IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 7, 2022, accepted March 23, 2022, date of publication April 6, 2022, date of current version April 19, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3165284

Helper Data Masking for Physically Unclonable Function-Based Key Generation Algorithms

AMIR ALI POUR[1,2], FATEMEH AFGHAH[2], (Senior Member, IEEE), DAVID HÉLY[3], VINCENT BEROULLE[1], GIORGIO DI NATALE[4], (Senior Member, IEEE), ASHWIJA REDDY KORENDA[5,6], AND BERTRAND CAMBOU[6], (Member, IEEE)

[1]LCIS, Grenoble INP, Université Grenoble Alpes, 26000 Valence, France
[2]Electrical and Computer Engineering Department, Clemson University, Clemson, SC 29634, USA
[3]CEA LETI, Université Grenoble Alpes, 38000 Grenoble, France
[4]TIMA, CNRS, Grenoble INP, Université Grenoble Alpes, 38000 Grenoble, France
[5]School of Informatics, Computing and Cyber Systems, Northern Arizona University, Flagstaff, AZ 86011, USA
[6]Department of Applied Physics and Materials Science, Northern Arizona University, Flagstaff, AZ 86011, USA

Corresponding author: Amir Ali Pour (amir.ali-pour@lcis.grenoble-inp.fr)

This work was supported in part by the French National Research Agency in the framework of the "Investissements d'avenir" Program under Grant ANR-15-IDEX-02, and in part by the U.S. National Science Foundation under Grant IIP-2204502.

ABSTRACT Key exchange protocols are a crucial part of the internet-based communication between connected devices in IoT. In this regard, Physically Unclonable Function (PUF) has been an enabler to provide intrinsic highly randomized source for key generation without requiring extra storage components. PUF however, is an unstable source. In that sense, Fuzzy Extractor (FE) methods with Error Correction Code (ECC) are used to ensure reliability of the key value. FE methods incorporate publicly available helper data to recreate an originally enrolled encryption key from the PUF in mission mode. It is crucial to ensure that the publicly available helper data leaks no valuable information from the source key value to allow untrusted parties to recreate the key. Here the adversary's work is to modify the helper data to decrease the entropy of the recovered codes by the ECC, and push the communicating parties in generating the key that is known to the adversary as well. In this work, we propose to protect helper data via a PUF-based masking mechanism with variable positioning. Masking with variable positioning adds a new fold of complexity for the adversary which is capable to considerably increase the guessing entropy. Our experimental results show that for 256-bit helper data, a 16-bit mask value can increase the guessing entropy by 5 folds against a Reed Muller majority logic vote decoder. Moreover, we show that an increased number of masking such as 4 times a 16-bit masking, can increase the guessing entropy against the same Reed Muller decoding function by 20 folds.

INDEX TERMS Fuzzy extractor, helper data manipulation attack, majority logic vote decoder, physically unclonable function, SRAM PUF.

I. INTRODUCTION

Physically Unclonable Functions (PUF) are known ubiquitously as ideal security primitive for light-weight and low-cost encryption key generation and device authentication [1]–[3]. PUF is mostly seen as a hardware function which incorporates the process variations (e.g., threshold voltage, critical dimensions) of a manufactured chip into a digital "fingerprint" that is unique to the hosting device [4], [5].

Memory-based PUF such as SRAM PUF had gained significant attention for key generation schemes. Memory-based PUF incorporates the micro physical features of a manufactured device memory, such as the initial value of the SRAM memory cells on device power-up state [5]. A good PUF is extracted from physical components' process variation and is consequently prone to erratic responses due to variations in temperature, voltage, etc. Therefore, PUF response should be used with error correction codes to build robust encryption keys [4], [6]–[9]. To utilize SRAM PUF in key generation schemes, a mechanism which enables perfect correction of the unstable output of SRAM cells with zero errors is required such as in [8], [10], [11].

Early encryption key generation methods based on memory-based PUF such as in [10] and [12], suggest using

The associate editor coordinating the review of this manuscript and approving it for publication was Pedro R. M. Inácio.

fuzzy extractor (FE) to generate an originally enrolled secret key from a noisy PUF response. In these methods a registration phase exists where an original secret key is generated with a helper data. Once the key and the helper data are generated, the key is stored and the helper data is shared publicly to be re-queried by the enrolled device for mutual key generation. Since helper data is assumed publicly available, it is prone to risks of being queried and manipulated by adversarial parties as well [13]–[16]. Various Helper Data Manipulation (HDM) Attacks exist which aim either at regenerating the original secret key by modifying the helper data and redirecting it to the PUF-enabled device [17], or derive the PUF-enabled device to regenerate a key value that is known to the adversary [18]. The second type of HDM Attack in fact aims to reduce the entropy of the recovered codes by FE while it uses the modified helper data for code recovery. The lowered entropy is also known to the adversary if he knows what ECC is used on the device. Therefore, with an enumerable number of guesses for the recovered code, the adversary can generate a mutual key same as it is generated on the target device.

Several countermeasures have been already proposed to protect PUF-based key generation against specific helper data manipulation attacks [19]–[21].However, these methods are designed either for a specific protocol implementation, or only provide manipulation detection of the helper data. Therefore, the overall security of the helper data remains an open problem since powerful HDM Attacks as we discussed above are able to considerably reduce the entropy of the recovered codewords and bias the key generation on the PUF-equipped devices.

In this work, we propose helper data masking to increase the guessing entropy for HDM Attacks. We propose using the embedded SRAM PUF as the source of mask value generation. This requires to use again the FE and ECC to build robust mask values which can be regenerated in the mission mode. By using PUF as the source of mask values, we assure first that the randomness of mask values is as high as for the generated secret keys. Moreover, we assure that there is no reliance on non-volatile memories to store the mask value.

Here we elaborate on a masking protocol with variable positioning to randomize the helper data. Variable positioning of the mask value in turn adds a new fold of complexity into the guessing structure of the HDM Attacks. Thus the adversaries should take into account every possible position that the mask is applied on the helper data. Depending on how large the helper data is, variable positioning can infer large possibilities of where the mask is applied on the helper data. This in turn can decrease the success-rate of the HDM Attacks by orders of magnitude. We support our work with experimental assessments of a simulated HDM Attack on a Key generation method based a Reed Muller code with Majority logic vote decoding mechanism that is protected with our proposed masking mechanism. We show how masking with variable positioning on helper data can drastically decrease the success-rate of HDM Attacks. The contributions delivered in this work are in the following:

1) Introducing a robust SRAM-PUF-based randomized mask value generation scheme.
2) Introducing helper data masking with variable positioning as a countermeasure against a higher-order HDM Attack where the knowledge of the masking with variable positioning is known.
3) Experimental assessment of helper data masking with variable positioning by reproducing the key generation and HDM Attack proposed in [18] in MATLAB.

The rest of the paper is organized as follows. In section II the preliminary information of PUF based key generation using FE and ECC, as well as HDM Attack are discussed. In section III, we elaborate on our helper data masking with the proposed variable positioning technique. Here we also elaborate on the adaptation of HDM Attack which is aware of the masking mechanism in the key generation technique. We also discuss the characteristic of weak mask values in this section. In section IV, we discuss our theoretical analysis of HDM Attack guessing entropy against our proposed masking mechanism. In section V we discuss our experimental setup and in section VI we discuss the experimental results of a case study HDM Attack against a candidate FE-based helper data masking. Section VII is the conclusion of our work and our future perspectives.

II. PRELIMINARIES

A. FUZZY EXTRACTOR-BASED ENCRYPTION KEY GENERATION USING MEMORY-BASED PUF AND HELPER DATA

The primary part of encryption key generation is to read the power-up binary values of memory cells. During the enrollment phase, the power-up binary values of memory cells are captured and stored on the server. These values are in turn the source to create encryption key values. An original key value is the base value which is hashed to create the encryption key on the server side. Meanwhile, an offset code from the original key value is also created as the helper data to send to PUF-enabled device. The PUF-equipped device uses the helper data in a fuzzy extractor to mutually generate the original key value known by the server as well. In this section, we will explain the *Robust Fuzzy Extractor*-like (RFE-like) construction as discussed in [18]. RFE is commonly used in key generation schemes to re-generate an original secret value from a noisy source (e.g., PUF) by using a publicly available helper data. To ensure a secure value recovery in an RFE construction, a 2 step verification is performed. First, a hash value generated from the recovered secret value is compared with a hash value given as part of the helper data. These two values should be the same to succeed in the first step. Secondly, the Hamming Distance (HD) between the recovered value and the regenerated value is checked with a pre-defined threshold. The HD value lesser than the threshold implies success in the second step. If any of the verification steps fail, the key generation is considered a failure.

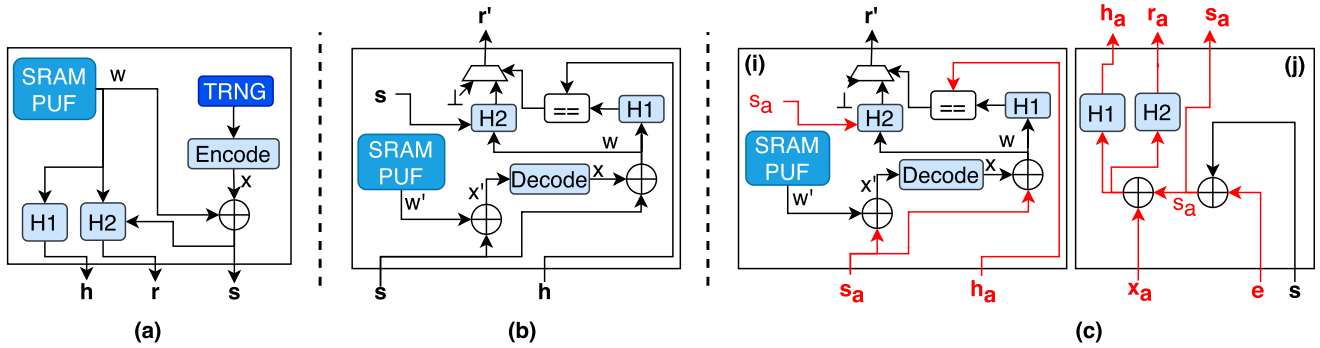


FIGURE 1. Illustration showing the RFE-like construction (a) the registration phase, (b) the recovery phase, and (c) the recovery phase with presence of HDMA, of the SRAM-PUF based encryption key generation.

