Reliable, Secure, and Efficient Hardware Implementation of Password Manager System Using Physical Unclonable Functions

MOHAMMAD MOHAMMADINODOUSHAN1, BERTRAND CAMBOU1, FATEMEH AFGHAH2, CHRISTOPHER ROBERT PHILABAUM1, AND IAN BURKE1
1School of Informatics, Computing, and Cyber System, Northern Arizona University, Flagstaff, AZ, USA
2Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, USA
Corresponding author: Mohammad Mohammadinodoushan (e-mail: mm3845@nau.edu).
This material is based upon the work supported by the National Science Foundation under Grant No. 1827753.

ABSTRACT Using Physical Unclonable Functions (PUFs) within the server-side has been recently proposed to address security vulnerabilities of the password (PW) authentication mechanism, including attacks on the database (DB) of user credentials. Practicing this idea using different available memory technologies and resource-constrained hardware modules may offer an additional hardware security layer. Therefore, finding the PWs would require the attacker to access both the hardware containing the PUF and the information stored in the DB. PUFs have been used with other cryptographic algorithms in previous studies to improve the system's security further. However, these studies have overlooked the challenges of implementing these cryptographic algorithms with limited resources of PUF hardware devices. Therefore, the trade-off between the achieved security and desired efficiency is still a challenge. The presented hardware-software PUF-based solution using embedded SRAM memory leads to faster computation in the hardware used on the server-side. Also, the protocol used on the client-side can cope with the resource limitations existing in essential applications, including low-power and memory IoT devices. Moreover, the scheme handles both the instability and bit alias of the Static Random-Access Memory (SRAM) PUF. This work presents a reliable, low-cost, and efficient prototype showing the functionality of a hardware-dependent protocol that is resistant to insider, PW guessing, and man-in-the-middle attacks. The presented hardware and software can be easily integrated with authentication servers. In the protocol used in this work, the PUF creates both DB addresses and contents. Statistical tests on the applied commercial SRAM in this article show that the protocol used in this paper reaches a better entropy in PUF responses stored in the DB. Besides, the experimental results of this work show the possibility of obtaining an SRAM PUF with very low intra-PUF variation without using any extra hardware overhead.

INDEX TERMS Password security, PUF-based security, database hacking, authentication, hash chain.

I. INTRODUCTION Authentication systems store user information in Lookup tables (LUTs) or databases (DBs). This data generally includes user identification such as username \( ID \) (\( ID \), hereafter) and associated authentication credentials such as passwords (PWs). Although using PWs is the most common way of authentication, the PW authentication mechanism suffers multiple vulnerabilities. One of the most commonly reported cyber-attacks is the hacking of the PW DBs [1, 2]. In this respect, several studies have shown that many DBs store the PWs in plaintext form [3, 4]. As a result, the PW of users would be revealed if the DB is accessed by a hacker or other unauthorized third parties. By hacking the DB of one single server, hackers can often cause substantial damages to the users, commercial vendors, and government organizations [1, 2, 5]. Other methods that have been used in current PW management (PWM) systems are hashing or encrypting the PWs, although they are not secure enough. For example, since many hashing functions are well-known, an attacker may use dictionary PWs, input that PW into a hashing function, and find the resulting Message Digest (MD) in the compromised DB.
A. PW Security
Using PWs involves several threats such as brute-force attacks, sophisticated PW guessing attacks, insiders’ access to DB of ID/PW, man-in-the-middle (MIM) attacks, and distributed denial-of-service attacks.

1) PW GUESSING
As many hashing functions are well-known, an attacker may guess a PW using a set of PWs, especially if the attacker knows user habits. In this way, the hacker puts the guessed PW into a hashing function and finds the output in the compromised table. Many studies have addressed this type of attack by increasing the resources needed to match a PW. The entropy of the PWs selected by users is typically low. Additional use of salt was proposed in the late 1970s to increase the entropy and the work required for cracking the PWs [6]. In addition, increasing the number of times the algorithms are run increases the resources needed to match a PW. A noteworthy drawback of these methods is that they are restricted by a maximum threshold that should not be passed; otherwise, authentication latency will be increased.

2) BRUTE-FORCE ATTACKS
In a brute-force attack on PWs, the attacker performs the exhaustive search on the PWs by testing the hash of each string combination for a chosen character set and string length. The calculated hashes are compared with MDs until a match is found or the attack is finished. Using salts is an effective strategy to prevent the use of rainbow tables. Nevertheless, brute-force attacks remain as fast as with unsalted hash functions. Brute forces only guarantee to find the PW, not the feasibility in terms of time required. As human-chosen PWs follow easy-to-guess patterns, a guessing and dictionary attack is the attack that is considered while designing a PW authentication scheme [7].

3) MIM ATTACKS
In this type of attack, the attacker has access to the communication channel between two endpoints and can eavesdrop, insert, modify, or simply replay their messages.

B. Threat Model and Contributions
This paper uses Physical Unclonable Functions (PUFs) on the server-side, unlike any existing PUF-based schemes. Thus, MDs are used as PUF inputs; instead of storing the MDs, the PUF responses are stored. This work aims to present a reliable, low-cost, and efficient prototype showing the functionality of a hardware-dependent protocol that is resistant to an insider, PW guessing, and MIM attacks. Other threats such as phishing, shoulder-surfing, PW reuse, and others can still be used to risk the system’s security. Our scheme does not address these issues but can be used with other remedies to minimize these risks. Studying PUF attacks such as fault injection is out of the scope of this work, and the interested readers are referred to [8].

1) GUESSING ATTACKS
Our scheme ensures that authentication data remains secure even if that data is accessed or stolen. If an attacker obtains access to the authentication data, attempting to “guess” PWs is computationally difficult because the system stores neither PWs nor the hash of the PWs. Because each PUF device is unique, the only way to identify valid PWs by guessing (other than a brute force attempt using every possible PW) would require the attacker to access both the DB and the hardware containing the PUF.

