applied sciences

MDPI

Article

A Novel Hardware Security Architecture for IoT Device: PD-CRP (PUF Database and Challenge–Response Pair) Bloom Filter on Memristor-Based PUF

Jungwon Lee [1], Seoyeon Choi [1,2], Dayoung Kim [1,2], Yunyoung Choi [1] and Wookyung Sun [3,*]

[1] Department of Electronic and Electrical Engineering, Ewha Womans University, Seoul 03760, Korea; jungwon0736@ewha.ac.kr (J.L.); irischoi518@gmail.com (S.C.); zlaek02@ewhain.net (D.K.); chidbsdud75@ewhain.net (Y.C.)
[2] Smart Factory Multidisciplinary Program, Ewha Womans University, Seoul 03760, Korea
[3] Department of Electrical and Computer Engineering, Seoul National University, Seoul 08826, Korea
* Correspondence: wksun@snu.ac.kr

Received: 24 August 2020; Accepted: 21 September 2020; Published: 24 September 2020

Abstract: Because the development of the Internet of Things (IoT) requires technology that transfers information between objects without human intervention, the core of IoT security will be secure authentication between devices or between devices and servers. Software-based authentication may be a security vulnerability in IoT, but hardware-based security technology can provide a strong security environment. Physical unclonable functions (PUFs) are hardware security element suitable for lightweight applications. PUFs can generate challenge–response pairs(CRPs) that cannot be controlled or predicted by utilizing inherent physical variations that occur in the manufacturing process. In particular, the pulsewidth-based memristive PUF (pm-PUF) improves security performance by applying different write pulse widths and bank structures. Bloom filters (BFs) are probabilistic data structures that answer membership queries using small memories. Bloom filters can improve search performance and reduce memory usage and are used in areas such as networking, security, big data, and IoT. In this paper, we propose a structure that applies Bloom filters based on the pm-PUF to reduce PUF data transmission errors. The proposed structure uses two different Bloom filter types that store different information and that are located in front of and behind the pm-PUF, reducing unnecessary access by removing challenges from attacker access. Simulation results show that the proposed structure decreases the data transmission error rate and reuse rate as the Bloom filter size increases; the simulation results also show that the proposed structure improves pm-PUF security with a very small Bloom filter memory.

Keywords: physical unclonable function; bloom filter; hardware security; PUF; memristor

1. Introduction

The advent of the Internet of Things (IoT) has led to a web of connections that provides network infrastructure and shares a massive amount of private information among smart devices [1]. The requirements for trustworthy communication have been one of the biggest concerns [2]. In recent years, it has been demonstrated that hardware-based security systems are more secure than software-based security systems due to innate architectural risks in software [3]. Hardware-based security systems typically store digital keys in non-volatile storage and the keys play an important role in encrypting shared information or in processing access authentication. However, it has been shown that the typical method is vulnerable to invasive attacks on physical systems to reveal their

secret information [4–6]. To defeat such potential threats, physical unclonable functions (PUFs) have been introduced as a promising alternative. PUFs primarily utilize inherent physical variations that naturally occur during hardware device manufacturing [7]. The major advantage of PUFs is that they are easy to build but practically impossible to duplicate since the formation of such physical variations is unpredictable and uncontrollable, even under the same conditions. In general, PUFs implement challenge–response protocols rather than extracting stored keys. When a challenge is applied to a PUF, a corresponding response is generated. Each response depends on the unique features of a PUF instance [8]. Therefore, a set of challenge–response pairs (CRPs) is regarded as the fingerprint of a PUF-integrated device; with unique CRPs, PUFs can be used in security applications such as authentication and identification as well as key generation [9]. However, a PUF generates and transmits a response to a challenge even if the challenge comes from an attacker [10], so eventually, attackers can obtain CRP information from a hardware PUF.

A Bloom filter (BF) is a simple and efficient bit vector structure that represents the membership information of a set of elements [11–15]. Bloom filters are used in various lookup systems to improve their performance by avoiding access to unnecessary data [16,17]. Bloom filters can reduce memory usage according to many studies in the storage field [18,19] and can remove data duplication [20]. The Bloom filters are also applied to many security fields [21,22]. A Bloom filter implemented in hardware can accelerate information retrieval and save memory [23–25]. Bloom filters are used in a wide variety of applications and will increasingly be the subject of various research studies in the future [26–28].

This paper describes a security approach that exchanges only the correct CRPs with the authentication server by removing unnecessary challenge access from attackers. The proposed approach is based on a pulsewidth-based memristive PUF (pm-PUF) [29]. To increase the accuracy of the exchange of CRP information between the pm-PUF and its authentication server, we propose to add a small-memory Bloom filter. When the CRPs of authentication server are stored in the Bloom filter and applied to the pm-PUF, its security is improved by removing unnecessary attacker access to CRPs and avoiding duplicate CRP usage.

The remainder of this paper is organized as follows. Section 2 describes related works including variant PUFs and Bloom filters. Section 3 introduces our proposed structure. Section 4 describes the simulation data set. Section 5 discusses the performance evaluation results, and Section 6 briefly concludes the paper.

2. Related Work

2.1. Conventional PUF and Memristor-Based PUF

Subsequent to the introduction of PUFs, various PUF structures have been proposed [30–33], analyzed, and implemented. Conventional PUFs like arbiter PUFs [30], ring oscillator PUFs [31], SRAM PUFs [32], and latch PUFs [33] seem attractive for their easy construction. However, as manufacturing technology trends toward smaller feature size, those structures are likely to face future physical limitations on scaling down [34]. Thus, emerging nanoelectronic devices such as memristor [35], PCM [36], and STT-MRAM [37] have been suggested. In particular, memristors are expected to be a suitable candidate for PUF implementation owing to their high fabrication process sensitivity and their compatibility with modern CMOS fabrication processes. The two-terminal memristor enables ultra-high integration density at low cost when organized into a cross-point array architecture. Many studies [38–40] that take advantage of memristors have been conducted to build reliable and secure PUFs. In [38,39], each memristor has a random 1 or 0 logic state at the same write time because each memristor has a different write time as a result of fabrication variations. In [40], the PUF sets the operating condition at a switching probability of 50%. Building a novel PUF relies on key extraction from the circuit implementation of the architecture. Applications of memristor-based PUFs have some challenges to resolve. One of challenges is that the number of CRPs increases in proportion to

the number of unit cells and ultimately in proportion to the array size. Consequently, several PUF architectures have been introduced and employed to obtain as many CRPs as possible with a limited array size. To construct a practical and secure PUF, methods have been proposed to evaluate PUF performance using performance metrics such as randomness, diffuseness, uniqueness, and steadiness which are defined as follows [41]:

- **Randomness**: Randomness indicates the frequency of 0 and 1 in the responses of a PUF instance. An ideal PUF should have these frequencies in balance; the ideal value of randomness is 1.
- **Diffuseness**: Diffuseness indicates the probability of identical responses from different challenges applied to the same PUF instance. Diffuseness can be calculated from the intra-Hamming distance (intra-HD) of all possible responses from a PUF instance. An ideal value of diffuseness is 1.
- **Uniqueness**: Uniqueness is the degree of response difference from a same challenge applied to different PUF instances. Uniqueness can be calculated from the inter-Hamming distance (inter-HD) of responses from different PUF instances. The ideal value of uniqueness is 1.
- **Steadiness**: Steadiness indicates whether the PUF operates stably enough to expect the same response over several challenges when subjected to environmental changes.

