Secure and Space Efficient Accounts Storage System Using Three-Dimensional Bloom Filter

Muhammad Arzaki, Raissa Henardianti Hanifa and Farah Afianti

***Abstract*—This paper investigates the application of a three-dimensional Bloom Filter (3DBF) to accomplish a secure and efficient accounts storage system by exploiting hashes of usernames and their corresponding passwords. We conducted numerical experiments and mathematical analysis to study the efficiency level of several 3DBF schemes. Our experimental results and analysis show that the level of occupancy for 3DBF is positively correlated to the value of its false positive rate, viz., if the occupancy level increases then so does the value of the false positive rate. Moreover, we also derive a formula for determining the minimum number of bits for storing some data in a 3DBF scheme given the value of its acceptable false positive rate and its occupancy level. We infer that the product of the dimensional parameter of a 3DBF scheme is inversely proportional to the false positive rate and occupancy level used in the scheme.**

***Index Terms*—Accounts storage system, False positive rate, Three-dimensional Bloom Filter.**

I. Introduction

THE number of websites that limit their access only to registered users is increasing. These types of websites require every user to create a membership account, i.e., a username and its corresponding password that is used for accessing the contents. As a consequence, the amount of storage required for securely storing usernames and their respective passwords is also increasing. These usernames and passwords must be kept in the protected format most of the time. One possible way to achieve this protection is by using a hash function to determine the digest of each username and its corresponding password. These digests have a fixed and identical length which depends on the choice of the hash function [1]. The utilization of the hash function also provides protection (although it is not always sufficient) to the website members in case the account information is compromised, such as in the case of e-commerce data leaks (see, e.g., [2], [3]).

As the number of users grows, a straightforward method of storing the hash values of all account information (such as the concatenation of username and password) is inefficient in terms of the required storage space. Bloom Filter (BF) is a data structure that can be employed to reduce storage consumption and thus can improve the performance of a data storing system. Instead of storing particular messages directly, we store the keys associated with this information. This data structure has been deployed in several systems, such as in big data storage [4], in data collection on wireless network communication [5], and in caching or analyzing data in the Internet of things [6]. There are five major classifications of BF and one of them is the r-dimensional BF which is an example of the multidimensional BF [7]. This r-dimensional BF, or rDBF for short, does not yield a *false negative match*—an error in which a key of an element is considered as a non-member of a particular set despite such an element is, in fact, a member of such a set. The absence of such an error also makes rDBF outperforms other probabilistic data structures in terms of accuracy, such as the Cuckoo Filter, which also offers storage reduction [7]. These conditions also cause rDBF to be preferred to Cuckoo Filter for reducing the memory consumption in an account storage system. Nevertheless, rDBF also has a drawback that makes its utilization limited, namely the *false positive error*—which happens when a particular element is considered as a member of a set while in reality, this condition is not true. If the false positive error rate of an rDBF scheme can be reduced, then this scheme can be used as a solution for reducing the storage consumption [4].

This paper investigates some preliminary experimental and theoretical aspects of three-dimensional BF or 3DBF for storing account information. The investigation of 3DBF is interesting—since theoretically the false positive rate of 3DBF is lower than that of 2DBF [8], yet the average time for insert and search operations of 3DBF is faster than that of 4DBF and 5DBF with relatively comparable false positive rates. In rDBF, the value r signifies the value of dimension in the BF used in the system, which also affects the data processing speed [7].

In our investigation, 3DBF schemes work by converting a digest value h of an input to a triple (i, j, k) by using the equations $i = h \bmod x$, $j = h \bmod y$, and $k = h \bmod z$, for some positive integers x, y, and z. Here, the triple (x, y, z) is called the *dimensional parameter* of a 3DBF scheme. In this paper, we investigate the influence of the value of (x, y, z) on the false positive rate of a particular 3DBF with several items as inputs. In our experiments, we randomly generated pairs of usernames and passwords as inputs and we computed their hash values using SHA256 algorithm. Our investigation is restricted only to insert and search (query) operations and we do not consider the delete operation. We perform empirical analysis from our experiments and we supplement these results with theoretical analysis.

M. Arzaki and F. Afianti are with the Computing Laboratory, School of Computing, Telkom University, Bandung 40257, Indonesia e-mail: arzaki@telkomuniversity.ac.id, farahafi@telkomuniversity.ac.id.

R. H. Hanifa was an undergraduate student at Computing Laboratory, School of Computing, Telkom University, Bandung 40257, Indonesia.

Manuscript received May 19, 2022; accepted December 12, 2022.

II. Related Works and Preliminaries

A. State of the Art

Bloom Filter (BF) is a probabilistic data structure that was first proposed by B. H. Bloom in 1970 [9]. This data structure is used to check whether an element is a member of a set by using a probabilistic approach. Instead of storing an item directly, BF stores a key associated with that item in a bit array. A one-dimensional BF is represented using a bit array of non-zero length. To store an element in a one-dimensional BF, we need $k \in \mathbb{N}$ hash functions, which can be denoted by $\mathsf{hash}_1, \mathsf{hash}_2, \ldots, \mathsf{hash}_k$. The value of k affects the false positive error rate in the BF.

In 2019, Patgiri et al. proposes a multidimensional BF called rDBF [7]. This scheme generalizes the one-dimensional BF by considering an r-dimensional bit array of a particular size (dimension). As in the one-dimensional BF, rDBF utilizes k hash functions to compute the digest of an input. However, unlike the one-dimensional BF, the digests are stored in an r-dimensional bit array using an r-tuple dimensional parameter $(x_1, x_2, \ldots, x_r)$ such that x_i are distinct prime numbers for all $1 \leq i \leq r$. For a digest value h of an input, an rDBF computes the value of $h \bmod x_i$ for each $1 \leq i \leq r$. The prime numbers are used as the dimensional parameter to avoid *collision*—a condition in which different inputs are associated with an identical key in an r-dimensional bit array.

Further improvement of BF was made by constructing a multilevel BF from rDBF. The resulting data structure is called a high accuracy BF or HFil [10]. HFil is used to construct a password database system called PassDB [11], which employs a 12-level HFil on a 3DBF. Notwithstanding, BF not only can be used to store passwords and usernames. Recently, Berardi et al. employed BF for determining the similarity among passwords [12].

Other recent developments related to BF were conducted by Zhong et al. who implemented a novel adaptive BF to decrease source-route length in the packet header facing frame size limitation in the resource-constraint environment [13] and An et al. who used the acquisition of BF to detect missing RFID tag in an efficient way [14]. Furthermore, integration between attribute BF and improvement of the ciphertext-policy attribute-based encryption (CP-ABE) is proposed by Ramu to guarantee confidentiality in the cloud framework [15]. BF is also used in another cloud system that supports an e-health environment to filter, classify, and guarantee data integrity [16].

