Dynamic Hierarchical Bloom Filters for Scalable Biometric Authentication Systems

Md Mashfiq Rizvee
University of Kansas
Lawrence, KS, USA
mashfiq.rizvee@ku.edu

Pallabi Ghosh
University of Florida
Gainesville, FL, USA
pallabighosh@ufl.edu

Domenic Forte
University of Florida
Gainesville, FL, USA
dforte@ece.ufl.edu

Sumaiya Shomaji
University of Kansas
Lawrence, KS, USA
shomaji@ku.edu

Abstract

Biometric authentication systems must handle large and constantly evolving datasets and maintain high authentication accuracy and scalability. Existing probabilistic data structures like Bloom Filters (BFs) efficiently handle membership testing, but their conventional forms are inherently static and lack tolerance for noisy data. It is also an inherent challenge in biometric systems, where factors such as lighting or camera angle can introduce noise because of these variability. This paper proposes the Dynamic Hierarchical Bloom Filter (DHBF), a novel variant of the Bloom Filter designed to integrate noise tolerance and substring handling capabilities, while simultaneously offering enhanced scalability and dynamic adaptability. The DHBF accommodates fluctuating dataset sizes and mitigates the impact of noisy biometric samples on authentication accuracy by supporting flexible memory allocation. In this paper, experimental validation was performed on a dataset of 30,000 facial images. The results demonstrate that the DHBF achieves 100% authentication accuracy in dynamic operations such as enrollment, querying, insertion, and deletion. Additionally, our experiments demonstrate a reduction ($\approx 30\%$) in storage requirements in terms of storing biometric templates compared to HBFs.

1. Introduction

In today's fast-paced digital landscape, managing and processing large datasets efficiently is more important than ever. This need is especially important in fields like biometrics where faster verification and privacy are required [14]. Biometric systems must adapt quickly to the changes in data volume as well as maintaining high levels of security, accuracy and performance. Therefore, efficient storage of biometric data has become increasingly crucial since biometric authentication systems are being rapidly adopted in various domains such as national security, healthcare, banking, and consumer electronics [11].

Biometric data typically includes sensitive information like facial recognition data or iris scans. The nature of such data poses several challenges. First, these datasets tend to be large and continuously growing as more users are enrolled in the system. Secondly, biometric data is inherently noisy [9]- i.e., an iris scan or facial recognition data can vary slightly due to environmental factors such as lighting or angle and make the system vulnerable to false positives and false negatives during the authentication. Additionally, the sensitivity of biometric data amplifies concerns around privacy and security. Unlike passwords or tokens, biometric information is immutable and irreplaceable [21], making it critical to prevent unauthorized access or breaches. Biometric data can potentially be exploited for identity theft or surveillance. Traditional data structures like the Bloom Filter (BF) are useful for probabilistically checking set membership with a highly efficient time complexity but they come with significant limitations. Even though BFs can be used implemented as an added security measure for biometric templates [10][6]; BFs are typically static, meaning they can't easily handle datasets where elements are frequently added or removed. Such rigidity often leads to higher false positive rates and suboptimal memory usage, especially in dynamic, large-scale environments which is very common in the field of biometrics.

Several advancements have been made to overcome these limitations, such as the Hierarchical Bloom Filter (HBF) and the Dynamic Bloom Filter (DBF). HBFs were designed to improve substring matching by breaking down strings into manageable components and checking parts of the string for membership, providing a more granular level of verification [25]. However, HBFs alone do not address the dynamic nature of datasets where elements may be added (without reconstruction) or deleted frequently. On the other hand, DBFs leverage multiple Counting Bloom Filters (CBFs) to offer flexible memory allocation and support dynamic changes in data, making them well-suited for applications that require scalability [12]. However, while DBFs are effective in handling dynamic data, they lack the noise tolerance capabilities necessary for more sensitive or noisy datasets, such as the biometric data.

Therefore, in this study, we introduce a novel biometric data storage and authentication scheme, Dynamic Hierarchical Bloom Filter (DHBF), an innovative framework that that is suitable for storing biometric templates (See Fig. 1). The DHBF not only accommodates the efficient handling of dynamically growing data where elements are frequently added and removed, but also enhances noise toler-

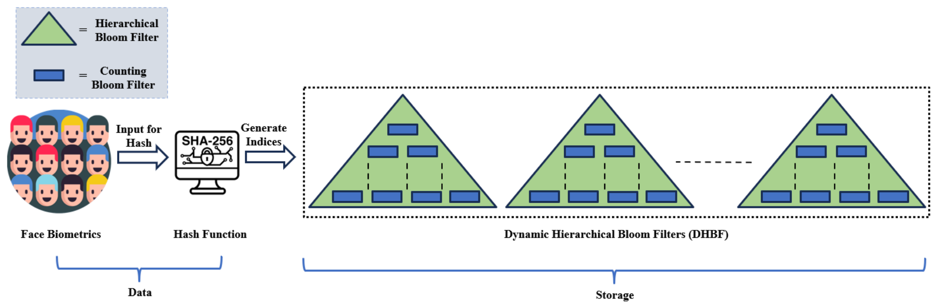

Figure 1. Dynamic Hierarchical Bloom Filter Overview. A More Detailed Version is Showcased in Fig. 3.

ance. The DHBF structure ensures that the system can grow and shrink as needed, maintain optimal memory usage, and control the false positive probability even as the set cardinality changes. Moreover, the usage of SHA256 ensures both uniformity and security [20]. We extract features from the biometric templates and then feed them into the irreversible SHA256 hash function. The generated output is further used as index for Counting Bloom Filters (CBF). We conducted experimental validation using a dataset of 30,000 human faces and demonstrated the robustness of DHBFs in managing dynamic data operations. The experiments included enrolling, querying, inserting, and deleting data, all of which showed that DHBFs maintained 100% accuracy in authenticating the biometric templates.