RFE-like construction is also a derivative of RFE construction with the difference that the security of the RFE-like is provided with only the verification procedure through a comparison of an original and a regenerated hash value. A detailed description of an RFE-like construction is in the following.

FIGURE. 1.(a) shows the enrollment phase of an RFE-like construction where a verification hash value h is created by hashing an SRAM-PUF response w with $H1$ hash function. Helper data s is also created by XORing the SRAM-PUF response w with a generated random codeword x. Noting that a *True Random Number Generator (TRNG)* is used here to generate a random message which is encoded to yield a random codeword x. The SRAM-PUF response is hashed with $H2$ to extract the encryption key r which is stored on the server to be used during the encryption key recovery phase.

FIGURE. 1.(b) shows the recovery phase of an RFE-like construction where helper data s and verification hash value h are requested from the server by the PUF-enabled device. As the device receives the queried data s and h, s is then used to recover the generated codeword x from the noisy SRAM-PUF response w', and then the original SRAM-PUF response w is recovered by XORing the x with s. The recovered w is then hashed with $H1$ and the regenerated hash value is compared with h that was received from the server, to verify the originality of w. If equal, w and helper data s are hashed with $H2$ to regenerate the encryption key r for final output r'. If not original, the extractor returns failed on r'.

B. HELPER DATA MANIPULATION ATTACK

HDM Attack refers to an attack against soft decision ECCs [16], wherein the attempt is to reconstruct the original PUF response in a divide and conquer fashion, by passing many attempts of sending modified helper data to the PUF-enabled device. During each failure in response reconstruction, the adversary learns information about the original response, leading to the reconstruction of the response. While proved in theory, in practice, it will take many attempts until the adversary obtains full knowledge of the original response. In contrast to the primary HDMA, another HDMA was also introduced in [18] where the attempt is not the reconstruction of the original PUF response but instead to try to set the reconstructed PUF response to a value known by the adversary.

In practice, often single long codes are not used as codewords, but instead, it actually consists of smaller concatenated codewords [22]–[25]. This approach is considered for the RFE-like construction we use in our evaluations. In this case, the first assumption is that the adversary knows the size of target codeword, and will apply helper data modifications repetitively on each sub-division of the helper data. FIGURE. 1.c shows the schematic of HDMA during the recovery phase. During the attack, we assume that the adversary is eavesdropping on the public data query and responds to the query from the server. Using helper data s and hash value h from the server, the adversary targets a subsection of the helper data s, by applying an error vector to each sub-part to manipulate the helper data and create *adversary's helper data*, s_a. The adversary also has a reduced set of codewords C_a which is a subset of the larger codeword set C that is used on the server during the enrollment phase to create helper data. This reduced set C_a is in fact extracted relative to the error vector the adversary is using. By choosing a codeword X_a from C_a, the adversary attempts the re-generating of a PUF response w_a, which in turn is used to create a candidate adversary verification hash value h_a and adversary encryption key r_a.

The adversary's' helper data s_a and adversary's' verification hash value h_a are then sent to PUF-enabled device, as a response to the public data query the PUF-enabled device has passed initially. While on the PUF-enabled device, the same steps of the recovery phase are taken, the way for the adversary to know if the re-generation of adversary's key value r_a is successful, is to receive the positive response of the comparison check of the adversary verification hash h_a with the one locally generated on PUF-enabled device.

Usually with the presence of HDMA, the failure rate of key generation will rise as expected. Therefore, the recovery process will be to repeat, until either the number of attempts for re-generation per query passes a pre-defined threshold, or the regeneration of a mutual key between the adversary

and the PUF-device becomes successful, leading to the result of having r' being equal to r_a.

As the RFE-like construction is explained, one can see that the reason why HDMA can be successful is that there is no check on the authenticity of the received helper data s during the recovery phase. In other words, during the recovery phase, any helper data $\bar{s}$ can be used to recover a codeword $\bar{x}$ and following that, produce a random PUF image $\bar{w}$ and a verification hash value $\bar{h}$. And as long as $\bar{h}$ is equal to a received verification hash value h, the regenerated key r' is valid. And since the adversary can also decide on the value of h, producing h_a, he can enforce a match between a $\bar{h}$ and h_a after multiple trial and errors.

In the following section it is shown how these notions were used in the baseline implementation of the experimental setup of this work.

III. HELPER DATA MASKING WITH VARIABLE POSITIONING

In this section, we elaborate on PUF-based helper data masking mechanism with variable positioning. First, we discuss how we employ PUF as the source of mask value generation. Then, we elaborate on the variable positioning mechanism. Later on, we discuss the threat model which is a derivative of HDM Attack that has knowledge of masked helper data with variable positioning. We then statistically show how the new threat model still has a large guessing field to explore breaking the key, compared to the conventional case of attacking helper data without masking as discussed in [18].

A. MASK VECTOR GENERATION

Commonly on a secured PUF-enabled device, using a non-volatile memory (NVM) on the PUF-enabled device to store secret values (e.g., mask values) is not suggested. That is due to its cost and the security issues with this type of memory. Moreover, for generating mask values on a PUF-enabled device, the values should be highly randomized to avoid any exploitable leaks that allow adversarial third parties to recreate the mask. Here we propose using SRAM-PUF itself as a source of mask generation. In our proposed scheme, the SRAM-PUF will be the source of both the key generation, and mask vector generation. This in turn eliminates the need for any storage component on the board to store the mask values. Therefore the method will be cost-efficient and physically secure. Moreover the PUF, assuming it has a good characteristic, is an ideal source of randomized value generation. That is a key criterion for key generation which makes at first place the PUF a good candidate for encryption key generation. For the same reason, it can be a good candidate for generating mask values with high randomness as well. Here we define the mask vector generation as a process to read the power-up binary values of several consecutive memory cells from an SRAM device.

Using PUF as the mask value source, one should guarantee that the generated mask vector is highly reliable since the source is inherently noisy. This means that the recovery of the mask value on the PUF side should yield exactly the same value used on the server side. To assure the reliability of mask vectors, one can consider using a fuzzy extractor that can suitably address the noisiness of the PUF source. The helper data based RFE-like construction in this case can also be used to assure reliability, equally to that in Key generation.

To elaborate on helper data masking, let us define the process of masking to be the XORing of a candidate mask vector on some sub-parts of the helper data bit stream. We assume the mask vector itself is a bit stream as well. The masking process is issued on the server side after the helper data is reloaded and just before answering the query from the PUF-enabled device. To answer the query, the server sends the masked helper data with an extended helper data for the recreation of the mask value along side with the rest of the public data to the PUF-enabled device for mutual key generation. On the PUF enabled device, the masked helper data is demasked using a mask vector that is recovered from the on-board PUF and the extended helper data.

For an adversary in the middle, we assume that there is no access to the source of the mask value. Thus the adversary has to undergo a guess-based procedure to discover the mask vector value in order to demask the helper data. Recalling that the adversary initially needs the original helper data in order to recreate the PUF response. Therefore, one can say that masking of helper data potentially increases complexity of the HDM Attack. This in turn can distance the adversary from the point of success in generating a mutual encryption key.

The general sketch of our proposed key generation protected with helper data masking is shown in FIGURE. 2. Here, the key generation process is the same as explained in Section II.A. The additional part is the masking of the helper data before answering a query from a device. Here also the helper data based RFE-like construction is used for mask vector generation during the recovery phase on the PUF-Enabled device. Following FIGURE. 2, during the registration phase, aside to enrolling PUF response w_1 for key generation, the PUF response w_2 as mask vector generation is also enrolled. Correspondingly, helper data s_2 of the enrolled mask vector is also created and also w_2 as the original mask vector, is XORed with s_1 to mask the primary helper data and produce Ms as masked primary helper data. All of the generated information in this phase are then stored on the server.

During the recovery phase, the public information sent to PUF-Enabled device includes 2-parts helper data, where the initiative part is the masked primary helper data Ms used for key generation. The latter part is the secondary helper data s_2 used for mask vector recovery. During the recovery phase, the primary attempt is to recover the original mask vector. Thus secondary helper data s_2 is XORed with captured PUF response w_2' from SRAM-PUF 2, and the output is decoded to generate a recovered codeword x_2. The output is then XORed with s_2 to recover w_2 as the original mask vector. w_2 is then XORed with the masked primary helper data Ms to demask the helper data and produce s.

Using this construction, one can assure that reliability is equally provided for the recovered mask vector as well as in PUF response for key generation, in addition to primarily securing primary helper data. However, the secondary helper data is now exposed and HDM Attack can exploit that to break the masking. However, with the extension of adding variable positioning mechanism to the masking scheme, one can assure that even with the exposure of the mask value helper data, the adversary has to undergo an exploration in a large guess-field in order to guess the position of the mask and correctly demask the helper data.

B. MASKING WITH VARIABLE POSITIONING

To mask the helper data with variable positioning, we propose applying a mask vector which is a bit vector smaller than the helper data bit string. Additionally, we propose to variate the mask vector's application point with respect to the value of the mask. Therefore, the address of the target region on the helper data string is defined by an address vector that is defined within the mask vector itself. FIGURE. 3 shows how this positioning mechanism works. Wherein the first n bits of a k-bit mask vector, w_2 is used to define the address of the mask vector (e.g., the ith block of the primary helper data s_1 as shown in FIGURE. 3). Given that the helper data and the mask vector are bit strings, the address checker block shown in the figure can in turn be a shift register which outputs the bit string with the same size of the helper data. In the string, the mask vector is shifted n times from the beginning of the string to be placed in the required address for masking. The output string then can be XORed with the helper data to produce masked helper data.

