Password Manager Combining Hashing Functions and Ternary PUFs

Bertrand Cambou(✉)

Northern Arizona University, Flagstaff, USA
Bertrand.cambou@nau.edu

Abstract. Hashing functions protect passwords against various hacking techniques because message digests can replace the passwords when stored in the network for future authentication. However, the message digests remain exposed to password guessing attacks, most hashing functions are known, and public. The objective of the protocols presented in this paper is to offer additional lines of defense using physical unclonable functions to convert the message digests into challenge-response pairs. The use of ternary physical unclonable functions reduces false rejection rates, and lowers the latencies during the processing of the authentications. Without having access to the PUFs, the look up tables storing challenge-response pairs are more difficult to attack than those storing message digests: they are unclonable, contain high levels of randomness, and quasi unique. The modeling efforts, and algorithms developed in this paper to validate the schemes, use commercially available components, and SRAM based ternary PUFs.

Keywords: Password management · Physical unclonable function · Hash functions · Ternary states

1 Introduction

The essence of access control is to match reference patterns such as passwords, secret keys, and biometric prints, against the same reference patterns of the users, which are stored in look up tables of the secure servers managing the networks [1–3]. One of the known exposures in the cyber space, is the potential loss to malicious entities of the look up table storing UserID-password pairs. Hackers are often able to damage institutions, and individual users by getting un-authorized access to databases of millions of passwords at once. Examples include attacks on internet providers, US government agencies, the IRS, health institutions, political parties, banks, and much more [4–7]. These types of cyber-attack, creates huge financial damages, and can be the result of insiders that are silent for years. A false sense of security can exposes millions of users overnight.

Ways to protect passwords and look up tables, from both external and internal attackers, have been proposed [8–14]. Several layers of protections for the access control are proposed in this paper; the first layer is the replacement of the userID-password pair by message digests of hashing functions [15–17]; the second layer is the replacement of the message digests by challenge response pairs (CRP) of physical

K. Arai et al. (Eds.): CompCom 2019, AISC 998, pp. 1–20, 2019.
https://doi.org/10.1007/978-3-030-22868-2_37

unclonable functions (PUFs) [18–26]. In both cases, the exposure to stolen databases is lower, because the content of the tables is not directly exploitable to gain access to the original UserID-password pairs. The protection offered by PUFs is much higher, and mitigates attacks based on the knowledge of look up tables using message digests, and hashing functions [27, 28]. However, the need to use an error correcting code can be extremely complex [29–35]. Therefore, novel methods based on ternary PUFs, that are error correction free, are proposed. The architectures developed in this paper, reuses known technology modules, such as memory based PUFs, true random number generators, functions, and public key infrastructure. Several prototypes, containing both hardware and software components, were designed to evaluate, and validate the proposed schemes.

2 Background Information

2.1 Hash Function for Password Managers

Hash functions [36–38] are one-way cryptographic methods to generate message digests of fixed lengths, from incoming data streams of variable lengths. Hash functions have the following properties:

- A single bit change in the incoming data stream results in a totally different message digest.
- The incoming data streams cannot be extracted from the message digests.
- The hash functions are collusion resistant: the probability that two different streams result in the same message digest is extremely low.

Hash functions are pivotal in multiple cryptographic applications such as multi-factor authentications, and blockchains [39–44]. Hash functions replace, the look up tables containing User ID-password pairs [45], as shown in Fig. 1.

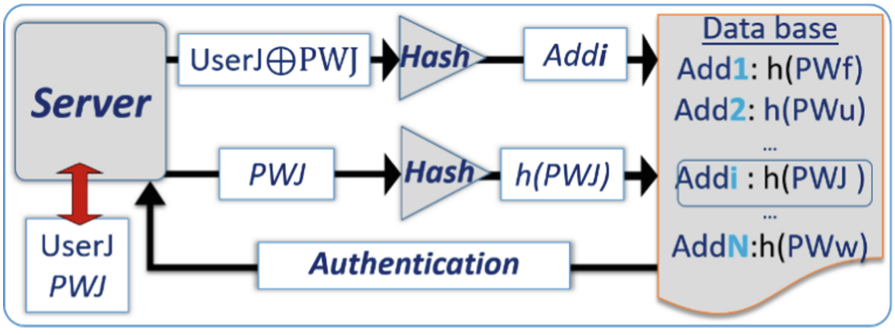

Fig. 1. Block diagram describing the data flow. Passwords and UserIDs are converted into message digests then into addresses (h(ID⊕pass)). On the right, the look up tables store the message digests.

The hashing of the password ***PWj*** results in the first message digest ***h(PWj)***; the user ID ***Userj*** and ***PWj*** are XORed; the hashing of ***Userj*** ⊕ ***PWj*** results in the second message digest ***h(Userj*** ⊕ ***PWj)***. A portion of the second message digest is then used to generate the coordinate **XY** of the look up table storing the first message digest. The XORing of the user ID, and the password, enhances the protection, and this because the hashing of the User ID alone is weak, considering that most hashing functions are known and public.

To explain this method we developed a small example shown in Tables 1 and 2. Here we are using SHA-1 for its simplicity; however, for the prototypes proposed in the remaining sections of this paper, we used SH-3, which is safer, even in cases of attacks from quantum computers. In this example, user#1 has a user ID "a6c26", a password "12ae5", and a first message digest "a639d" generated with SHA-1; only the first five characters of the message digest are kept. The XORing the first three hexadecimal characters of the user ID and the password have a value of "b86". The hashing of "b86" with SHA-1 generates a message digest that has "3e" as first two hexadecimal characters. We then place "a639" in the address "3e" of Table 2. Five more UserID-password pairs are stored in a similar way in the look up Table 2.

Table 1. Example showing the conversion of user IDs, and passwords into message digests, and an addresses, with hash functions.

User	ID	PW	h(PW)	ID⊕PW	h(ID⊕PW)
1	a6c26	12ae5	a639d	b86	3e
2	a311f	221ab	44d46	81f	90
3	ed011	22131	a3171	dfe	8c
4	15cab	320f1	7d5ff	27c	af
5	87a1b	775ed	0454b	f0f	93
6	1acf6	523c0	9b4fc	48f	ba
…	…	…	…	…	…

Table 2. Look up table storing the hashing of the passwords.

YX	0	1	2	3	4	5	6	7	8	9	A	B	c	d	e	f
0																
1																
2																
3															**a639d**	
4																
5																
6																
7																
8													**a3171**			
9	**44d46**			**0454b**												

(*continued*)

Table 2. (*continued*)

YX	0	1	2	3	4	5	6	7	8	9	A	B	c	d	e	f
A																**7d5ff**
B											**9b4fc**					
C																
D																
E																
F																

Such a method is applicable to store very large quantities of UserID-Password pairs. During authentication cycles, the users provide their UserID/Password pair; the information extracted from the table at the corresponding address is then compared to the message digest provided by the password. When SHA-1 is replaced by more powerful hashing functions (ex: SHA-3), such look-up tables are much more secure than the tables directly storing UserID-Password pairs. The method can handle low levels of collisions, i.e. multiple messages digests located in the same address, or multiple Users sharing the same message digest.

