3, will be chosen to determine the position of the m bits and the values of the m bits. Note that in our example we only list 4 ($p = 4$) possible full length challenges, C, to one C' to simplify our description. Subsequently, the challenge control block determines a full length challenge C based on C' and the selected challenge obfuscation *Pattern*. Consequently, the response, R, is generated according to the full length challenge C as shown in Fig. 2.

The notable aspect of our concept is the fact that full length challenge C is hidden from not only the adversary but also the server because the generation of the full length challenge C is invisible to both parties. However, as we will show in Section 3.2, only the server has the knowledge to recover the correct C given C' and R to authenticate the device.

Challenge Control How does a server (verifier) ensure a successful authentication of a legitimate device? In other words, how does the server discover the exact full length challenge C or *Pattern* used by the device? The answer lies in taking advantage of the large Hamming distance among different responses cor-

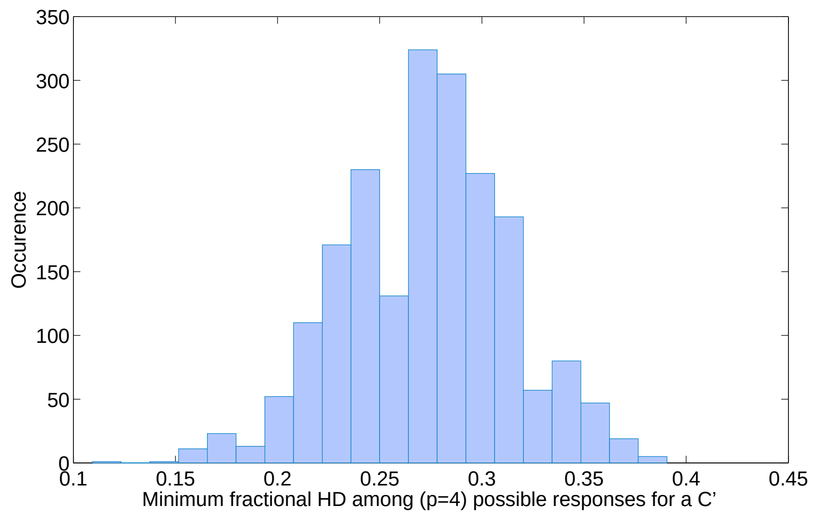


Fig. 4. Minimum fractional hamming distance (HD) evaluated using the example in Fig. 3 for 2000 randomly selected partial challenges illustrating that the server can authenticate a PUF without any knowledge of the exact full length challenge C used by the device. Here, the minimum fractional HD is 10.95%.

responding to different challenges applied to the same PUF, where HD is defined as the number of positions at which two strings, x and y of the same length over a finite alphabet $\sum$ differ, i.e., $\triangle(x, y) = |i\{x_i \neq y_i\}|$.

Ideally, if one bit flips between two different challenges, the fractional Hamming distance (where the fractional HD between $x, y \in \sum^n$ is given by $\frac{\triangle}{n}$) between two responses corresponding to the two different challenges is expected to be 50%. However, in reality, this is not the case for a PUF. Mostly, the Hamming distance among responses corresponding to nearly similar challenges is very small. Thus if the p responses for the possible p full length challenges corresponding to a specific partial challenge C' are nearly similar to each other, the small HD between nearly similar responses can essentially prevent the server from identifying the exact full length challenge C used by the device to generate the response R.

The challenge control block can be implemented to appropriately select *Patterns* —determining the positions and values of these eliminated m bits in challenge—to ensure that challenge recovery results in a successful authentication. This can be realized by ensuring that the m bits for different *Patterns* have a relatively large HD. Subsequently the HD among the possible full length challenges corresponding to a given C' will also be large and consequently the possible responses will also have large Hamming distances. Thus, the server will be able to identify the full length challenge C used by the device by selecting *Patterns* that have larger Hamming distances.

Furthermore, appropriate selection of a partial challenge, C', by the server can further ensure the correct recovery of the challenge C generated by the device. This can be realized by pre-calculation of the HD among all possible responses to a randomly selected partial challenge C'. If the minimum HD is small, then this partial challenge C' will be discarded to ensure that the minimum HD among all the possible responses are large enough. Notably, these per-calculations will not leak information to the adversary or result in any additional overhead to the device because the partial challenge C' is randomly generated by the sever, therefore the adversary has no knowledge of the discarded partial challenges without having access to a parameterized model of a PUF.