B. Measuring the Performance of a BF

BF can be viewed as a membership filtering algorithm that returns either *true* (also denoted by 1) or *false* (also denoted by 0) [4]. Suppose $x \in S$ is an element to be inserted into a conventional (one-dimensional) BF where S is a finite set. In practice, S may be the set of strings obtained from the concatenation of usernames and their corresponding passwords. Initially, BF works by applying one or more hash functions to x to get the associated hash value of x. Let us denote this value by $\mathbf{h}(x)$. The value of $\mathbf{h}(x)$ is associated with one or more elements of a one-dimensional bit array $\mathbf{B}$, called a key. Suppose for an element $x \in S$, we denote its associated key by $\mathbf{k}(x)$. Given an element y, BF can be used to check whether $y \in S$ by determining if $\mathbf{k}(y) \in \mathbf{B}$. The correctness performance of a filtering algorithm is typically measured using a true positive, false positive, true negative, or false negative match as in Definition 1.

Definition 1 ([4]): Let $x \in U$ be an element of a (finite) universal set U and $\mathbf{k}(x)$ be its associated key obtained from its hash value $\mathbf{h}(x)$. Suppose $S \subset U$ is a finite set and $\mathbf{B}$ is a bit array used in a BF scheme. Then:

1) if $\mathbf{k}(x) \in \mathbf{B}$ and $x \in S$, then the result of BF is called a true positive match,
2) if $\mathbf{k}(x) \in \mathbf{B}$ and $x \notin S$, then the result of BF is called a false positive match,
3) if $\mathbf{k}(x) \notin \mathbf{B}$ and $x \in S$, then the result of BF is called a false negative match, and
4) if $\mathbf{k}(x) \notin \mathbf{B}$ and $x \notin S$, then the result of BF is called a true negative match.

For an rDBF, we consider $\mathbf{B}$ as an r-dimensional bit array. An example of a three-dimensional bit array is $[[[10, 11], [01, 00]], [[00, 11], [01, 10]]]$. For $r \geq 2$, we notice that an r-dimensional bit array contains one or more $(r-1)$-dimensional bit array.

In a perfect filtering algorithm, we have $x \in S$ if and only if $\mathbf{k}(x) \in \mathbf{B}$. However, this is impossible if the filtering algorithm is probabilistic. We typically measure the correctness performance of filtering algorithms through *false positive rate* and *false negative rate*. A false positive rate (or false positive error rate) of a filtering algorithm measures the probability of an element $x \in U$ such that $\mathbf{k}(x) \notin \mathbf{B}$ and $x \in S$ where $S \subset U$, whereas a false negative rate is defined analogously. A functioning filtering algorithm is expected to have sufficiently low false positive and false negative rates. According to Patgiri et al., rDBF (including 3DBF) does not produce false negative matches and thus its false negative rate is zero [7, Theorem 4]. As a consequence, the correctness performance of a 3DBF is typically measured from its false positive rate.

C. Hash Functions in BF

Hash functions are used in BF to transform the inputs of various sizes into values of uniform and fixed lengths using a key [17]. These functions typically convert a string (also referred to as a message) of any length into its digest or hash value using mathematical operations. For a secure BF, the hash function should be irreversible, that is, given the output of a hash function, it is computationally infeasible to determine its corresponding input. In addition, a hash function in BF needs to satisfy weak and strong collision properties, namely, it is computationally infeasible to find different messages whose hash values are identical [17]. We refer the reader to [18] for a more comprehensive discussion regarding the hash function we use in this paper.

III. Proposed 3DBF for Accounts Storing System

Our proposed 3DBF uses a three-dimensional bit array in its filtering algorithm and it is adapted from the description of rDBF in [7, Section 3]. A two-dimensional bit array of dimension (x, y) can be viewed as an $x \times y$ matrix whose entries are bit strings of a fixed length. This matrix can be represented in a nested array notation $[r_0, r_1, \cdots, r_{x-1}]$ such that $r_i = [r_{i,0}, r_{i,1}, \ldots, r_{i,y-1}]$ and $r_{i,j}$ are bit strings of a fixed size for all $0 \leq i \leq x-1$ and $0 \leq j \leq y-1$. Notice that, for algorithmic purposes, we use a 0-based indexing in our matrices and array. As a consequence, we refer to the first entry of an array as the 0-th component of such an array. We can generalize the notion of a two-dimensional bit array to a three-dimensional bit array as in Definition 2

Definition 2: A three-dimensional bit array $\mathbf{B}$ of dimension (x, y, z) is an array $\mathbf{B} = [B_0, B_1, \ldots, B_{x-1}]$ such that $B_i = [B_{i,0}, B_{i,1}, \ldots, B_{i,y-1}]$ and $B_{i,j} = [B_{i,j,0}, B_{i,j,1}, \ldots, B_{i,j,z-1}]$ where $B_{i,j,k}$ are bit strings of a fixed size for all $0 \leq i \leq x-1$, $0 \leq j \leq y-1$, and $0 \leq k \leq z-1$. The location (i,j,k) in the bit array $\mathbf{B}$, namely the location of $B_{i,j,k}$, is called as the (i,j,k)-th cell, where $0 \leq i < x$, $0 \leq j < y$, and $0 \leq k < z$.

Example 1: The array $[[[01, 11], [10, 00]], [[11, 11], [00, 10]]]$ is an example of a three-dimensional bit array of dimension $(2, 2, 2)$ whose entries are bit strings of size 2. For example, the $(0, 1, 1)$-th entry of this array is 00.

Observe that, if each bit string $B_{i,j,k}$ is of size β, then the total bits available in a three-dimensional bit array of dimension (x, y, z) is $\beta \cdot x \cdot y \cdot z$. A 3DBF uses a three-dimensional bit array as in Definition 2 whose dimensional parameter (x, y, z) is chosen to be a triple of distinct prime numbers [7]. Suppose $m \in S$ is a message we want to input to a 3DBF scheme. In our proposed system, we first need to find (at least) one hash value of m.

Assume that we use one hash function hash to compute the hash value of m and we define $\mathsf{hash}(m) = h$. We then represent h as a decimal number and compute $i = h \bmod x$, $j = h \bmod y$, and $k = h \bmod z$. The last operations yield a triple (i,j,k) such that $0 \leq i < x$, $0 \leq j < y$, and $0 \leq k < z$. This triple also signifies a particular (i,j,k)-th cell in the bit array $\mathbf{B}$. Observe that the value of (i,j,k) is unique for all h such that $0 \leq h < xyz$. The values in the three-dimensional bit array are constructed by changing the value of $B_{i,j,k}$ in $\mathbf{B}$ according to the hash value of m. We further explain this process in our data insertion algorithm.