2) MIM ATTACKS
In the protocol used in this paper, at each authentication request, the users hash their PW multiple times but at levels lower than those used previously. Hence, the client uses different authentication credentials for each authentication request which mitigates the MIM attack (see Figure 1). Two types of protocol are implemented in this paper to trade-off between security and efficiency.

3) SIDE-CHANNEL ATTACKS
This article chooses Static Random-Access Memory (SRAM) as a PUF since it is a widely accessible memory technology in various devices. The SRAM PUFs are susceptible to cloning attacks using side channels, which compromises the PWs of the users. However, the computation and time involved to implement side-channel attacks for large entropy PUFs such as the SRAM PUF used in this work make these types of attacks hard in practice.

4) MICROCONTROLLER IMPLEMENTATION
The idea of using PUFs as additional hardware (HW) security layer was patented by the authors of this paper [9]. This idea is combined with other cryptographic algorithms to improve the system’s security in previous studies, which will be reviewed in section II. However, these previously reported works have disregarded the challenges of implementing these cryptographic algorithms with limited resources of PUF HW devices. The hardware-software solution proposed in this paper aims to improve mainly the system’s latency and then the entropy and randomness of the streams generated by SRAM PUF.

The remainder of this paper is organized as follows. Section II provides background information and a literature review. Details of the design and protocol is provided in Section III. Section IV focuses on implementation and prototype description. Section V shows the results of SRAM PUF characterization and step-by-step results for all protocol steps. Section VI discusses the reliability, security, and efficiency of the system. Eventually, section VII describes the conclusions and future works.

II. RELATED WORK AND BACKGROUND
PW manager (PWM) systems employ several techniques to help users select stronger PWs and manage them [10]. However, regarding the difficulty in remembering the PWs and having too many PWs, users apply insecure mechanisms
such as selecting PWs that are memorable but easy for hackers to guess and reusing those PW across multiple accounts [11, 12].

The most reliable methods used by current PWM systems regarding dictionary attacks are salted hashing or encrypting the user information [13, 14]. In this way, hackers cannot quickly determine the users’ credentials by obtaining unauthorized access to data stored in the DB. Despite such advantages, these PWM systems suffer from several security issues. For instance, if attackers access a DB storing the user information, they can apply various computational methods to decrypt or decode it. The main reason for this issue is that the methods used by current PWM systems are based on known and public algorithms. In [15], dictionary attacks on relatively strong PWs were completed in a relatively short time, even when using more additional sophisticated algorithms. Furthermore, computing devices are getting faster, allowing these attacks to get even more effortless in the short future.

PUFs are fingerprints of the HW devices. PUFs have been used widely in various security applications such as authentication, key generation, and IP protection based on the Challenge-Response Pair (CRP) approach [16-18]. Ideally, physically reproducing the entire PUF is unlikely due to random variations in manufacturing. However, research has shown that many current PUFs are vulnerable to machine learning attacks [19].

Many forms of PUFs designed in previous studies are reviewed in [20-23]. One type of PUFs is memory PUFs, which can be made from available memory technologies such as Flash [24], Magnetic Random-Access Memory (MRAM) [25], and SRAM. SRAM PUFs were discovered independently and concurrently in [26] and [27]. A typical cell of SRAM contains two cross-coupled inverters with two stable states. Manufacturing variability always leads to a physical disparity between two symmetrical halves of the SRAM circuit. This random physical mismatch is controlled by the power-up behavior. During different power-up cycles, responses of the SRAM PUFs are the same for most of the SRAM cells. Nevertheless, few SRAM cells have a weak preference or no particular preference [28], called fuzzy cells in this article.

In [29], PUFs are used on the client-side to strengthen the PWs. [29] uses both software and HW to generate PWs for users. The key idea in [29] is to leverage the uniqueness and uniformity of PUF on the client-side to strengthen PWs and prevent attacks. Unlike hashing, which has a deterministic output for a given input, PUFs provide different outputs for the same input. The scheme presented in [29] can be implemented without any need for the user being aware of it. However, the designed scheme reduces the transportability since the same ID and PW would not work on other device due to uniqueness of PUF.

The authors of this paper have recently proposed applying PUFs in the server-side of PWM systems [9, 30]. The idea behind these works is that instead of storing the PWs or Message Digests (MDs) produced by the hash functions, the MDs are fed to a PUF as a challenge that refers to a cell address of the memory-based PUF. Then, the first measurement of a PUF parameter, defined as PUF original response, is stored in the DB. As a result, getting access to the authentication data and guessing PWs by an attacker becomes computationally difficult because the system stores neither the PWs nor hashes of PWs. Because of the uniqueness of each PUF, the only way to identify a valid PW by guessing is when the attacker has access to both the data stored in DB and the HW containing the PUF. Thus, by taking advantage of PUFs, the system can ensure authentication data remains secure even if that data is stolen.

The structure of the PWM system in [9] is based on using an Addressable PUF Generator (APG). As explained in [9], the APG can include a memory PUF and a microcontroller (MCU) that handles computation tasks, including hashing. APGs can be used in two modes. First, the challenge specifies the cell addresses in the memory and the cell readings as the original response are stored in the enrolment phase. In authentication mode, the APG generates fresh responses from the same addresses of memory PUF, which differ from the original response due to environmental changes and aging effects.

The idea proposed in [9] is first implemented in [31] by the authors of this paper, using commercial SRAMs. Because the PUF is secured in the server, an attacker cannot simply read the entire PUF array and use the information to uncover the saved PWs in the DB. Nonetheless, the method in [31] is vulnerable to MIM attacks, where the PWs can be disclosed by watching the same network flow several times.