Consider the challenge control block example implementation outlined using the four different *Patterns* listed in the Fig. 3. Fig. 4 shows the distribution of *minimum* fractional HD among the responses corresponding 2000 randomly selected partial challenges using *Patterns* listed in the Fig. 3. It can be seen that 100% of the possible responses that use this control block example have a *minimum* fractional HD$>$10.95%.

3.2 Challenge Recovery

The server now has to emulate all possible responses corresponding to all different *Patterns*, for example the *Patterns* shown in Fig. 2 using the parameterised model of the PUF securly stored on the server. Only the response corresponding to the hidden full length challenge C used by the device will match the response sent from the device because the hamming distance between the emulated responses by the server and the response sent from the device is large enough (see Section 3.1) to ensure that the server can reject the authenticity of the device otherwise.

However given that PUF responses exhibit a degree of unreliability characterised as the bit error rate (BER) we define a response HD threshold τ which is the Hamming distance between a possible emulated response by the server and the received response R from the device. Then a response HD threshold that is less than the average BER of the PUF used by the device is needed to minimize the FRR (False Rejection Rate) and FAR (False Acceptance Rate) of the device [15] as illustrated in Fig. 5. Therefore, in practice, if one of the emulated responses by the server *closely* matches, that is with a response HD below τ, the response sent from the device is accepted as a valid match, otherwise, if none of the emulated responses meets the threshold criterion, the response sent from the device is rejected by the server.

Recall that partial challenges can be discarded by the server, therefore, the randomly selected partial challenges employed by the server further ensures a higher minimum HD among possible responses. Consequently, it is highly unlikely for a full length challenge C to be selected by the device with a HD lower than the BER of a PUF. For instance we can observe from Fig. 4 that the *minimum* hamming distance value is no less than 10.95%, which is 5.95% higher than

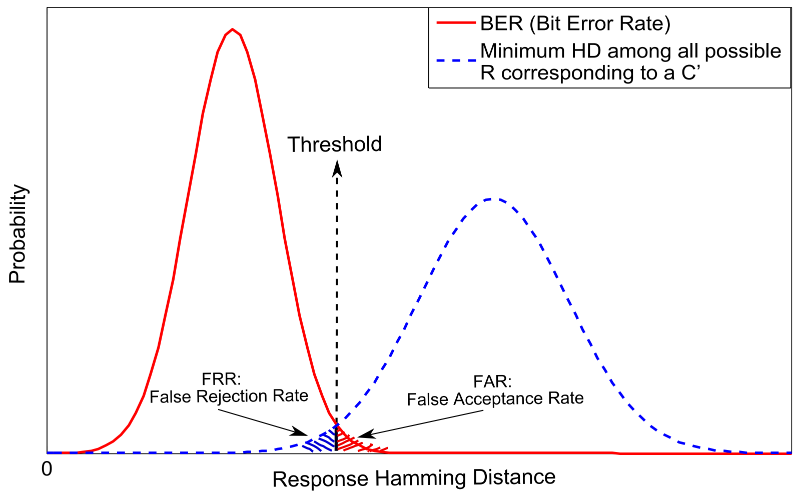


Fig. 5. Illustration of a Bit error rate (BER) distribution and a minimum Hamming distance (HD) distribution among all possible responses, R, corresponding to a given C'. The threshold τ shown is the optimal value that minimizes both FRR and FAR.

the 5% BER of an APUF [1] under worst case power supply and temperature fluctuations.

3.3 Modified Parameter-Based Authentication

Here we present a modified version of the parameter-based authentication protocol in [20]. Our protocol is illustrated in Fig. 6 and outline in detail below:

First: A server measures a specific number of CRPs of a PUF to train a model and securely stores the parameterised model. This is called the provisioning phase.

Second: Whenever a device (prover) needs to be authenticated, the device requests an authentication service from the server.