A. Insertion Algorithm

Our proposed 3DBF scheme uses a hash function hash to compute the hash value $\mathsf{hash}(m)$ for any input m. Here, m is a concatenation of a username and its corresponding password. This scheme also employs a three-dimensional bit array $\mathbf{B}$ of dimension (x, y, z) such that all x, y, and z are distinct prime numbers. Each cell value $B_{i,j,k}$ in $\mathbf{B}$ stores a bit string of size β where typically β is either 32 or 64. Initially, the array $\mathbf{B}$ is set to zero, i.e., $B_{i,j,k}$ is set to be a string of zeros of length β for every $0 \leq i < x$, $0 \leq j < y$, and $0 \leq k < z$.

Suppose we want to insert a message m into the three-dimensional bit array $\mathbf{B}$. As in the one-dimensional BF, we associate the hash value of m, namely $\mathsf{hash}(m)$, instead of m itself in the array $\mathbf{B}$. We first need to compute the hash value of m and convert this value into a decimal number, let us denote this number by $d(m)$. We use $d(m)$ to indicate that a key associated with m exists in our filter using the following steps. First, we need to determine the (i,j,k)-th cell in $\mathbf{B}$ using the equations $i = d(m) \bmod x$, $j = d(m) \bmod y$, and $k = d(m) \bmod z$. This (i,j,k) indicates the cell position in our bit array which initially is filled with a string of zeros of length β. Next, we compute $\rho = d(m) \bmod \beta$. Observe that ρ satisfies $0 \leq \rho \leq \beta - 1$. To indicate that we store m in our filter, we set the $(\rho+1)$-th rightmost digit of $B_{i,j,k}$ to 1. This operation can also be expressed using the left-shift operator, that is, $B_{i,j,k} \leftarrow B_{i,j,k} \mid (1 << \rho)$ where $\mid$ denotes the *bit-wise or operation*. If this position has been previously set to 1, our 3DBF generates a notification that m has been stored in our filter. In other words, we use a non-blind insertion technique, i.e., we check the existence of a key of an element before we add it to our filter (see [7, Section 3.1.] for more explanations about non-blind and blind insertion method). A false positive match occurs when this digit has been set to 1 but $d(m)$ has not been associated with any entry in our filter using the aforementioned process.

To explain our insertion process in detail, we first describe the procedure SETBIT$(\mathbf{B}, i, j, k, \rho)$ in Algorithm 1 which is adapted from [7, Algorithm 2]. This procedure requires a three-dimensional bit array $\mathbf{B}$ of dimension (x, y, z), the cell position (i,j,k), and the value ρ indicating the bit position within $B_{i,j,k}$. The output is an updated value of $B_{i,j,k}$ such that its $(\rho+1)$-th rightmost bit is set to 1.

Algorithm 1 SETBIT$(\mathbf{B}, i, j, k, \rho)$ sets the $(\rho+1)$-th rightmost bit of $B_{i,j,k}$ to 1.

```
Input: A three-dimensional bit array $\mathbf{B}$, a cell position $(i, j, k)$, and a value $\rho$ indicating the bit position within $B_{i,j,k}$.
Output: The $(\rho+1)$-th rightmost bit of $B_{i,j,k}$ is set to 1.
1: $v \leftarrow B_{i,j,k}$
2: $w \leftarrow (1 << \rho)$
3: $B_{i,j,k} \leftarrow v \mid w$   ▷ bit-wise or operation between $v$ and $w$
4: return $B_{i,j,k}$
```

In Algorithm 1, line 1 takes a bit string $B_{i,j,k}$, line 2 constructs a bit string whose values are 0 except at the $(\rho+1)$-th rightmost bit, and line 3 updates the value of $B_{i,j,k}$ with a new one such that the $(\rho+1)$-th rightmost bit of $B_{i,j,k}$ is set to 1. Notice that the value $B_{i,j,k}$ remains unchanged if initially the rightmost $(\rho+1)$-th position of $B_{i,j,k}$ has been already set to 1. Assuming that the bit-wise operation takes $\mathcal{O}(1)$ time, then it is obvious that Algorithm 1 takes $\mathcal{O}(1)$ time.

Next, we explain the function CHECKBIT$(\mathbf{B}, i, j, k, \rho)$ in Algorithm 2 that is adapted from [7, Algorithm 3]. This function checks whether the $(\rho+1)$-th rightmost bit of $B_{i,j,k}$ has been set to 1. The function returns true if such a condition happens and false otherwise. In Algorithm 2, line 3 computes the *bit-wise xor* between the bit string $B_{i,j,k}$ and a bit-string w that contains 0 except at the $(\rho+1)$-th rightmost position and stores its result in a variable s. Line 4 computes the *bit-wise and* between s and the previous bit string w and stores its result in a variable t. The bit string $B_{i,j,k}$ contains 1 at its $(\rho+1)$-th rightmost bit if the decimal representation of t is zero but the decimal representation of v is non-zero. By abuse of notation, Algorithm 2 uses the formula $(B_{i,j,k}{}^\wedge(1 << \rho))\&(1 << \rho)$ where $^\wedge$ denotes the bit-wise xor operation and $\&$ denotes the bit-wise and operation, and checks whether the result of this formula is non-zero. If the result is zero but $B_{i,j,k}$ is non-zero, then we infer that the $(\rho + 1)$-th rightmost bit of $B_{i,j,k}$ is 1. Assuming that all bit-wise operations require $\mathcal{O}(1)$ time, we infer that Algorithm 2 takes $O(1)$ time as well.

Algorithm 2 CHECKBIT($\mathbf{B}, i, j, k, \rho$) checks whether the $(\rho+1)$-th rightmost bit of $B_{i,j,k}$ is 1.

```
Input: A three-dimensional bit array $\mathbf{B}$, a cell position $(i, j, k)$, and a value $\rho$ indicating the bit 1 position within $B_{i,j,k}$.
Output: The function returns true if the $(\rho + 1)$-th rightmost bit of $B_{i,j,k}$ is 1 and false otherwise.
1: $v \leftarrow B_{i,j,k}$
2: $w \leftarrow (1 << \rho)$
3: $s \leftarrow v$ ^ $w$                 ▷ bit-wise xor operation
4: $t \leftarrow s \,\&\, w$               ▷ bit-wise and operation
5: $t_d = d(t)$                ▷ converting $t$ to decimal value
6: $v_d = d(v)$                ▷ converting $v$ to decimal value
7: return ($t_d = 0$ and $v_d \neq 0$)
```

As in [7], we also use the function SETCOUNT($\mathbf{B}$) to compute the total number of unique input counts. Here, our SETCOUNT($\mathbf{B}$) definition is identical to that in [7, Algorithm 4]. This SETCOUNT($\mathbf{B}$) function is important if we use a non-blind insertion method. We also use SETCOUNT($\mathbf{B}$) for another algorithm as a subroutine to determine whether the three-dimensional bit array $\mathbf{B}$ is full.