Following this construction, the address of the target block depends on the true value of the mask vector. In specific, it is depended on the part of the mask vector which the address is extracted from. Moreover, to break the key from a key generation procedure protected with masking with variable positioning, the HDM Attack model needs to adapt as well. In the following we explain the HDM Attack that has knowledge of masking with variable positioning.

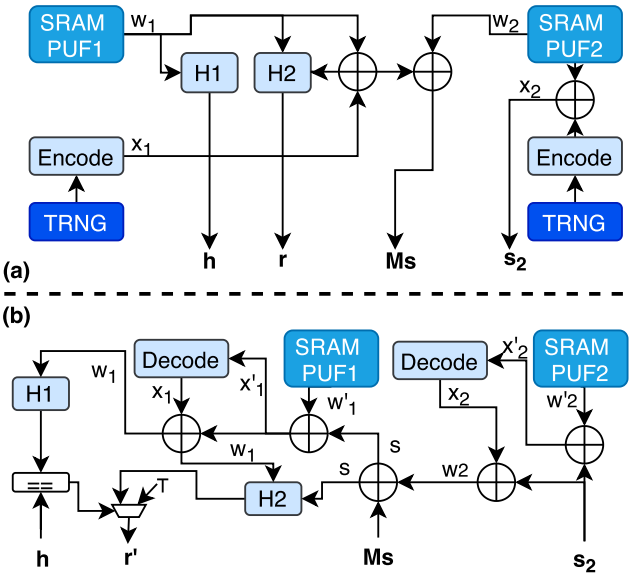


FIGURE 2. Illustration showing the structure of helper data based RFE-like construction protected with helper data masking. Here (a) is the registration phase, and (b) is the recovery phase.

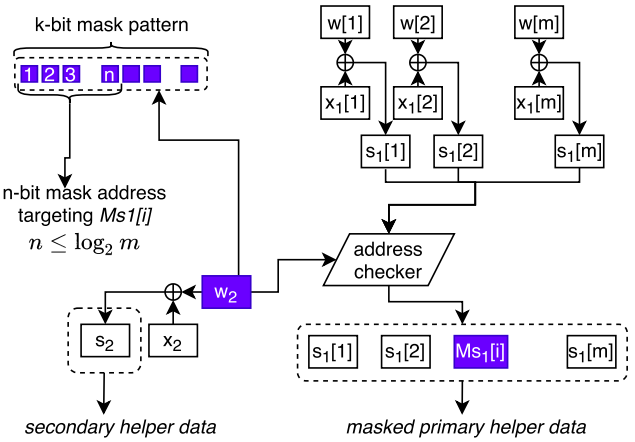


FIGURE 3. Illustration showing how masking with variable positioning is applied on helper data.

C. VULNERABILITY OF MASKING AGAINST HDM ATTACK

With the application of masking over helper data, the HDM Attack as proposed by [18] will not be applicable anymore to break the key. However realistically, we can assume that at some point the adversary will obtain knowledge that the helper data is masked. We assume here the worst case in which the mechanism of helper data masking with variable positioning is known to the adversary. Thus, the adversary will try at the same time to find a combination of guessable values for the codeword with reduced entropy for mask value regeneration, the position where the mask is applied on the masked helper data, and the codewords with reduced entropy for the regeneration of the encryption key. The key point here is that with reducing the entropy of guessable codeword to break the mask value, the adversary is still faced with the same entropy of guessing the position of the masked region on the helper data. In other words, since the position of the masked region is dependent on the true value of the mask, its entropy will stay the same even after modifying the helper data by the adversary. In the following, we explain in details the threat model against masked helper data.

The new threat model of HDM Attack which is a derivative of Becker's HDM Attack proposed in [18] is shown in FIGURE. 4. The primary phase of the new HDM Attack is to modify the mask value. At this phase the adversary first elects an error vector e_i and accordingly a $X_{a,i}$ form a reduced set of codewords with k possible codewords, accordingly $\{X_{a,1}, \ldots, X_{a,k}\}$. The helper data s_2 for the mask value is then XORed with the elected e_i and Xa_j. At this stage the adversary's modified mask value wa_2 is generated. wa_2 is then passed into a mask stream generator function which takes in addition an elected $Addr_i$ which is a number indicating

the position of the mask on the stream. The mask stream is then XORed with the received masked helper data *Ms*. The product of the last XOR is an elected demasked helper data *dmsa* which goes into a phase two of modification which is similar to that as discussed in section II.B.

As can be seen, in this model, two guessing fields in phase 1 are similar to that of phase 2. Specifically, the guessing field for electing an error vector and a codeword. However, an additional guessing field is also needed to elect the address of the mask. Noting that the guessing field for the address of the mask cannot be reduced similar to the guess field for the reduced codewords to elect a Xa_i. This is due to the fact that the address of the masked region depends on the original value of the mask. Recalling that we proposed to define the address space within the mask value itself (e.g., the first n bits of the mask value). Therefore at any case, the adversary needs to brute-force the guessing of the address value. This means that if a user initially defines a large address space for masking the helper data, the guessing field for the address value on the adversary's side would consequently be large proportionally.

We recall that the intention of masking the helper data is for further randomization of the helper data stream. However, at some point, the mask value could result in a neutralized product which could make the masking ineffective. Such a case can appear if the product of the mask vector on the specified region of the helper data would yield a product where the affected codeword(s) of that region are new codewords. This in turn means that the baseline HDM model can break the key without going through phase 1 as shown in FIGURE. 4. In the following, we elaborate on a process to detect such mask values as we refer to as weak mask vectors, in order to build a more robust masking scheme.

D. DETECTION OF WEAK MASK VECTOR

In a noise-free setting for the SRAM PUF source for mask vector generation, the first milestone is the selection of mask vectors. In the specific case against HDMA, not any arbitrary value for a mask vector is secure. Potentially against HDMA, certain values of mask vectors would still allow HDMA the equal chance of success as in the case of no masking, if they fit in the equation brought in (1). We refer to these mask vectors as weak mask vectors.

$$C_a \subset C \quad \forall c_i \in C_a, \ \exists m \Rightarrow m \oplus c_i = c_i'; \quad c_i' \in C_a \quad (1)$$

wherein c_i and c_i' are codeword elements of the main codeword set C and C_a is a subset of C that is used by an adversary.

Against such mask vectors, the adversary can yield success without attempting to demask the helper data. Accordingly, he first queries the server, receiving a masked helper data *Ms* and a verification hash value h. Then, XORs error vector e with the masked helper data to create *Msa*. Noting that the adversary has no knowledge of helper data being masked. In XORing a candidate codeword Xa_i with *Msa*, the adversary then creates a predictable PUF response w_a which then is passed to Hash function $H1$ to produce adversary's verification hash value h_a.

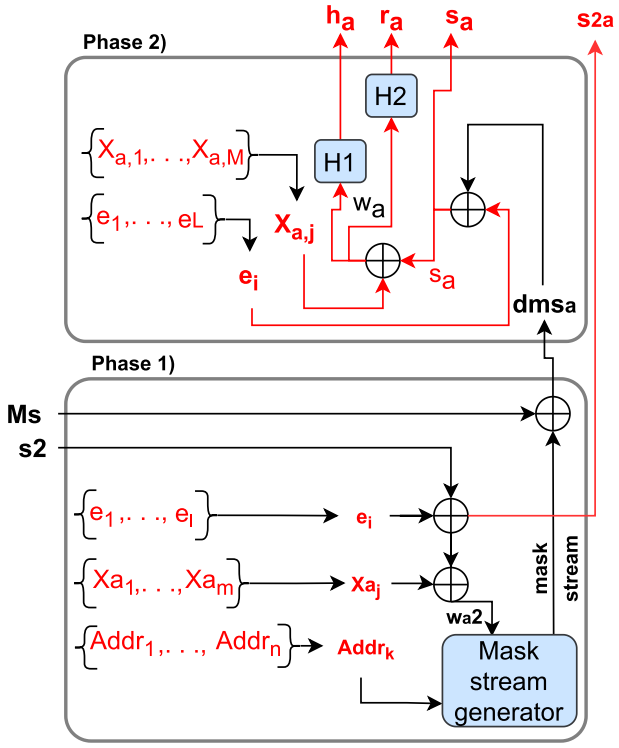


FIGURE 4. Illustration showing the structure of HDM Attack with a new guess field for the address of the masked region on the helper data. Mentioned $Addr_k$ is an elected value from the possible positions of the masked region on the masked helper data *Ms*.

On the PUF-enabled device, the pair of Ms_a and h_a are given. Primarily, the helper data is demasked by XORing the locally regenerated mask vector m, and adversary's helper data s_a is resulted. Then, the helper data is XORed with captured PUF response w' and noisy code x' is produced which is then passed to the decoder function to produce recovered codeword x. At this point, the mask vector will be called weak, if the recovered codeword x, assuming to be one of the guessable codewords by the adversary, is itself a combination of the mask vector and a codeword guessable by the adversary. In this case, after the final XORing of helper data s_a and the codeword to generate recovered PUF response, it will result in the same w_a computed on adversary's side, which will lead to producing the same verification h_a as sent by the adversary to the PUF-enabled device. Therefore at the end, a mutual key will be generated between the adversary and PUF-enabled device, despite using helper data masking to prevent it. There in helper data masking, it would be necessary to avoid considering a potential weak mask vector.