Third: The server randomly selects a partial challenge C' and sends it to the device. Subsequently, the device generates a challenge C from C' and obtains a response R using the challenge obfuscation method.

Fourth: The device sends the obtained response back to the server.

Fifth: The server performs the authentication by implementing the challenge recovery method, if the response HD between one of the emulated possible responses given a C' and the response sent from the device is lower than the threshold τ, the device is authenticated successfully; otherwise, the authentication fails and the device is rejected.

Sixth: The specific partial challenge C' used is discarded by the server to ensure that is it used only once.

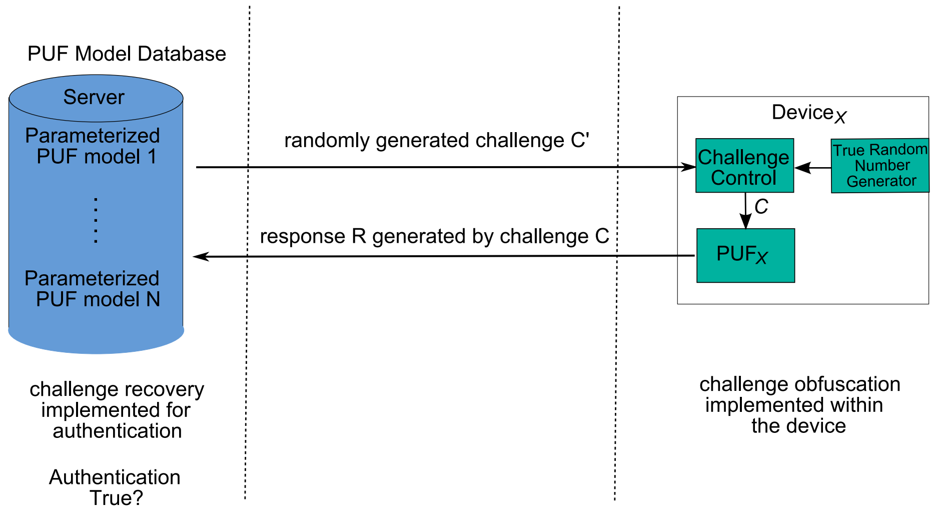

PUF Model Database
Server
Parameterized PUF model 1
Parameterized PUF model N
randomly generated challenge C'
response R generated by challenge C
Device$_X$
Challenge Control
True Random Number Generator
C
PUF$_X$
challenge recovery implemented for authentication
Authentication True?
challenge obfuscation implemented within the device

Fig. 6. Modified parameter based authentication protocol.

The modified authentication protocol can be distinguished from the basic parameter-based authentication [20] in three aspects: (i) The server sends out a randomly selected partial challenge C' instead of a full length challenge C; (ii) The device implements challenge obfuscation to hide the exact full length challenge C used not only to the adversaries but also to the server; (iii) The server is implements a challenge recovery method in contrast to a response recovery method to successfully authenticate a device. An adversary now faces the challenge of uncovering the exact relationship between the partial challenge C' and the response R.

3.4 Key Exchange Protocol

Considering Fig. 2, if the true random number generator and the control logic are implemented in software rather in hardware in the device. The prover who has already securely got the PUF can determine which *Pattern* is selected to stimulate the PUF. In this case, the modified parameter-based authentication protocol can be also utilized to secure key exchange.

For the example given in Fig. 3, where an prover is given 4 options, the prover can select *Pattern 1* to *Pattern 4* on behalf of value 0 to 3 correspondingly. These 4 random numbers is encoded into 2 bits that can be exchanged during one authentication round, if the secret key needs to be exchanged is 128-bit, the authentication should take $64 = \frac{128}{2}$ rounds to transfer all of the secret key. It should be noted that the required transfer rounds can be reduced by increasing the number of *Patterns*. In this way, the authentication protocol can also act as key exchange protocol and securely transfer keys between two parties (here are the server and prover).

4 Analysis

After implementing challenge obfuscation, the adversary faces a challenge to find out the full length challenge C used to generate the response, since the adversary can only acquire the partial challenge C' and the response R regardless of possible concern about eavesdropping or direct measurements. However, the adversary has no knowledge on which *Pattern* is randomly picked up in Fig. 3. In other words, the adversary has no knowledge of the exact full length challenge C and the corresponding response relationship.