Vulnerability of the Method. When look-up tables storing message digests are lost to the enemy, information can also be lost overtime with password guessing methods, big data analysis, and brute force hashing of commonly used passwords. For example, if the attacker knows that a particular user trends to use a set of passwords, the attacker can hash these passwords, and see if the message digests exist in the table. When millions of users are involved, the loss of can have a considerable impact, with legal exposure for the operator. Other issue of this password manager is the potential loss of the password by the user. The password manager cannot erase the message digest because the address in the look up table is also lost. One remedy is to implement a counter, which is incremented every time a user forgets the password. When the user login after the loss, its password can be concatenated with an index that is incremented at each loss. If the previously lost password is used again, the concatenated version with the index will not be recognizable.

2.2 Physical Unclonable Functions

PUFs generate patterns from hardware components that act as human fingerprints, which are unclonable, random, and make each device authenticable from each other [18–25]. The design of the PUF exploits the manufacturing variations created during their fabrication. In this paper, we are calling challenges, or initial responses, the patterns generated upfront by the PUF, while the responses are generated during each access control cycle. The authentications are positive when the matching challenge-response-pairs (CRPs) are high, with a low rate of error, i.e. mismatch. Methods to design PUFs include the use of logic components with gate delays, arbiters, and ring oscillators [18–20]. A PUF designed with ring oscillators is shown in Fig. 2.

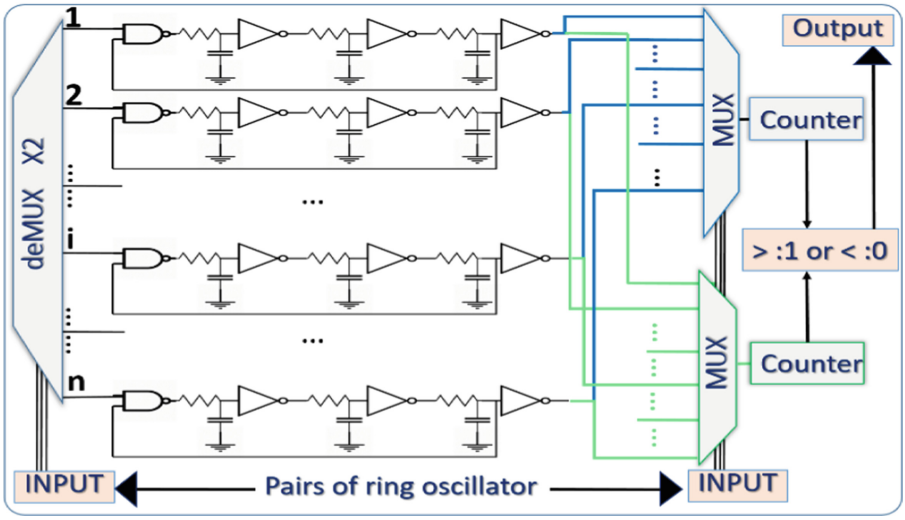

Fig. 2. Block diagram of a PUF designed with ring oscillators.

Pairs of ring oscillators selected among the n possible pairs, are analyzed using two counters. If the number of oscillations of the ring going to the top MUX is greater than the number of oscillations of the ring going through the bottom MUX, the challenge (during initial set up, or response during authentication) is a "1", if not, the challenge (or response) is a "0". If for example, the address of 128 successive pairs is transmitted to the PUF, 128-bit challenges (or responses) are generated.

Memory structures [46–55], SRAM [46, 47], DRAM [48], Flash [49, 50], ReRAM [51–53], and MRAM [54, 55], are also excellent elements to generate strong PUFs. The method to generate PUFs from SRAM arrays is to subject the device to power-off-power-on cycles. A significant proportion of the cells always restart as either a 0, or a 1, thereby generating CRPs. One of the generic methods to generate CRPs from memory devices is to characterize a particular parameter $\mathcal{P}$ of the cells of the array. The values of parameter $\mathcal{P}$ vary cell to cell, and follow a distribution with a median value **T**. In order to generate challenge and response pairs, all cells with $\mathcal{P} < \mathbf{T}$ generate "0" states, and the others generate "1" states. The resulting streams of data become PUF CRPs to authenticate the array.

Various PUFs have different figures of merit, and strength against attackers. A hostile party cannot simply read the entire PUF array of a strong PUF, and use the reading to communicate with a network of client devices. The cloning of the entire array would be a security threat; however, the level of randomness of such a component is so high that this type of attack is highly unlikely. The use of ternary PUFs reduces the challenge-response-pair (CRP) error rates [56–60].

3 Password Managers with PUFs

3.1 Password Generators with PUFs

One time password generators can be designed with addressable PUF generators (APG) [45, 61]. The block diagram of the APG is shown in Fig. 3.

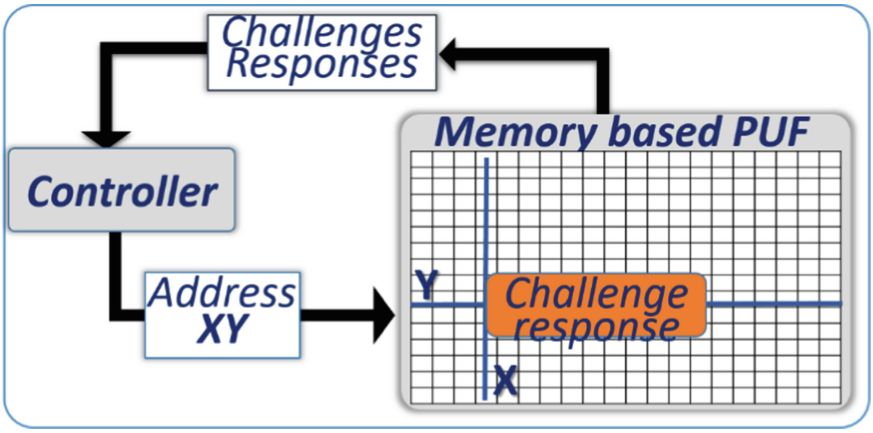

Fig. 3. Block diagram of the APG

In this architecture, the controller sends an address XY to the memory array to read a block of cells from which challenges and responses are generated. For a SRAM PUF, 0 or 1 are read in the block of cells located at address XY after power-off-power-on cycles. For other memory PUFs, parameter $\mathcal{P}$ of the block of cells located as XY is characterized. The ring oscillator of section II-B is also used as APG. In this case series of pairs are characterized at each address. The CRP generation varies with the relative value of the parameters within the multiple cells that are selected at a particular address. Therefore, as a particular cell could be a "0" when part of one group of cells, and a "1" when part of a different group, or when read with different instructions.

As described in [45, 61], APGs can be used to generate temporary passwords, and authenticate users. A block diagram of a temporary password generator with APG is shown in Fig. 4:

- For *password generation*, a random number ***TRN*** is generated and concatenated with the UserID, to feed the hash function. An address XY is extracted from the message digest to point on a particular block of cells in the APG. The reading of the PUF generates challenges, i.e. the initial responses. The resulting temporary password is the combination of ***TRN***, and the challenges.
- For *authentication*, the user transmits the password and its user ID. The same ***TRN*** and ***UserID*** are generating the same message digest, the same address ***XY***; the same block of cells is read to generate responses. These responses are matched with the challenges provided as part of the temporary password. If the CRP error rate is low, the authentication is positive.