Going back to (1) to identify the weak mask vectors, the equation in fact simply suggests that if a candidate mask vector in XORing with a predicable codeword from C_a lead to another predictable codeword from the same set, then the mask application on Helper Data is not effective against first order HDM Attack. In this case, we address the Becker's baseline HDM Attack as the first order HDM Attack.

TABLE 1. Number of rejected mask vectors out of 1600 candidates per SRAM according to (1).

SRAM1	SRAM2	SRAM3	SRAM4	SRAM5
18	19	15	19	19

SRAM6	SRAM7	SRAM8	SRAM9	SRAM10
14	20	11	22	6

TABLE 2. Number of rejected mask vectors out of 1600 candidates per SRAM according to (2).

SRAM1	SRAM2	SRAM3	SRAM4	SRAM5
608	608	480	608	608

SRAM6	SRAM7	SRAM8	SRAM9	SRAM10
448	640	384	704	192

However, in order to utilize equation (1) to identify weak mask vectors, one would require prior knowledge of predictable codewords, which itself requires knowing the error vectors that are the most effective on the decoder function onboard according to Becker's suggestion in [18]. Alternatively, a relaxed version of this equation can coexist which is brought in (2). This alternative version suggests that if a candidate mask vector in XORing with a valid codeword from set C is equal to another valid codeword from the same set, then the mask vector is not effective against first order HDMA.

$$\forall c_i \in C, \quad \exists m \Rightarrow m \oplus c_i = c_i'; \quad c_i' \in C \tag{2}$$

wherein c_i and c_i' are codeword elements of the main codeword set C. Using this alternative equation however, may come with the cost of reducing the chance of graduating a mask vector.

To experimentally assess the rate of mask vector rejection in our case, and to see how many candidate mask vectors are rejected according to (1), we took an statistical analyis over each of our SRAM-PUF dataset, considering each one at a time being a source of mask vector generation. Results of this statistical check are brought in Table 1. In comparison, we also brought the statistical analysis of mask vector rejection according to the relaxed equation (2), in Table 2.

IV. THEORETICAL ANALYSIS OF MASKING WITH VARIABLE POSITIONING

In theory, one can measure the min-entropy of the PUF responses w_1 and w_2 used for secret key generation and mask vector generation, respectively. Let us first discuss the measurement of uncertainty for an adversary w.r.t to correctly guessing the address of a given masked block within the masked primary helper data. We note that our formulation of the min-entropy is considered in a noise-free case.

Let us first denote the probability of masking a sub-division of a given helper data s as $P(s)$ with a given mask vector w_2. We can then define the entropy of the address to a masked sub-division for a given mask vector, H(Addr), as shown in (3).

$$H(Addr) = \sum_{i=1}^{x} P(s)_i \times (log_2 \frac{1}{P(s)_i}) \tag{3}$$

where x is the number of sub-divisions in the given helper data s that can be addressed for masking. Noting that in this formulation, we assume that the starting point of the address in the given mask vector w_2 is already known by the adversary.

Now we can consider a specification for $P(s)$, by defining the total number of sub-divisions in a given helper data s as $x = \frac{l}{k}$ where k is the size of a given mask vector and l is the size of the helper data vector. If we consider that $P(s)$ is equally distributed for all sub-divisions, we can then define the entropy of the address to a masked sub-division for a given mask vector as in (4).

$$H(Addr) = -\log_2 \frac{k}{l} \tag{4}$$

we can also develop (4) for cases in which two or more mask vectors are considered. Let us denote in masking with variable positioning, using m mask vectors, the length of the ith mask vector by k_i, and the length of the helper data vector by l. Noting that in this setting, we consider each mask vector out of the m blocks, having a variable size, thus denoting the length of the ith mask vector as k_i and not k. We would then have the entropy of masking address as in (5):

$$H(Addr) = -\log_2 \prod_{i=1}^{m} \frac{k_i}{l} \tag{5}$$

we can now use (5) in defining the min-entropy of the mask vector w_2 and the masking address. We would define it as $H_\infty(w_2, Addr)$ and refer to it as the min-entropy of masking.

$$H_\infty(w_2, Addr) = -(\log_2 \prod_{i=1}^{m} Max(P_r\_\beta_i) + \log_2 \prod_{i=1}^{m} \frac{k_i}{l}) \tag{6}$$

where $Max(P_r\_\beta_i)$ is the maximum probability of recovering one of the predictable codewords for the ith vector of the m mask vectors during the recovery phase.

Using equation (6) we can now compute the min-entropy of masking for the experimental cases we discussed in the previous section. Accordingly, the min-entropy for which we used 1 mask vector to mask a 256-bit helper data would increase by 5. This comprises the min-entropy of mask vector which is 1. Recalling that the maximum probability after error vector is applied to a [16, 5, 8] Reed Muller code (w.r.t the notation $[n, k, t]$, where n is the size of the codeword, k is the size of the binary random number and t is the order of the code) with majority logic vote decoder is $\frac{1}{2}$, and the min-entropy of the masking address is 4. Likewise for 2 mask vectors and 4 mask vectors to mask a 256-bit primary helper data, the min-entropy is increased by 10 and 20, respectively.

We can also define the total min-entropy, which comprises the sum of the min-entropy of PUF response

w_1 and min-entropy of masking. We would define it as $H_\infty(w_1, w_2, Addr)$:

$$H_\infty(w_1, w_2, Addr) = -(\log_2 \prod_{i=1}^{n} Max(P_r\_\alpha_i) + \log_2 \prod_{i=1}^{m} Max(P_r\_\beta_i) + \log_2 \prod_{i=1}^{m} \frac{k_i}{l}) \quad (7)$$

where $Max(P_r\_\alpha_i)$ is the maximum probability of recovering one of the predictable codewords for the i^{th} block of the n blocks of the primary SRAM-PUF response (i.e., the source of encryption key generation), during the recovery phase.

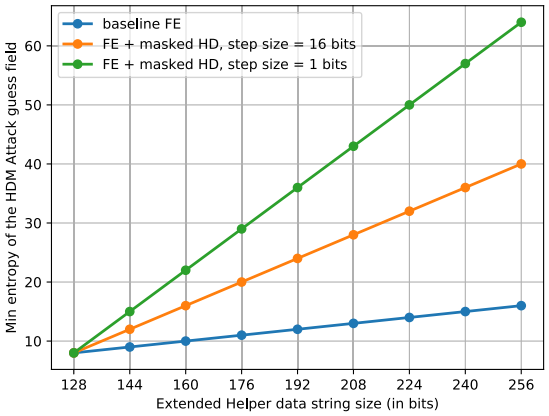
Noting that this specification of min-entropy is not bound to the limits of the primary specifications of our helper data masking with variable positioning scheme. One for instance can expand the address space of adaptive masking, but should however mind that it will then require more bits of the mask vector to define the address. In a noise-free setting, for S number of available maskable regions on primary helper data, the min-entropy of the HDMA against the key generation scheme would be:

$$H_\infty(w_1, w_2, Addr) = -(\log_2 \prod_{i=1}^{n} Max(Pr\_\alpha_i) + \log_2 \prod_{i=1}^{m} Max(Pr\_\beta_i) + \log_2 (\frac{1}{S})^m) \quad (8)$$

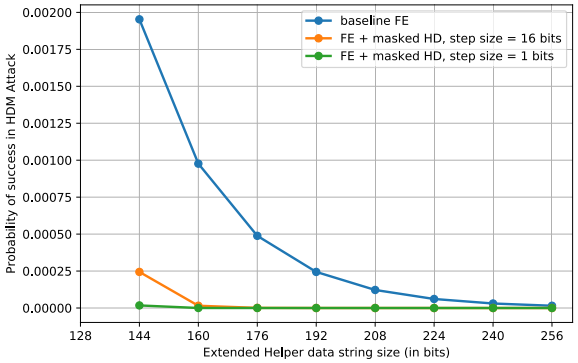
To theoretically analyze the efficiency of our helper data masking, we compare 3 approaches in strengthening the security of an FE based key generation scheme. At one approach, the w_1 string is extended without considering to mask the helper data. At another approach, the w_2 string which is the mask value, is extended and the positioning step is defined to be bit-wise. Therefore, the number of possible positions of the mask over the helper data string for w_1 is $n-m+1$, where n is the size of w_1 and and m is the size of w_2. The other approach here is to consider for each added extension block for w_2, the positioning step to be the size of the added block. Therefore, the total number of possible positions of the mask over helper data s would be $\frac{\text{size of}(w_1)}{\text{size of}(w_2)}$.

Given equation (8), we can demonstrate the increase in the min entropy of beckers HDM Attack against a [16, 5, 8] Reed Muller code and a majority logic vote (MLV) decoder. Initially when encoding, the number of possible codewords are 32 given the specification above. However, as discussed in [18], HDM Attack can reduce it to 2 possible codewords after modifying the helepr data.

We consider the initial size of the key to be 128 bits. FIGURE. 5 shows the increase in the min entropy for the three approaches with respect to extending the helper data



(a) Min entropy of guessing in Becker's HDM Attack against [16,5,8] Reed Muller Code and Majority Logic Vote decoder.



(b) Probability of success of Becker's HDM Attack against [16,5,8] Reed Muller Code and Majority Logic Vote decoder.

FIGURE 5. Illustrations showing the min-entropy and probability of HDM Attack success against increasing size of helper data.

string. It is apparent that the 'FE + masked Helper Data' with 1 bit positioning step size has the lead compared to the other two. The 'FE + masked Helper data' with 16-bits positioning step also has a significant lead compared to the baseline FEs. Recalling that in baseline FE we just increase the size of the w_1 sting which is the source value for generating the encryption key.