If we consider a three-dimensional bit array $\mathbf{B}$ of dimension (x, y, z) such that each $B_{i,j,k}$ stores β bits, then the maximum number of bits that can be stored in this array is $\beta \cdot x \cdot y \cdot z$. However, as in [7], we can choose not to use all available bits for storing the data. We define the threshold τ as $(\beta/\alpha) \cdot xyz$ where α is an integer between 1 and β (inclusive). Observe that τ satisfies $xyz \leq \tau \leq \beta \cdot xyz$. If $\tau = \beta \cdot xyz$, then theoretically all bits in $\mathbf{B}$ can be used.

The value β/α is between 1 and β and it is called as the *criticality factor*, while the value α itself is referred to as the *false positive intolerance factor* [7]. If the criticality factor of a BF scheme is β, then all bits in $\mathbf{B}$ can be used and the system can tolerate a higher false positive rate. In addition, if the criticality factor is 1, then the false positive rate is zero, but only xyz bits in the array can be used. According to Patgiri et al., there is a trade-off between the false positive rate and the number of bits that can be used in the system, namely, the more bits are used, the higher the false positive rate is [7].

Furthermore, observe that the higher the intolerance factor used in the system, the fewer bits can be used to store the data. For example, if we consider a three-dimensional bit array $\mathbf{B}$ of dimension (x, y, z) with $\beta = 64$ and intolerance factor $\alpha = 8$, then only $(64/8) \cdot xyz = 8xyz$ bits of $\mathbf{B}$ can be used, which is one-eighth of the original size. Nevertheless, theoretically, this scheme has a lower false positive rate than the system that uses all available bits.

We describe the function ISFULL($\mathbf{B}, \tau$) that examines whether a three-dimensional bit array $\mathbf{B}$ of dimension (x, y, z) is full if the threshold of the array is $\tau = (\beta/\alpha) \cdot xyz$ in Function 1. Each cell value $B_{i,j,k}$ in $\mathbf{B}$ holds β bits. This method is adapted from [7, Algorithm 5] and uses the aforementioned SETCOUNT($\mathbf{B}$) function and runs in $\mathcal{O}(1)$ time.

```
Input: A three-dimensional bit array $\mathbf{B}$ of dimension $(x, y, z)$ and a threshold value $\tau = (\beta/\alpha) \cdot xyz$.
Output: The function returns true if the three-dimensional bit array $\mathbf{B}$ is full with the threshold value $\tau$ and false otherwise.
1: if SETCOUNT($\mathbf{B}$) $= \tau$ then
2:     return true
3: else
4:     return false
5: end if
```

Function 1: ISFULL($\mathbf{B}, \tau$) examines whether a three-dimensional bit array $\mathbf{B}$ of dimension (x, y, z) is full if the threshold value is τ where $\tau = (\beta/\alpha) \cdot xyz$.

We are now ready to describe our non-blind data insertion procedure in our 3DBF scheme which is adapted from [7, Algorithm 6]. This process is described in Algorithm 3 using the procedure INSERT($\mathbf{B}, m$) that executes a non-blind insertion of a message m into a three-dimensional bit array $\mathbf{B}$ of dimension (x, y, z) such that each $B_{i,j,k}$ stores a bit string of length β. We also assume that the threshold for $\mathbf{B}$ is τ.

Algorithm 3 initially computes the hash value of m and converts its value into a decimal number denoted by $d(m)$. Here, we use the cryptographically secure SHA256 algorithm to determine $\mathsf{hash}(m)$. Subsequently, line 2 of Algorithm 3 determines the triple (i, j, k) such that $i = d(m) \bmod x$, $j = d(m) \bmod y$, and $k = d(m) \bmod z$. The triple (i, j, k) indicates the (i, j, k)-th cell in the three-dimensional bit array $\mathbf{B}$ that contains a bit string $B_{i,j,k}$ of size β. In line 3 of Algorithm 3 we compute the value ρ to determine the bit position of $B_{i,j,k}$ that is associated to the message m.

Line 4 of Algorithm 3 is performed to check whether the three-dimensional bit array $\mathbf{B}$ is not full and the $(\rho + 1)$-th rightmost bit of $B_{i,j,k}$ is 0. If both conditions are satisfied, then we increment the number of unique inputs using SETCOUNT($\mathbf{B}$) function and subsequently set the $(\rho + 1)$-th rightmost bit of $B_{i,j,k}$ to 1 to indicate that m has been inserted in our filter. If at least one of the conditions is not satisfied, then either the three-dimensional bit array $\mathbf{B}$ is full or the $(\rho + 1)$-th rightmost bit of $B_{i,j,k}$ has been set to 1. In either case, Algorithm 3 provides a pertinent notification as in line 9 or line 11. Assuming that all modulo operations in lines 1, 2 , and 3 take $\mathcal{O}(1)$ time each and every aforesaid subroutine runs in $\mathcal{O}(1)$ time, then we infer that Algorithm 3 requires $\mathcal{O}(1)$ time.

B. Search (Query) Algorithm

Suppose we consider a message m and a three-dimensional bit array $\mathbf{B}$ of dimension (x, y, z) such that each cell (i, j, k) stores a bit string of length β for every $0 \leq i < x$, $0 \leq j < y$, and $0 \leq k < z$. The objective of the search (query) algorithm in our 3DBF scheme is to determine whether the key associated with the message m exists in $\mathbf{B}$. If such a key occurs, then we infer that m has been stored in our filter. Our proposed search algorithm is adapted from [7, Algorithm 7]

Algorithm 3 INSERT($\mathbf{B}, m$) performs a non-blind insertion of a message m to a three-dimensional bit array $\mathbf{B}$ used in 3DBF scheme.

Input: A message m, a three-dimensional bit array $\mathbf{B}$ of dimension (x, y, z) such that each cell consists of β bits, and a threshold value τ of the 3DBF scheme.

Output: Modified three-dimensional bit array $\mathbf{B}$ or a relevant information about $\mathbf{B}$.

```
1: $h \leftarrow \mathsf{hash}(m); d(m) \leftarrow$ decimal value of $h$
2: $i \leftarrow d(m) \bmod x; j \leftarrow d(m) \bmod y; k \leftarrow d(m) \bmod z$
3: $\rho \leftarrow d(m) \bmod \beta$
4: if IsFull($\mathbf{B}, \tau$) = false and CheckBit($\mathbf{B}, i, j, k, \rho$) = false then
5:     SetCount($\mathbf{B}$)                  ▷ increment the number of unique inputs
6:     SetBit($\mathbf{B}, i, j, k, \rho$)     ▷ the $(\rho + 1)$-rightmost bit of $B_{i,j,k}$ is set to 1
7: else
8:     if IsFull($\mathbf{B}, \tau$) = true then
9:         "The three-dimensional bit array $\mathbf{B}$ is full."
10:    else
11:        "The element $m$ has been added to $\mathbf{B}$."
12:    end if
13: end if
```

and uses the function CHECKBIT explained in Algorithm 2 as a subroutine.