Overall, this paper makes the following contributions-

- Introduces a novel probabilistic data structure, DHBF, which combines optimal management of dynamically growing data and robust noise handling properties for storing biometric templates. To the best of the authors' knowledge, this is the only probabilistic data structure exhibiting these combined features in terms of biometric template storage.
- Derives an optimal authentication threshold selection process mathematically to determine the minimum threshold for accurate biometric authentication.
- Conducts experimental authentication on facial biometric data that demonstrate the robustness of DHBF in dynamic data environments.
- Demonstrates comparison of the operational efficiency and space-time complexity of different variants of BFs.

The rest of the paper is organized as follows: Sec. 2 provides an overview of the potential applications of DHBF in biometric enabled watchlists, while Sec. 3 reviews related work. Sec. 4 details the methodology, focusing on the DHBF architecture and its operations. Sec. 5 presents the results for each DHBF operation, Section 6 evaluates the scalability of DHBF in comparison to other BF variants. Sec. 7 briefly discusses about the security aspect of DHBF. Finally, the paper concludes in Sec. 8.

2. Re-imagining Biometric Watchlists

A watchlist is a systematic tool used across various domains to monitor, identify, and flag individuals, entities, or activities that pose potential risks or require special attention [27]. In the domain of **law enforcement**, watchlist screening is a mandatory mechanism of national and international security [3][7]. In the field of **education**, in smart classroom scenarios, watchlists can be integrated with to streamline automated attendance tracking [23]. This ensures that attendance records are accurate and tamper-proof. It can also identify any unauthorized individuals attempting to access the classroom. In **healthcare**, biometric watchlists help verify patient identities, reducing medical fraud and ensuring that only authorized personnel access sensitive patient data.

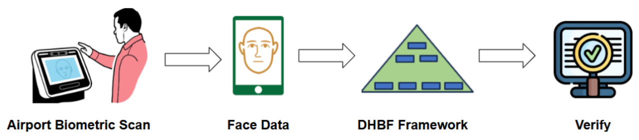

Figure 2. DHBF-based Database in Watchlist Technology.

Work in [16] outlines a data-driven model for biometric watchlist technology used in e-border applications. This model incorporates a central database to store physical and virtual evidence for traveler risk assessment and screening. The traditional database here can be replaced with a Dynamic Hierarchical Bloom Filter (DHBF) (See Fig. 2), in which the biometric features from face data is extracted and hashed before the hash outputs are securely stored rather than raw biometric data. This transition significantly enhances the privacy and data security while retaining the operational integrity of core components, such as the host system for data collection and the Authentication and Risk Assessment (ARA) station for traveler verification.

3. Related Works

In this section of the paper, we discuss related studies of the existing dynamic variants of BFs.

3.1. Preliminaries

Bloom Filter (BF): A BF is a space-efficient probabilistic data structure used for membership testing—i.e., deter-

Table 1. Comparison of Features Across Various Bloom Filter Techniques, Including the Proposed DHBF.

Feature	Scalable Bloom Filter	Cuckoo Filter	Dynamic Bloom Filter	Hierarchical Bloom Filter	Split Bloom Filter	Proposed DHBF
Memory Efficiency	No	No	Yes	Yes	No	**Yes**
Insertion	Yes	Yes	Yes	No	Yes	**Yes**
Deletion	No	Yes	Yes	No	No	**Yes**
Noisy Data Authentication	No	No	No	Yes	No	**Yes**
Dynamicity	Yes	Yes	Yes	No	Yes	**Yes**

mining whether an element is in a set. However, false positives are possible in BFs, but false negatives are not.

Hierarchical Bloom Filter (HBF): HBF is designed to enhance security, space efficiency, and rapid query handling in systems requiring large-scale biometric identification [25]. BFs, while effective in various applications, face limitations when dealing with noisy biometric data due to their inherent inability to handle such variability without significant false positives. The HBF addresses these challenges by structuring multiple layers of BFs.

Dynamic Bloom Filter (DBF): Proposed in [12], the DBF is another variant of the classic BF which handles sets that are not static but evolve over time. It extends the traditional BFs by allowing incremental expansions and contractions in its capacity, which is controlled by parameters that can be adjusted as the dataset grows or shrinks.

3.2. Other Dynamic Variants of BF

Scalable Bloom Filter: Scalable Bloom Filters [1] also employs a series of traditional BFs in an incremental manner but suffers from a drawback due to the use of heterogeneous BFs featuring different sizes and hashes.