4.1 Statical Analysis

It has been demonstrated that the number of CRPs needed to train a machine learning model is a function of the prediction accuracy $1-\epsilon$ (ϵ is the error prediction rate of the machine learning model) and the number of stages, k, in an APUF [17, 18, 16]

$$N_{\text{CRP}} = O(\frac{k}{\epsilon}) \tag{4}$$

Usually, to impersonate a physical PUF, the model should achieve a prediction rate higher than the reliability of the PUF. For example, more than N_{CRP} of CPRs is needed to break a n-bit APUF. To achieve a prediction accuracy of $1-\epsilon$, the adversary needs $N_{\text{CRP},1-\epsilon}$ CRPs for training to break an APUF. However, in our case, the adversary has no knowledge of full length challenge C, and hence has to guess it based on guessing the *Pattern* that is randomly selected in the device. To achieve the same prediction accuracy of $1-\epsilon$ from machine learning through challenge-replication strategy [24], which the adversary substitute all possible p CRPs given a C' to his/her model. As for each possible CRP, the adversary needs build up a model. The adversary has to conduct $N_{\text{CRP},1-\epsilon}$ rounds of authentication with the server and each time use one of his/her trained models. If the adversary correctly guesses all the precisely full length challenge C used in the device for all of these $N_{\text{CRP},1-\epsilon}$ rounds. Therefore, one of his/her models will pass authentication. Then the number of models the adversary has to try/build is expressed as [16]:

$$N_{\text{Model}} = p^{N_{\text{CRP},1-\epsilon}}, \tag{5}$$

where N_{Model} is the number of models the adversary needs to build up, and then gains one out of them, to pass the authentication or impersonate a physical PUF. In the above equation, we see by only using $p=4$ *Patterns*, the adversary has already faced a great challenge to find out the correct model. Moreover, the p can be increased to an even large number.

4.2 Model Building Attack Test

CRP Generation The CRPs used in the following are generated through simulations that can effectively model a physical APUF architecture [13, 17, 18].

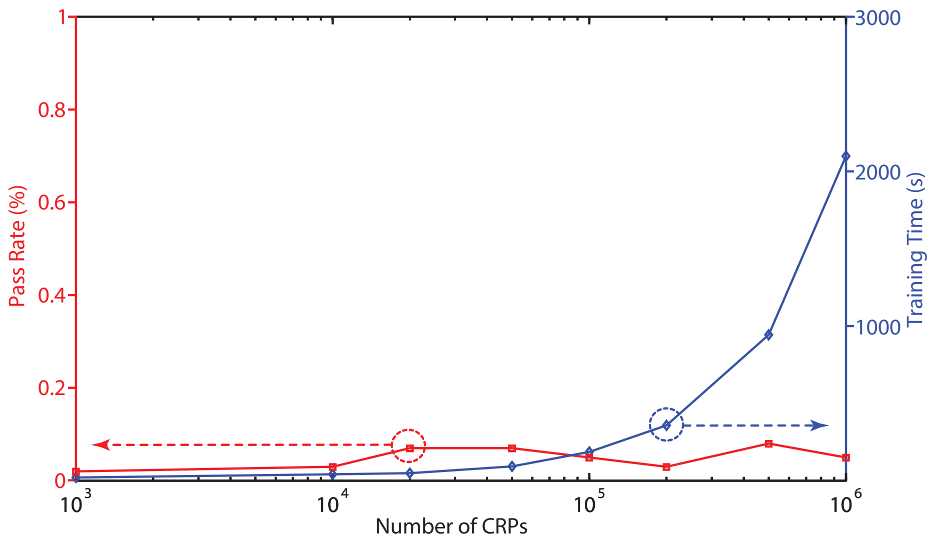


Fig. 7. Pass rate and training time as function of the number of CRPs when challenge obfuscation is implemented.

The delay values for different stages in APUF are randomly produced following a standard normal distribution. When a specific response is required, a given challenge C is applied to the APUF to determine which delay values $\sigma_i^{0/1}$ in each stage will be selected according to the logic value of c_i. Then according to the linear delay model, the selected delay values corresponding to two electrical signal paths are simply added up and compared with each other to generate a response bit.