We explain this process using the function SEARCH($\mathbf{B}, m$) in Algorithm 4 that takes a three-dimensional bit array $\mathbf{B}$ and a message m as inputs. Observe that lines 1, 2, and 3 of Algorithm 4 are respectively identical to the corresponding lines of Algorithm 3. This function subsequently determines whether the $(\rho + 1)$-th bit of $B_{i,j,k}$ is 1 using the function CHECKBIT($\mathbf{B}, i, j, k, \rho$) and stores its value in a Boolean variable $found$ as explained in line 4. Notice that, if $found$ is true, then we infer that the key associated with m exists in $\mathbf{B}$, and thus the algorithm returns true. Analogously, if $found$ is false, then we infer that the key associated with m does not exist in $\mathbf{B}$, and hence the algorithm returns false. Since CHECKBIT function runs in $\mathcal{O}(1)$ time and using the assumption that all modulo operations in lines 1, 2, and 3 require $\mathcal{O}(1)$ time each, then we infer that Algorithm 4 requires $\mathcal{O}(1)$ time.

Algorithm 4 SEARCH($\mathbf{B}, m$) determines whether a key associated to a message m exists in $\mathbf{B}$.

Input: A three-dimensional bit array $\mathbf{B}$ of dimension (x, y, z) and a message m.

Output: The function returns true if the bit associated with m in $\mathbf{B}$ is set to 1 and false otherwise.

```
1: $h \leftarrow \mathsf{hash}(m); d(m) \leftarrow$ decimal value of $h$
2: $i \leftarrow d(m) \bmod x; j \leftarrow d(m) \bmod y; k \leftarrow d(m) \bmod z$
3: $\rho \leftarrow d(m) \bmod \beta$
4: $found \leftarrow$ CheckBit($\mathbf{B}, i, j, k, \rho$)
5: return $found$
```

C. Correctness Performance Indicator

Our 3DBF uses a three-dimensional bit array $\mathbf{B}$ with dimensional parameter (x, y, z) where each cell value $B_{i,j,k}$ stores β bits of information. Consequently, the total number of bits available is $\beta \cdot xyz$. However, as previously mentioned in Section III-A, we can choose not to use all bits in $\mathbf{B}$ by considering an intolerance factor $\alpha \in \{1, 2, \cdots, \beta\}$. If we consider a 3DBF scheme with an intolerance factor α, then the total number of usable bits in this scheme is $\beta/\alpha \cdot xyz$.

The correctness performance of an rDBF scheme is measured using the false positive rate. In a conventional one-dimensional BF, this quantity, denoted by FPR, is defined as

$$FPR = 1 - \left(1 - \frac{1}{m}\right)^{nk}, \tag{1}$$

where m is the total number of bits usable in BF (total bits that can be used in the bit array), n is the number of elements inserted into the BF, and k is the total number of hash functions used [4]. In a 3DBF scheme of dimension (x, y, z), we have $m = \beta/\alpha \cdot xyz$, and thus (1) becomes

$$FPR = 1 - \left(1 - \frac{1}{\beta/\alpha \cdot xyz}\right)^{nk}. \tag{2}$$

Eq. (2) is a special case of the more general formula in rDBF (see [7, p. 112]). Since our proposed 3DBF scheme uses one hash function in its insert and search procedures, the false positive rate is measured using the following formula

$$FPR = 1 - \left(1 - \frac{1}{\beta/\alpha \cdot xyz}\right)^{n}. \tag{3}$$

The value β/α is referred to as the criticality factor of a 3DBF scheme. This value is related to the *occupancy level* of the scheme that quantifies the number of bits that can be used in the bit array. Formally, we define the occupancy level as a rational number within the interval $(0, 1]$. Notice that if $\alpha = 1$, then the occupancy level of the scheme is 100%, meaning that all bits in the associated three-dimensional bit array $\mathbf{B}$ can be used. Similarly, if $\alpha = 2$, then the occupancy level of the scheme is 50%, indicating that only half of all bits in the associated three-dimensional bit array $\mathbf{B}$ is usable.

Observe that the original definition of criticality factor in [7] is somewhat restricted since one must be able to express its value as β/α where α is an integer between 1 and β

(inclusive). For example, this definition does not allow an occupancy level of 0.1 in a 3DBF that uses 64 bit in each cell of its three-dimensional bit array, since $64/\alpha = 0.1$ implies $\alpha = 640$, which is not an integer between 1 and 64. However, it is possible to use around 10% of all available bits in a three-dimensional bit array $\mathbf{B}$ if the dimension (x, y, z) is sufficiently large. As an illustration, in a 3DBF scheme that uses a three-dimensional bit array $\mathbf{B}$ of dimension $(2, 3, 5)$ whose each cell contains 64 bit string, then 10% of all available bits is 192 bits. In other words, our definition of occupancy level is more general than the criticality factor proposed by Patgiri et al. in [7], since it is not always necessary for the occupancy level to be expressed as a ratio between β and α, so long as its value is a rational number in the interval $(0, 1]$. In this paper, we use this notion of occupancy level instead of the criticality factor to evaluate the performance of a 3DBF scheme. As a consequence, we measure the false positive rate of 3DBF of dimension (x, y, z) and an occupancy level C as

$$FPR = 1 - \left(1 - \frac{1}{C \cdot \beta \cdot xyz}\right)^n. \tag{4}$$

IV. Experimental Results and Mathematical Analysis

Our computational experiments were mainly carried out to investigate the false positive rates of several 3DBF schemes of various dimensional parameters for various values of occupancy levels and the number of inputs. In other words, we investigated the effect of the values of (x, y, z), C, and n, to the value of FPR according to (4). We initially generated 1 000 000 random pairs of strings of usernames and passwords. Each username contains between 7 to 10 alphanumeric characters, while its corresponding password is constructed of more random ASCII characters of length 7 to 11 (including some non-alphanumeric characters). Thus, a message is a string of ASCII characters of length 14 to 21. To associate a message in a 3DBF scheme, we used the SHA256 algorithm as our hash function and compute one hash value of such a message. This hash value is later converted to a decimal number.