Counting Bloom Filter: Counting Bloom Filters [5] extend Bloom Filters by using arrays of counters instead of simple bit arrays. Each position in this array contains a counter rather than a single bit (0 or 1).

3.3. Protecting Biometric Templates with SHA-256

The use of SHA-256 (i.e., works in [2, 15, 18]) ensures the stored face template is non-invertible (reveals no sensitive facial data) and is fixed-length, which is convenient for storage and comparison. Their system achieved state-of-the-art recognition accuracy on standard face databases while providing high security and template cancelability (meaning if a stored template is compromised, a new one can be issued by changing the encoding procedure).

4. Methodology

In this section, we describe the construction of the DHBF followed by its core operations, including enrollment, insertion, deletion, and query processing. The construction of the DHBF begins with the specification of three fundamental input parameters that define the structure and its operational constraints. The first input is the number of individuals (n) enrolled in the system, which determines the scale of the dataset. The second input is the false positive rate ($\mathrm{FP}_{\mathrm{CBF}}$) of the counting bloom filters (CBF), which establishes the likelihood of incorrect membership queries and impacts both accuracy and storage efficiency. The third input is the capacity of each instance of the DHBF (c). Once these inputs are defined, the DHBF is constructed using the method described in 4.1. Following the construction of the DHBF, the system performs a sequence of operations. The first operation is enrollment, during which individuals are registered in the system by enrolling their biometrics' hashed outputs into the hierarchical structure. As the dataset grows beyond the initial set cardinality and as the people who are not registered at the initial phase of enrollment, the insertion operation allows additional individuals to be dynamically added to the system. If a previously enrolled individual needs to be removed, the deletion operation is performed to remove a template. Throughout these operations, the query processing step can be executed at any stage to verify whether a given input is present in the DHBF. Querying is performed iteratively during each stage of the process to evaluate the performance and accuracy of the DHBF, as presented in Section 5.

4.1. Constructing a DHBF with CBF

Table 2. Dynamic Hierarchical Bloom Filter Notations.

Notation	Description
d	# of layers/ levels in HBF
U	upper bound of set cardinality
e	# of bits flipped due to noise in a template
k	# of hash functions in a CBF
l_i	# of bits in a block of a template in i^{th} layer of HBF
m	length of CBF in bits
n	# of biometric templates
N	total # of CBFs in an HBF
p_{CBF}	probability of having a falsely-set-to-1 bit in a CBF
s	number of HBF in DHBF
c	capacity of a single HBF instance
P	# of bits in a template
w	# of subsequent blocks concatenated in each layer of the HBF
N_t	# of threshold

The DHBF is designed as a hierarchical structure with multiple levels, forming a pyramid-like architecture. At the topmost level, biometric features extracted from a person's full face are hashed, producing a random numeric hash output. This hash output is treated as indices of a zero-bit array (or CBF), where the corresponding array positioned elements are incremented by 1 to record the occurrences of specific hashed feature values. This top level serves as the most general representation of the data as the whole biometric template (full face) is provided to the k hashes. As we progress down the hierarchical levels, the representation

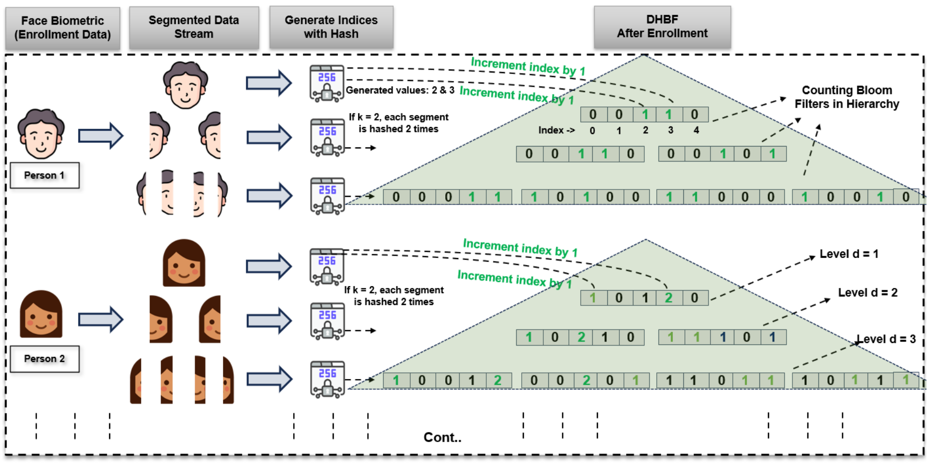

Figure 3. Storing Biometric Data in DHBF.