Furthermore, in the protocol used in [31], the PUF is just used to generate the PUF responses stored as content in the DB based on PW. Let us consider a case in which the hacker accesses the information in the DB. Here, the hacker can test various common PWs, XOR them with the user’s ID, hash the result, and check to determine if it matches the same address in the DB. Thus, the hacker can decode DB information with more effort, even without controlling the PUF. This challenge was dealt with by using the PUF to extract not only the DB content but also the DB address. In a scheme that only the content of what is stored in the DB address incorporates the patterns generated by the PUFs, small error rates in PUF responses due to environmental conditions are acceptable. In this paper, security is improved using the addressing system extracted from the PUF. However, an error in the address would send the search engine to the wrong part of the DB, which needs matching algorithms used in [32]. The matching algorithms could be computationally challenging to handle, especially they add latencies when using APG with resource-constrained HW. In this paper, because of using ternary PUFs with extremely low error rates to solve this issue, the matching algorithms used in [32] are not required.
Lamport [33] applied One-way Hash Chains (HCs) as a cryptographic primitive for the first time. HCs have low communication overheads and have some public-key cryptography features. Therefore, they have been used by the protocol designers in many security applications such as one-time PWs generation and IoT devices authentication [34]. The hash length finiteness is the primary concern with the design of an HC-based security protocol. On the one hand, if the HC’s length is too big, the memory-computation overload for generating each output will grow linearly with \( n \), where \( n \) is the length of the HC. On the other hand, a low value of \( n \) makes the HC exhaust quickly, and reinitializing is required.

The PWM systems protocol proposed in [35] uses hash chains that mitigate MIM attacks. In the registration step, the user hashes their PWs \( N \) times and sends encryption of \( H^n(PW) \) to the server. The server generates the “original response” from the PUF at the address extracted from \( H^n(PW) \). In the authentication step, the user hashes their PWs \( M \) times (\( M < N \)). As the first step to authenticate the user, the server reads the PUF at the address extracted from \( H^M(PW) \) and it compares it to the “original response” stored in the DB. The server authenticates a user by reiterating the previous step and increasing the hashing times one by one (\( M+1, M+2, ..., N-2, N-1 \)). Next, it reads the PUF at the corresponding address and compares it to the “original response” stored in the DB. During future authentication, the users again hash their PW multiple times but at levels lower than those used previously. Accordingly, the client uses different authentication credentials for each authentication request, thereby mitigating the MIM attack.

The protocol presented in [35] has several advantages, including the authentication of the users by the server without knowing their PWs. However, the server should repeat hashing and reading the PUF for multiple times (\( N \) time in the worst-case scenario). On the one hand, lengthy HCs (larger \( N \)) introduce increased memory-computation overhead on the APG connected to the server. As a result, the authentication time is prolonged, and the manageable number of clients is reduced. On the other hand, a low value of \( N \) makes the hash chain exhaust quickly, and thus reinitializing is required. Therefore, this method is not feasible for authentication scenarios in which at least one of the communication parties is a device with low power capabilities or even low memory capacities. More importantly, the protocol in [35] requires reading the PUF multiple times for each authenticating request. Consequently, the latency increases severely because the generation of responses from a PUF is slower than reading a LUT with ID s and PW s. For example, reading 128 bits from non-volatile memory such as flash memory typically takes 10 \( \mu s \), while generating a PUF response from a flash memory may take as much as 10 ms, which is 1000 times more [9].

In [35], a PC handles all tasks, and the utilized HW is just used to read the PUF. However, the idea proposed in [9] aims to have a customized HW (APG) that handles all protocol steps. Generally, MCUs used in APGs have low computation and low memory capabilities. Therefore, the protocol in [35] cannot be used due to its high computation burdens. In this paper, a new protocol is offered and implemented to deal with this issue. Also, an appropriate MCU is selected that supports the hash via the HW crypto engine. This HW is considerably different from the one used in [35]. Furthermore, the protocol used in this paper is resistant to MIM attacks because of using a random number generated by an HW crypto engine.

On another note, a block of SRAM cells next to each other was selected to extract each PUF output sequence in [35]. Nevertheless, the results in [32] reported existing patterns and bias for the homologous cells close to each other. Therefore, reading the block of cells used in [35] lowers the entropy of the output sequence extracted from the SRAM PUF. In this paper, just 1 bit is extracted each time, and the next bit is extracted at another random address coming from generated MD. In this way, the entropy in each PUF output sequence is improved. As a longer MD is needed to point to more addresses in the PUF, a block responsible for making the MD longer is used in the present study.

Another contribution of this paper compared to [35] is using a different method for handling the PUF error. In [35], an error threshold was defined to accept the noisy responses while the error rate between the original and fresh responses was less than 6%. In this work, the PUF characterization (enrollment) phase is added to the protocol. The protocol is revised to discard the unstable cells and reach an extremely low error rate. Hence, the latency of matching algorithms used in [32] for handling the addressing error is mitigated. In the implementation, the characterization results are stored on MCU flash.

III. METHODS AND PROTOCOL

In this section, the proposed architecture, including PUF characterization and the protocol, is discussed. TABLE I shows the notations used in this section.

| TABLE I NOTATIONS |
|--------------------|
| \( n \)           | Hash chain length     |
| \( L_i \)         | \( n+1 \) th link of the Hash Chain |
| \( Re_i \)        | PUF response stored as the DB content for user \( i \) during registration |
| \( Cdd_i \)       | PUF response as the DB address for user \( i \) during registration |
| \( Re_{ij} \)     | PUF response stored as the DB content for user \( i \) at \( j \) th positive authentication request |
| \( Cdd_{ij} \)    | PUF response as the DB address for user \( i \) at \( j \) th positive authentication request |
| \( XY1 \)         | Memory PUF cell addresses used for extracting DB content |
| \( XY2 \)         | Memory PUF cell addresses used for extracting DB address |
| \( \oplus \)      | Exclusive OR          |
| \( E(\cdot) \)    | Encryption            |
A predefined secret key on both sides for encryption or decryption of some part of the data, based on the type of protocol used (type 1 or type 2). Our primary focus is to emphasize the novelty of the protocol using hash chains and PUF, which results in resistance to MIM and PW guessing attacks. Our scheme does not use a standard key sharing protocol. In practice, a detailed protocol should be defined and used for key sharing between client and server.