In general, the attacker collects a very large number of CRPs, determines the correlation between the challenge and the response, and performs a modeling attack. In earlier research, CRP has been obfuscated or a security protocol has been applied to prevent such attacks. By applying the random hash function to the challenge and response of PUF, the CRP of the actual PUF is hidden [8], and Majzoobi et al. [42] selected the substring of a response to minimize the exposure of the response. Ye et al. [43] proposed an isolated primitive that the security agent monitors the CRP request growth rate and separates the DoS attackers requesting a large amount of CRP into a gray/blacklist to delay or completely block response transmission. The optimal time delay algorithm (OTDA) [44] significantly improves PUF reliability under various operating conditions. OTDA has been shown to more efficiently enhance of dynamic Ring Oscillator PUF(d-ROPUF) ability to generate a fairly large set of trusted secret keys to protect PUF structures from new cyberattacks, including machine learning and modeling attacks.

2.2. Bloom Filter

A Bloom filter [45] represents the membership information of a set using a simple bit-vector-based data structure. Bloom filters have recently been applied in many applications [46–49] and proposed in others [50–52]. A Bloom filter uses a hash function k on an m bit array that is initialized to 0 to store the elements n of a set into an array. The Bloom filter has two different operations: programming and querying. In programming, the Bloom filter sets the corresponding cell from 0 to 1 using the hash indices of the element n of the set obtained by the hash function k. Querying is performed to determine whether a given input is included in the set. The query results for a standard Bloom filter (SBF) are positive or negative. In querying, if all the cells corresponding to the hash indices are 1, then the Bloom filter produces a positive result, which means that the given input is in the elements of the set. If at least one of the cells corresponding to the hash index is 0, the Bloom filter produces a negative result, indicating that the input is not an element of the set. However, the Bloom filter can produce a false positive result even if the given input is not an element of the set but never produces a false negative result. The probability of false positives is as follows [53,54]:

$$f = (1-p)^k = (1-(1-\frac{1}{m})^{kn})^k \quad (1)$$

where p is the probability that a cell in the Bloom filter remains in state 0 after programming the Bloom filter as element n, assuming that the hash function k is perfectly random.

The optimal number of hash functions k that minimizes the false positive probability is as follows [53,54]:

$$k = \frac{m}{2^{\lceil \log_2 n \rceil}} ln2 \quad (2)$$

A counting Bloom filter (CBF) [55] stores multiple bits in one cell; the number of cells indicates the number of mapped elements. The CBF structure improves on the SBF disadvantage that its elements are impossible to delete. However, CBFs consume a massive amount of memory and can produce false negative results. The quaternary Bloom filter (QBF) [56] is a proposed structure that can eliminate false negative CBF results. A QBF has two bits per cell and allows the elements of a set to be inserted and deleted. When more than two elements are mapped to a cell, the corresponding cell is represented with an X. In querying, if all mapped cells of a given input are X, the QBF result is indeterminable, so the given input cannot be identified as positive or negative. In deleting, if all cells mapped to an element are X, the element cannot be deleted.

3. Proposed Structure

PUFs are widely used to protect sensitive information in a data transmission environment. However, if attackers access a PUF in an untrusted environment, they will eventually obtain CRP information from the PUF. Our proposed architecture improves the security of a pm-PUF by removing the PUF access of the attacker with Bloom filter querying. In this paper, we propose a structure to improve security between authentication servers and devices by applying two different types of Bloom filters to a PUF. The proposed structure applies the Bloom filter to a pm-PUF. The pm-PUF is based on a memristive PUF with its wide resistance variation under various pulses. The proposed structure reduces unnecessary CRP transmission between the device and the authentication server by applying a Bloom filter and avoids reuse by verifying the duplicates of the CRP. Figure 1 shows the proposed PUF database Bloom filter (PDBF) and challenge–response pair Bloom filter (CRPBF).

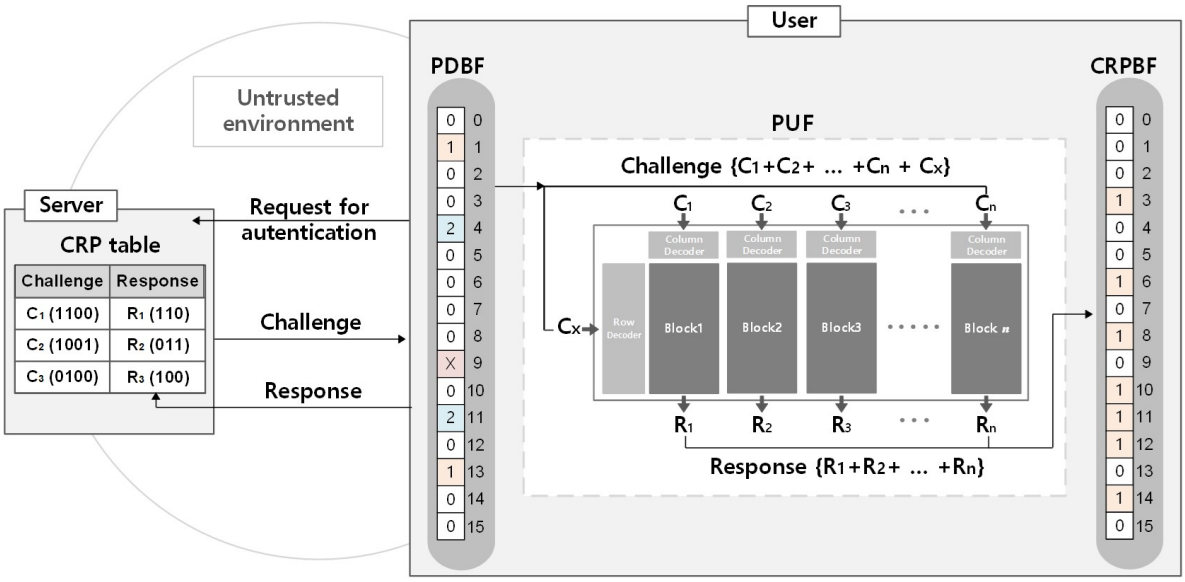
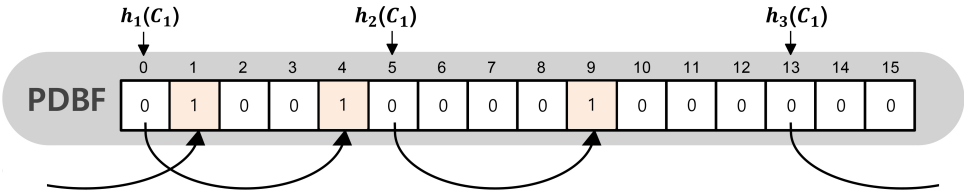


Figure 1. Proposed physical unclonable function database Bloom filter (PDBF) and challenge–response pair Bloom filter (CRPBF).

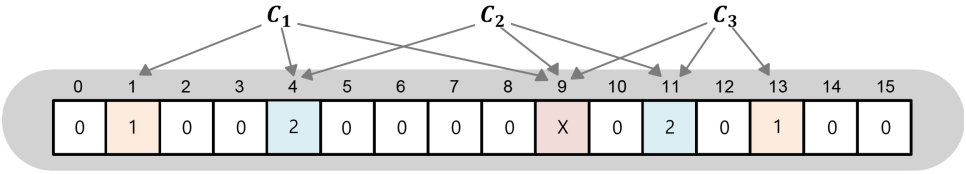
The PDBF, located in front of the pm-PUF, stores all the challenge information in the CRP table of the authentication server to prevent a challenge access of the attacker to the pm-PUF. Additionally, the PDBF improves the security of pm-PUF by removing the challenge information of the CRP used. The CRPBF behind the pm-PUF stores all CRP information in the CRP table of the authentication server to prevent transmission of responses that do not exist in the authentication server.