becomes increasingly granular. The lower levels partition the facial image into multiple segments, each of which is independently hashed k times (As an example shown for $k = 2$ in Fig. 3 where each segment of the face template is hashed $k = 2$ times across the 3 different levels). This hierarchical decomposition enables a more granular-level control over the stored information. Mathematically, if a biometric template with P bits is considered in which the noise has flipped e bits. Assuming that every bit can be noisy with equal probability, we obtain bit flip probability, $p_t = e/P$. When enrolling a set of a templates in a single DHBF instance, which is nothing but an HBF, has a static set cardinality, the templates are divided into blocks with length $l_i (0 \leq i \leq d)$. Therefore, the probability of having no error in a block would be $(1-p_t)^{l_i}$. As a result, the probability of correctly setting a bit to "1" is $(1-p_t)^{l_i}/m$ when enrolling a block in a CBF. Therefore, it would also be conclusive that the probability of having at least one erroneous bit after inserting a template would be $\left(1-(1-p_t)^{l_i}/m\right)^k$. If there are no collisions that occur during the insertion of all n blocks (e.g., the first block of all n templates), the probability of having at least one error in the CBF is $\left(1-(1-p_t)^{l_i}/m\right)^{kn}$. This probability is an approximation of p_{CBF} that can possibly occur in practice. Our HBF is composed of standard CBFs with an application-defined false positive ($\mathrm{FP_{CBF}}$). When processing a block of a query, k bit positions in a CBF are checked for a "1". As a matter of fact, we have the false positive as, $\mathrm{FP_{CBF}} = (1-p_{CBF})^k$. From the work mentioned in [7], we can say that, $\mathrm{FP_{HBF}} \approx 0$, even if $\mathrm{FP_{CBF}} \approx 10\%$. Therefore, in order to make sure that a noisy data gets authenticated, we have to define a threshold on the number of CBFs (N_t), depending on k, p_{CBF}, N, and l_i, so that if the number of CBFs authenticating a template exceeds that, it is determined as "found". More details on calculating the threshold (N_t) is discussed in the next subsection.

4.2. Threshold Calculation

When a query is executed, each CBF inside the HBF structure either matches or does not match the query, resulting in a number of "hits" at each level. To authenticate the query, a minimum number of hits, N_t, is required such that the confidence level α is satisfied. Mathematically, this problem is framed as determining N_t such that the cumulative probability of achieving N_t or more hits satisfies:

$$P(R \geq N_t) \geq \alpha \tag{1}$$

where R is a binomial random variable representing the number of successful matches. The binomial distribution models the probability of N_t successes out of N independent trials, with the probability of success in a single trial being t. The probability mass function of R is given by:

$$Prob(R = k) = \binom{N}{N_t} t^{N_t} (1-t)^{N-N_t} \tag{2}$$

where $\binom{N}{N_t}$ denotes the binomial coefficient. The cumulative probability, $Prob(R \geq N_t)$, is then computed as:

$$Prob(R \geq N_t) = \sum_{i=N_t}^{N} \binom{N}{i} t^i (1-t)^{N-i} \tag{3}$$

To ensure the desired confidence level α, we require the smallest N_t such that:

$$1 - \sum_{i=0}^{N_t-1} \binom{N}{i} t^i (1-t)^{N-i} \geq \alpha \tag{4}$$

This problem is efficiently solved using the percent-point function (PPF), which is the inverse of the cumulative distribution function (CDF). The PPF identifies the smallest N_t such that:

$$Prob(R \leq N_t) \geq 1 - \alpha \tag{5}$$

After that, this threshold is computed as:

$$N_t = \lceil \mathrm{PPF}(\alpha, N, t) \rceil \quad (6)$$

4.3. Enrollment in DHBF (Static Set Cardinality)

DHBF is built using HBF as its unit. In order to initialize, the input template (pixel intensities of grayscale images are considered as inputs) is normalized and quantized to get transformed as a binary string. Next, the string is split into fixed-size non-overlapping blocks. Then, each block is independently inserted into the hash functions. The generated values from the hashes are considered as index values of the CBFs. The corresponding CBF indices increment by 1. This may result in an increase in False Positive due to combinations of substrings that are incorrectly reported as being in the CBF. To tackle this problem, along with the substrings, their concatenations are also inserted into the filters. The HBF is made up of standard CBFs of N of the same size m. When a string is inserted, the smallest blocks of the string are added to the CBFs that are present at the highest level of the HBF hierarchy at level d. Consequently, the whole string, altogether is inserted into the CBF at level 1, that is, the lowest level (See Fig. 3). An external variable keeps track of the total number of items enrolled inside the DHBF framework. (Relevant notations shown in Tab. 2).

Algorithm 1 Insert(x)

```
Require: $x$ is not null
 1: ActiveHBF ← GetActiveHBF()
 2: if ActiveHBF is null then
 3:     ActiveHBF ← CreateHBF($m$, $k$, $U$, $c$)
 4:     Add ActiveHBF to this DHBF
 5:     $s \leftarrow s + 1$
 6: end if
 7: for $i = 1$ to $k$ do
 8:     ActiveHBF[$\mathrm{hash}_i(x)$] ← ActiveHBF[$\mathrm{hash}_i(x)$] + 1
 9: end for
10: ActiveHBF.$item$ ← ActiveHBF.$item$ + 1
11: return null
```