The overall system architecture is shown in FIGURE 1. On the client-side, the users set up an HC from their PW. Also, they communicate different links of the HC in the reversed direction in a stepped manner for registration and at each authentication request. User credentials are supplied to known functions such as XOR or hashing function to create two different streams on the server-side. These streams are fed to “Block1”, which increases the entropy by shifting, hashing, and concatenating to create two MDs (MD1 and MD2). The details of “Block1” are presented in [31]. MD1 and MD2 produce “Raw” addresses that point to the PUF cells.

Raw addresses (Raw XY1 and Raw XY2) can include the address of both fuzzy and non-fuzzy cells. In the “Masking block” shown in FIGURE 1, raw addresses are revised to disregard the address of fuzzy cells. The details of the “Masking” block will be discussed later in this paper. The output of the “Masking” block is the “Revised” or “Refined” addresses. “Revised XY1” and “Revised XY2” are cell addresses fed to the PUF as challenges. PUF responses to “Revised XY1” and “Revised XY2” are the DB content and DB address, respectively.

A. PUF Characterization

SRAM PUFs are utilized in this study since they are readily accessible on many devices. The instability of SRAM PUFs for some cells can increase the false rejection rate (FRR) of the PWM system in this paper. Ternary SRAM PUFs are used in this paper to solve the problem. Ternary PUFs are based on three states, including a “fuzzy state” to denote the unstable cells.

As previously explained in section II, during different power-up cycles, responses of the SRAM PUFs are the same for most of the SRAM cells. However, few SRAM cells have a weak preference, called fuzzy cells. To manage fuzzy cells in the ternary PUF, the SRAM is characterized in advance. The enrollment objective is to define the unstable cells that should not be used due to their unstable behavior. During the characterization (enrollment) phase, the SRAM cells were subjected to successive power-off/power-on cycles. In each cycle, the cells with different responses against their previous cycle are represented by fuzzy cells. In this way, it is possible to recognize the cells that can produce stable ‘0’ and ‘1’ and remove those with the ‘X’ state. The higher number of reads is accompanied by more fuzzy cells, more stable responses, and lower error rates. The characterization (enrollment) result is called “mask data” in this paper.

The level of enrollment is selected to target an FRR below $10^{-4}$. Based on the characterization results discussed later in this paper, the level of enrollment is increased to 1000. This reduces the error rate to an acceptable range required for the addressing scheme used in this paper.

In this paper, the “Masking” block controls the error. The mask data information is saved in MCU flash memory, which indicates the location of fuzzy cells. If a fuzzy cell is selected in the “Masking” block, the address will be revised to the address of the following non-fuzzy cells.

B. Protocol

This subsection describes the protocol, including a single registration phase followed by one or more authentication phases.

5) REGISTRATION

Here, the operations done in the client-side and server-side for registration are discussed.

- Operation In the Client-Side

The $i$th client should setup the HC, by computing the values $L_1^i, \ldots, L_{n-1}^i, L_n^i$ as shown in FIGURE 2, through performing the following operations:

1) Chooses the value of $PW$.
2) Applies $Hash$ to $PW$ repeatedly $n$ times until it derives the value of $L_n^i$. 

![FIGURE 1. An overview of the architecture](image-url)
3) Sends encryption of $L_n^i$ and ID to the server.
-Operation in the Server-Side

FIGURE 3 presents how the server registers a new user for the first time. The steps of the registration of a new user are listed in the following.

-FIGURE 2. Hash Chain Setup in the Client-Side (Registration).

3) Sends encryption of $L_n^i$ and ID to the server.

5) Each bit of the $Re_n^i$ is extracted at “Revised XY1”

6) On the other path, $(ID \oplus L_n^i)$ is fed into "Block1". The output of "Block1" generates "Raw XY2".

7) If "Raw XY2" points to a fuzzy cell, the address is revised to the address of the next non-fuzzy cell in the “Masking” block. The output of the “Masking” block is named “Revised XY2”.

8) Each bit of the $Cdd_n^i$ is extracted in "Revised XY2".

9) $Re_n^i$ generated in step 5 is stored at $Cdd_n^i$.

6) AUTHENTICATION

In this subsection, the operations done on the client-side and server-side for authentication are discussed, respectively.

-Operation in the Client-Side

For each authentication request, the client must output the links of the HC in the reversed direction in a stepped manner (FIGURE 4). The client sends the value of $L_{n-j}^i$ at step $j$ (i.e., it sends the value of encryption of $H_{n-1}^i(\text{PW}_j)$ at the initial step and continues until sending encryption of $H_1^i(\text{PW}_j)$ at the final step)

-Operation in the Server-Side

The steps on the server-side for authentication are listed in the following. Steps 1-3 are shown in FIGURE 5, and steps 4-5 are shown in FIGURE 6.

1) APG receives the encrypted ID and the encrypted value of $L_{n-j}^i$ or $H_{n-j}^i$.

2) APG performs one HW hash and calculates $L_{n+1-j}^i$ or $H_{n+1-j}^i$.

3) APG generates a fresh PUF response and compares it with the PUF response previously stored in the DB. If these two do not match, the authentication is not approved. If these two match, the authentication is approved, and the server goes through steps 4 and 5.