3.1. PUF Database Bloom Filter (PDBF)

The role of the PDBF is to store all the challenge information and challenge length held on the authentication server to improve the security of the pm-PUF. The PDBF is the structure referred to in [52,56,57] in which one cell stores two bits, and the false negative of the CBF is removed. Additionally, the cell positions are shifted from the hash indices by the challenge length. Figure 2 shows the PDBF programming procedure. As an example of the programming procedure, let $S_{crp} = \{C_1, C_2, C_3\}$ represent a challenge in the authentication server. Assume that the challenge length is four bits and the number of hash indices k is three. Program it in the cell positions shifted by four bits of challenge length from the hash indices. The hash indices of C_1 are 0, 5, and 13, and the challenge length is 4, so C_1 is programmed in the cell positions 4, 9, and 1 shifted by 4 as in Figure 2a. If two challenges are mapped to cells 4 and 11 as in Figure 2b, the cells are represented by 2. If more than two challenges are mapped as shown for cell 9 in Figure 2b, the corresponding cell is represented by X.



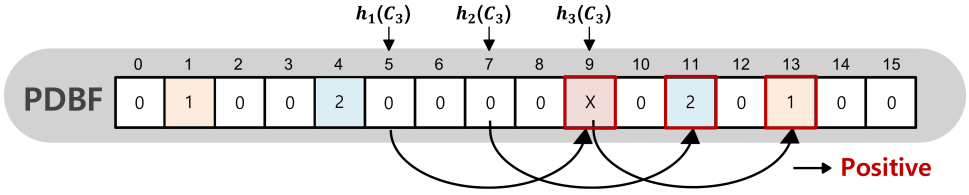
(**a**) Shift from the hash indices by the challenge length of C_1



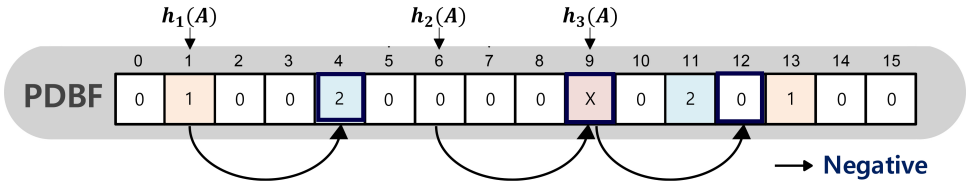
(**b**) Insert all challenges into the PDBF

Figure 2. Programming procedure for a PDBF.

The purpose of querying the PDBF is to determine whether the input challenge exists in the authentication server. Figure 3 shows the PDBF querying procedure. If all of the cells contain 1, 2, or X, the result of PDBF is termed a positive. If any one of the cells contains 0, the result of PDBF is termed a negative. For example, in Figure 3a, the challenge C_3 of input is queried. The hash indices of C_3 are 5, 7, and 9, and the challenge length of C_3 is 4, so C_3 is queried to the cell positions 9, 11, and 13 shifted by 4. Since the values of the corresponding cells are X, 2, and 1, it is termed positive and challenge C_3 is transmitted to the pm-PUF. The PDBF querying for an attack challenge A is shown in Figure 3b. Assuming the hash indices of A are 1, 6, and 9, and the challenge length of A is 3. A is queried to the cell positions 4, 9, and 12 shifted by 3. The value in cell 12 of the PDBF is 0, so the result is termed a negative. When the PDBF produces a negative result, the input challenge is dropped and is not transmitted to the pm-PUF.



(**a**) The PDBF querying for an input challenge C_1



(**b**) The PDBF querying for an attack challenge A

Figure 3. Querying procedure for a PDBF.

Another role of the PDBF is to avoid the reuse of CRPs. The PDBF deletes a used challenge and produces a negative result when the same challenge is presented again. The deletion procedure for the PDBF is performed after the response is transmitted to the authentication server as a positive result of CRPBF in Section 3.2 and the transmitted response is successfully authenticated. If the cell values are 1 and 2, they are reduced to 0 and 1, respectively. If the cell contains X, it is not deleted. For example, assume that CRPBF produces a positive result for challenge C_3 and transmitted response succeeds in authentication. The hash indices of C_3 are 5, 7, and 9, and the cell positions shifted by the challenge length are 9, 11, and 13. Since the values of cells 11 and 13 are 2 and 1, respectively, they are changed to 1 and 0. However, since the value of the cell 9 is X, it is not deleted. The PDBF procedure is shown in Algorithm 1.

3.2. Challenge–Response Pair Bloom Filter (CRPBF)

The role of the CRPBF is to verify if the output response from the pm-PUF exists in the authentication server; the CRPBF is a Bloom filter that stores the CRPs contained in an authentication server. The hash index programmed into the CRPBF is obtained by concatenating the challenges and responses in the CRP table of authentication server. Figure 4 shows a CRPBF programming example. If a challenge C_1 is 1100 and its response R_1 is 110 in the authentication server, their concatenation is 1,100,110. The hash indices of 1,100,110 are 3, 6, and 8, and the corresponding cells of the CRPBF are set to 1.

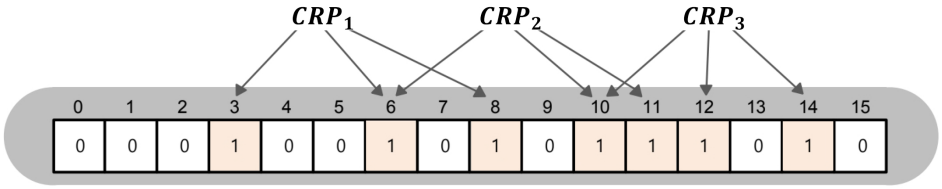


Figure 4. Programming procedure for a CRPBF.

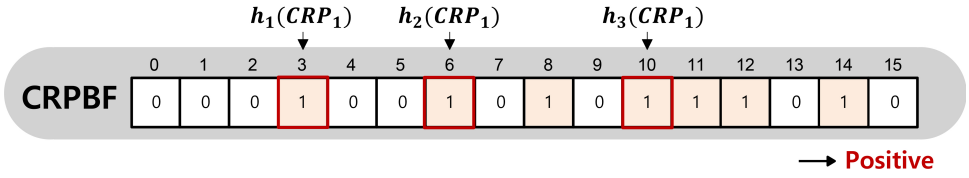
Algorithm 1: PDBF procedure

Function *Programming_PDBF*($challenge$, $length_{challenge}$)
  for $i = 1$ *to* k **do**
    if BF[$h_i(challenge) + length_{challenge}$] == 0 || BF[$h_i(challenge) + length_{challenge}$] == 1 **then**
      BF[$h_i(challenge) + length_{challenge}$] = BF[$h_i(challenge) + length_{challenge}$] + 1;
    end
    else if BF[$h_i(challenge) + length_{challenge}$] == 2 **then**
      BF[$h_i(challenge) + length_{challenge}$] = X;
    end
    else if BF[$h_i(challenge) + length_{challenge}$] == X **then**
      continue;
    end
  end
end