4.4. Insertion in DHBF (Dynamic Set Cardinality)

To demonstrate the dynamic nature of DHBF, we implemented an insertion process that showcases its ability to adapt to growing data. Before inserting a data into the DHBF, we initialize the parameter c, which defines the capacity of a single HBF in the DHBF. It is defined as if the number of items enrolled surpass a predefined capacity c, the DHBF dynamically adds another instance of an HBF. This feature highlights the scalability of the approach and ensures that the data structure can efficiently handle varying data loads while maintaining its core properties. A DHBF consists of s number of HBFs. The initial value of s is 1, and the HBF is active. An HBF is considered active when the number of items enrolled is less than c. The HBF only inserts items of a set into the active HBF and appends a new HBF as an active HBF when the previous active HBF becomes full. The newly inserted items will be placed on the new HBF until it reaches $2c$. Moreover, in order to represent a dynamic set with an upper bound, U is defined. In standalone applications in which it is not possible to know the total data in advance e.g., total internet users or distributed systems, the distribution of set cardinality covers a large range [8]. In such a distribution, the upper bound is sometimes set to be several orders of magnitude larger than the mean or minimum cardinality [13]. Thus, the total number of items U is also defined by the user. Therefore, given a dynamic set X with U items, the DHBF should first discover an active HBF when inserting an item x of X. If there are no active HBFs, the DHBF creates a new HBF as an active HBF and increments s by one. The steps for this operation is shown in Algo. 1.

Algorithm 2 Query(x)

```
Require: $x$ is not null
 1: for $i = 1$ to $d$ do
 2:     counter ← 0
 3:     for $j = 1$ to $k$ do
 4:         if $\mathrm{HBF}_i\left[\mathrm{hash}_j(x)\right] = 0$ then
 5:             break
 6:         else
 7:             counter ← counter + 1
 8:         end if
 9:     end for
10:     if counter $\geq N_t$ then
11:         return true
12:     end if
13: end for
14: return false
```

4.5. Querying in DHBF

It is convenient to represent X as a DHBF by invoking process of insertion repeatedly in order to enroll items dynamically inside the DHBF. After achieving the DHBF, we can answer any set membership queries based on the DHBF instead of X. It takes an item x as input. The $hash(x)$ directs to the indices where there are CBFs in the HBF. If all the $hash(x)$ counters are set to a nonzero value in the first HBF, then the item x is a member of X. Otherwise, it goes down the structure on the preceding levels to check the same non-zero values. It is important to note that, each level of the HBF has different threshold (N_t) values. If at any level, the threshold criteria, $counter > N_t$ (See Algo. 2) is met, the search is concluded. Else, the search moves on to the next one as the DHBF checks its second HBF, and so on. In summary, x is not a member of X if it is not found in all HBFs of the DHBF and is a member of X if it is found in any HBF of the DHBF.

4.6. Deletion in DHBF

The process of deletion in DHBF follows Algo. 4. The deletion takes place in the multi-level HBF structures. If an item x is removed from set X, the corresponding DHBF must execute deletion with x as the input in order to reflect X as consistently as possible. First of all, the DHBF identifies the HBF in which all the $hash(x)$ counters are set to a nonzero. If no HBF exists that satisfies the constraint in the DHBF, the item deletion operation will be rejected since x does not belong to X. If there is only one HBF satisfying the constraint, the counters $hash(x)$ are decremented by one. If there are multiple HBFs satisfying the constraint, then x may appear to be in multiple HBFs of the DHBF. Thus, it is impossible for the DHBF to know which is the right one. If the DHBF persists in removing membership information of x from it, the wrong HBF may perform the item deletion operation with given probability. The wrong item deletion operation destroys the DHBF and leads to, at most, k potential false negatives. To avoid producing false negatives, the membership information of such items is kept by the DHBF but removed from X. Therefore, ensuring that the item is uniquely identified within a single instance before performing the deletion is essential to avoid errors. When enough items are deleted and two consecutive HBFs have a total number of items less than the combined capacity of a single HBF, these two HBFs can be merged into one to optimize memory usage (See Algo. 3).

Algorithm 3 Merge()

```
1: for $d = j$ to $l$ do
2:     for $j = 1$ to $s$ do
3:         if HBF$_j$.i $< c$ then
4:             for $q = j + 1$ to $s$ do
5:                 if HBF$_j$.i + HBF$_k$.i $< c$ then
6:                     HBF$_j$ $\leftarrow$ HBF$_j$ $\cup$ HBF$_k$
7:                     HBF$_j$.i $\leftarrow$ HBF$_j$.i + HBF$_k$.i
8:                     Clear
9:                     Break
10:                end if
11:            end for
12:        end if
13:    end for
14: end for
```

Algorithm 4 Delete(x)

```
Require: $x$ is not null
1: for $d = 1$ to $l$ do
2:     $index \leftarrow null$
3:     $counter \leftarrow 0$
4:     for $i = 1$ to $s$ do
5:         if $HBF[i].Query(x)$ then
6:             $index \leftarrow i$
7:             $counter \leftarrow counter + 1$
8:             if $counter > 1$ then
9:                 break
10:            end if
11:        end if
12:    end for
13:    if $counter = 1$ then
14:        for $i = 1$ to $k$ do
15:            $CBF[index][hash_i(x)] \leftarrow CBF[index][hash_i(x)] - 1$
16:        end for
17:        $CBF[index] \leftarrow CBF[index] - 1$
18:        Merge()
19:
20:        return true
21:    else
22:
23:        return false
24:    end if
25: end for
```