4) As shown in FIGURE 6 by number 1, the server erases the pattern stored in the previous authentication request ($Re_{n+1-j}^i$).

5) As shown in FIGURE 6 by number 2, the server repeats the tasks discussed in registration but without hashing $L_{n-j}^i$. The server finds $Re_{n-j}^i$ and stores it in the address $Cdd_{n-j}^i$. This step reduces the computation burden on the server-side for the following authentication requests, improving the authentication latency.

-FIGURE 3. Operation in the Server-Side (Registration).

-FIGURE 4. Operation on the Client-Side (Authentication).
IV. IMPLEMENTATION

A. Prototype Description

The server is a laptop running MATLAB software. MATLAB provides a GUI to enter the user’s information and control APG data flow. All other tasks, including calculations and reading SRAM PUF, are carried on the APG HW. MCU (i.e., the central part of APG) handles most tasks, including hashing and encryption. The processing speed was increased using SAMV71 MCU to support the hash via an HW crypto engine. SAMV71 MCU with Cortex M7 core running at 300 MHz is the heart of the Xplained Ultra Evaluation Kit. This kit is used in the prototype to validate the protocol.

In this paper, CY62256N, a 32 kB Cypress SRAM, is used as a PUF. A custom SRAM shield (FIGURE 7) is designed as a compact HW package to interact reliably with the evaluation kit via its GPIO pins. The shield is used to interface SRAM PUF with SAMV71 MCU during the characterization, registration, and authentication phases. The SRAM PUF is characterized in advance to find the unstable cells with a higher response variation. The characterization (enrollment) result shows which cells are fuzzy or non-fuzzy stored in the MCU flash memory.

For each authentication request in the hash chain-based PWM system, the \(PW\) is hashed \(M\) times where \(M\) should be smaller than \(N\). Regarding how the number of applying hash decreases \((N-M)\), two types of operation are considered in the implementation. Type 1 refers to when \(M\) decreases randomly for each authentication request in the range of 2 to 9. Type 2 refers to the case where \(M\) decreases by one for each authentication request. The steps of the protocol explained in section III are based on type 2. In Type 1 operation, called "random reduction," \(M\) should be encrypted and transmitted with the encryption of ID and encryption of \(H(\{4,5\})(PW)\). The process will be done in reverse at the server side to extract the user ID and \(H(\{4,5\})(PW)\). In Type 2 operation, called "one by one," \(M\) is reduced by one for each authentication request. This number is synchronized with the server in the registration step. In type 1, the client does not need to transmit (and encrypt) \(M\).

V. RESULTS

In the following, subsection A provides the results of SRAM PUF characterization. Also, in subsection B, step-by-step results of the protocol are provided. It has to be noted that most of the registration and authentication steps and results are the same.

A. SRAM PUF Characterization

Some statistical parameters and quality metrics, including intra-PUF, and inter-PUF are presented in [31]. The new results are presented in this section. The enrollment size in the ternary PUF is selected 1000. Such an enrollment size
improves the error rate to an acceptable range for the protocol used in this paper.

1) ENROLLMENT AND MASKING

The experiments are carried out on 10 SRAM chips. FIGURE 8 shows the number of detected fuzzy cells versus the size of the enrollment. This figure indicates how finding new fuzzy cells becomes more difficult with each additional read. As shown in FIGURE 8, each enrollment size N analyzes all the bits that have changed their state in the previous cycles. The percentage of fuzzy masked cells (‘X’) and stable (‘0’ + ‘1’) cells are independently plotted against the enrollment size. The results are taken from the median of the ten individual chips. As can be seen at a given cycle, there is no other previous cycle, so that every cell is labeled as stable by default. Over half the masked (fuzzy) cells are found after ten cycles, whereas the remaining half is found after cycle 1000. After 1000 cycles, 17.87% of cells are fuzzy, while the other 82.13% are stable.

2) INTRA- AND INTER-PUF COMPARISON

In another experiment, the enrollment of the SRAMs was done first. The fuzzy cells are detected after 1000 reads. Then, the data of intra-chip and inter-chip variations are collected when just non-fuzzy cells are considered. FIGURE 9 shows the results of intra/inter-chip variation after enrollment with 1000 cycles. TABLE II shows the value of minimum, median and maximum of intra- and inter-chip variation. This table shows that the average flipping probability of cells is 6.33E-05. The inter-PUF distance has worsened compared to previously published work. However, for handling the addressing error, intra-PUF distance is more important in this paper.

3) UNIFORMITY

In this research, uniformity is defined as the ratio of ‘0’s to ‘1’s for each read from an SRAM. The response of each of the 10 Cypress SRAM chips is analyzed when they are queried 100 times. FIGURE 10 illustrates the distribution of 100 uniformity indices for each chip as a boxplot. Table III shows the value of the mean and standard deviation of the distribution of the uniformity index. It has been observed that the uniformity mean values are very close to 1 for all the chips. In this regard, the global mean for all chips is 0.9993762, and the global standard deviation is 0.005012068, respectively.
TABLE II
INTRA-/INTER-CHIP VARIATION

|          | Min   | Median  | Max   |
|----------|-------|---------|-------|
| Intra    | 0     | 6.33E-05| 0.003108|
| Inter    | 0.349555 | 0.374765 | 0.39439 |

B. Step by Step results of the protocol

1) TERMINAL RESULTS

FIGURE 11 indicates the terminal results of the client-side for registration. As shown in FIGURE 11, MCU receives an encrypted ID, PW, and the last N (initial N for registration step). FIGURE 12 shows the results of calculations performed in the Server-Side for registering a user. These calculations include a rotating and hashing process (to generate the long MD) that generates a raw address from long MD generates refined address using the mask data and reads PUF bits at refined addresses. As FIGURE 12 shows just the results of generating DB addresses, a similar calculation is necessary to generate the DB content, which is not included in FIGURE 12.