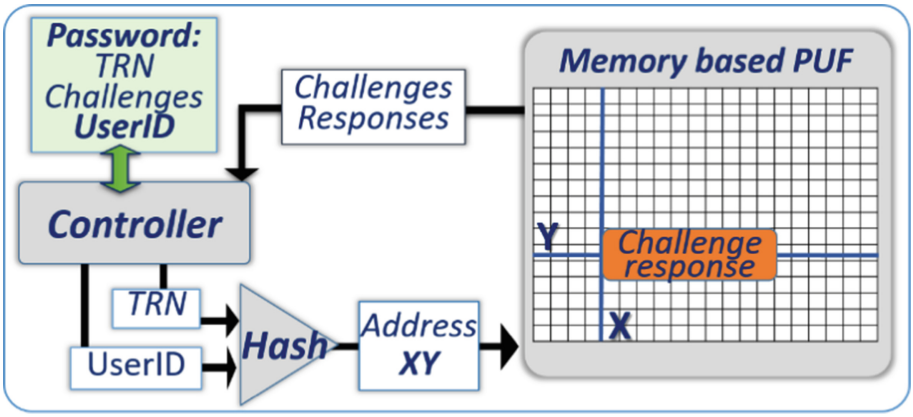

Fig. 4. Block diagram of a temporary password generator with APG

Multi-factor authentication and encryption was included in our implementation to protect the password. In the application the PUFs reside in the server, therefore the device is not subject to significant thermal and other environmental variations, this results in low CRP error rates. The method uses new random numbers ***TRN*** at each operation for one-time use of the passwords and concatenate ***TRN,*** with an index to eliminate the relevance of previous used passwords.

3.2 Password Manager Architecture with PUFs

Unlike the temporary password generator shown in section III-A, a password manager allows the user to pick its own password *PWj*, and user ID *Userj*. The design of password managers with PUFs is more complicated than designing password generators with PUFs. In the architecture presented in this paper, addressable PUF generators (APG) are combined with the hash function based password manager described in section II-A. A block diagram describing the scheme is shown in Fig. 5.

- During *enrollment*, the hashing of ***Userj*** $\oplus$ ***PWj*** results in the message digest ***h (Userj*** $\oplus$ ***PWj)*** from which the address ***Addi*** is extracted. The hash of password ***PWj*** generates the address ***XY*** in the APG. In our implementation, the memory space of the APG is an array of 4096 $\times$ 4096 cells, the first 12 bits of the message digest were used to find the ***X*** in the array, the next 12 bits to find the ***Y***. Starting from this address ***XY***, challenges are generated from the n-cells following this address. The parameter ***P*** is then measured, and challenges ***Chj***, the stream of bits *{Ch1, Ch2, ...Chn}*, are generated. The challenges ***Chj*** are finally stored in the look up table at the address ***Addi***.
- The *authentication* cycles follow the same step by step scheme, resulting in the same address ***Addi***, and a stream of responses ***Rej***. These responses are matched with the challenges ***Chj***. Assuming that the CRP error rates are low enough, the authentication is positive.

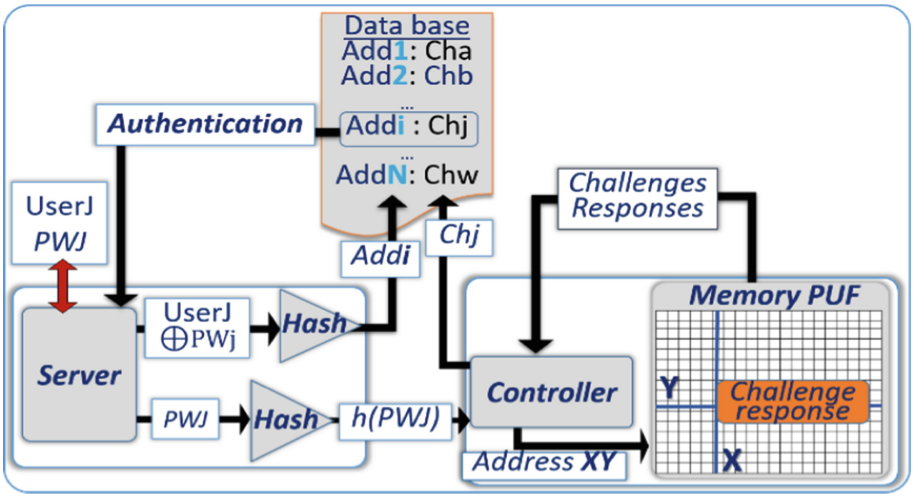

Fig. 5. Block diagram showing a password manager with APG.

Since hash functions are one-way functions, it is impossible to deduce the input of the hash function by looking at the address of a PUF array. Unlike traditional data storage units, the memory arrays used in the APG do not store information; CRPs cannot be generated from these PUFs without knowing the necessary instructions. If an attacker is able to have access to the database, the knowledge of the challenges do not disclose the message digest of the password. This offers additional layers of security, and makes password guessing more difficult.

3.3 Algorithm of Password Managers with PUFs

A summary of the algorithm developed to store the user ID and password in the look-up table with APGs is presented in Table 3:

Table 3. Algorithm to generate, and store challenges, replacing the passwords.

Step	Description of the instructions	Data stream/information	Where
1.0	*Hashing of the password* ***PWj***	***h(PWj)***	***Server***
2.0	***h(PWj)*** *converted into the address* ***XY***	***h(PWj)*** → ***X Y***	***APG***
3.0	*Select n-cells in the array located after* ***XY***	***{n-cells}***	***APG***
3.1	*Measure parameter* $\mathcal{P}$ *for the n-cells*	***{P1, P2, …, Pn}***	***APG***
3.2	*Generate the challenges* ***Chj*** ϵ *{0, 1}*	***Chj = {C1, C2, …, Cn}***	***APG***
4.0	*XORing the user ID* ***Userj*** *with* ***PW***	***Userj*** ⊕ ***PW***	***Server***
5.0	*Hashing of* ***Userj*** ⊕ ***PW***	***h(Userj*** ⊕ ***PW)***	***Server***
6.0	***h(Userj*** ⊕ ***PW)*** *converted into address* ***Addi***	***h(Userj*** ⊕ ***PW)*** → ***Addi***	***Server***
7.0	*Store* ***Chj*** *in the database at address* ***Addi***	***Addi; Chj***	***Database***

- *Step-1:* The password ***PWj*** feeds the hash function to generate a message digest ***h (PWj)***. Additional security features are not described in this paper, which include the use of cryptography to protect the passwords, and multi-factor authentication to protect the user. The hash function can be based on some standard hash algorithms (SHA).
- *Step-2:* The message digest of step one is used to generate the coordinate ***XY*** of the memory array. As suggested above, this could be the first 24-bits of the message digest, if the size of the memory array is 4,096 × 4,096. Different possible schemes are presented in Sect. 4.
- *Step-3:* The generation of a stream of challenges ***Chj = {Ch1,….Chn}*** from the ***n-cells*** located in the memory array which passes the address ***XY,*** can be done differently based on the type of memory technology, and various generation protocols. Some methods, as suggested Sect. 2.1, can be based on the measurement of a particular parameter Ƥ.
- *Step*-4 and 5: The user ID ***Userj***, and the password ***PW*** are Xored to generate the data stream ***Userj*** ⊕ ***PW***. The Boolean function XOR can be replaced by other methods to create a data stream from the user ID.
- *Step*-6: After the hashing of ***Userj*** ⊕ ***PW***, an address ***Addi*** is generated from the message digest. If, for example, the look-up table has 4,096 × 4,096 positions, the first 24 bits of the message digest can be used to generate ***Addi***.
- *Step*-7: The challenges ***Chj = {Ch1,….Chn}*** is stored at the address ***Addi***. In case of a collusion, i.e. multiple users at the same address, multiple challenges are stored at the same location, which does not make the authentication more difficult.