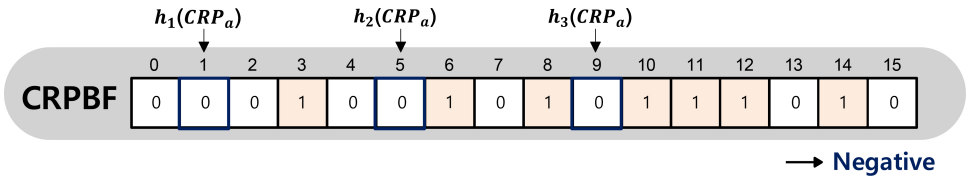
Function *Querying_PDBF*($challenge$, $length_{challenge}$)
  for $i = 1$ *to* k **do**
    if BF[$h_i(challenge) + length_{challenge}$] == 0 **then**
      return negative;
    end
  end
  return positive;
end

Function *Deleting_PDBF*($challenge$, $length_{challenge}$)
  for $i = 1$ *to* k **do**
    if BF[$h_i(challenge) + length_{challenge}$] == 1 || BF[$h_i(challenge) + length_{challenge}$] == 2 **then**
      BF[$h_i(challenge) + length_{challenge}$] = BF[$h_i(challenge) + length_{challenge}$] - 1;
    end
    else
      continue;
    end
  end
end

Figure 5 shows a CRPBF query example. In querying the CRPBF, the hash indices are a concatenation of the input challenge and the output response from the pm-PUF. In Figure 5a, it is assumed that the output response R (110) obtained by the pm-PUF from the challenge C_1 (1100) of input produces a positive result from the PDBF. The hash indices of 1,100,110, the concatenation of challenge C_1 (1100), and response R (110) are 3, 6, and 10. In Figure 5a, since the corresponding cells of the CRPBF all contain 1, the CRPBF produces a positive result, and the output response R from pm-PUF is transmitted to the authentication server for comparison with the CRP table. If any one of the cells contains 0, the CRPBF produces a negative result and the output response is dropped because the output response from the pm-PUF does not exist in the authentication server. In another query example shown in Figure 5b, it is assumed that the output response R_a from an challenge A of attacker results in a false positive from the PDBF. The hash indices of CRP_a are 1, 5, and 9. The values in the cells are 0, so the result is negative, and the output response is dropped and is not transmitted to the authentication server. The CRPBF procedure is shown in Algorithm 2.



(**a**) The CRPBF querying for CRP_1



(**b**) The CRPBF querying for an attack CRP_a

Figure 5. Querying procedure for a CRPBF.

Algorithm 2: CRPBF procedure

```
Function Programming_CRPBF(challenge, response)
    CRP = concatenate(challenge, response);
    for i = 1 to k do
        if BF[h_i(CRP)] == 0 then
            BF[h_i(CRP)] = 1;
        end
        else
            continue;
        end
    end
end

Function Querying_CRPBF(challenge, response)
    CRP = concatenate(challenge, response);
    for i = 1 to k do
        if BF[h_i(CRP)] == 0 then
            return negative;
        end
    end
    return positive;
end
```

3.3. Operation Procedure

Figure 1 illustrates the operating procedure of the proposed architecture. For example, the challenge C (1100) has hash indices of 0, 5, and 13, and a challenge length of 4, so the PDBF cell shifted by 4 is queried. Since all the values in the cell are not 0, the PDBF produces a positive result, and the corresponding challenge C (1100) is transmitted to the pm-PUF, resulting in the output response R (110). To query the CRPBF, a hash index is obtained with the CRP (1,100,110), the concatenation of the input challenge C (1100) and the output response R (110) of the pm-PUF. The hash indices of CRP (1,100,110) are 3, 6, and 10, each CPRBF cell accessed contains a 1, and the CPRBF produces a positive result. The output response R from the pm-PUF is transmitted to the authentication server for comparison with its CRP table. When the server indicates successful authentication, since the CRP has been used, the challenge C of input is deleted in the PDBF. The hash indices of challenge C (1100) are 0, 5, and 13, and the challenge length is 4. In the PDBF, the cells shifted by 4 are 4, 9, and 1, and since the

contents of cells 1 and 4 are not X, their values are each decreased by 1. However, since the value of cell 9 is X, it is not deleted. Algorithm 3 describes the operation procedure in detail.

To summarize these operations, the security of the pm-PUF is improved because the proposed structure transmits the challenge identified by the PDBF to the pm-PUF and passes the pm-PUF response to the CRPBF, transmitting only the response that exists in the authentication server. To prevent the reuse of CRPs, the challenge of an authenticated CRP is deleted from the PDBF.

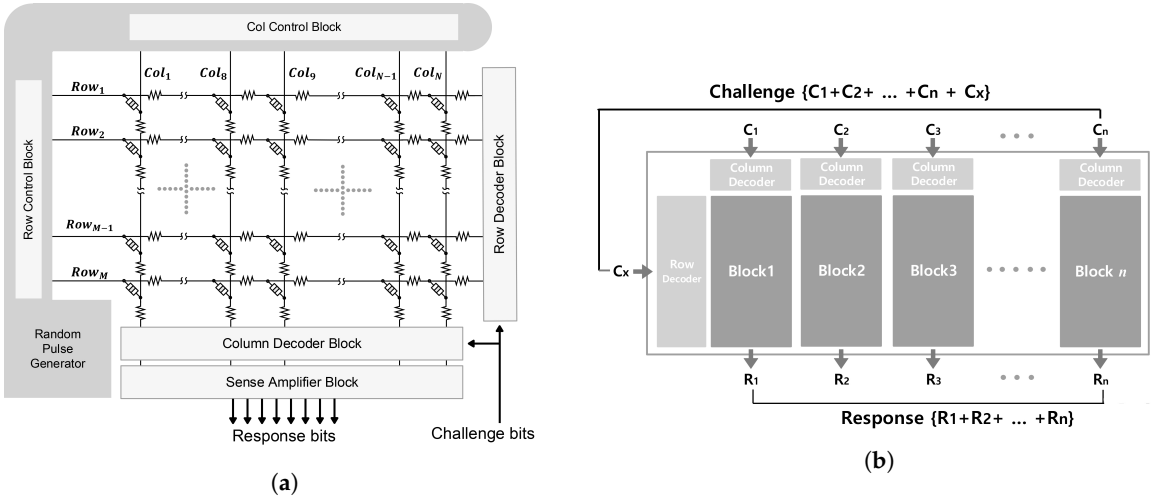
Algorithm 3: Operation Procedure

```
Function QueryBF(challenge)
    PB_RESULT = Querying_PDBF(challege, length_challenge);
    if PB_RESULT == positive then
        Input challenge passed to pm-PUF;
    end
    else
        Input challenge drop;
    end

    //Response from pm-PUF
    CRP = concatenate(challenge, response);
    CRP_RESULT = Querying_CRPBF(CRP);
    if CRP_RESULT == positive then
        Response passed to Authentication Server;
        if Authentication Success then
            Deleting_PDBF(challenge, length_challenge);
        end
    end
    else
        Response drop;
    end
end
```

4. Data Set Analysis for Simulation

In this paper, we utilized a data set obtained from a pm-PUF circuit for simulation [29]. As shown in Figure 6a, the pm-PUF circuit consists of a 64 × 64 memristor crossbar array with row and column control blocks made up of multiplexers, random pulse generators, row and column decoder blocks, and voltage sense amplifiers. The line resistance is 1.2 Ω and each memristor in the array follows the electrical characteristics of a Ta_2O_5 memristor device. The characteristics were modeled in [58] and coded in Verilog-A for HSPICE circuit simulation. The pm-PUF applies a bank design in which several unit blocks are integrated to form a large structure and therefore can generate a multi-bit response in a single cycle. The pm-PUF bank design is described in Figure 6b. The entire PUF is composed of N small PUF blocks. Each small block receives a corresponding challenge bit string and generates a response bit string. In the simulated circuit, a 64 × 64 array consists of 8 blocks of 8 × 64 cells. We maximized resistance variation by applying various pulses with varied pulse widths and used this random variation as an entropy source. The pulse amplitude was set to 1.5 V, which is higher than the positive threshold voltage (1 V). The read voltage was set to 0.75 V, which is lower than the threshold voltage. The entire challenge consisted of 30 bits with 6 bits used as row select bits and 24 bits used as column select bits, allowing one memristor cell to be selected in each block. Two bits of the state of selected cell are output as its response. A total of eight blocks generates a 16-bit response. Therefore, the CRP data set consisted of 30-bit challenge and 16-bit response pairs.