We can also assess the probability of success of HDM Attack. The probability of success can also be defined as in (5b).

$$Pr(w_1, w_2, Addr) = \prod_{i=1}^{n} Max(Pr\_\alpha_i) \times \prod_{i=1}^{m} Max(Pr\_\beta_i) \times \prod_{i=1}^{m} \frac{k_i}{l} \quad (9)$$

where $Max(P_r\_\alpha_i)$ is the maximum probability of recovering one of the predictable codewords for the ith block of the n blocks of the primary SRAM-PUF response. And $Max(P_r\_\beta_i)$ is the maximum probability of recovering one of the predictable codewords for the ith vector of the m mask vectors during the recovery phase. In FIGURE. 5b, we can observe the decreasing probability of success for the HDM

TABLE 3. Min entropy of various codes and decoding methods. The two different codes mentioned here are Hard-in Soft-out.

Code	Decoder	Reliability	Min-entropy (per codeword) No Masking	Min-entropy (per 175 bits) No Masking	**Min-entropy (per all bits) 1 block mask 5-bit step size**	**Min-entropy (per all bits) 1 block mask 1-bit step size**	**Min-entropy (per all bits) 2 blocks mask 1-bit step size**
[7,1,7] Repetition Code [16,5,8] Reed-Muller Code	SDML Soft-decision [26]	100%	1	35	**40.09**	**42.38**	**49.66**
		95%	4	138.8	**143.89**	**146.18**	**153.46**
	SDML hard-decision [26]	100%	1	35	**40.09**	**42.38**	**49.66**
		95%	1.5	51.8	**56.89**	**59.18**	**66.46**
	Without Attack		5	175	**180.09**	**182.38**	**189.66**
	GMC [27], [28]	100%	0	0	**5.09**	**7.38**	**14.66**
		95%	2.4	82	**87.09**	**89.38**	**96.66**
	Without Attack		5	175	**180.09**	**182.38**	**189.66**
	Majority logic [18]	100%	0.2	6.8	**11.89**	**14.18**	**21.46**
		85%	0.2	11	**16.09**	**18.38**	**25.66**
	Without Attack		5	175	**180.09**	**182.38**	**189.66**
[8,1,8] Repetition Code [24,12,8] Golay Code	Soft Decision [29]	100%	0	0	**3.32**	**6.77**	**13.2**
		95%	4.1	44.9	**48.22**	**51.67**	**58.1**
	Without Attack		12	132	**135.32**	**138.77**	**145.2**

Attack with respect to extending the helper data string. It can be seen that overall the 'FE + masked Helper Data' with 1 bit positioning step size is the strongest countermeasure. Nonetheless, 'FE + masked Helper Data' with 16 bits positioning step size has a similar characteristic. Nonetheless, both masking approaches seem to be considerably stronger compared to the baseline FE. This suggests ultimately that masking helper data using PUF data seem to be a reliable and strong countermeasure. Thus at the end that considering to use a part of w_1 to construct a w_2 mask value and mask the helper data, can lead to a stronger FE based key generation scheme. In the following, we demonstrate our evaluation of HDM Attack against some real SRAM PUF data we collected from an in-house developed SRAM device.

A. EVALUATION ON VARIOUS CODING AND DECODING METHODS

Reed Muller (RM) codes have been used in abundance for error correction codes in several applications where biometric data is used to generate secret values. Some of the use cases of RM codes are mentioned in [26]–[30]. An evaluation of HDM Attack against RM codes is presented by Becker [18] where the codes are used to recover original secret values generated from PUF source. This work shows how the new HDM Attack can decrease the min-entropy of the recovering codeword by injecting an error vector e into the decoding process. We restate his analysis of HDM Attack over various codes and decoding methods in TABLE. 3 and demonstrate as well the effect of using our masking countermeasure on the entropy of the recovering codeword. Here the results show that using our helper data masking masking mechanism overall increases the entropy of the codeword. One can observe here that the entropy of codeword after using HDM Attack on decoders such as on SDML soft-decision, where the reliability is not 100%, is considerably high. Therefore the use-case of our masking mechanism, although it adds to the overall entropy, may not be justified for such cases.

On the other hand, the increase in entropy is relatively high using our helper data masking mechanism for cases such as Soft Decision Hackett at 100% reliability, Majority Decoding at both 100% and 85% reliability and GMC decoder at 100% reliability. We can assume here that masking helper data can in turn be accounted as an effective countermeasure against HDM Attack where the overall entropy of the codes are low. Given that so far we only considered maximum 2 block of the entire bits to be the mask values. In turn, if the overall overhead of computing large number of masking is justified (i.e., considering half of the bits as the stem value for key generation, and the other half for masking), we can count on the linear addition they provide on the overall entropy of the codewords against HDM Attacks.

V. EXPERIMENTAL SETUP

A. DESCRIPTION OF SRAM PUF DEVICE

The SRAM-PUF device we developed is an in-house device with a software package allowing us to interact with and control the device. FIGURE. 6 shows the device housing the micro-controller with our SRAM-PUF. Some of the functions implemented as part of our SRAM-PUF include a function for reading the real time PUF response after refreshing the SRAM every 2 seconds, and a function for enrolling the PUF, where it performs 100 read-out from the PUF source in the SRAM device and stores the responses. Our SRAM-PUF follows the addressable PUF generator (APG) technique, which is described in [31]. This allows us to extract unique responses from the PUF-device. The APG mechanism randomly selects memory cells to feed the PUF. For each memory address queried to the PUF device, there will be several response values coming from the memory cells. After an arbitrary amount of acquisition, the memory addresses and the captured responses are stored into a PUF dataset.

B. PUF DATA DESCRIPTION

We employed 10 SRAM-PUF devices for evaluation. For a given SRAM-PUF device, at the same memory address, we read the responses from each memory address 100 times. Noting that each memory address yields a 256-bit binary

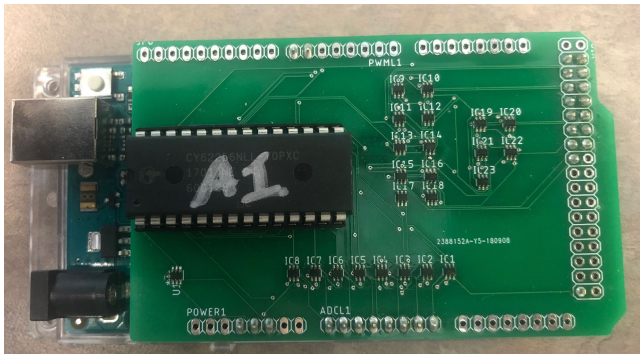


FIGURE 6. Our in-house developed SRAM device.

vector as the response. Ideally, the captured 100 response vectors should be exactly the same since they belong to the same address. However, due to the process variation between each response acquisition, there will be a chance that each cell's binary value would flip. Therefore, the PUF response vectors in-between different acquisitions have dissimilarities in some cell values. This implies the inherent physical characteristic of a real PUF.

Here we assess $\{16, 32, \ldots, 128,$ and $256\}$-bit of PUF responses. Noting a PUF response smaller than 256-bit, is a sub part of the 256-bit PUF response we captured initially from SRAM-PUF instances. For instance, a 16-bit PUF response is the first block of the captured 256-bit PUF response. Similarly, a 32-bit PUF response is the first two blocks of the captured 256-bit PUF response and so on.

In addition, we also measured the frequency of bit-flipping within each subset, to identify the unstable cells corresponding to each PUF images. Following Table. 4 at the end of the paper, it shows the percentage of cells with bit flipping per block of 16-bits within the entire 256-bits of a PUF image for a given memory address, per SRAM. Nothing that this statistical analysis is performed on the 100 captured PUF images from each SRAM for a given memroy address.

C. IMPLEMENTATION OF BECKER HDM ATTACK

For our experiments, we used MATLAB to build an emulation of Helper Data based RFE-like construction for PUF-based key generation, and Becker's HDM Attack as the adversary. For the encoding function in registration phase, we used Reed Muller encoding algorithm. For encoding, we use the specification of [16, 5, 8] (w.r.t the notation [n,k,t], where n is the size of the codeword, k is the size of the binary random number and t is the order of the code). Consequently, the encoded codewords are 16-bits sized. Noting that in this scheme, we use concatenated codes. Thus in an iterative fashion, to construct a helper data s, we XOR a 16-bit codeword x, where x is the encoded codeword of a randomly generated binary number rnd, over the entire PUF response w wherein the size of w should be greater than or equal to 16 bits. During the registration phase, we also generate a verification hash value h and an encryption key r using the two hash functions $H1$ and $H2$, respectively. To create the hash values, we used an existing MATLAB library to generate MD5 128-bits hash values. All the generated data will then be saved for a given SRAM device, for a given memory address. In implementing the recovery phase, we used Reed Muller Majority Logic Vote. We acknowledge that MD5 is the most secure hashing algorithm, and we used the algorithm only due to the ease of availability to carry out our evaluation. However, for a sophisticated implementation of the key generation for real applications, we suggest using more secure hashing algorithms such as SHA-2.

Our MATLAB-based implementation of Becker's HDM Attack is the following. The adversary first intervenes in the query of PUF-enabled device, by receiving the query from the device and passing it to the server, and in return, awaits the response from the server, instead of the device (i.e adversary as a man in the middle). After receiving the public data, using the error vector $e = [0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0]$, the adversary builds first his helper data s_a by XORing the error vector e with all the sub blocks of the helper data s which the adversary has received from the server. With respect to the reduced list of codewords known to the adversary, the adversary will create a pool of predictable responses w_a_pool and accordingly, a pool of predictable verification hash values h_a_pool. These lists comprise the items that the adversary will use after each failed attempt. Recalling that a failed attempt is an attempt in which the key generation is not decided as valid on the PUF-enabled device during recovery mode. In respond to the query from the PUF-enabled device, the adversary answers returns a tuple comprising an address value $Addr$ the same as the one received from the server. Also the modified helper data s_a and an elected hashed value. The adversary then awaits the response from the PUF-enabled device w.r.t whether the key generation is valid. If it is valid, it means that the adversary has successfully re-generated the encryption key as r_a. If the adversary receives a key generation invalid response, then the adversary will send another tuple with a different hash value.