Authentication involves more steps than registration. More input data can be seen in FIGURE 13 compared to what we have seen in FIGURE 11. In FIGURE 13, the reduction type 1 is selected. Since it is aimed to reduce the calculation burden on the client-side, in the protocol used in this paper, N is replaced with(N − 1)in type 2 (or (N − rand[2,9]) in type 1) if the authentication was successful. Thus, the same calculations are used to extract the DB address and content based on the client's new N.
FIGURE 11. Terminal results of registration on the client-side.

FIGURE 12. Calculation results on the server for registration.

FIGURE 13. Terminal results of the client at each authentication request.
A few more steps are on the server-side to extract $H^N(PW)$ and $(ID \oplus H^N(PW))$, while $N$ in this notation is the last $N$ used in the previous authentication request. Since the digest we received on the server is $H^{N-rand[2,9]}(PW)$, it makes $\text{rand}[2,9]$ or one more hash to generate the $H^N(PW)$. First, raw user $ID$ is obtained from the address, and $H^M(PW)$ is received in the server by XORing them together. As shown in FIGURE 14, one more hash is performed to obtain the $H^N$ and XOR the raw $ID$ with the final digest $H^N$. This result is used to generate the DB address and content. First, raw user $ID$ was obtained from the address, and $H^M(PW)$ received in the server by XORing them together. In FIGURE 15, one more hashing is applied to obtain the $H^N(PW)$ and again XOR the raw user-ID with the final $H^N(PW)$ digest.

2) MATLAB GUI

The results of each registration and authentication step of the users are illustrated in the MATLAB GUI. The GUI scheme used for this paper is shown in FIGURE 16. The GUI has three main parts: 1) "Account Info" in the upper part, 2) "Details" in the middle part, and 3) "CRP Current" in the lower part. "Account Info" is similar for registration and authentication.

FIGURE 16 represents the results of the registration of a new user. As shown in the upper part of FIGURE 16, in the "Account Info" section, the input data that the user enters for registration are "ID ", "PWD", and "LastN". The user should enter $(N-1)$ or $(N - \text{rand}[2,9])$ in the "LastN" box based on the selected type and the random number generated by the HW. The random number generated by the HW crypto engine is shown in the box close to the "Registration" button. In the middle part of FIGURE 16 (i.e., the "Details" section), four parameters are shown: $H(N)(PW)$, "ID XOR H(N)(PW)", and the responses extracted from them. The responses extraction process is presented in the lower part (i.e., the "CRP Current" section). The left part of this section shows the DB address, and the right part shows the content that should be saved in that address. This information has four columns. "RawAddr" presents the raw address, which is the input of the "Masking" block in FIGURE 5. The second column from the left (i.e., "MaskData") shows the fuzzy and non-fuzzy cells are by "1" and "0", respectively. The third
column (i.e., “RefinedAddr”) shows the revised addresses, which are the output of the “Masking” block in FIGURE 5. Finally, the fourth column (i.e., “RefinedBits”) indicates the bits of the response read in the revised addresses.

FIGURE 17 displays the results of the authentication of a user. Compared to the registration result shown in FIGURE 16, the section "New" is filled in the middle part of FIGURE 17. The "Current" is the header of those rows representing the results generated using the last N and is common for both registration and authentication steps, and "New" is the header of those rows representing the results generated using the new N selected on the client-side.

The server checks the data in the "Current" section for approving or rejecting the authentication (step 3 of authentication in the server). If the authentication is approved, the data shown in the "New" section will be stored in the DB (step 4 of authentication in the server).

The only information that is stored in DB for clients is their corresponding PUF response. As shown in FIGURE 12, the PUF response size used as DB content for each client is 128 bits. Also, the size of the utilized MATLAB DB is 1MBit. Thus, 8192 users can be covered, and 13 bits can address a specific DB location. The least significant 13 bits of the “DataBase addr" in FIGURE 12 are used to address a location for a specific user in the DB.

VI. DISCUSSION ON RELIABILITY, SECURITY, AND PERFORMANCE

A. Reliability and FRR

The results of some of the previous publications on the comparison of silicon PUFs are listed in TABLE IV. In [36], a promising crossover Ring Oscillator (RO) PUF is presented. The crossover RO PUF introduced in [36] has low HW overheads. Moreover, their proposed crossover RO PUF can drastically mitigate the effect of the environmental changes on PUF responses and generate more reliable responses. The crossover RO PUF presented in [36] is also resistant to side-channel attacks based on the claim in [36, 37].

| Ref. # | Inter(%) | Intra(%) | Ref. # | PUF TYPE |
|---|---|---|---|---|
| [36] | 49 | 5.03 | Crossover RO PUF |
| [36] | 50.36 | 1.78 | RO |
| [39] | 45.83 | 5 | SRAM |
| [32] | 48.9 | 3.63 | SRAM (without enrollment) |
| [32] | 48.7 | 0.00216 | SRAM (with enrollment) |
| [31] | 38.2 | 3.5 | SRAM (without enrollment) |
| [31] | 36.9 | 0.0467 | SRAM (with enrollment) |
| [31] | 36.9 | 3.5 | SRAM (without enrollment) |
| [31] | 37.4 | 6.33E-05 | SRAM (with enrollment) |
The inter-PUF distance has worsened compared to previously published work. However, as previously mentioned, the address of DB is coming from PUF in our protocol. Thus, reaching a very low intra-PUF index is more important in this paper. Using the Poisson statistical model, the probability of at least 1-bit error in the 13-bit address stream that leads to FRR is estimated at $8 \times 10^{-5}$. Such levels of FRRs are minimal and acceptable to users [9].