(**a**) (**b**)

Figure 6. Simplified pulsewidth-based memristive PUF (pm-PUF) architecture [29]. (**a**) Simplified pulsewidth-based memristive PUF (pm-PUF) architecture. (**b**) A concept of bank design for pm-PUF.

We evaluated the CRP data set in terms of randomness, diffuseness, uniqueness, and steadiness. Mathematical expressions of these performance metrics are described in detail in [29,41]. The notation used in this paper is used in [29,41].

- **Randomness**: Let p_n be the relative frequency of 1 appearing in all the response bits generated in device n. Then, p_n is given by

$$p_n = \frac{1}{K \cdot T \cdot L} \sum_{k=1}^{K} \sum_{t=1}^{T} \sum_{l=1}^{L} b_{n,k,t,l} \tag{3}$$

Then, the device randomness H_n is defined as follows [29,41]:

$$H_n = -\log_2 max(p_n, 1 - p_n) \tag{4}$$

When p_n = 0.5, H_n takes the highest value 1 and is considered as ideal value. We randomly applied 5500 different challenges and investigated 5500 sets of 16-bit length responses. The randomness of the PUF is 0.9828 (98.28%), which is slightly lower than the ideal value of 1. The probability for a 0 response is 50.6% and for a 1 is 49.4% as shown in Figure 7a.

- **Diffuseness**: Let $d_{n,l}$ be the sum of Hamming distance of the possible bit-combinations in response [29,41]:

$$d_{n,l} = \sum_{i=1}^{K-1} \sum_{j=i+1}^{K} (b_{n,i,l} \oplus b_{n,j,l}) \tag{5}$$

Then, since each bit of L-bit responses is independent of other bits, the upper bound of the sum of Hamming distance of the possible response combination is $L \cdot (K/2)^2$. Therefore, the device diffuseness D_n is defined as follows [29,41].

$$D_n = \frac{1}{L} \sum_{l=1}^{L} \frac{d_{n,l}}{(K/2)^2} \tag{6}$$

When the sum of Hamming distance reaches the upper bound, D_n takes the highest value 1 and is considered as the ideal value. We randomly applied 5500 different challenges and investigated 5500 sets of 16-bit length responses. The diffuseness of the PUF is 0.9871, which is close to the

ideal value of 1. The distribution of the intra-HD among the obtained responses is shown in Figure 7b; it is mean is 49.29%.

- **Uniqueness**: The upper bound of the sum of Hamming distance of the possible responses among all devices is $L \cdot (N/2)^2$. Therefore, the device uniqueness U_n is defined as follows [29,41].

$$U_n = \frac{4}{K \cdot L \cdot N \cdot} \sum_{k=1}^{K} \sum_{l=1}^{L} \sum_{j=1, j \neq n}^{N} (b_{n,k,l} \oplus b_{j,k,l}) \tag{7}$$

When the sum of Hamming distance reaches the upper bound, U_n takes the highest value 1 and is considered as the ideal value. In our data set, we randomly applied 5500 different challenges for 100 different PUF instances, and investigated 5500 sets of 16-bit length responses. The uniqueness of the PUF is 0.9507 which is close to the ideal value of 1. The mean of the inter-HD is 47.93%.

- **Steadiness**: Let $S_{n,k,l}$ be the steadiness of the l-th bit of $R_{n,k}$. Then, $S_{n,k,l}$ is defined using min-entropy as follows [29,41]:

$$S_{n,k,l} = 1 + \log_2 max(p_{n,k,l}, 1 - p_{n,k,l}) \tag{8}$$

Then, the steadiness S_n is defined by taking the mean of $S_{n,k,l}$.

$$S_n = \frac{1}{K \cdot L \cdot} \sum_{k=1}^{K} \sum_{l=1}^{L} S_{n,k,l} \tag{9}$$

To evaluate steadiness, we obtained 256-bit length responses at temperatures of 0 °C, 25 °C, 50 °C, and 85 °C and compared them to see if they were identical. A reference response was obtained at room temperature (25 °C). The results are shown in Figure 7d. The worst steadiness of the PUF is 0.9102 for a temperature of 85 °C.

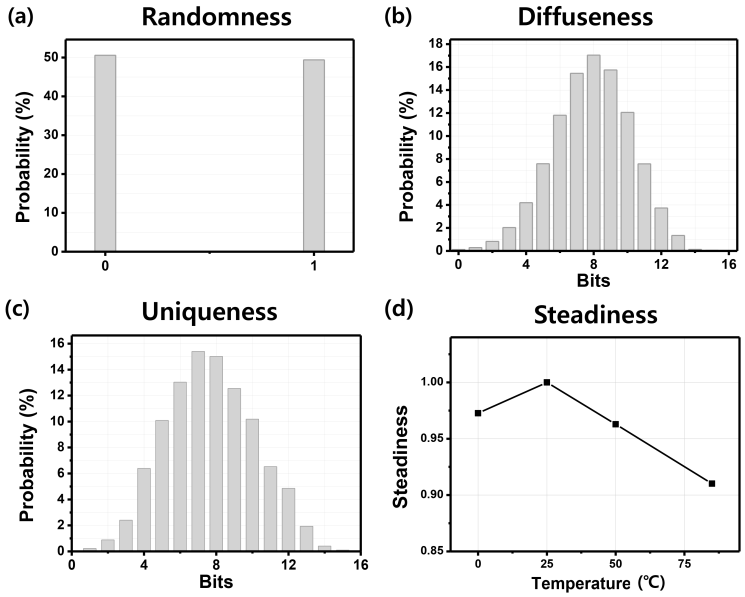


Figure 7. pm-PUF performance evaluation [29]: (**a**) randomness: occurrence probability of a 0 and 1 in the responses, (**b**) diffuseness: possibility of generating different responses with the same challenge, (**c**) uniqueness: possibility of generating different responses among different PUFs, and (**d**) steadiness under temperature variations (0 °C, 25 °C, 50 °C and 85 °C).

5. Performance Evaluation

We evaluated the performance of proposed architecture by implementing its structure in C++ using the original CRPs obtained from the pm-PUF [29]. The hash function used for our proposed structure is a 64-bit cycle redundancy check (64-CRC). We assumed that the tables of authentication server held 2^9, 2^{10}, 2^{11}, and 2^{12} CRPs. Table 1 shows the PDBF, CRPBF, and total Bloom filter memory requirements when the sizing factor α of the Bloom filter is increased to two, four, and eight in the proposed structure. The total memory requirement for the Bloom filter is $M_{PDBF} + M_{CRPBF}$. The memory requirement for the PDBF is equal to $2 \cdot \alpha \cdot 2^{\lceil \log_2 n_c \rceil}$ where n_c is the number of challenges in the authentication server. The memory requirement for the CRPBF is $\alpha \cdot 2^{\lceil \log_2 n_{crp} \rceil}$ where n_{crp} is the number of CRPs in the authentication server.