5. Experiments and Results

5.1. Dataset Preparation

Four different datasets- FRGC [19], Faces94 [26], Pinellas (Pinellas County Sheriff's Office (PCSO) database) and MORPH [22] were used during the experiments. A subset of 30,000 unique samples were selected to compute the mean vector. In the remaining databases, only individuals who had at least four noisy samples in addition to one genuine sample were retained. Consequently, this process resulted in taking 152 classes from Faces 94, 302 from FRGC, and 557 from Morph. Because the four datasets varied in resolution, pose, and illumination, all images underwent a standardized preprocessing pipeline prior to being fed into the HBF framework. Overall, the images were subjected to three major steps: (1) The face detection process, during which each input image was analyzed for the presence of a face and 68 facial landmark points, using dlib's pretrained face detector [24]; (2) Aligning the faces, in which affine transformations based on the detected landmark coordinates were applied to normalize facial orientation; and (3) Masking, wherein a binary elliptical mask was imposed on each aligned image to retain only the essential frontal facial region. After that, each image was resized to 70×70 for a common dimension across all the datasets. Then, histogram equalization was applied to the resulting images to ensure a uniform distribution of intensities. Finally, binary quantization was applied to compress the features.

5.2. DHBF Initialization

In order to initialize the DHBF, it begins by defining desired levels of $\mathrm{FP_{CBF}}$ and $\mathrm{FP_{HBF}}$. Although it is suggested to have a low rate of the $\mathrm{FP_{CBF}}$, since CBFs are included, a higher rate of the $\mathrm{FP_{CBF}}$ is defined and at the same time $\mathrm{FP_{HBF}} \approx 0$, similar to what has been devised in [7]. $\mathrm{FP_{CBF}} \approx 10\%$ is set and $n = 30000$ is defined. Considering that $FP \approx 0.6185^{m/n}$ [8], for given n and a desired FP, one can compute m, e.g., in our setting $m \approx 140530$. The optimal k is further computed as $k = (m \ln 2)/n$ [7], and in our setting it was $k \approx 4$. SHA-256 hash function was used along with FNV [17] to enhance the randomness. Additionally, for the HBF, $\mathrm{FP_{HBF}}$ as $\mathrm{FP_{HBF}} \ll \mathrm{FP_{CBF}}$ was defined. Moreover, $p_{CBF} \approx 0.5$ was set to account for the worst-case scenario that is the probability of falsely setting a bit to 1 is approximately 50%. As k has already been computed for the desired $\mathrm{FP_{CBF}}$, the approximation of the total number of BFs(N) and N_t in a recursive manner (Eqn. 7). In this setting, $N \approx 165$ was obtained.

$$FP_{th} = \sum_{i=0}^{|N|} \binom{N}{i} (1 - p_{BF})^{ki} \left(1 - (1 - p_{BF})^k\right)^{(N-i)} \tag{7}$$

As the next step in the design, in order to find the number of levels in the HBF and l_i in an extreme scenario, the smallest blocks were assumed (i.e., the blocks inserted in the HBF at level 0) to contain at least one bit flipped, with the probability close to one: $1-(1-e/P)^{l_0} \approx 1$. It is worth noting here that this probability can vary from one application to another and can be adjusted according to experimental results. When the values of e and P are known, l_0 can be computed. According to our experimental results, on average $e/P \approx 16\%$. Setting $1-(1-e/P)^{l_0} \approx 0.995$, $l_0 \approx 32$ was obtained. Note that in each level of the HBF, e.g., i^{th} level $(1 \leq i \leq d)$, we concatenate w blocks from the level $i-1$, therefore, $l_i = w.\ l_{i-1} (1 \leq i \leq d)$. Knowing l_i, our goal is to find the minimum value of d, and accordingly, the maximum value of w to fulfill the condition stated by the equation $N = \sum_{i=0}^{d} [P/l_i]$. The goal was to minimize d, and simultaneously, maximize w to improve space utilization. Solving the above equation in an iterative fashion, $w \approx 13$ and $d = 2$ were obtained.

5.3. Threshold (N_t) Determination

The threshold determination process requires parameterization of the binomial distribution for each level of the HBF system. Specifically, the parameters N, t, and α must be defined (Eqn. 6). For this study, the number of Bloom filters at each level is given by $N = [1, 11, 153]$, corresponding to the three levels of the HBF hierarchy. The probability of success, t, represents the false positive rate of a Bloom filter under noisy conditions and is empirically determined to be $FP_{CBF} = t = 0.10$. The confidence level α is set to 0.95 , ensuring that the system achieves at least 95% confidence in authenticating a genuine query.

For each level, the minimum number of hits, N_t, is computed using the PPF of the binomial distribution. This computation yields the following thresholds: $N_{t1} = 1, N_{t2} = 3$, and $N_{t3} = 21$. These thresholds represent the minimum number of BF matches required at levels (d) 1, 2, and 3, respectively, to achieve the desired confidence level. At level, $d = 1$, which contains only a single BF ($N_1 = 1$), a single hit ($N_{t1} = 1$) suffices to achieve the required confidence level. In contrast, at level, $d = 2$, with 11 BFs, requires at least three hits ($N_{t2} = 3$), reflecting the increased granularity of encoding at this level. Similarly, at $d = 3$, with 153 BFs, requires a minimum of 21 hits ($N_{t3} = 21$). The confidence level of $\alpha = 0.95$ ensures that the system maintains a high true positive rate (TPR) while minimizing the false positive rate (FPR). The parameters apply the same to every HBF in the DHBF.