Notice that a 3DBF dimensional parameter is defined as a triple (x, y, z) such that x, y, and z are distinct prime numbers. Suppose we consider a three-dimensional bit array $\mathbf{B}$ of dimension (p, q, r) whose each cell stores β bits. The total number of bits in this array is identical to the number of bits in the arrays whose dimensions are obtained from the permutations of (p, q, r), e.g., three-dimensional bit arrays of dimensions (p, r, q), (q, p, r), and (q, r, p). All of these arrays accommodate $\beta \cdot pqr$ bits in total if each cell stores β bits. Using this argument, we restrict our investigation to the 3DBF whose dimensional parameter is (x, y, z) such that $x < y < z$. Moreover, due to some limitations of our computational environment, we only consider the prime numbers between 2 and 47 (inclusive). In addition, we also define $\beta = 64$ in all of our three-dimensional bit arrays. As a result, the smallest three-dimensional bit array in our experiments stores $64 \cdot 2 \cdot 3 \cdot 5 = 1920$ bits whereas the largest one holds $64 \cdot 41 \cdot 43 \cdot 47 = 5\,303\,104$ bits. Our experiments are performed using Python 3 language in Google Colaboratory using its default specification. We put our source code, datasets, and other pertinent experimental materials related to our research for interested readers at `https://github.com/csmarz/SecureSpaceEfficient3DBF`.

A. Empirical Relationship Between Occupancy Level and the False Positive Rate

In our first experiment, we determined the false positive rates of several 3DBF schemes whose occupancy levels are between 0.10 to 0.95 with 0.05 increment. We computed the maximum, minimum, and average values of FPR in several 3DBF schemes whose dimensions vary from $(2, 3, 5)$ to $(41, 43, 47)$. The number of data to be inserted varies between 192 and 530 311 items from 1000 000 available records.

Notice that 192 is 10% of the available total bits of a three-dimensional bit array of dimension $(2, 3, 5)$ whose each cell contains 64 bits, while 530 311 is 10% of the available total bits in a three-dimensional bit array of dimension $(41, 43, 47)$ whose each cell stores 64 bits. The relationship between the occupancy levels with the maximum and minimum FPR values is described in Fig. 1. The average FPR values are not plotted in this graph since it is obvious that these values are located between the minimum and maximum FPR values. Here, we see that the values of FPR are directly proportional to the values of occupancy levels. That is, the more bits are used, the higher the value of FPR. In our first experiment, the smallest value of $FPR = 0.095386$ occurs when the occupancy level is 0.1.

We conducted a further experiment to see the relationship between FPR values of several 3DBF schemes whose occupancy levels are between 0.010 and 0.095 with 0.005 increment. The objective of this experiment is to see the FPR values of 3DBF schemes if the number of bits used is between 1% and 9.5% from the total available bits. We use the same parameters as in the first experiment. The relationship of occupancy levels with the maximum and minimum value of FPR is depicted in Fig. 2

B. Mathematical Relationship Between Occupancy Level and the False Positive Rate

Suppose we consider a 3DBF scheme of dimension (x, y, z) whose three-dimensional bit array $\mathbf{B}$ holds β bits in each of its cells. Suppose we denote A as the total number of available bits in this scheme, that is, $A = \beta \cdot xyz$. If the occupancy level of this scheme is $C \in (0, 1]$, then the total usable bits in this scheme is $C \cdot \beta \cdot xyz = C \cdot A$.

From the experimental results in Section IV-A, we see that the false positive rate of a 3DBF scheme is directly proportional to its occupancy level. Now we are interested to determine the minimum number of bits required for storing n items of data in a 3DBF scheme of dimension (x, y, z) given an acceptable value FPR of false positive rate and the value C of occupancy level. In other words, we want to determine the value $A = \beta \cdot xyz$ if the values FPR of false positive rate and C of occupancy level are known. This result can be used to further determine the dimensional parameter of the 3DBF

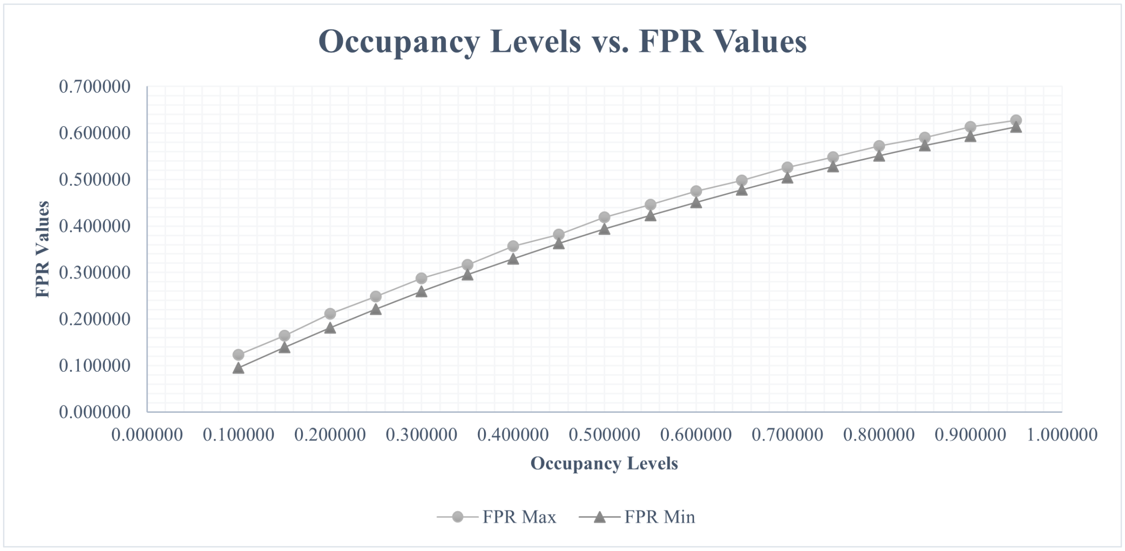

Fig. 1. The relationship between occupancy levels with the maximum and minimum FPR values. Here, the occupancy levels are observed for any value between 0.10 and 0.95 (inclusive) with 0.05 increment.

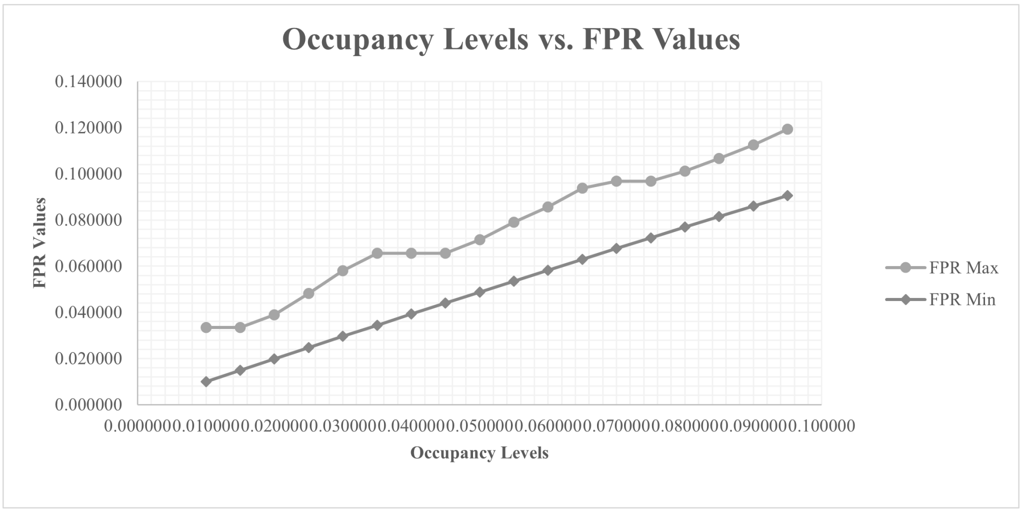

Fig. 2. The relationship between occupancy levels with the maximum and minimum FPR values. Here, the occupancy levels are observed for any value between 0.010 and 0.095 (inclusive) with 0.005 increment.

needed to store the data given the description of its acceptable false positive rate, occupancy level, and the number of bits in each cell of the three-dimensional bit array. We state these important relationships in Theorem 1 and Corollary 1.