We repeat our experiment independently for each PUF response size, which leads to 5 iterations of performing HDM Attack per SRAM-PUF device. Finally, noting that per pair of SRAM dataset and PUF response size, our experiment is iterated 32 times w.r.t to our attempt to exhaustively enlarge our experimental space by iterating the experiment each time using one of the possible codewords during PUF registration and recovery. Noting the 32 times iteration is w.r.t the total number of possible values for a 5-bit binary value. Recalling that this binary value is the value which is encoded during the registration phase to produce a codeword x.

Noting also that the assessment in our experimental results is based on a metric we define as the success-rate. Here, the success-rate is defined as number of successful HDM Attack attempts over the total number of attempts. We also define the best-case success-rate as the least number of successful attempts out of total number of attempts. The worst case success-rate is the highest number of successful attacks out

TABLE 4. Percentage of unstable cells, per block of 16-bits, in a PUF response size of 256 bits per SRAM.

	SRAM1	SRAM2	SRAM3	SRAM4	SRAM5	SRAM6	SRAM7	SRAM8	SRAM9	SRAM10
Block1	18.75%	12.50%	18.75%	12.50%	18.75%	12.50%	6.25%	12.50%	12.50%	12.50%
Block2	18.75%	6.25%	18.75%	25.00%	18.75%	12.50%	25.00%	6.25%	18.75%	18.75%
Block3	12.50%	18.75%	6.25%	6.25%	12.50%	12.50%	37.50%	37.50%	6.25%	6.25%
Block4	18.75%	12.50%	18.75%	18.75%	25.00%	18.75%	31.25%	6.25%	18.75%	18.75%
Block5	12.50%	25.00%	12.50%	18.75%	12.50%	0%	31.25%	25.00%	6.25%	6.25%
Block6	0%	18.75%	6.25%	6.25%	31.25%	12.50%	0%	0%	6.25%	6.25%
Block7	12.50%	18.75%	25.00%	12.50%	18.75%	25.00%	6.25%	25.00%	25.00%	25.00%
Block8	18.75%	18.75%	6.25%	18.75%	25.00%	6.25%	18.75%	37.50%	6.25%	6.25%
Block9	25.00%	6.25%	18.75%	25.00%	25.00%	18.75%	31.25%	25.00%	6.25%	6.25%
Block10	0%	12.50%	12.50%	18.75%	18.75%	0%	6.25%	12.50%	12.50%	12.50%
Block11	25.00%	12.50%	31.25%	37.50%	18.75%	25.00%	12.50%	6.25%	12.50%	12.50%
Block12	12.50%	25.00%	12.50%	18.75%	12.50%	18.75%	12.50%	12.50%	0%	0%
Block13	0%	12.50%	18.75%	18.75%	18.75%	6.25%	18.75%	25.00%	0%	0%
Block14	31.25%	25.00%	18.75%	6.25%	6.25%	12.50%	12.50%	12.50%	6.25%	6.25%
Block15	25.00%	18.75%	6.25%	18.75%	18.75%	18.75%	18.75%	31.25%	12.50%	12.50%
Block16	18.75%	18.75%	18.75%	18.75%	18.75%	37.50%	12.50%	25.00%	0%	0%

TABLE 5. Definition of virtual devices.

Virtual device	SRAM for key Generation	SRAM for mask vector generation
Vdevice1	SRAM1	SRAM2
Vdevice2	SRAM3	SRAM4
Vdevice3	SRAM5	SRAM6
Vdevice4	SRAM7	SRAM8
Vdevice5	SRAM9	SRAM10

of total number of attempts. Finally, the average success-rate is the average of the success-rate of all the cases of key generation w.r.t all possible different codewords for a given SRAM-PUF device dataset, and a given PUF response size.

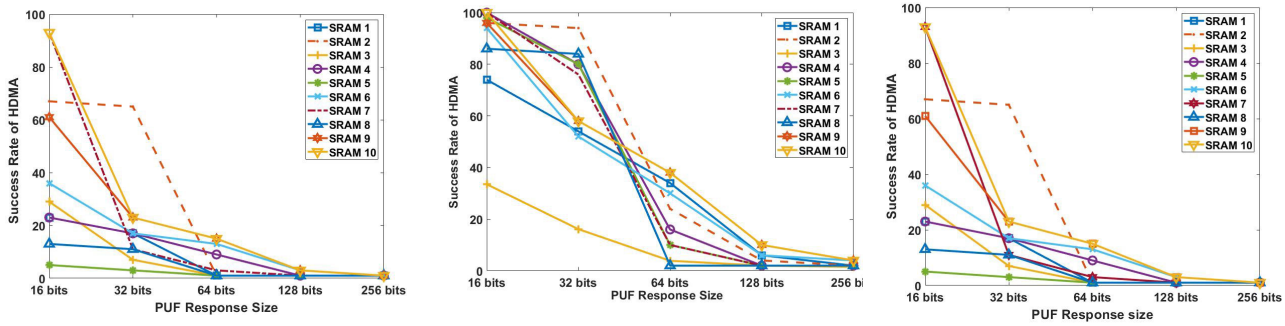
VI. EXPERIMENTAL RESULTS

In this section, the results of performing HDMA against our SRAM-PUF instances datasets will be discussed. FIGURE. 7 shows the best case, the worst case and the average success-rate of HDMA, respectively. We clarify first that the difference in the attack success-rate between the best case and the worst case in plots (a) and (b) in FIGURE. 7 is due to the difference in the value of the codeword used per SRAM-PUF dataset per given PUF response. Recalling that the possible number of codewords are 32, due to using a 5-bit binary random number for encoding. We can see that HDM Attack against the employed Reed Muller code and the majority logic vote decoder has some significant chance of success if the PUF response is small. Although we can see that at 128-bits size, there still is a chance of success, even in the best case scenario. At 256-bits obviously due to the doubled entropy of the guess field, no success was observed in the HDM Attack at 100 attempts per device.

We now observe the success-rate of HDM Attack against FE-based key encryption with helper data masking with variable positioning. To define one source for key value and one for mask value generation, we define here 5 virtual devices wherein each device employs 2 SRAMs. Thus, each virtual device is defined to comprise 2 SRAM datasets according to Table 5. Accordingly, there is a primary SRAM dataset that is the source of key generation and the secondary SRAM dataset for mask vector generation. We note also that we employed the variable positioning step size of 16-bits which is equal to the size of a codeword in this work.

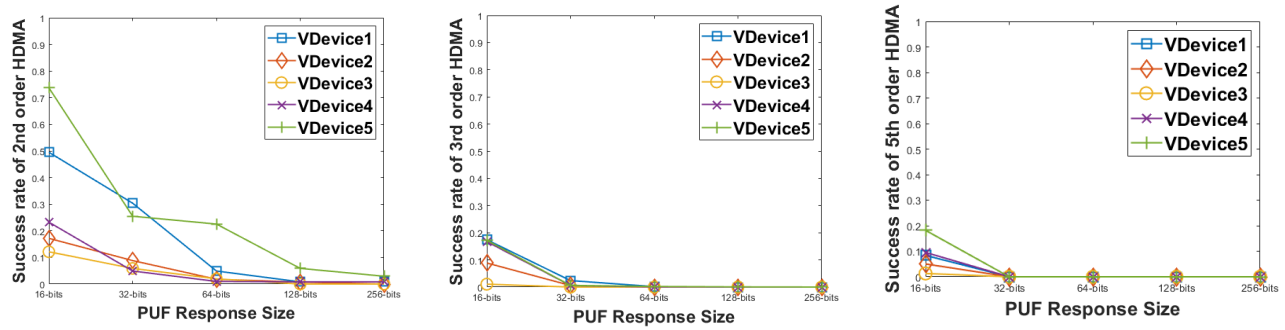
This experiment is performed in three settings wherein each setting, different sizes of mask vector is considered. FIGURE. 8 shows the results of this experiment in settings of using 1-block mask vector, 2-block mask vector, and 4-block mask vector. The plots shown in the figure represent the decreasing success-rate of second order HDM Attack against 5 virtual devices as the size of PUF response for key generation increases. Noting that for the assessment using 1-block mask vector, the datasets of PUF responses for key generation are augmented from 100 responses per dataset to 10000 responses. This augmentation is performed by simply repeating each response 100 times. For the assessment using 2-block mask vector and 4-block mask vector also, the datasets are augmented to 100000 responses per dataset using the same augmentation method. Recalling also that the responses per dataset correspond to one address of the SRAM-PUF and the differentiation between them is due to the instability of the SRAM-PUF.

First, we point out to the case where the success-rate of the attack against a PUF response size of 16 bits is still high regardless of the masking. We expect it since for a 16 bits of helper data, there is only one addressable block to mask. Therefore, there is no entropy in the guessing field for the address space on the adversary's side. Other than that, we can see that using masking with variable positioning significantly decreases the success-rate of second order HDM Attack while the PUF response is larger than the mask block. Unlike the key generation without masking, the success-rate drops drastically at even smaller w_1 sizes such as 32-bits and 64-bits while we employ more than 1 mask vector. This is similar to the demonstration of probability of success as we discussed in Section IV. Noting in the case of the



(a) Best case success-rate (b) Worst case success-rate (c) Average success-rate

FIGURE 7. Success-rate of Becker's HDMA out of 100 attempts on 10 different SRAMs.



(a) 1 × 16-bits mask value (b) 2 × 16-bits mask value (c) 4 × 16-bits mask value

FIGURE 8. Success-rate of HDM Attack with a) 10000 attempts. b) 100000 attempts. c) 100000 attempts.

settings using 2-block and 4-block mask vector, the success-rate has been measured from 100000 attempts of second order and fourth order HDMA, of which the results are shown in FIGURE. 8(b) and FIGURE. 8(c), respectively.