A summary of the algorithm developed to authenticate a user with password is summarized in Table 4.

Table 4. Algorithm to authenticate users with CRPs.

Step	Description of the instructions	Data stream/information	Where
1.0	*Hashing of the password* ***PWj***	***h(PWj)***	***Server***
2.0	***h(PWj)*** *converted into the address* ***XY***	***h(PWj)*** → ***X Y***	***APG***
3.0	*Select n-cells in the array located after* ***XY***	***{n-cells}***	***APG***
3.1	*Measure parameter* ***Ƥ*** *for the n-cells*	***{P1, P2, …, Pn}***	***APG***
3.2	*Generate responses* ***Rej*** ϵ *{0, 1}*	***Rej = {C1, C2, …, Cn}***	***APG***
4.0	*XORing the user ID* ***Userj*** *with* ***PW***	***Userj*** ⊕ ***PW***	***Server***
5.0	*Hashing of* ***Userj*** ⊕ ***PW***	***h(Userj*** ⊕ ***PW)***	***Server***
6.0	***h(Userj*** ⊕ ***PW)*** *converted into address* ***Addi***	***h(Userj*** ⊕ ***PW)*** → ***Addi***	***Server***
7.0	*Read* ***Chj*** *in the database at address* ***Addi***	***Addi; Chj***	***Database***
8.0	*Find Hamming distance* ***Hj Chj*** *to* ***Rej***	***Hj***	***Server***
9.0	*Authentication if* ***Hj*** *below the threshold*	***Yes*** *or* ***No***	***Server***

- *Step*-1 to 5: These steps are similar to the one described above, to program the look-up tables. Rather than generating challenges, responses are generated by the APG at the same addresses.
- *Step*-6 to 9: The challenges previously stored in the look-up table are compared with the freshly generated responses. When the Hamming distance ***Hj*** between challenges and responses is bellow a set threshold, CRP error rates are low enough, the authentications are positives.

4 Password Manager with Ternary PUFs

4.1 Ternary APGs

Typically, PUFs experience 3 to 10% CRP error rates. This is usually acceptable for authentication when the length of challenge-response-pairs is high enough. The ternary PUFs are based on three states, (0, 1, X), X being the fuzzy state that is used to characterize the cells that are not predictable. The CRP error rate measured on the non-fuzzy states is significantly lower. The results shown in Fig. 6 are based on the characterization of commercially available SRAM memories. Due to micro-manufacturing differences, the flip-flop of each SRAM cell randomly emerges from power-off-power-on as a 0, or a 1. The vast majority of the cells are responding in a predictable way, therefore acting as a "finger print" of the device. We characterized 32 KB SRAMs, and measured 3 to 5% CRP error rate at each power-off power-on cycles. We then subjected the memory cells with successive power-off power-on cycles, and masked with a ternary state "X" the cells having CRP errors. As a function of the number of queries, the percentage of the cells that are not stable (the one marked "X"), are plotted at the bottom portion of Fig. 6. The corresponding CRP error rates of the non-fuzzy states are plotted at the top portion of Fig. 6. Below 10 cycles, the CRP error rates are in the 0.3 to 5% range, above 80 cycles the rates are in the 0.01% to 0.03% range. In our software implementation, the cells were cycled 100 times during enrollment, 12% of the cells are marked X, 44% are "0"s, and 44% are "1"s. The average CRP error rates of the non-fuzzy cells were at 0.01%. Two possible modifications of the protocol described in Sect. 3 are suggested to process the ternary PUFs.

First method: To store the ternary states in the look up table. In this method, during the enrollment of the password ***PWj***, the server generates streams of trits (0, 1, X) ***Chj*** from the PUF array. ***Chj*** is stored at the address ***Addi*** in the look-up table shown in the block diagram of Fig. 4. In our implementation, the "0" are stored in the look up table as (01), the "1" as (10), and the fuzzy "X" as (11). During authentication, the response generation is binary, only "0"s and "1"s are extracted. Only the non-fuzzy states stored in the look up table are counted for CRP error rates estimate. This protocol is implementable with SRAM PUFs, and with memory PUFs that are based on a parameter ***P***. Within the n-cells that are used for challenge generation, at a particular address ***XY*** of the array, the bottom 1/3 are "0"s, the middle 1/3 are "X"s, and the top 1/3 are "1"s. During response generation, the same n-cells are sorted in the bottom half for the "0"s, and into the top half for the "1"s. In this method, the error rates are extremely low; no error correcting methods are needed. The latency of the enrollment of each new password is here higher, while the authentication cycles are fast.

Second method: Upfront characterization of the entire array. In this method, the entire PUF array is characterized upfront to identify all fuzzy cells of the array, and mark them with an "X". During the enrollment of the passwords, and the corresponding challenge generation, the fuzzy cells located pass the address ***XY*** are ignored, and the n-bits kept are non-fuzzy, i.e. binary. The resulting challenges consist of streams of bits that are stored in the look-up table. During response generation, the same fuzzy cells are ignored, resulting in binary streams that are compared with the challenges stored in

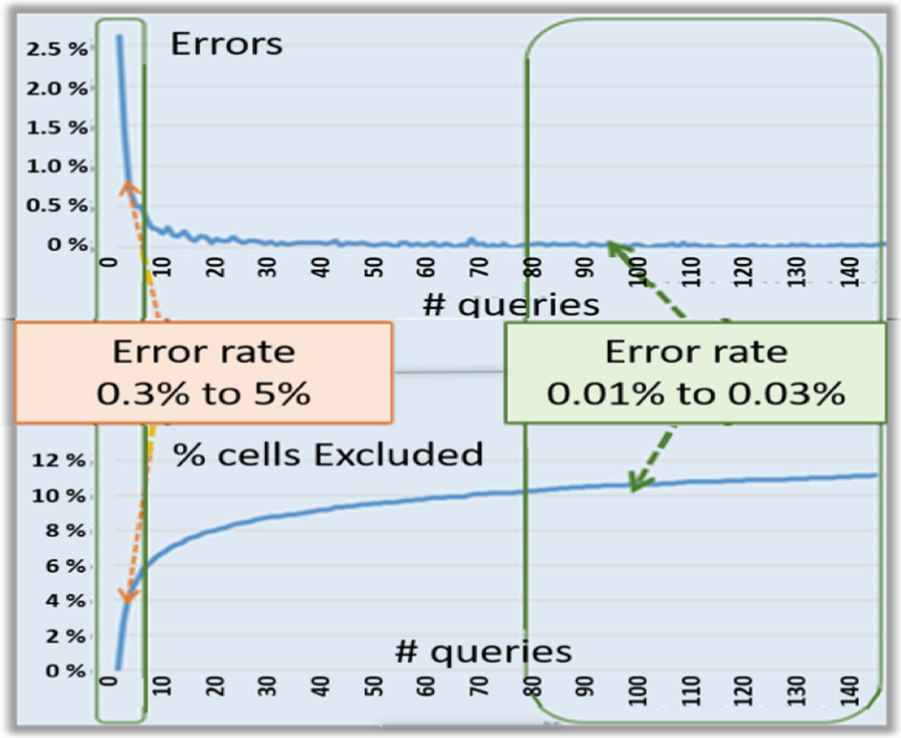

Fig. 6. CRP error rate reduction by masking the fuzzy cells of a SRAM.

the lock up table. The lengthy upfront characterization is done only once, both enrollment and authentication cycles are relatively fast.