Table 1. Memory requirement of total Bloom filter (KB).

CRPs	α	M_{PDBF}	M_{CRPBF}	Total Memory
Set 1 (2^9)	2	0.25	0.13	0.38
	4	0.50	0.25	0.75
	8	1.00	0.50	1.50
Set 2 (2^{10})	2	0.50	0.25	0.75
	4	1.00	0.50	1.50
	8	2.00	1.00	3.00
Set 3 (2^{11})	2	1.00	0.50	1.50
	4	2.00	1.00	3.00
	8	4.00	2.00	6.00
Set 4 (2^{12})	2	2.00	1.00	3.00
	4	4.00	2.00	6.00
	8	8.00	4.00	12.00

Table 2 shows the simulation result for the input set which is the sum of the authentication challenges and attacker challenges. To evaluate the performance of the proposed structure, we created input challenge sets with three times the challenges of authentication servers. It is assumed that one-third of the input challenges are authentic challenges, and the remaining two-thirds are attacker challenges. If authentication succeeded, the challenge is deleted from the PDBF. In Table 2, the CRBPF false positives are equal to the number of data transmission errors in the proposed structure. When the sizing factor of the Bloom filter is increased to four, the CRPBF false positives from all sets are reduced to zero. In other words, the proposed structure only transmitted data that existed on the authentication server.

Table 2. Bloom filter performance.

	Set 1		***Set 2***		***Set 3***		***Set 4***	
α	2	4	2	4	2	4	2	4
total no. of PDBF queries	1536		3072		6144		12288	
total no. of CRPBF queries	524	512	1054	1024	2112	2048	4214	4096
no. of PDBF false positives	12	0	30	0	64	0	118	0
no. of CRPBF false positives	5	0	11	0	25	0	34	0
no. of transmission data	517	512	1035	1024	2073	2048	4130	4096

Table 3 shows the simulation results for an input set consisting only of attacker challenges. We created the input challenge sets with two times the challenges of authentication servers. In this simulation, since the challenge of the authentication server is not included in the input set, deletion of the PDBF did not occur. In the proposed structure, when the sizing factor of the Bloom filter is increased to eight, the CRPBF false positives of all sets are reduced to a range of zero to three. It means that the data transmission error rate is 0–0.098%.

Table 3. Bloom filter performance.

	Set 1			***Set 2***			***Set 3***			***Set 4***		
α	2	4	8	2	4	8	2	4	8	2	4	8
total no. of PDBF queries		1024			2048			4096			8192	
no. of PDBF false positives	409	144	30	819	257	52	1514	597	90	3224	1236	190
no. of CRPBF false positives	153	32	1	324	47	0	580	81	1	1245	182	3

Figure 8 is the false positive probability of the proposed structure using the input from Table 3. Figure 8 can be compared with the false positive of theoretical analysis. The theoretical analysis is referenced to in [57]. If n_c is the number of challenges in the authentication server, n_{crp} is the number of CRPs in the authentication server, k is the number of hash functions, and m is the number of cells, and p_{pd} is the probability that a particular cell is set at least once by n_c elements in the PDBF, p_{pd} is defined in Equation (10):

$$p_{pd} = 1 - \left(1 - \frac{1}{m}\right)^{kn_c} \tag{10}$$

If p_{crp} represents the probability that a particular cell is set at least once by n_{crp} elements in CRPBF, p_{crp} is defined as in Equation (11):

$$p_{crp} = 1 - \left(1 - \frac{1}{m}\right)^{kn_{crp}} \tag{11}$$

In querying, for the input y not included in the set S, false positives occur if the PDBF produces false positives from input y and the CRPBF also produces false positives from input y. Hence, the false positive probability $P(F)$ is defined as follows:

$$P(F) = {p_{pd}}^k \cdot {p_{crp}}^k \tag{12}$$

Figure 8 shows that the data transmission error rate decreases for all data sets with increasing the sizing factor of the Bloom filter. Additionally, the simulation results for all data sets are close to the theoretical results. If a PUF is implemented without a Bloom filter, responses corresponding to all challenges by an attacker are transmitted, and CRP table information or the PUF model in the server may eventually be exposed.

Figure 9 shows the delete performance of the PDBF of the proposed structure. To evaluate the deletion performance of the proposed structure, all CRPs from the authentication server are used once and then again used as inputs. The PDBF should produce negative results for reused CRPs. If the CRPBF produces a positive result, our proposed structure incurs a data transmission error. When the size factor of the Bloom filter is eight, most of the sets do not transmit the response to the server, confirming the successful removal of the reused CRP.

The proposed structure applies small Bloom filters to the PUF. The Bloom filters identify an attacker challenge and remove its access to the PUF. Additionally, the Bloom filters verify the response generated by the PUF and transmit only the data that exists in the authentication server. The performance of the proposed structure is confirmed by simulation.

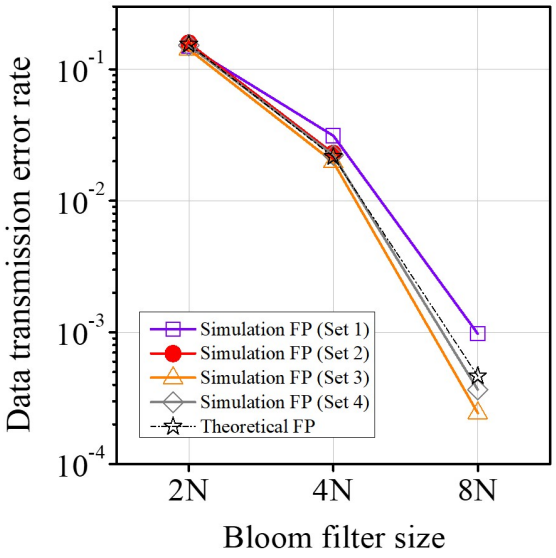


Figure 8. Data transmission error rate.

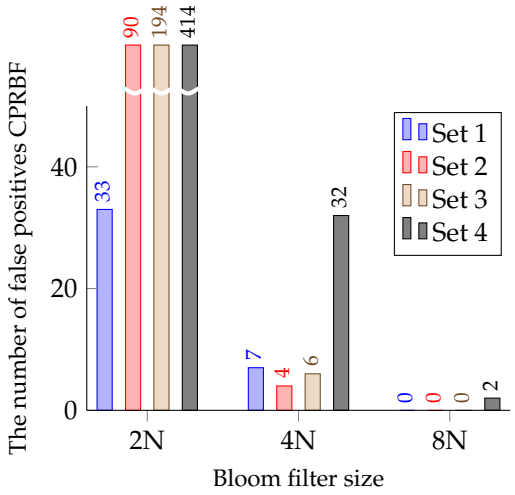


Figure 9. The number of false positives after deletion.

6. Conclusions

In this paper, we propose a new hardware security architecture consisting of a memristor-based PUF and Bloom filters to improve security. In our proposed structure, two different types of Bloom filters are used, the PDBF and CRPBF, that store information about the challenge and response of the authentication server; the structure detects arbitrary challenge attacks and reduces error rates originating from the instability of the PUF. At the front of the configuration, the PDBF identifies whether an input challenge is valid by satisfying two conditions before applying it to pm-PUF directly; one condition is that the challenge is to be included in the CRP table shared by the authentication server, and the other condition is that it should not have been previously used for authentication. When an input challenge turns out to be appropriate, the pm-PUF generates the response corresponding to the input challenge. Next, the result from the PUF is checked by the CRPBF to prevent sending the wrong response to the authentication server. The proposed structure presents a practical solution to remove unnecessary access to PUF by the attack and prevent the reuse of a CRP and detect the wrong PUF response with the error. Moreover, because the data set used to verify the performance of the Bloom filter is circuit-simulated data from the pm-PUF that we proposed in previous work, it is more reasonable and realistic than using arbitrary data from all the theoretical data possibilities.