VII. RELATED WORKS AND COMPARISON

Our proposed helper data masking is not the first work to provide a countermeasure against HDM attacks. Several countermeasures have already been proposed in [19]–[21]. Here we discuss some of the existing countermeasures and compare them qualitatively with our method.

The work of Delvaux *et al.* [20] explains four schemes that secure a Helper Data Algorithm (HDA)-based key generation process. The primary HDM attack detection method is originally proposed in [32]. The method suggests to provide a hash value using collectively the helper data and the key value on the server side, and publicize it for devices to query during key generation. During key generation, the device queries the server for the hash value and compares it with a locally generated version of the hash. If equal, the generated key is considered valid and the key can be used for encryption and decryption of the exchanged message. We have shown a similar key generation structure as shown in FIGURE. 1 which is similar to the RFE-like process discussed in [18]. The problem with this scheme is that an HDM attacker can impersonate the server and provide the hash value itself, which is based on the manipulated helper data and the guessed key value. Therefore the proposed countermeasure can be easily compromised. Considering our countermeasure however, we showed that the demasking process cannot be compromised by any modification since the position of the mask over the helper data vector is based on the true value of the mask.

Similar to [32], another HDM detection scheme has been proposed [33], in which only the key value is used to generate the hash. However, the same security issue as we explained above can be discussed here. The HDM detection can be biased just by the attacker impersonating the server and providing its own hash value for key validity check on the PUF-enabled device.

Another solution has been discussed in [20] where it is suggested to program the public key and the helper data all into a one-time programmable NVM memory (EEPROM) during the enrollment phase of the PUF-enabled device [34]. During the key regeneration then, the IC housing or working with a PUF component for key generation, queries the NVM memory to retrieve the helper data for key generation. This solution ultimately negates possibility of HDM attack as we discussed in this work, since there is no out of device communication with a server system to query for helper

data. One can assume however that device can undergo some physical attacks using side channel analysis methods to read the information on the NVM device, and also using fault injection attacks to change some bits in order to impose helper data manipulation. These attacks however need precise tools to perform such modifications from outside of the device. This means that the attackers are enforced to employ some expensive methods. Nonetheless, the countermeasure itself is an expensive implementation, since it requires an external device to store the public key and the helper data. Moreover, using a memory component accompanying a PUF is fundamentally questioning the employment of PUF itself. Since the existence of PUF is set to replace conventional methods which save the secret values on device memories in order to provide protection against physical attacks.

A countermeasure against HDM attack is proposed by Hiller *et al.* [35]and discussed also in [20]. The problem discussed in their work is the practice of HDM attacks that focus on single codeword modification by observing several key generation operation outputs. The authors then propose a scheme where a hash value of the helper data is XORed with the recovered PUF response to generate the key. This way, the authors ensure that at list 50% of the key values are changed and not a single one, which in turn disturbs the entire attack mechanism. The issue with this countermeasure is that it is targeted for attack methods which aim to recover the original key. While there exists other HDM attacks such as the one discussed by Becker [18], for which this countermeasure is equally vulnerable to that of the RFE-like method we discussed in this work.

As we explained earlier, the mechanism of robust fuzzy extractors (RFE) can also provide some level of security. This has been discussed already in [18] and [32]. In that mechanism, despite the equality check between a localy generated and a publicly received hash value on the PUF device, the hamming distance between the recovered PUF response and the raw extracted PUF response is measured as well. If the distance is larger than a threshold, then the key generation fails. This suggests that only recovered codes with certain Hamming distance from the original ones are valid when recovered. However, such mechanism can also discard regenerated keys which have not been affected by an HDM attack. Simply due to PUF instability, there could be more noisy bits than what the RFE's Hamming distance threshold allows for a valid recovered codeword. Therefore, this method, although providing security against HDM attacks, also imposes extensive sensitivity to PUF instability.

Gao *et al.* [19] proposes a PUF-based key generation technique and assures its security using BCH codes and syndrome decoding. Using BCH codes and syndrome decoding has good level of security against HDM Attacks as discussed in [18]. However, it is also discussed that for $[n, k, 2t + 1]$ BCH codes with small k, it is not as efficient to use syndrome decoding compared to other simpler decoders such as maximum likelihood decoding.

Merli *et al.* [21] proposes a codeword masking scheme against helper data manipulation attacks based differential power analysis (DPA). The DPA HDM attack in their work aims to read the processed output of an inner decoder which decodes the noisy codeword that is the product of the XOR of the noisy PUF response and the received helper data. The codeword masking countermeasure in turn is implemented in a way to obfuscate the output of the inner code, so that DPA is not capable of reading the raw output of the first stage decoder. In order to mask the inner code, a random locally generated mask value is first encoded and the encoded mask value is XORed with the helper data, which is then XORed with the noisy PUF response. The output is then decoded and then demasked using the raw mask value to yield the secret key value. The countermeasure proposed here is the closest to our work. The similarity is that by masking the inner code using a random value, the DPA cannot find the correlation between the captured power traces. Theoretically, when masking is applied to secure an implementation, the DPA will require considerably more traces in order to bypass the masking and find the correlation to extract the target value. This is similar to increasing the guessing entropy of the HDM attack as we discussed in this work. The increased entropy in turn requires more observations on the data transmitting channel to discover the correct guess for the location of the mask on the helper data.

Most of the countermeasures discussed here were methodical and aimed at preventing the HDM attack using hardware or algorithmic solutions. Few, such as [21] suggested methods focus on the theoretical aspect of HDM attack and provide solutions that make the attacks more difficult to succeed. Our method also sits in this class of countermeasures. Since we also try in general to increase the guess entropy of the attacker using the proposed helper data masking method.

VIII. CONCLUSION AND FUTURE PERSPECTIVE

In this work, we discussed a new masking mechanism to protect publicly available helper data in FE-based key generation protocols. Here we proposed masking helper data with variable positioning. We also proposed PUF as the source of mask value generation. Our theoretical analysis showed that helper data masking with variable positioning is a potential mechanism to drastically decrease the probability of success in attacks whose aim is to modify helper data for mutual key generation on a target device. We also performed experimental evaluations to prove the efficiency of our method over real data. Here we used real-time SRAM PUF data captured from PUF-enabled devices developed in-house. Our experimental results show that masking with variable positioning can practically decrease the probability of success of HDM Attack even in a large number of attempts. Our 32-bit mask value in turn could reduce drastically the success-rate of HDM Attack 20% to less than 5%. With a 64-bit mask vector also, we showed that we can render HDM Attacks unsuccessful even in a extensive number of attempts, where the attack

against key generation per given response has been attempted 100000 times.

In terms of the threat model, this work focused on the vulnerability of the FE-based key generation against HDM Attacks. However, other metrics are also important to evaluate the practicality of our masking countermeasure. For that we foresee assessing the performance of SRAM-PUF based key generation in the presence of masking with variable positioning. This measurement is required in fact to evaluate the overhead of the extra computation due to the positioning procedure. In this evaluation, we could assess whether the performance overhead due to the extra computation for positioning the mask is significant or tolerable. Comparison cases can also be made. For instance, for variable sizes of PUF responses for encryption key generation, we can analyze the trade-off between the performance overhead and the security against HDM Attack in different cases where a block of a PUF response is chosen for key generation or chosen for mask vector generation.

REFERENCES

[1] A. Shamsoshoara, A. Korenda, F. Afghah, and S. Zeadally, "A survey on physical unclonable function (PUF)-based security solutions for Internet of Things," *Comput. Netw.*, vol. 183, Dec. 2020, Art. no. 107593. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128620312275

[2] A. A. Pour, V. Beroulle, B. Cambou, J.-L. Danger, G. Di Natale, D. Hely, S. Guilley, and N. Karimi, "PUF enrollment and life cycle management: Solutions and perspectives for the test community," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2020, pp. 1–10.

[3] A. R. Korenda, F. Afghah, and B. Cambou, "A secret key generation scheme for Internet of Things using ternary-states ReRAM-based physical unclonable functions," in *Proc. 14th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jun. 2018, pp. 1261–1266.

[4] B. Halak, *Physically Unclonable Functions: From Basic Design Principles to Advanced Hardware Security Applications*. Springer, 2018.

[5] C. Böhm, M. Hofer, and W. Pribyl, "A microcontroller SRAM-PUF," in *Proc. 5th Int. Conf. Netw. Syst. Secur.*, Sep. 2011, pp. 269–273.

[6] C.-H. Chang and M. Potkonjak, *Secure System Design and Trustable Computing*. Springer, 2015.

[7] J. Lee, D.-W. Jee, and D. Jeon, "Power-up control techniques for reliable SRAM PUF," *IEICE Electron. Exp.*, vol. 16, no. 13, 2019, Art. no. 20190296.

[8] A. Hosey, M. T. Rahman, K. Xiao, D. Forte, and M. Tehranipoor, "Advanced analysis of cell stability for reliable SRAM PUFs," in *Proc. IEEE 23rd Asian Test Symp.*, Nov. 2014, pp. 348–353.

[9] P. P. Aung, K. Mashiko, N. B. Ismail, and O. C. Yee, "Evaluation of SRAM PUF characteristics and generation of stable bits for IoT security," in *Emerging Trends in Intelligent Computing and Informatics*, vol. 1073, F. Saeed, F. Mohammed, and N. Gazem, Eds. Cham, Switzerland: Springer, 2020, pp. 441–450.

[10] B. Chen, T. Ignatenko, F. M. J. Willems, R. Maes, E. van der Sluis, and G. Selimis, "A robust SRAM-PUF key generation scheme based on polar codes," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2017, pp. 1–6.

[11] K. Xiao, M. T. Rahman, D. Forte, Y. Huang, M. Su, and M. Tehranipoor, "Bit selection algorithm suitable for high-volume production of SRAM-PUF," in *Proc. IEEE Int. Symp. Hardware-Oriented Secur. Trust (HOST)*, May 2014, pp. 101–106.