The first method is more desirable than the second one, on a security standpoint. The entropy of the stored challenges ***Chj*** is higher. The number of possible configurations of a stream of n trits is 3^n versus 2^n for a stream of bits. Attacks such as password guessing are therefore more complex.

4.2 Double Addressing with Ternary APGs

In the scheme presented above in section IV-A, only the content of what is stored in the address Addi incorporate the patterns generated by the PUFs. Small error rates, below 7%, in CRPs do not add latencies in the matching algorithm. The use of ternary PUF reduces the error rates, and latencies. The question addressed in this section is the potential use of PUF CRPs to replace both password *PWj* by *Chj*, and address *Addi* by the address *Cddi*, also extracted from the PUFs. An error in the address would send the search engine to the wrong part of the look up table, which could be difficult to handle. The block diagram of a ternary PUF password manager with double addressing is shown in Fig. 7.

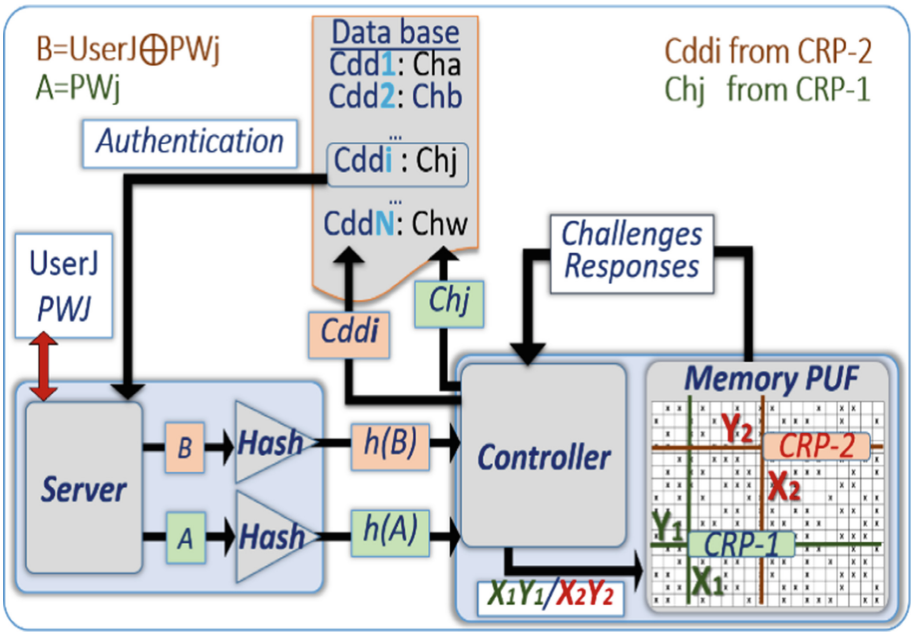

Fig. 7. Block diagram of a password manager with two addressing schemes.

- The *enrollment* of a new UserID-Password pair has two steps. The first one, as described previously, where the APG generated at the address X_1Y_1, the challenges Chj from the password ***PWj***, to store it in the look up table. In the second step, the message digest $h(Userj \oplus PWj)$ is used to find a second address X_2Y_2 in the APG. The challenges extracted at this second address are then used to find the address ***Cddi*** in the look up table where ***Chj*** is stored. For example, if the size of the look-up table is 4096 × 4096, the first stream of the 12 bits of the challenges is used as the first coordinate, and the second stream of 12 bits for the second coordinate.

As a result, not only the content of the look up table is generated from the PUF, but the addressing system is also extracted from the PUF.

- The *authentication* of the user follows a similar method. The password generates, at the address X_1Y_1, the responses ***Rej***, and the message digest $h(Userj \oplus PWj)$ is used to find the address ***Cddi*** from the responses generated at the address X_1Y_1. If ***Cddi/Rej*** is matching the reference ***Addi/Chj***, the authentication is positive.

The use of ternary PUFs, as described below in section IV-C, is important to mitigate potential mismatches between ***Addi*** and ***Cddi*** in case of CRP errors, which could result in pointing to the wrong stream in the look up table.

4.3 Error Matching Algorithms

The two figures of merit of the password generator that are analyzed in this section are the false rejection rate (FRR), and latency of the matching algorithm. The objective of the matching algorithm is to minimize FRR to an acceptable level, while keeping the latency low enough. To keep the user experience enjoyable, we target an FRR below 10^{-4}, and latencies under 3 s.

With PUFs having CRP error rates of 3%, the rate of bad addressing is estimated with statistical models such as Poisson.

- If the CRP error rate is 3%, with 24-bit address, the probability to get at least 1bit error in the address is 51%, and the probability to get at least 2 is 22%. This error rate is prohibitive.
- With ternary PUFs, if the CRP error rate is 0.03%, the probability to get at least 1 bit error is 0.7%, and the probability to get at least 2 is $2.6 \cdot 10^{-5}$. The probability to get at least 3 errors is only $6 \cdot 10^{-8}$.

To find the matching address we designed our codes based on the algorithm shown in Fig. 8. The challenge stored in the look up table, at the address ***Cddi*** is ***Chi***. The authentication is positive when the hamming distance between ***Rej***, the response generated by the APG from the password ***PWj***, and ***Chi*** is below an acceptable threshold ***Hmax***.

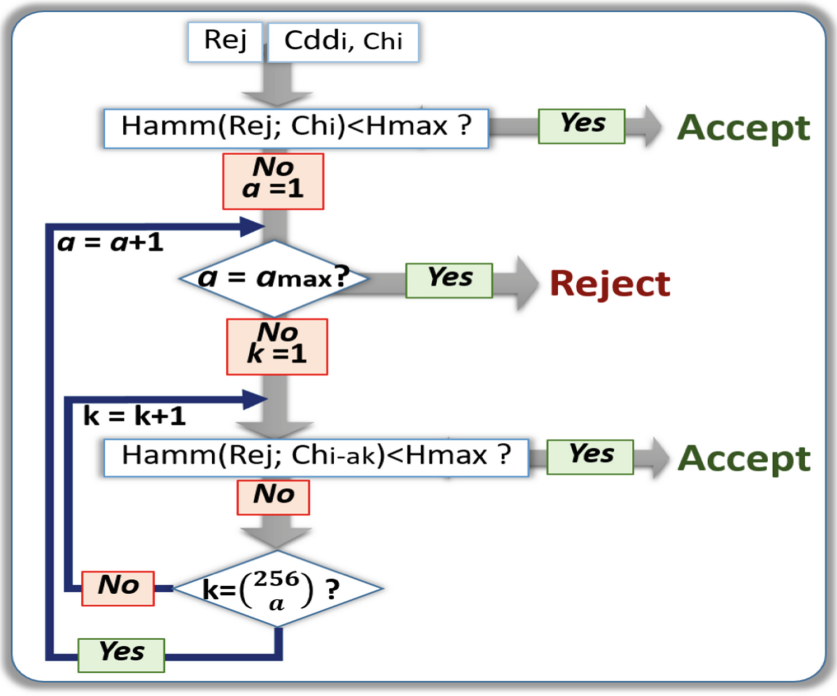

Fig. 8. Algorithm to find the matching address in the look up table.