Tests First, we define *pass rate* is the possibility that the fractional hamming distance between the predicted response and the response from measurement is less than a threshold (τ). Here $\tau = 6.25\%$ that is greater than the bit error rate (BER) of n-bits APUF (5.0%) used for tests. The $\tau = 6.25\%$ can ensure successful authentication even considering worst-case BER than caused by a wide range of fluctuations on voltage supply and ambient temperature. As for a 128-bits response, if the hamming distance between predicted response and measured response is less than 8 bits, the server will let the adversary pass the authentication, which means the adversary's model successfully impersonate a physical n-bits APUF.

The LR (logistic regression) machine learning algorithm is used for building a machine learning model. Because this algorithm shows best performance to train a model to break PUFs [17]. The *pass rate* after implementing the proposed challenge obfuscation method is shown in Fig. 7. It can be seen that the *pass rate* is less than 0.1% even after 1×10^6 CRPs used to train a model. Moreover, increasing the number of CRPs does not help improve the *pass rate*. The reason is that $\boldsymbol{\Phi}$ used to evaluate vector $\boldsymbol{\omega}$ is always missing several features due to the elimination of 4 bits from the full length challenge C. Only increasing the number of CRPs for training without inferencing the full 64-bit challenge does not result in increasing the *pass rate*.

To make a direct comparison, we test the *pass rate* without implementing the challenge obfuscation by also using LR machine learning algorithm. It is shown that the *pass rate* achieves 100% only after 2000 number of CRPs used to train the model within only 46.81 seconds. To sum up, it can be seen that our proposed challenge obfuscation method can significantly increase the resistance to model building attacks through machine learning algorithms, LR, which is the most effective algorithm when it is deployed to break PUFs [17].

5 Conclusion

In this paper, we propose a challenge obfuscation and a challenge recovery method to thwart model building attacks on PUFs, in particular, APUF and k-sum PUF, which build upon linear additive blocks. We provide a modified parameter-based authentication mechanism that is able to obfuscate the adversary, while still allowing a server (or verifier) to successfully authenticate a device (or prover). Moreover, we demonstrate that the modified parameter-based authentication protocol can further be used for securing key exchange. Furthermore, the presented statistical analysis demonstrates that the number of models needed to build up after using the proposed challenge obfuscation method increases exponentially. Finally, we implement model building attacks using the most effective machine learning algorithm against PUFs, Logistic Regression, to evaluate our proposed method. Our tests confirm that our method significantly enhances the security of APUFs, and therefore linear additive blocks based PUFs in general, by making model building attacks infeasible.

The limitation of our work is that we have not conducted a full-challenge-replication strategy through inferencing the full challenge based on *Patterns*, although it has already been shown that a full-challenge-replication strategy is not efficient once an adversary has to guess the exact challenge-response relationships [24]. Moreover, trade-offs among the number of *Patterns*, computational load on the server, the FAR rate, the FRR rate, robustness to machine learning attacks need to be investigated in more detail. These tasks will be our future work.

6 Acknowledgment

This research was supported by a grant from the Australian Research Council (DP140103448). The authors also appreciate sponsorship from the China Scholarship Council.