Theorem 1: The minimum number of bits for storing n items of data in a 3DBF scheme of dimension (x, y, z) given the value of the acceptable false positive rate of FPR and the value of occupancy level C is A where

$$A = \left\lceil \frac{1}{\left[1-(1-FPR)^{1/n}\right] \cdot C} \right\rceil, \tag{5}$$

where $\lceil \dots \rceil$ denotes the ceiling function, i.e., a function that returns an integer greater than or equal to the input.

Proof: From (4), we have

$$FPR = 1 - \left(1 - \frac{1}{C \cdot \beta \cdot xyz}\right)^n, \text{thus}$$

$$\begin{aligned} 1 - FPR &= \left(1 - \frac{1}{C \cdot \beta \cdot xyz}\right)^n \\ (1-FPR)^{1/n} &= 1 - \frac{1}{C \cdot \beta \cdot xyz} \\ \frac{1}{C \cdot \beta \cdot xyz} &= 1 - (1-FPR)^{1/n}, \text{consequently} \\ C \cdot \beta \cdot xyz &= \frac{1}{1-(1-FPR)^{1/n}} \\ \beta \cdot xyz &= \frac{1}{\left[1-(1-FPR)^{1/n}\right] \cdot C}. \end{aligned}$$

Since $A = \beta \cdot xyz$ and the number of bits must be an integer, then the result follows. ■

The result in Theorem 1 also tells us that the product of the dimensional parameter (xyz) is inversely proportional to the value of the false positive rate (FPR) and occupancy level (C). The following corollary is an immediate consequence of

Theorem 1.

Corollary 1: Suppose we consider a 3DBF of dimension (x, y, z) that is used to store n items of data using an acceptable positive rate of FPR and occupancy level of C. If each cell in the three-dimensional array used in this scheme stores β bits, then (x, y, z) satisfies

$$xyz \geq \frac{1}{\left[1 - (1 - FPR)^{1/n}\right] \cdot C \cdot \beta} \tag{6}$$

We illustrate the application of Theorem 1 and Corollary 1 in the following example.

Example 2: Suppose we want to store $n = 1000$ data using a 3DBF whose value of false positive rate is no more than $FPR = 0.1$ and its occupancy level is at most $C = 0.2$. Then, using (5), the minimum number of bits required to store these items, namely A, satisfies

$$\left\lceil \frac{1}{\left[1 - (1 - 0.1)^{1/1000}\right] \cdot 0.2} \right\rceil = 47\,459 \text{ bits.} \tag{7}$$

If we consider a 64 bit scheme, i.e., a 3DBF scheme that employs a three-dimensional bit array whose every cell contains 64 bits, then the product of dimensional parameter (x, y, z) satisfies

$$xyz \geq \frac{1}{\left[1 - (1 - 0.1)^{1/1000}\right] \cdot 0.2 \cdot 64} = 741.540749. \tag{8}$$

This means we need to use a three-dimensional bit array of dimension (x, y, z) such that $x \cdot y \cdot z \geq 741.540749$. We can choose a 3DBF scheme of dimension $(x, y, z) = (7, 11, 13)$. Observe that $7 \cdot 11 \cdot 13 = 1001$. Furthermore, the FPR of a 3DBF scheme of dimension $(7, 11, 13)$ with $C = 0.2$ and $\beta = 64$ is $0.075081 < 0.1$.

C. Empirical Running Time for Insert and Search (Query) Operations

In addition to the experiments related to the investigation between the false positive rates and the occupancy levels of the 3DBF scheme, we also performed an experiment to determine the empirical running time for insert and search (query) operations. As previously stated in Section III-A, the insert operation is carried out using the procedure INSERT explained in Algorithm 3 and it requires $\mathcal{O}(1)$ time for inserting one element. This procedure calls the algorithms ISFULL, CHECKBIT, SETCOUNT, and SETBIT as subroutines, each of them also runs in $\mathcal{O}(1)$ time. On the other hand, the search operation is performed using the function SEARCH described in Algorithm 4 and it runs in $\mathcal{O}(1)$ time for searching one element in a 3DBF scheme. As in the INSERT procedure, this algorithm also uses the CHECKBIT function as a subroutine.

Although both INSERT and SEARCH algorithms run in $\mathcal{O}(1)$ time, we notice that the INSERT procedure in Algorithm 3 involves more operations and uses more subroutines than the SEARCH procedure in Algorithm 4. As a result, intuitively inserting an element into a 3DBF scheme takes more time than searching for such an element in an identical scheme, although the difference is only up to a constant factor. We performed an experiment to insert n items where $192 \leq n \leq 530\,311$ in a 3DBF scheme of false positive rate $FPR = 0.1$ and occupancy level $C = 0.1$. In this experiment, the total bits in the 3DBF schemes vary from 1920 to $5\,303\,104$. For each of these items, we also performed a search (query or look-up) operation. We compared the empirical running time for the insert and search operations and plot the result in Fig. 3. From this result, we infer that the running time for inserting or searching items is linearly proportional to the number of items. Furthermore, the time required for inserting an item is slower than that for searching for such an item in an identical 3DBF scheme by a constant factor.

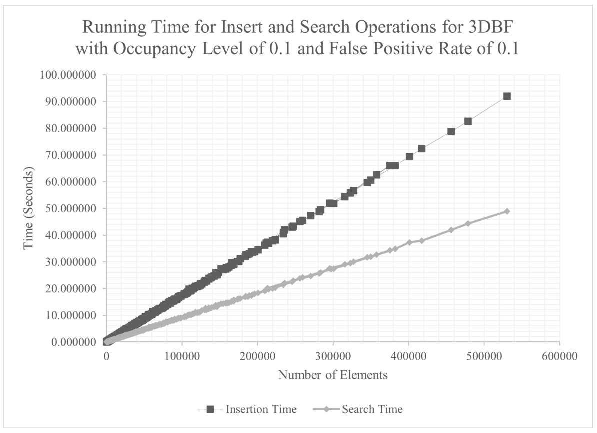

Fig. 3. The processing time for inserting and searching n items where $192 \leq n \leq 530\,311$ in 3DBF scheme of false positive rate $FPR = 0.1$ and occupancy level $C = 0.1$.