If negative, all ***k*** challenges ***Chi-1k*** located in the look-up table at the addresses ***Cddi-1k*** are considered. The addresses ***Cddi-ak*** have a Hamming distance of one with the address ***Cddi***. The authentication is positive when the Hamming distance between one of the k challenges ***Chi-1k***, and ***Rej*** is below the threshold ***Hmax***.

If negative, the process iterates to consider all ***k*** challenges; ***Chi-ak*** located in the look up table at the addresses ***Cddi-ak*** having a Hamming distance ***a*** with ***Cddi***. The iteration stops when the Hamming distance reaches the threshold ***amax***, therefore rejecting the authentication.

With 24-bit addresses, there are only 24 addresses with a Hamming distance of 1 within a given response, so step-2 is quick. The likelihood to have to go to the next step, ***a = 2***, is small (If CRP error rate is 0.03%, the probability is $2.6 \cdot 10^{-5}$). The number of addresses with a Hamming distance of 2 with ***Cddi*** is $\binom{24}{2} = 273$.

Based on the ternary SRAM PUF that we characterized, if $a_{max} = 2$, the resulting false rejection rate (FRR) of the protocol is $2 \cdot 10^{-5}$, which is the probability to have the hamming distance at 2 or higher. When $a_{max} = 3$, the resulting FRR is $6 \cdot 10^{-8}$, with the probability to have the hamming distance at 3 or higher. Such levels of FRRs are very small, and acceptable to users. The latencies of the methods are also acceptable.

To find the Hamming distance between two 256-bit long streams usually takes less than 10 clock cycles, or 10 ns for a 1 GHz processor. With the ternary PUFs that we characterized, and ***a*** = 2, the sever needs to find the Hamming distance between ***Rej*** and ***Chi***, then the 24 Hamming distances between ***Chi-1 k*** and ***Chi***, a total of 24 searches, or 250 ns for a 1 GHz processor. With ***a*** = 3, the maximum number of searches needed to iterate is 297, and 2.97 μs latency.

This protocol is also scaling well when the size of the lock-up table increases. For example, look-up tables with 36-bit addresses can store 68G addresses, which is 65,000 larger than a look-up table bit 24-bit addresses. There are only 36 addresses with a Hamming distance of 1 within a given response, and $\binom{36}{2} = 630$ addresses with a Hamming distance of 2.

4.4 Double Addressing with Two Ternary APGs

The same method as the one described above can be implemented with two different ternary APGs, as shown in Fig. 9. One APG, in green in the figure, generates the challenges *Chj* to be stored in the look-up table based on the passwords *PWj*. The second APG, in red in the figure, generates the addresses *Cddi* in the look-up table based on *Userj* ⊕ *PWj*. To enhance security the two APGs can use different PUF technologies. The first APG for *Chj* generation is less sensitive to CRP error rates, and does not have to be based on Ternary PUFs. The control circuitry shown in Fig. 8 can also be separated into two different circuits to increase the protection of the system.

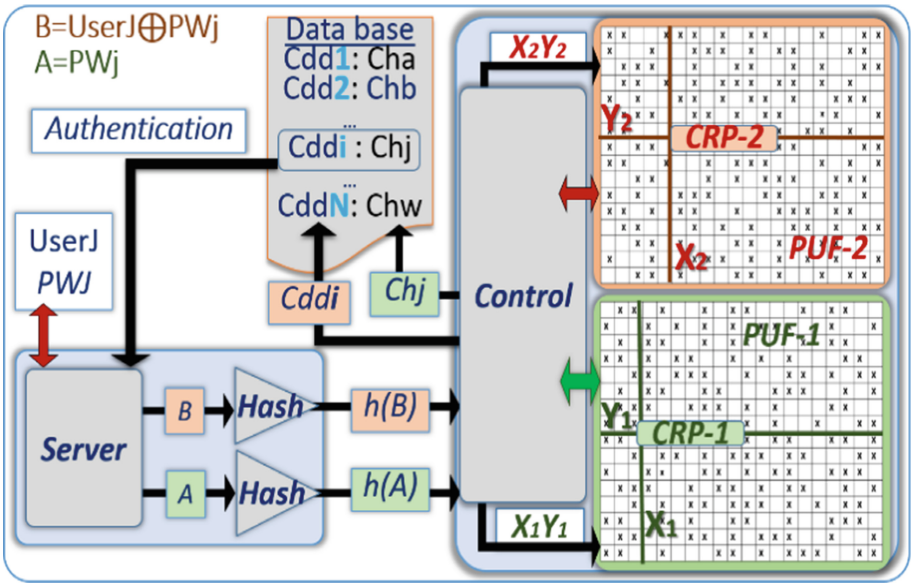

Fig. 9. Block diagram of a password manager with two ternary APG, driving two addressing schemes.

4.5 Double Addressing with PUF Based on Ring Oscillators

PUF designed with component different than SRAM were studied, such as the ones designed with gate delays and ring oscillators. The diagram of such password manager is shown in Fig. 10. Pairs are formed to group the stream of bits, which are generated from the message digest. Each pair generates a challenge (or response), in such a way that a message digest of 512 bits can generate 256 CRPs. Ternary states can be generated with ring oscillator PUFs. During challenge generation, a particular pair of ring oscillators generates the state "0" or "1" when the difference between the numbers of oscillation per milli-second of the two rings is large enough. When this difference is below a given threshold, the pair generates a fuzzy state "X". During the response generation, all pairs are generating only "0", or "1".The CRP matching estimate is based on the non-fuzzy pairs that were generated during challenge generation, while the pairs with fuzzy state are ignored. This results in a reduction of the CRP error rates.

4.6 Entropy Enhancement

In the software implementation presented above, the randomness, and entropy was increased with longer message digests. The protocols presented in Sects. 3 and 4 are based on the conversion of message digest into addresses. The example given is the one of an APG based on an array of 4,096 × 4096 cells. The first 12 bits are used to find the ***X*** coordinate, and the next 12 bits are used to find the ***Y*** coordinate. The n-cells located after that address are used to generate PUF challenges (or responses) that consist of n-bits. Message digests contain long streams of bits, typically 512, so only 24