5.4. Enrollment & Query

Enrollment in the DHBF is similar to the enrollment process in a single HBF. Except, instead of flipping bits from 0 to 1 , the bits are incremented by 1 of the CBF. The querying process in the DHBF is initiated by checking each instance of the DHBF sequentially until the item is found. If the item is located in any HBF instance, the process stops and returns a positive result. If not, the query continues through all instances. For the enrollment experiment, we enrolled 30000 biometric data all at once inside a single HBF of the DHBF and queried them with themselves to make sure the membership tests are consistent. The results are shown in Table 3 as it refers to a 100% authentication rate.

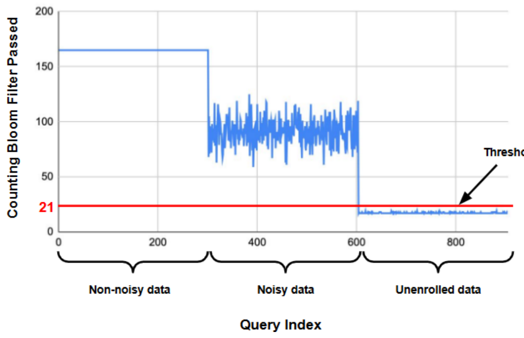

Figure 4. 100% Accuracy as Noisy Data Passes Threshold.

Table 3. Query After Enrollment with an Accuracy of 100%.

	Positive	Negative
Positive	30000	-
Negative	-	0

5.5. Insertion & Query

To demonstrate the dynamic nature of DHBF, insertion experiment was conducted in which 10000 faces were enrolled in 3 different instances of HBFs in the DHBF. In other words, the capacity (c) of each instance of HBF was kept to 10,000. As mentioned earlier, when the number of enrolled items surpasses this threshold, the automatically adds another instance of the HBF. Since HBF is the base structure of the DHBF, parameters will be set according to the capacity that is defined by the user. Therefore, as mentioned in Section 5.2, all the parameters will be re-calculated according to the capacity, $c = 10000$. Thus, to accommodate 10000 items in one single instance of HBF, the size and the number of hash functions would be updated as $m \approx 47926$ and $k \approx 3$. When a new face enters beyond the capacity (c), a new instance of HBF will be added to DHBF.

Table 4. Query After Insertion with an Accuracy of 100%.

	True Positive	False Positive	True Negatives
Enrolled Data	302	0	-
Noisy Data	302	0	-
Unenrolled Data	-	-	302

Once the DHBF architecture is ready and gallery data is inserted, another query experiment with 906 templates was performed from Faces 94, FRGC, and Morph. The query results are shown in Fig. 4. These 906 query templates can be categorized into 3 different types: (1) Already Enrolled in DHBF : The first 302 query data are already enlisted in

DHBF which means DHBF already has seen them. Thus, the DHBF approved the membership of these individuals; (2) Noisy template of already enrolled person in DHBF: The next 302 samples are also known to DHBF, but DHBF has not seen the exact same query template. In other words, the query is based on the same individuals' noisy templates (e.g., different expressions or lighting). Ideally, the DHBF approved the membership of these individuals because they are noisy versions of the members of the database and it gets authenticated by the least number of N_t individual CBFs on each level; (3) Never Enrolled in DHBF: The last 302 templates are from those individuals who were never enrolled in the DHBF, so-called imposters, and should be rejected. The second set of query set will claim to be a member of the database. Insertion results are shown in Tab. 4.

5.6. Deletion

The deletion operation intends to clear the database. For this experiment, all the data were deleted one by one and eventually clearing all the 30000 data. In order to make sure that all the data was cleared from the DHBF, the same query experiment was done as mentioned in 5.4. The query returned an empty value.

6. Scalability Analysis

In this section, we analyze the operational complexities across various Bloom filter-based data structures and compare them with our proposed method (See Tab. 5).

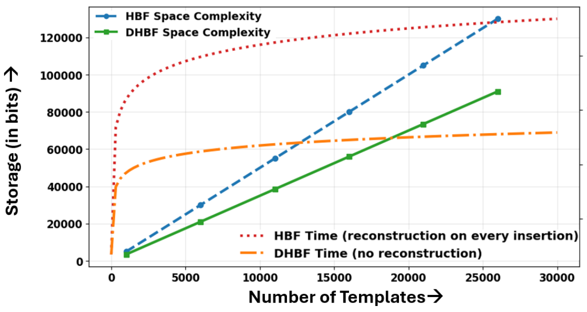

Figure 5. Comparison of Storage and Insertion Time Comparison Between DHBF and HBF.