V. CONCLUDING REMARKS

From experimental results in Section IV-A, we conclude that the value of the false positive rate is directly proportional to the value of occupancy level in a 3DBF scheme. We describe our mathematical analysis in Section IV-B. To begin with, we provide a mathematical formulation for determining the minimum number of bits required to store n items of data in a 3DBF scheme of false positive rate FPR and occupancy level C in Theorem 1. The result in Theorem 1 also tells us that the product of the dimensional parameter is inversely proportional to the value of the false positive rate and occupancy level in the 3DBF scheme. This formulation is further refined in Corollary 1 to affirm a sufficient condition for the dimensional parameter (x, y, z) in a 3DBF scheme to store n items given a false positive rate of FPR and occupancy level C, assuming that each cell of the three-dimensional bit array used in the 3DBF holds β bits. We provide a relevant illustration of this corollary in Example 2.

Our recommended scheme is adapted from the system proposed by Patgiri et al. [7]. This scheme only uses one hash function to determine the associated key of an item. However, the optimal number of hash functions used in a conventional one-dimensional BF of capacity m with n input items satisfies $m/n \cdot \ln 2$ (see, e.g., [4], [12]). Nevertheless, to our knowledge, investigations regarding the optimal number of the hash function used in 3DBF (or rDBF in general) have not been carried out rigorously.

In our investigation, the dimensional parameter of a 3DBF scheme is defined as a triple (x, y, z) such that x, y, and z are distinct prime numbers. This criterion is taken from the argument by Patgiri et al. to avoid collision and thus keep the false positive rate sufficiently low [7]. Nevertheless, there is no rigorous mathematical argument for this stipulation. We conjecture that to keep the false positive rate sufficiently low, we can use a dimensional parameter (x, y, z) such that x, y, and z are pairwise relatively prime (i.e., the greatest common divisors of every pair of distinct numbers are 1). This conjecture needs further experimental and theoretical explorations.

Finally, our experiments are somewhat limited since the prime numbers used are no more than 47. The maximum number of available bits in the 3DBF scheme is around five million. We suggest further investigation by considering 3DBF schemes whose dimensional parameters are (x, y, z) such that x, y, and z are pairwise relatively prime numbers between 1 and 100. Notice that there are $47\,086$ of such triples and if each cell of a three-dimensional bit array holds 64 bits, then the maximum number of bits is $61\,459\,200$ (around twelve times larger than that in our experiments), which is obtained when the dimension is $(97, 99, 100)$.

References

[1] S. Long, "A Comparative Analysis of the Application of Hashing Encryption Algorithms for MD5, SHA-1, and SHA-512," *Journal of Physics: Conference Series*, vol. 1314, no. 1, 2019.

[2] W. K. Pertiwi, "*Anggota Forum Hacker Klaim Punya Data 13 Juta Akun Bukalapak* (In Bahasa Indonesia)," 2020. [Online]. Available: https://tekno.kompas.com/read/2020/05/06/09390047/anggota-forum-hacker-klaim-punya-data-13-juta-akun-bukalapak?page=all

[3] R. Dwinanda, "*Tokopedia Laporkan Kasus Kebocoran Data Pengguna* (In Bahasa Indonesia)," 2020. [Online]. Available: https://www.republika.co.id/berita/qd18a0414/tokopedia-laporkan-kasus-kebocoran-data-pengguna

[4] R. Patgiri, S. Nayak, and S. K. Borgohain, "Role of Bloom Filter in Big Data research: A survey," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 11, pp. 655–661, 2018.

[5] A. Singh, S. Garg, S. Batra, N. Kumar, and J. J. P. C. Rodrigues, "Bloom filter based optimization scheme for massive data handling in IoT environment," *Future Generation Computer Systems*, vol. 82, pp. 440–449, 2018. [Online]. Available: https://doi.org/10.1016/j.future.2017.12.016

[6] W. Liu, Z. Xu, J. Tian, and Y. Zhang, "Towards In-Network Compact Representation: Mergeable Counting Bloom Filter Vis Cuckoo Scheduling," *IEEE Access*, vol. 9, pp. 55 329–55 339, 2021.

[7] R. Patgiri, S. Nayak, and S. K. Borgohain, "rDBF: A r-Dimensional Bloom Filter for massive scale membership query," *Journal of Network and Computer Applications*, vol. 136, no. March, pp. 100–113, 2019. [Online]. Available: https://doi.org/10.1016/j.jnca.2019.03.004

[8] F. Afianti, C. Asrini, and D. R. Wijaya, "Scalable Two-Dimensional Bloom Filter Membership Scheme on Massive Scale Wireless Sensor Networks," *International Journal of Intelligent Engineering and Systems*, vol. 14, no. 1, pp. 474–481, 2021.

[9] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[10] R. Patgiri, "HFil: A high accuracy bloom filter," *Proceedings - 21st IEEE International Conference on High Performance Computing and Communications, 17th IEEE International Conference on Smart City and 5th IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2019*, pp. 2169–2174, 2019.

[11] R. Patgiri, S. Nayak, and S. K. Borgohain, "PassDB : A password database with strict privacy protocol using 3D Bloom filter," *Information Sciences*, vol. 539, pp. 157–176, 2020. [Online]. Available: https://doi.org/10.1016/j.ins.2020.05.135

[12] D. Berardi, F. Callegati, A. Melis, and M. Prandini, "Password Similarity Using Probabilistic Data Structures," *Journal of Cybersecurity and Privacy*, vol. 1, no. 1, pp. 78–92, 2020.

[13] X. Zhong and Y. Liang, "Scalable downward routing for wireless sensor networks actuation," *IEEE Sensors Journal*, vol. 19, no. 20, pp. 9552–9560, 2019.

[14] Z. An, Q. Lin, L. Yang, W. Lou, and L. Xie, "Acquiring Bloom Filters across commercial RFIDs in physical layer," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1804–1817, 2020.

[15] G. Ramu, "A secure cloud framework to share EHRs using modified CP-ABE and the attribute Bloom Filter," *Education and Information Technologies*, vol. 23, no. 5, pp. 2213–2233, 2018.

[16] C. Xu, N. Wang, L. Zhu, K. Sharif, and C. Zhang, "Achieving searchable and privacy-preserving data sharing for cloud-assisted E-healthcare system," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8345–8356, 2019.

[17] AEPD and EDPS, *Introduction to the Hash Function as a Personal Data*, 2019, no. October.

[18] S. H. Lee and K. W. Shin, "An efficient implementation of SHA processor including three hash algorithms (SHA-512, SHA-512/224, SHA-512/256)," *International Conference on Electronics, Information and Communication, ICEIC 2018*, vol. 2018-January, pp. 1–4, 2018.