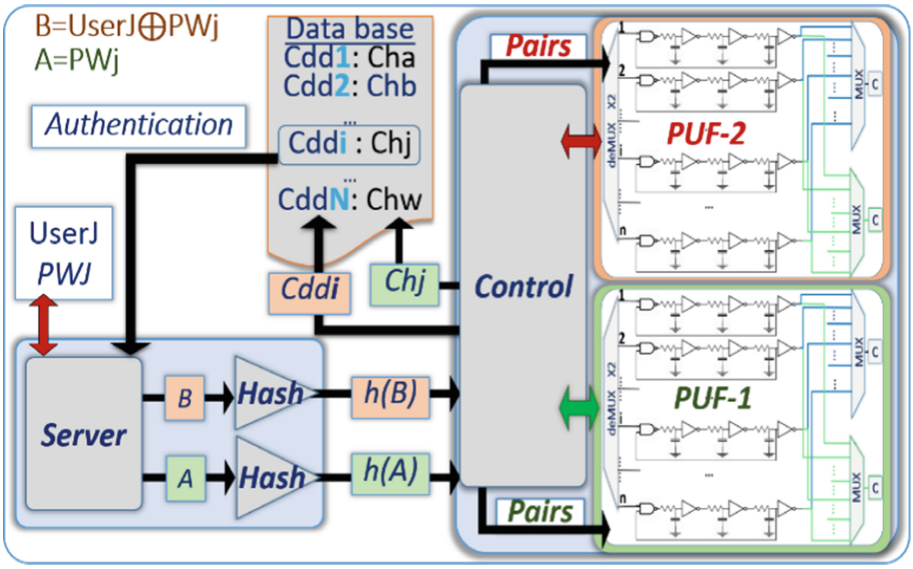

Fig. 10. Block diagram of a password manager with two PUFs based on ring oscillator driving two addressing schemes.

bits out of these streams are used in the protocol. The opportunity to increase randomness is to find multiple addresses in the APG from the message digests, and to reduce the number of cells involved at each address. Assuming that the size of the streams is kept at **n**, a total of f addresses in the APG are selected from each message digest, and m cells are selected by address to generate the n-bit challenges (or responses), with $n = fm$. For example if $n = 512$, and $f = 16$ (addresses selected from the message digests), then $m = 32$ bits are generated at each address. The number of bits of the message digest used for addressing in the protocol is now $16 \times 24 = 384$. There are 2^{24} different 24-bit possible addresses, and entropy of 24 ($\text{Log}_2(2^{24}) = 24$). The number of possible ways to pick 16 addresses having 24 bits is $(2^{24})^{16}$, an entropy of 384.

5 Conclusion and Future Work

As part of the design of password managers, hash functions are effective to replace the storage of the password into message digests. Hash functions can also protect the address in look up tables where these message digests are stored. Such methods offer reasonable protections against the exposure of the look up tables to malicious parties. However, certain attacks, such as password guessing, are still effective against such schemes, because the hash functions are usually known, and public. The use of encrypted hashing functions is stronger, but can create a false sense of security when the cryptographic keys are also lost to the hackers. The combination of hashing functions with addressable PUF generators allows the replacement of the message

digests of the passwords by challenge-response-pairs, which provide an additional layer of security. Even if the hash functions are known, the PUFs, which can be unclonable, random, and unique, mitigate attacks against lost look up tables. The task of malicious entities is thereby much more challenging; in order to break the system they have to steal the hardware containing the PUFs, or find a way to download a representation of the PUFs. The suggested ternary PUF architecture handles the fuzzy states, which reduces the challenge-response pair error rates, making the method more reliable, and reduces latencies. The proposed matching algorithms are critical to protect the addresses in look up tables using ternary PUFs, with low false rejection rates (FRR) at low computing latencies. The implementation using ternary SRAM PUF is showing FRR lower than the part per million, and latencies in the microsecond range. This suggested additional protection can use a single PUF, two PUFs, or PUFs based on different technologies. The hardware-software implementation that was developed, and described in this paper, uses existing components, and is extremely low cost. This method has the objective to complement other cryptographic protections, not to replace them. For example, the use of multi-factor authentication, and the encryption of the passwords with public key infrastructure (PKI) are recommended.

Moving forward the suggested algorithms should be further improved, and use stronger PUFs beside SRAMs. The functionality of commercial password managers, including password expiration and replacement, verification of their strength, firewall protection, and others, will be incorporated, and tested in future prototypes. SRAM PUFs are relatively easy to break, and the proposed password managers should be designed with tamper resistant PUFs. Examples of stronger candidates include memristor and ReRAM based low power PUFs [51–53, 56, 57]. Finally, the development boards used in this work should be replaced with application specific integrated circuits (ASIC), which incorporate hardware protection against differential power analysis, and other side channel attacks, and using a cryptoprocessor, to provide encryption of the passwords with PKI, and hardware implementation of SHA-3.

Acknowledgments. The author is thanking the contribution of several graduate students at Northern Arizona University, in particular Sareh Assiri, Christopher Philabaum, Duane Booher, Vince Rodriguez, Ian Burke, and Mohammad Mohammadi.

References

1. Jeong, Y.S., Park, J.S., Park, J.H.: An efficient authentication system of smart device using multi factors in mobile cloud service architecture. Int. J. Commun. Syst. **28**(4), 629–674 (2014)
2. Saxena, N., Choi, B.J.: State of the art authentication, access control, and secure integration in smart grid. Energies **8**(10), 11883–11915 (2015)
3. Zhang, M., Zhang, J., Zhang, Y.: Remote three-factor authentication scheme based on Fuzzy extractors. Secur. Commun. Netw. **8**(4), 682–693 (2014)
4. Keane, J.: Security researcher dumps 427 million hacked Myspace passwords, July 2016. https://www.digitaltrends.com/social-media/myspace-hack-password-dump/
5. Morgan, S.: 2017 Cybercrime Report, Cybercrime damages will cost the world $6 trillion annually by 2021. Cybersecurity Ventures, Herjavec Group (2017)

6. Target: Data stolen from up to 70 million customers: USA Today. https://www.usatoday.com/story/money/business/2014/01/10/target-customers-data-breach/4404467/
7. Blocki, J., Harsha, B., Zhou, S.: On the economics of offline password cracking. In: IEEE Symposium on Security and Privacy (SP) (2018)
8. Lee, H.W., Noh, M.J., Chol, H.M., Feng, X.: Password system, method of generating password, and method of checking password. Patent application US2009/0228977A1 (2009)
9. Fitzgerald, J.: Systems and methods for providing a covert password manager. US patent 9,571,487 B2 (2017)
10. Harper, R.: STARTS password manager. Patent publication, US2005/0125699A1 (2005)
11. Mimlitsch, J.: User-administrated single sign-on with automatic password management for WEB server authentication. Patent publication US2007/0226783A1 (2007)
12. Safriel, M.: Portable password manager. Patent publication US2004/0193925A1
13. Tsai, C.-S., Lee, C.-C., Hwang, M.-S.: Password authentication schemes: current status and key issues. IJ Network Security (2006)
14. Zhang, Z., Yang, K., Hu, X., Wang, Y.: Practical anonymous password authentication and TLS with anonymous client authentication. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1179–1191. ACM (2016)
15. Forler, C., List, E., Lucks, S., Wenzel, J.: Overview of the candidates for the password hashing competition. In: Conference on Passwords, Springer (2014)
16. Tsai, J.-L.: Efficient multi-server authentication scheme based on one-way hash function without verification table. Comput. Secur. **27**(3–4), 115–121 (2008)
17. Janzen, W.S.: Iterated password hash systems and methods for preserving password entropy. US patent 8,769,637 B2 (2014)
18. Pappu, R., Recht, B., Taylor, J., Gershenfield, N.: Physical one-way functions. Science **297** (5589), 2026–2030 (2002)
19. Jin, Y.: Introduction to hardware security. Electronics **4**, 763–784 (2015). https://doi.org/10.3390/electronics4040763
20. Gassend, B., et al.: Silicon physical randomness. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS'2002, pp. 148–160 (2002)
21. Naccache, D., Frémanteau, P.: Unforgeable identification device, identification device reader and method of identification. Patent US5434917, August 1992
22. Gao, Y., et al.: Emerging physical unclonable functions with nanotechnologies. IEEE. https://doi.org/10.1109/access.2015.2503432
23. Delavor, M., et al.: PUF based solution for secure communication in advanced metering infrastructure. ACR Publication (2014)
24. Herder, C., Yu, M.-D., Koushanfar, F., Devadas, S.: Physical unclonable functions and applications: a tutorial. Proc. IEEE **102**(8), 1126–1141 (2014)
25. Maes, R., Verbauwhede, I.: Physically unclonable functions: a study on the state of the art and future research directions. In: Towards Hardware-Intrinsic Security (2010)
26. Wang, D., Zhang, Z., Wang, P., Yan, J., Huang, X.: Targeted online password guessing: an underestimated threat. In: Proceedings of the ACM CCS, pp. 1242–1254 (2016)
27. Pass the Hash attack: Microsoft research as on 12 August 2015. http://www.microsoft.com/PTH
28. Bonneau, J., Herley, C., van Oorschot, P., Stajano, F.: Passwords and the evolution of imperfect authentication. Commun. ACM **58**(7), 78–87 (2015)
29. Taniguchi, M., Shiozaki, M., Kubo, H., Fujino, T.: A stable key generation from PUF responses with a fuzzy extractor for cryptographic authentications. In: 2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE), Tokyo (2013)
30. Price, N.E., Sherman, A.T.: How to generate repeatable keys using physical unclonable functions, correcting PUF errors with iteratively broadening and prioritized search