The simulation results of the proposed architecture with the Bloom filter show that with use of a small-memory filter, the possible access from attackers decreases, and the used CRPs are mostly deleted. Therefore, the PD-CRP Bloom filter structure proposed in this study can contribute to improving the performance of a hardware-based security system.

Author Contributions: Conceptualization, J.L., and W.S.; software, J.L., and S.C.; investigation, J.L., S.C., D.K., Y.C., and W.S.; writing—original draft preparation, J.L., S.C., D.K., and Y.C.; writing—review and editing, W.S.; project administration, J.L., and W.S.; funding acquisition, J.L., and W.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (No. NRF-2018R1A6A3A11040736 and NRF-2020R1I1A1A01065622).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Tedeschi, S.; Mehnen, J.; Tapoglou, N. Secure IoT devices for the maintenance of machine tools. *Procedia CIRP* **2017**, *59*, 150–155. [CrossRef]
2. Arias, O.; Wurm, J.; Hoang, K.; Jin, Y. Privacy and security in Internet of Things and wearable devices. *IEEE Trans. Multi-Scale Comput. Syst.* **2015**, *1*, 99–109. [CrossRef]
3. Rostami, M.; Koushanfar, F.; Karri, R. A Primer on Hardware Security: Models, Methods, and Metrics. *Proc. IEEE* **2014**, *102*, 1283–1295. [CrossRef]
4. Shamsi, K.; Jin, Y. Security of emerging non-volatile memories: Attacks and defenses. In Proceedings of the IEEE VLSI Test Symposium, Las Vegas, NV, USA, 25–27 April 2016; pp. 1–4.
5. Van der Leest, V.; Tuyls, P. Anti-counterfeiting with hardware intrinsic security. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 18–22 March 2013; pp. 1137–1142.
6. Sadeghi, A.; Naccache, D. *Toward Hardware-Intrinsic Security: Foundations and Practice*; Springer: New York, NY, USA, 2010.
7. Herder, C.; Yu, M.; Koushanfar, F.; Devadas, S. Physical Unclonable Functions and Applications: A Tutorial. *Proc. IEEE* **2014**, *102*, 1126–1141. [CrossRef]
8. Gassend, B. Clarke, D. Van Dijk, M. Devadas, S. Silicon physical random functions. In Proceedings of the ACM Conference on Computer and Communications Security(CCS), Washington, DC, USA, 18–22 November 2002; pp. 148–160.
9. Uddin, M.; Majumder, B.; Rose, G. Nanoelectronic Security Designs for Resource-Constrained Internet of Things Devices: Finding Security Solutions with Nanoelectronic Hardwares. *IEEE Consum. Electron. Mag.* **2018**, *7*, 15–22. [CrossRef]
10. Rührmair, U.; Sehnke, F.; Sölter, J.; Dror, G.; Devadas, S.; Schmidhuber, J. Modeling attacks on physical unclonable functions. In Proceedings of the ACM Conference on Computer and Communications Security (CCS), Chicago, IL, USA, 4–8 October 2010; pp. 237–249.
11. Mitzenmacher, M. Compressed Bloom filters. *IEEE-ACM Trans. Netw.* **2002**, *10*, 604–612. [CrossRef]
12. Mosharraf, N.; Jayasumana, A.; Ray, I. Compacted Bloom filter. In Proceedings of the IEEE International Conference on Collaboration and Internet Computing (CIC), Pittsburgh, PA, USA, 1–3 November 2016; pp. 304–311.
13. Bonomi, F.; Mitzenmacher, M.; Panigrahy, R.; Singh, S.; Varghese, G. An improved construction for counting Bloom filters. In Proceedings of the European Symposium on Algorithms, Zurich, Switzerland, 11–13 September 2006; pp. 684–695.
14. Ahmadi, M.; Wong, S. A memory-optimized Bloom filter using an additional hashing function. In Proceedings of the IEEE GLOBECOM, New Orleans, LA, USA, 30 November–4 December 2008; pp. 1–5.
15. Geravand, S.; Ahmadi, M. A novel adjustable matrix Bloom filterbased copy detection system for digital libraries. In Proceedings of the International Conference on Computer and Information Technology, Pafos, Cyprus, 31 August–2 September; pp. 518–525.

16. Kaya, I.; Kocak, T. Energy-efficient pipelined Bloom filters for network intrusion detection. In Proceedings of the IEEE International Conference on Communications(ICC), Istanbul, Turkey, 11–15 June 2006; pp. 2382–2387.
17. Kocak, T.; Kaya, I. Low-power Bloom filter architecture for deep packet inspection. *IEEE Commun. Lett.* **2006**, *10*, 210–212. [CrossRef]
18. Paynter, M.; Kocak, T. Fully pipelined Bloom filter architecture. *IEEE Commun. Lett.* **2008**, *12*, 855–857. [CrossRef]
19. Zhou, T.; Song, T.; Wang, X. EABF: Energy efficient self-adaptive Bloom filter for network packet processing. In Proceedings of the IEEE International Conference on Communications(ICC), Ottawa, ON, Canada, 10–15 June 2012; pp. 2729–2734.
20. Kapoor, A.; Arora, V. Application of bloom filter for duplicate url detection in a web crawler. In Proceedings of the IEEE International Conference on Collaboration and Internet Computing (CIC), Pittsburgh, PA, USA, 1–3 November 2016; pp. 246–255.
21. Durham, E.; Kantarcioglu, M.; Xue, Y.; Toth, C.; Kuzu, M.; Malin, B. Composite Bloom filters for secure record linkage. *IEEE Trans. Knowl. Data Eng.* **2014**, *26*, 2956–2968. [CrossRef]
22. Moreira, M; Laufer, R.; Velloso, P.; Duarte, O. Capacity and robustness tradeoffs in Bloom filters for distributed applications. *IEEE Trans. Parallel Distrib. Syst.* **2012**, *23*, 2219–2230. [CrossRef]
23. Peir, J.; Lai, S.; Lu, S.; Stark, J.; Lai, K. Bloom filtering cache misses for accurate data speculation and prefetching. In Proceedings of the ACM International Conference on Supercomputing, New York, NY, USA, 22–26 June 2002; pp. 347–356.
24. Bender, M.; Farach-Colton, M.; Johnson, R.; Kraner, R.; Kuszmaul, B.; Medjedovic, D.; Montes, P.; Shetty, P.; Spillane, R.; Zadok, E. Don't thrash: How to cache your hash on flash. *VLDB Endow.* **2012**, *5*, 1627–1637. [CrossRef]
25. Lyons. M.; Brooks. D. The Design of a Bloom Filter Hardware Accelerator for Ultra Low Power Systems. In Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design, San Fancisco, CA, USA, 19–21 August 2009; pp. 371–376.
26. Singh, A.; Garg, S.; Batra, S.; Kumar, N.; Rodrigues, J. Bloom filter based optimization scheme for massive data handling in iot environment. *Futur. Gener. Comp. Syst.* **2018**, *82*, 440–449. [CrossRef]
27. Yang, D.; Tian, D.; Gong, J.; Gao, S.; Yang, T.; Li, X. Difference bloom filter: A probabilistic structure for multi-set membership query. In Proceedings of the IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017; pp. 1–6.
28. Busch, H.; Katzenbeisser, S.; Baecher, P. PUF-Based Authentication Protocols. In Proceedings of the International Workshop on Information Security Applications(WISA), Busan, Korea, 25–27 August 2009; pp. 296–308.
29. Choi, S.; Kim, D.; Choi, Y; Sun, W.; Shin, H. Multi bit-generating Pulse width-based Memristive-PUF Structure and Circuit Implementation. *Electronics* **2020**, *9*, 1446. [CrossRef]
30. Fruhashi, K.; Shiozaki, M.; Fukushima, A.; Murayama, T.; Fujino, T. The arbiter-PUF with high uniqueness utilizing novel arbiter circuit with Delay-Time Measurement. In Proceedings of the IEEE International Symposium of Circuits and Systems (ISCAS), Rio de Janeiro, Brazil, 15–18 May 2011; pp. 2325–2328.
31. Eiroa, S.; Baturone, I. An analysis of ring oscillator PUF behavior on FPGAs. In Proceedings of the International Conference on Field-Programmable Technology, New Delhi, India, 12–14 December 2011; pp. 1–4.
32. Garg, A.; Kim, T. Design of SRAM PUF with improved uniformity and reliability utilizing device aging effect. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Melbourne, VIC, Australia, 1–5 June 2014; pp. 1941–1944
33. Ardakani, A.; Baradaran Shokouhi, S. A secure and area-efficient FPGA-based SR-Latch PUF. In Proceedings of the International Symposium on Telecommunications (IST), Tehran, Iran, 27–28 September 2016; pp. 94–99.
34. Rostami, M.; Wendt, J.B.; Potkonjak, M.; Koushanfar, M. Quo vadis, PUF? Trends and challenges of emerging physical-disorder based security. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 21 April 2014; pp. 1–6.
35. Chua, L. Memristor-The missing circuit element. *IEEE Trans. Circuit Theory* **1971**, *18*, 507–519. [CrossRef]
36. Wong, H.; Raoux, S.; Kim, S.; Liang, J.; Reifenberg, J.; Rajendran, B.; Asheghi, M.; Goodson, K. Phase Change Memory. *Proc. IEEE* **2010**, *98*, 2201–2227. [CrossRef]
37. Kim, J.; Ryu, K.; Kang, S.; Jung, S. A Novel Sensing Circuit for Deep Submicron Spin Transfer Torque MRAM (STT-MRAM). *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2012**, *20*, 181–186. [CrossRef]