References

1. B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas. Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 148–160. ACM, 2002.
2. B. Gassend, D. Lim, D. Clarke, M. Van Dijk, and S. Devadas. Identification and authentication of integrated circuits. *Concurrency and Computation: Practice and Experience*, 16(11):1077–1098, 2004.
3. C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas. Physical unclonable functions and applications: A tutorial. *Proceedings of IEEE*, 102:1126–1141, 2014.
4. D. E. Holcomb, W. P. Burleson, and K. Fu. Initial SRAM state as a fingerprint and source of true random numbers for RFID tags. In *Proceedings of the Conference on RFID Security*, volume 7, 2007.
5. D. E. Holcomb, W. P. Burleson, and K. Fu. Power-up SRAM state as an identifying fingerprint and source of true random numbers. *Computers, IEEE Transactions on*, 58(9):1198–1210, 2009.
6. S. S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls. The butterfly PUF protecting ip on every FPGA. In *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, pages 67–70. IEEE, 2008.
7. J. W. Lee, D. Lim, B. Gassend, G. E. Suh, M. Van Dijk, and S. Devadas. A technique to build a secret key in integrated circuits for identification and authentication applications. In *2004 Symposium on VLSI Circuits, Digest of Technical Papers*, pages 176–179. IEEE, 2004.
8. D. Lim. *Extracting secret keys from integrated circuits.* PhD thesis, Massachusetts Institute of Technology, 2004.
9. D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. Van Dijk, and S. Devadas. Extracting secret keys from integrated circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(10):1200–1205, 2005.
10. R. Maes, P. Tuyls, and I. Verbauwhede. Intrinsic PUFs from flip-flops on reconfigurable devices. In *3rd Benelux Workshop on Information and System Security (WISSec 2008)*, volume 17, 2008.
11. R. Maes, A. Van Herrewege, and I. Verbauwhede. PUFKY: A fully functional PUF-based cryptographic key generator. In *Cryptographic Hardware and Embedded Systems–CHES 2012*, volume 7428, pages 302–319. Springer, 2012.
12. M. Majzoobi, F. Koushanfar, and Potkonjak. Lightweight secure PUFs. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 670–673. IEEE, 2008.
13. M. Majzoobi, F. Koushanfar, and M. Potkonjak. Testing techniques for hardware security. In *IEEE International Test Conference, ITC*, pages 1–10. IEEE, 2008.
14. 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 *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*, volume 6917, pages 33–44. IEEE, 2012.
15. M. Roel. *Physically Unclonable Functions: Constructions, Properties and Applications.* PhD thesis, Ph. D. thesis, Dissertation, University of KU Leuven, 2012.
16. M. Rostami, M. Majzoobi, F. Koushanfar, D. S. Wallach, and S. Devadas. Robust and reverse-engineering resilient puf authentication and key-exchange by substring matching. *IEEE Transactions on Emerging Topics in Computing*, 2(1):37–49, 2014.
17. U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber. Modeling attacks on physical unclonable functions. In *Proceedings of the 17th ACM*

Conference on Computer and Communications Security, pages 237–249. ACM, 2010.

18. U. Ruhrmair and M. Van Dijk. PUFs in security protocols: Attack models and security evaluations. In *2013 IEEE Symposium on Security and Privacy (SP)*, pages 286–300. IEEE, 2013.
19. Y. Su, J. Holleman, and B. P. Otis. A digital 1.6 pJ/bit chip identification circuit using process variations. *IEEE Journal of Solid-State Circuits*, 43(1):69–77, 2008.
20. G. E. Suh and S. Devadas. Physical nclonable functions for device authentication and secret key generation. In *Proceedings of the 44th Annual Design Automation Conference*, pages 9–14. ACM, 2007.
21. D. Suzuki and K. Shimizu. The glitch PUF: A new delay-PUF architecture exploiting glitch shapes. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 366–382. Springer, 2010.
22. V. van der Leest, G.-J. Schrijen, H. Handschuh, and P. Tuyls. Hardware intrinsic security from D flip-flops. In *Proceedings of the fifth ACM workshop on Scalable trusted computing*, pages 53–62. ACM, 2010.
23. M. van Dijk and U. Rührmair. Physical unclonable functions in cryptographic protocols: Security proofs and impossibility results. *IACR Cryptology ePrint Archive*, 2012:228, 2012.
24. M.-D. Yu, D. M'Raihi, I. Verbauwhede, and S. Devadas. A noise bifurcation architecture for linear additive physical functions. In *Hardware-Oriented Security and Trust (HOST), IEEE International Symposium on*, pages 124–129. IEEE, 2014.
25. M.-D. M. Yu, D. MRaihi, R. Sowell, and S. Devadas. Lightweight and secure PUF key storage using limits of machine learning. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 358–373. Springer, 2011.
26. L. Zhang, Z. H. Kong, and C.-H. Chang. PCKGen: A phase change memory based cryptographic key generator. In *Proc. of IEEE International Symp. on Circuits and Systems (ISCAS)*, pages 1444–1447. IEEE, 2013.