31. Boehm, H.M.: Error correction coding for physical unclonable functions. In: Austrochip, Workshop in Microelectronics (2010)
32. Yu, M., Devadas, S.: Secure and robust error correction for physical unclonable functions. IEEE Design Test Comput. Verifying Phys. Trustworthiness ICs Syst. **27**, 48–65 (2010)
33. Kang, H., Hori, Y., Katashita, T., Hagiwara, M., Iwamura, K.: Cryptographie key generation from PUF data using efficient fuzzy extractors. In: 16th International Conference on Advanced Communication Technology, Pyeongchang (2014)
34. Becker, G.T., Wild, A., Güneysu, T.: Security analysis of index-based syndrome coding for PUF-based key generation. In: 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), Washington, DC (2015)
35. Hiller, M., Weiner, M., Rodrigues, L., Birkner, M., Sigl, G.: Breaking through fixed PUF block limitations with differential sequence coding and convolutional codes. In: TrustED'13 (2013)
36. Paar, C., Pezl, J.: Understanding Cryptography - A Text Book for Students and Practitioners. Springer, Berlin (2011)
37. Mel, H.X., Baker, D.: Cryptography Decrypted. Addison-Wesley, Boston (2001)
38. Pfleeger, C.P., et al.: Security in Computing, 5th edn. Prentice Hall, Upper Saddle River (2015)
39. Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A., Miller, A.: On scaling decentralized blockchains. In: Springer International Conference on Financial Cryptography and Data Security, Berlin, Heidelberg (2016)
40. Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., Saxena, P.: A secure sharing protocol for open blockchains. In: ACM SIGSAC Conference on Computer and Communication Security (2016)
41. Eyal, I., Gencer, A.E., Sirer, E.G., Renesse, R.V.: Bitcoin-NG: a scalable blockchain protocol. In: NSDI (2016)
42. Dorri, A., Kanhere, S.S., Jurdak, R.: Blockchain in internet of things: challenges and solutions. arXiv preprint arXiv: 1608.05187 (2016)
43. Gervais, A., Karame, G.O., Wüst, K., Glykantzis, V., Ritzdorf, H., Capkun, S.: On the security and performance of proof of work blockchains. In: ACM SIGSAC Conference on Computer and Communications Security (2016)
44. Zheng, Z., Xie, S., Dai, H.-N., Wang, H.: Blockchain challenges and opportunities: a survey. Int. J. Web Grid Serv. 1–25 (2016)
45. Cambou, B.: Addressable PUF generators for database-free password management system. In: CryptArchi (2018)
46. Holcomb, D.E., Burleson, W.P., Fu, K.: Power-up SRAM state as an identifying fingerprint and source of TRN. IEEE Trans. Comput. **57**(11), 1198–1210 (2008)
47. Maes, R., Tuyls, P., Verbauwhede, I.: A soft decision helper data algorithm for SRAM PUFs. In: 2009 IEEE International Symposium on Information Theory (2009)
48. Christensen, T.A., Sheets II, J.E.: Implementing PUF utilizing EDRAM memory cell capacitance variation. Patent No.: US 8,300,450 B2, 30 October 2012
49. Prabhu, P., Akel, A., Grupp, L.M., Yu, W.-K.S., Suh, G.E., Kan, E., Swanson, S.: Extracting device fingerprints from flash memory by exploiting physical variations. In: 4th International Conference on Trust and Trustworthy Computing, June 2011
50. Plusquellic, J., et al.: Systems and methods for generating PUF's from non-volatile cells. WO20151056887A1 (2015)
51. Chen, A.: Comprehensive assessment of RRAM-based PUF for hardware security applications. IEEE (2015). 978-1-4673-9894-7/15/IEDM

52. Cambou, B., Afghah, F., Sonderegger, D., Taggart, J., Barnaby, H., Kozicki, M.: Ag conductive bridge RAMs for physical unclonable functions. In: 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean (2017)
53. Korenda, A., Afghah, F., Cambou, B., A secret key generation scheme for internet of things using ternary-states ReRAM-based physical unclonable functions. In: Submitted to International Wireless Communications and Mobile Computing Conference (IWCMC 2018)
54. Zhu, X., Millendorf, S., Guo, X., Jacobson, D.M., Lee, K., Kang, S.H., Nowak, M.M., Fazla, D.: PUFs based on resistivity of MRAM magnetic tunnel junctions. Patents US 2015/0071432 A1, March 2015
55. Vatajelu, E.I., Di Natale, G., Barbareschi, M., Torres, L., Indaco, M., Prinetto, P.: STT-MRAM-based PUF architecture exploiting magnetic tunnel junction fabrication-induced variability. ACM Trans. **13**(1), 5 (2015)
56. Cambou, B., Orlowski, M.: Design of PUFs with ReRAM and ternary states. CISR 2016, April 2016
57. Cambou, B., Afghah, F.: Physically unclonable functions with multi-states and machine learning. In: 14th International Workshop on Cryptographic Architectures Embedded in Logic Devices (CryptArchi), France (2016)
58. Cambou, B., Telesca, D.: Ternary computing to strengthen information assurance, development of ternary state based public key exchange. In: Computing Conference, IEEE, London, July 2018
59. Cambou, B., Flikkema, P., Palmer, J., Telesca, D., Philabaum, C.: Can ternary computing improve information assurance? Cryptography, MDPI, February 2018
60. Cambou, B.: Physically unlonable function generating systems and related methods. US patent 9,985,791 (2018)
61. Cambou, B.: Encoding ternary data for PUF environment. US patent 10,050,796 (2018)