[12] F. Armknecht, R. Maes, A.-R. Sadeghi, F.-X. Standaert, and C. Wachsmann, "A formalization of the security features of physical functions," in *Proc. IEEE Symp. Secur. Privacy*, May 2011, pp. 397–412. [Online]. Available: http://ieeexplore.ieee.org/document/5958042/

[13] R. Cramer, Y. Dodis, S. Fehr, C. Padró, and D. Wichs, "Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), N. Smart, Ed. Berlin, Germany: Springer, 2008, pp. 471–488.

[14] J. Delvaux and I. Verbauwhede, "Attacking PUF-based pattern matching key generators via helper data manipulation," in *Topics in Cryptology—CT-RSA* (Lecture Notes in Computer Science), J. Benaloh, Ed. Cham, Switzerland: Springer, 2014, pp. 106–131.

[15] J. Delvaux and I. Verbauwhede, "Key-recovery attacks on various RO PUF constructions via helper data manipulation," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, 2014, pp. 1–6.

[16] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede, "Helper data algorithms for PUF-based key generation: Overview and analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 6, pp. 889–902, Jun. 2015.

[17] V. van der Leest, B. Preneel, and E. van der Sluis, "Soft decision error correction for compact memory-based PUFs using a single enrollment," in *Cryptographic Hardware and Embedded Systems—CHES* (Lecture Notes in Computer Science), E. Prouff and P. Schaumont, Eds. Berlin, Germany: Springer, 2012, pp. 268–282.

[18] G. T. Becker, "Robust fuzzy extractors and helper data manipulation attacks revisited: Theory versus practice," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 5, pp. 783–795, Sep. 2019.

[19] Y. Gao, Y. Su, W. Yang, S. Chen, S. Nepal, and D. C. Ranasinghe, "Building secure SRAM PUF key generators on resource constrained devices," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2019, pp. 912–917.

[20] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede, "Helper data algorithms for PUF-based key generation: Overview and analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 6, pp. 889–902, Jun. 2015.

[21] D. Merli, F. Stumpf, and G. Sigl, "Protecting PUF error correction by codeword masking," Cryptol. ePrint Arch., Tech. Rep., 2013.

[22] R. Maes, P. Tuyls, and I. Verbauwhede, "Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs," in *Cryptographic Hardware and Embedded Systems—CHES* (Lecture Notes in Computer Science), C. Clavier and K. Gaj, Eds. Berlin, Germany: Springer, 2009, pp. 332–347.

[23] R. Maes, A. Van Herrewege, and I. Verbauwhede, "PUFKY: A fully functional PUF-based cryptographic key generator," in *Cryptographic Hardware and Embedded Systems—CHES* (Lecture Notes in Computer Science), E. Prouff and P. Schaumont, Eds. Berlin, Germany: Springer, 2012, pp. 302–319.

[24] S. Puchinger, S. Mueelich, M. Bossert, M. Hiller, and G. Sigl, "On error correction for physical unclonable functions," in *Proc. 10th Int. ITG Conf. Syst., Commun. Coding (SCC)*, 2015, pp. 1–6.

[25] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient helper data key extractor on FPGAs," in *Cryptographic Hardware and Embedded Systems—CHES* (Lecture Notes in Computer Science), E. Oswald and P. Rohatgi, Eds. Berlin, Germany: Springer, 2008, pp. 181–197.

[26] V. V. D. Leest, B. Preneel, and E. V. D. Sluis, "Soft decision error correction for compact memory-based PUFs using a single enrollment," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* Berlin, Germany: Springer, 2012, pp. 268–282.

[27] R. Maes, P. Tuyls, and I. Verbauwhede, "A soft decision helper data algorithm for SRAM PUFs," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun. 2009, pp. 2101–2105.

[28] S. Puchinger, S. Müelich, M. Bossert, M. Hiller, and G. Sigl, "On error correction for physical unclonable functions," in *Proc. 10th Int. ITG Conf. Syst., Commun. Coding (SCC)*, 2015, pp. 1–6.

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

[30] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient helper data key extractor on FPGAs," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* Berlin, Germany: Springer, 2008, pp. 181–197.

[31] A. R. Korenda, F. Afghah, B. Cambou, and C. Philabaum, "A proof of concept SRAM-based physically unclonable function (PUF) key generation mechanism for IoT devices," in *Proc. 16th Annu. IEEE Int. Conf. Sens., Commun., Netw. (SECON)*, Jun. 2019, pp. 1–8.
[32] X. Boyen, Y. Dodis, J. Katz, R. Ostrovsky, and A. Smith, "Secure remote authentication using biometric data," in *Advances in Cryptology—EUROCRYPT*, vol. 3494, R. Cramer, Ed. Berlin, Germany: Springer, 2005, pp. 147–163.
[33] Z. Paral and S. Devadas, "Reliable and efficient PUF-based key generation using pattern matching," in *Proc. IEEE Int. Symp. Hardware-Oriented Secur. Trust*, Jun. 2011, pp. 128–133.
[34] P. Tuyls, G.-J. Schrijen, B. Škorić, J. V. Geloven, N. Verhaegh, and R. Wolters, "Read-proof hardware from protective coatings," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* Berlin, Germany: Springer, 2006, pp. 369–383.
[35] M. Hiller, M. Weiner, L. Rodrigues Lima, M. Birkner, and G. Sigl, "Breaking through fixed PUF block limitations with differential sequence coding and convolutional codes," in *Proc. 3rd Int. Workshop Trustworthy Embedded Devices*, 2013, pp. 43–54.

AMIR ALI POUR is currently a visiting Ph.D. student at Clemson University. He received the B.S. degree in software engineering from Azad University of Tehran South Branch, Tehran, Iran, and the M.Eng. degree from Grenoble INP, Valence, France, in 2016 and 2019, respectively. He also was a Master student at Iran University of Science and Technology, Tehran, Iran, for two years since 2016. He is currently a Ph.D. student at Université Grenoble Alpes (UGA), Grenoble, France. He was also a student co-lead of CSAW Europe Applied Research Competition event for the years 2020 and 2021, respectively. His research interests are machine learning and cryptography.

FATEMEH AFGHAH (Senior Member, IEEE) is currently an Associate Professor with the Department of Electrical and Computer Engineering, Clemson University, where she is also the Director of the Intelligent Systems and Wireless Networking (IS-WiN) Laboratory. Prior to joining Clemson, she was an Associate Professor with the School of Informatics, Computing and Cyber Systems, Northern Arizona University (NAU), Flagstaff, AZ, USA, from 2015 to 2021. She has coauthored more than 100 technical papers. Her research interests include wireless communication networks, decision making in multi-agent systems, radio spectrum management, hardware-based security, and artificial intelligence in healthcare. She was a recipient of several awards, including the NSF CRII Award, in 2017; the Air Force Office of Scientific Research Young Investigator Award, in 2019; and the National Science Foundation (NSF) CAREER Award, in 2020. She serves as an Editor for several journals, including *Ad Hoc Networks* journals and *Computer Networks* journal.

DAVID HÉLY received the master's degree from the National Institute of Applied Sciences of Lyon, in 2002, and the Ph.D. degree from the University of Montpellier 2, in 2005, with a focus on the design for testability of secure IC in collaboration with STMicroelectronics and the LIRMM Laboratory. From 2005 to 2009, he held several positions in research and development with STMicroelectronics and then Sagem Défense et Sécurité, where he was focused on the architecture, design, and certification of safe and secure system on chip. Since 2009, he has been an Associate Professor with the Grenoble Institute of Technology and is currently a member of the LCIS Laboratory. His research interests include design and test of safe and secure embedded systems.

VINCENT BEROULLE received the M.S. and Ph.D. degrees in microelectronics from the University of Science of Montpellier, France, in 1999 and 2002, respectively. In 2002, he joined the Grenoble Institute of Technology (Grenoble INP) as an Assistant Professor and became a Full Professor, in 2019. He is currently the Director of the LCIS Laboratory, Valence. His main research interests include security and robustness of integrated systems.

GIORGIO DI NATALE (Senior Member, IEEE) received the Ph.D. degree in computer engineering from the Politecnico di Torino, in 2003. He currently works as the Director of research for the French National Research Center (CNRS) and has been the Director of the TIMA Laboratory, Grenoble, since January 2021. His research interests include hardware security and trust, secure circuits design and test, reliability evaluation and fault tolerance, software implemented hardware fault tolerance, and VLSI testing. He is a Golden Core Member of the Computer Society. He has been serving as the Chair for the IEEE Computer Society TTTC, since 2020.

ASHWIJA REDDY KORENDA received the B.Sc. degree in electronic and communication engineering from Jawaharlal Technological University (JNTU), Hyderabad, India, in 2015, and the M.Sc. degree in electrical engineering from Northern Arizona University (NAU), Flagstaff, USA, in 2018, where she is currently pursuing the Ph.D. degree with the School of Informatics, Computing and Cyber Systems. Her main research interests include security, physically unclonable functions, error correction coding, and cognitive radios.

BERTRAND CAMBOU (Member, IEEE) received the Ph.D. degree in electronics from the University of Paris-Saclay. He worked in the smart card/secure microcontroller industry at Gemplus (now Gemalto) and in the POS/secure payment industry at Ingenico. He spent 15 years at Motorola Semiconductor (now NXP-Freescale), where he worked in multiple capacities, including a CTO and was named as a Distinguished Innovator and a Scientific Advisor of BOD. He is currently a Professor at Northern Arizona University (NAU). He has 71 granted patents. His primary research interests include cyber-security and how to apply microelectronics to strengthen hardware security. This includes the design of novel secure elements, physically unclonable functions (PUF), true random generators (TRNG), and the usage of nanotechnologies, such as ReRAM.