### B. Security

In the protocol used in this paper, the client uses different authentication credentials at each authentication request. Hence, the protocol is resistant to MIM attacks. Also, the proposed PWM system is resistant to PW guessing and offline dictionary attacks since the PUF is inside the server. In addition, since the DB address is extracted from the PUF, decoding the DB information becomes more challenging when the data is stolen. Nevertheless, as SRAMs PUFs are relatively easy to break, it is necessary to design and prototype the proposed PWM system with tamper-resistant PUFs.

### C. Efficiency

TABLE VI compares the similar protocol used in previous publications regarding computation cost, communication cost, and the hash-chain reinitialization time. Since the client-side memory and computational power are critical, we consider the comparison on the client-side. $HT$ is the time needed to hash the $PW$ and $ID$. Also, $AL$ is the length of one encrypted string, $T$ is the required time before a new chain reinitialization, and $n$ is the length of the hash chain. $ET$, i.e., the time needed to do encryption, is added to the computation cost in “random reduction.”

Our proposed method uses PUFs on the server-side, unlike any PUF-based scheme in the existing research. All the previous works are based on using PUF on the client-side. The only similar work to ours is [35]. However, in [35], a PC handles all tasks, and the utilized HW is just used to read the PUF. Any analytical or numerical evaluation has not been presented in [35]. In [35], the length of the used hash chain is 1000; the reinitializing time is $500 \times T$. The reinitializing time for type 1 of our protocol (“Random decreasing”) is $5.5 \times T$ as the random number is selected in $[2,9]$.

**TABLE V**

| Protocol                  | Computation cost | Communication cost | Reinitializing time |
|---------------------------|------------------|--------------------|---------------------|
| Random decreasing         | $HT + ET$        | $5 \times AL$      | $\left(\frac{n}{2}\right) \times T$ |
| One by one                | $HT$             | $4 \times AL$      | $T$                 |
| [35]                      | Not reported     | Not reported       | $500 \times T$      |
VII. CONCLUSION AND FUTURE WORK

This article presents novel techniques and HW solutions for enhancing PUF-based PWM systems' efficiency that may be utilized in various real-world situations. The proposed PWM system is immune to offline dictionary attacks since the PUF is secure inside the server (and maybe further protected through anti-tampering techniques). Additionally, using hash chains makes the system immune to MIM attacks.

The HW implementation described in this article employs extremely inexpensive components. The 32kB SRAM device used in this study is readily accessible commercially for less than $2. A significant issue with PUFs, namely their response instability, is addressed in this study by utilizing a ternary SRAM PUF. This article chooses SRAM as a PUF since it is a widely accessible memory technology in various devices. They are, however, susceptible to cloning attacks using remanence decay side channels [40].

In this research, ternary PUF with three possible states for each cell is used. When an attacker has access to the HW, ternary PUFs are more challenging to read and break than binary PUFs. Additionally, this research employs the SRAM's middle eight kB as a PUF. The location of the PUF used in this system might be a piece of secret information that adds a low degree of security. However, attacking the SRAM PUF can compromise the PWs of the users. Therefore, tamper-resistant PUFs such as MRAM PUFs [41] are considered for future work.

As previously stated, latency is a significant disadvantage of using long HCs with PUFs and long HCs. This factor prolongs the time required for authentication and thus reduces the number of users that can be handled. As a result, our future effort will increase the number of APGs capable of handling more clients.

ACKNOWLEDGMENT

The authors thank Dr. Tolga Yalcin, Cryptography Hardware Architect at Google.