38. Koeberl, P.; Kocabaş, Ü.; Sadeghi, A. Memristor PUFs: A new generation of memory-based physically unclonable functions. In Proceedings of the Conference on Design, Automation and Test in Europe, Grenoble, France, 18–22 March 2013; pp. 428–431.
39. Rose, G.S.; McDonald, N.; Yan, L.; Wysocki, B. A write-time based memristive PUF for hardware security applications. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Jose, CA, USA, 18–21 November 2013; pp. 830–833.
40. Chen, A. Reconfigurable physical unclonable function based on probabilistic switching of RRAM. *Electron. Lett.* **2015**, *51*, 615–617. [CrossRef]
41. Hori, Y.; Yoshida, T.; Katashita, T.; Satoh, A. Quantitative and Statistical Performance Evaluation of Arbiter Physical Unclonable Functions on FPGAs. In Proceedings of the International Conference on Reconfigurable Computing and FPGAs, Quintana Roo, Mexico, 13–15 December 2010; pp. 298–303.
42. Majzoobi, M.; Rostami, M.; Koushanfar, F.; Wallach, D.; Devadas, S. Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching. In Proceedings of the IEEE Symposium on Security and Privacy Workshops, San Francisco, CA, USA, 24–25 May 2012; pp. 33–44.
43. Ye, M.; Shahrak, M.; Wei, S. PUFSec: Protecting physical unclonable functions using hardware isolation-based system security techniques. In Proceedings of the Asian Hardware Oriented Security and Trust Symposium (AsianHOST), Beijing, China, 19–20 October 2017; pp. 7–12.
44. Amsaad, F.; Niamat, M.; Dawoud, A.; Kose, S. Reliable Delay Based Algorithm to Boost PUF Security Against Modeling Attacks. *Information* **2018**, *9*, 224. [CrossRef]
45. Bloom, B. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* **1970**, *13*, 422–426. [CrossRef]
46. Zhang, R.; Liu, J.; Huang, T.; Pan, T.; Wu, L. Adaptive Compression Trie Based Bloom Filter: Request Filter for NDN Content Store. *IEEE Access* **2017**, *5*, 23647–23656. [CrossRef]
47. Zhang, W.; Xu, Y.; Li, Y.; Zhang, Y.; Li, D. FlameDB: A Key-Value StoreWith Grouped Level Structure and Heterogeneous Bloom Filter. *IEEE Access* **2018**, *6*, 24962–24972. [CrossRef]
48. Moralis-Pegios, M.; Terzenidis, N.; Mourgias-Alexandris, G.; Vyrsokinos, K. Silicon Photonics towards Disaggregation of Resources in Data Centers. *Appl. Sci.* **2018**, *8*, 83. [CrossRef]
49. Lee, J.; Sim, M.; Lim, H. Name Prefix Matching Using Bloom Filter Pre-Searching for Content Centric Network. *J. Netw. Comput. Appl.* **2016**, *65*, 36–47. [CrossRef]
50. Bonomi, F.; Mitzenmacher, M.; Panigrah, R.; Singh, S.; Varghese, G. Beyond Bloom filters: From approximate membership checks to approximate state machines. In Proceedings of the Conference on Applications,Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM), Pisa, Italy, 11–15 September 2006; pp. 315–326.
51. Lim, H.; Lee, J.; Yim, C. Complement Bloom Filter for Identifying True Positiveness of a Bloom Filter. *IEEE Commun. Lett.* **2015**, *19*, 1905–1908. [CrossRef]
52. Yang, T.; Liu, A.; Shahzad, M.; Zhong, Y.; Fu, Q.; Li, Z. A shifting Bloom filter framework for set queries *VLDB Endow.* **2016**, *9*, 408–419. [CrossRef]
53. Broder, A.; Mitzenmacher, M. Network Applications of Bloom Filters: A Survey. *Int. Math.* **2004**, *1*, 485–509. [CrossRef]
54. Tarkoma, S.; Rothenberg, C.; Lagerspetz, E. Theory and practice of Bloom filters for distributed systems. *IEEE Commun. Surv. Tutor.* **2012**, *14*, 131–155. [CrossRef]
55. Fan, L.; Cao, P.; Almeida, J.; Broder, A. Summary cache: A scalable wide-area Web cache sharing protocol. *IEEE/ACM Trans. Netw.* **2000**, *8*, 281–293. [CrossRef]
56. Lim, H.; Lee, J.; Yim, C. Ternary Bloom Filter Replacing Counting Bloom Filter. *IEEE Commun. Lett.* **2017**, *21*, 278–281. [CrossRef]
57. Lee, J.; Byun, H.; Lim, H. Dual-Load Bloom Filter : Application for Named Lookup. *Comput. Commun.* **2020**, *151*, 1–9. [CrossRef]
58. Kim, B.; Jo, S.; Sun, W.; Shin, H. Analysis of the Memristor-Based Crossbar Synapse for Neuromorphic 437 Systems. *J. Nanosci. Nanotechnol.* **2019**, *19*, 6703–6709. [CrossRef] [PubMed]