REFERENCES

[1] H. Saleem and M. Naveed, "SoK: anatomy of data breaches," Proc. Priv. Enhancing Technol., vol. 2020, no. 4, pp. 153-174, 2020.
[2] F. Ullah, M. Edwards, R. Ramdhany, R. Chitechyan, M. A. Bahar, and A. Rashid, "Data exfiltration: A review of external attack vectors and countermeasures," Journal of Network and Computer Applications, vol. 101, pp. 18-54, 2018.
[3] J. Hendrickson. "Why are companies still storing passwords in plain text?" [Online].Available: http://www.usatoday.com/story/money/business/2014/01/10/targetcustomer-data-breach/44044467/ (accessed 2020).
[4] Z. Whittaker. "GitHub says bug exposed some plaintext passwords." [Online] Available: https://www.zdnet.com/article/github-says-bug-exposed-account-passwords/ (accessed 2020).
[5] J. Blocki, B. Harsha, and S. Zhou, "On the economics of offline password cracking," in 2018 IEEE Symposium on Security and Privacy (SP), 2018: IEEE, pp. 853-871.
[6] D. V. Klein, "Foiling the cracker: A survey of, and improvements to, password security," in Proceedings of the 2nd USENIX Security Workshop, 1990, pp. 5-14.
[7] S. Mishra, D. Chang, and S. K. Sanadghy, "Design and analysis of password-based authentication systems," IIIT-Delhi, 2017.
[8] S. Tajik, H. Lohrke, F. Ganji, J. Seife, and C. Boit, "Laser fault attack on Physically Unclonable Functions," in 2015 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), 13-13 Sept. 2015, pp. 85-96.
[9] B. F. Cambou, "Password management with addressable physical unclonable function generators," ed: Google Patents, 2019.
[10] M. Shirvani, N. Saxena, S. Jarecki, and H. Krawczyk, "Building and studying a password store that perfectly hides passwords from itself," IEEE Transactions on Dependable and Secure Computing, vol. 16, no. 5, pp. 770-782, 2019.
[11] D. Florencio and C. Herley, "A large-scale study of web password habits," in Proceedings of the 16th international conference on World Wide Web, 2007: ACM, pp. 657-666.
[12] R. Wash, E. Rader, R. Berman, and Z. Wellner, "Understanding password choices: How frequently entered passwords are re-used across websites," in Twelfth Symposium on Usable Privacy and Security (SOUPS) (2016), 2016, pp. 175-188.
[13] W. S. Janzen, "Iterated password hash systems and methods for preserving password entropy," ed: Google Patents, 2014.
[14] R. Aremburg, S. Chawla, A. Mathur, and C. Skawratananond, "Method, apparatus and program storage device for providing a secure password manager," ed: Google Patents, 2007.
[15] L. Bošnjak, J. Šrej, and B. Brumen, "Brute-force and dictionary attack on hashed real-world passwords," in 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2018, pp. 1161-1166.
[16] A. Shamossinha, A. Korenda, F. Afgah, and S. Zeaally, "A survey on physical unclonable function (PUF)-based security solutions for Internet of Things," Computer Networks, vol. 183, p. 107593, 2020.
[17] A. R. Korenda, F. Afgah, B. Cambou, and C. Philabaum, "A proof of concept SRAM-based physically unclonable function (PUF) key generation mechanism for IoT devices," in 2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), 2019: IEEE, pp. 1-8.
[18] B. Cambou, M. Mohammadi, C. Philabaum, and D. Booher, "Statistical analysis to optimize the generation of cryptographic keys from physical unclonable functions," in Science and Information Conference, 2020: Springer, pp. 302-321.
[19] J. Zhang and C. Shen, "Set-Based obfuscation for strong PUFs against machine learning attacks," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 68, no. 1, pp. 288-300, 2021.
[20] Y. Gao, S. F. Al-Sarawi, and D. Abbott, "Physical unclonable functions," Nature Electronics, vol. 3, no. 2, pp. 81-91, 2020.
[21] J. McGrath, I. E. Bagci, Z. M. Wang, U. Roedig, and R. J. Young, "A PUF taxonomy," Applied Physics Reviews, vol. 6, no. 1, pp. 011303, 2019.
[22] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, "Physical unclonable functions and applications: A tutorial," Proceedings of the IEEE, vol. 102, no. 8, pp. 1126-1141, 2014.
[23] R. Maes and I. Verbauwhede, "Physically unclonable functions: A study on the state of the art and future research directions," in Towards Hardware-Intrinsic Security: Springer, 2010, pp. 3-37.
[24] C. A., “A review of emerging non-volatile memory (NVM) technologies and applications,” Solid-State Electron., vol. 125, p. 25, 2016.

[25] J. Das, K. Scott, S. Rajaram, D. Burgett, and S. Bhanja, “MRAM PUF: A novel geometry based magnetic PUF with integrated CMOS,” IEEE Transactions on Nanotechnology, vol. 14, no. 3, pp. 436-443, 2015.

[26] 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, 2007, vol. 7, no. 2, p. 01.

[27] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in International workshop on cryptographic hardware and embedded systems, 2007: Springer, pp. 63-80.

[28] A. Vijayakumar, V. Patil, and S. Kundu, "On improving reliability of SRAM-based physically unclonable functions," Journal of Low Power Electronics and Applications, vol. 7, no. 1, p. 2, 2017.

[29] Q. Guo et al., "PUFPass: A password management mechanism based on software/hardware codesign," Integration, vol. 64, pp. 173-183, 2019.

[30] B. Cambou and M. Mohammadinodoushan, "Resilient password management system using an array of addressable physical unclonable functions," US Patent, Filed: April, vol. 15, 2020.

[31] M. Mohammadinodoushan, B. Cambou, C. Philabaum, D. Hely, and D. D. Booher, "Implementation of password management system using ternary addressable PUF generator," in 2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), 2019: IEEE, pp. 1-8.

[32] M. Mohammadinodoushan, B. Cambou, C. R. Philabaum, and N. Duan, "Resilient password manager using physical unclonable functions," IEEE Access, vol. 9, pp. 17060 - 17070, 2021.

[33] L. Lamport, "Password authentication with insecure communication," Communications of the ACM, vol. 24, no. 11, pp. 770-772, 1981.

[34] A. Pinto and R. Costa, "Hash-chain-based authentication for IoT," ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal, vol. 5, no. 4, p. 43, 2016.

[35] S. Assiri and B. Cambou, "Homomorphic password manager using multiple-hash with PUF," Cham, 2021: Springer International Publishing, in Advances in Information and Communication, pp. 772-792.

[36] Z. Pang, J. Zhang, Q. Zhou, S. Gong, X. Qian, and B. Tang, "Crossover ring oscillator PUF," in 2017 18th International Symposium on Quality Electronic Design (ISQED), 2017: IEEE, pp. 237-243.

[37] J. Zhang and G. Qu, "Physical unclonable function-based key sharing via machine learning for IoT security," IEEE Transactions on Industrial Electronics, vol. 67, no. 8, pp. 7025-7033, 2019.

[38] D. Deng, S. Hou, Z. Wang, and Y. Guo, "Configurable ring oscillator PUF using hybrid logic gates," IEEE Access, vol. 8, pp. 161427-161437, 2020.

[39] A. Kumar, S. Sahay, and M. Suri, "Switching-time dependent PUF using STT-MRAM," in 2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID), 2018: IEEE, pp. 434-438.

[40] S. Zeitouni, Y. Oren, C. Wachsmann, P. Koeberl, and A.-R. Sadeghi, "Remanence decay side-channel: The PUF case," IEEE Transactions on Information Forensics and Security, vol. 11, no. 6, pp. 1106-1116, 2015.

[41] A. Nejat, F. Ouattara, M. Mohammadinodoushan, B. Cambou, K. Mackay, and L. Torres, "Practical experiments to evaluate quality metrics of MRAM-based physical unclonable functions," IEEE Access, vol. 8, pp. 176042-176049, 2020.