For the HBF [25], there are N legacy BFs across all the levels of an HBF. To calculate the total number of BFs, we simply add the BFs of each level and each of the BF access will require k hash operations. Therefore, the membership query complexity of HBF is $(k \times N)$. For the DHBF, if the capacity of each HBF is defined as c, the number of HBF instances that will be required to store n templates will be $s = n/c$. Thus, the total number of CBF in a DHBF will be $N \times s$. Now, in order to access all the CBFs inside the DHBF, k number of hash operations will be required. Thus, we can conclude that the membership query time complexity of a DHBF is $O(k \times N \times s)$.

Table 5. Operational Complexities of BF-based Frameworks.

Operation/ Algorithm	BF	DBF	HBF	Proposed DHBF
Enrollment/ Insertion	$O(k)$	$O(k)$	$O(k)$	$O(k)$
Membership Query	$O(k)$	$O(k \times s)$	$O(k \times N)$	$O(k \times N \times s)$
Deletion	-	$O(k \times s)$	-	$O(k \times N \times s)$

For the deletion operations in DHBF, if we need to delete all the elements inside the DHBF structure, we will have to access all the CBFs. Deletion complexity will be similar to the membership query complexity of the DHBF, which is $O(k \times N \times s)$ since a query operation will take place initially to check whether the CBF is empty or not.

Table 6. Complexity of BF-based Frameworks vs Proposed DHBF.

	BF [4]	DBF [12]	HBF [25]	Proposed DHBF
Time	$O(k)$	$O(k \times s)$	$N \times O(k)$	$N \times O(k \times s)$
Space	$O(m)$	$O(m \times s)$	$N \times O(m)$	$N \times O(m \times s)$

An experimental analysis is shown in Fig. 5 where the DHBF exhibits a notable space & time efficiency. While it incurs DHBF requires additional (Tab. 6) time for membership queries compared to the HBF, this overhead stems from its need to traverse multiple instances of DHBF to ensure the queried item is not present. However, there are no reconstruction time cost required for the DHBF while doing insertions (Fig. 5a). Additionally, there are significant storage savings ($\approx 30\%$) (See Fig. 5b) as the merge operation dynamically eliminates unused space.

Time complexity in DHBF: In order to access every CBF cell in the DHBF, we need k operations to be conducted and the total number of CBFs inside the DHBF is $N \times s$. Thus, the time complexity of DHBF is $O(k \times N \times s)$.

Space complexity in DHBF: If each of the CBF sizes has been defined with m and the total number of CBFs inside the DHBF is $N \times s$. The space complexity of the DHBF is $O(m \times N \times s)$.

7. Security & Future Work

While inherent security mechanism (use of SHA-256 hashing) have been integrated, a comprehensive security analysis against targeted adversarial models remains an open research question and will be addressed extensively along with more dataset samples in our future work.

8. Conclusion

This study introduces the Dynamic Hierarchical Bloom Filter (DHBF), a novel framework designed to enhance the scalability and dynamic capabilities of biometric authentication systems. We thoroughly reviewed the related works, providing a detailed discussion on the evolution. Our exploration into the methodology employed by the DHBF highlighted its unique approach to managing dynamic data sets, using adaptations of both hierarchical and dynamic features to address the limitations found in previous studies. DHBF also demonstrated high authentication accuracy.

References

[1] P. S. Almeida, C. Baquero, N. Preguiça, and D. Hutchison. Scalable bloom filters. *Information Processing Letters*, 101(6):255–261, 2007.

[2] S. Arora, M. Bhatia, H. Kukreja, and S. Jain. Privacy protection of biometric templates using deep learning. In *Innovations in Cyber Physical Systems: Select Proceedings of ICICPS 2020*, pages 19–27. Springer, 2021.

[3] D. Bigo, S. Carrera, B. Hayes, N. Hernanz, and J. Jeandesboz. Justice and home affairs databases and a smart borders system at eu external borders: An evaluation of current and forthcoming proposals. *CEPS Papers in Liberty and Security in Europe*, 52, 2012.

[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[5] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese. An improved construction for counting bloom filters. In *European Symposium on algorithms*, pages 684–695. Springer, 2006.

[6] J. Bringer, C. Morel, and C. Rathgeb. Security analysis and improvement of some biometric protected templates based on bloom filters. *Image and Vision Computing*, 58:239–253, 2017.

[7] U. Customs, B. Protection, J. R. Cantor, and A. C. P. Officer. Automated targeting system. *Contact Point*, 2017.

[8] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *ACM SIGCOMM computer communication review*, 29(4):251–262, 1999.

[9] M. Ghayoumi. A review of multimodal biometric systems: Fusion methods and their applications. In *2015 IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS)*, pages 131–136. IEEE, 2015.

[10] M. Gomez-Barrero, C. Rathgeb, G. Li, R. Ramachandra, J. Galbally, and C. Busch. Multi-biometric template protection based on bloom filters. *Information Fusion*, 42:37–50, 2018.

[11] S. Guennouni, A. Mansouri, and A. Ahaitouf. Biometric systems and their applications. *Visual impairment and blindness-what we know and what we have to know*, 2020.

[12] D. Guo, J. Wu, H. Chen, and X. Luo. Theory and network applications of dynamic bloom filters. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–12. Citeseer, 2006.

[13] F. Hao, M. Kodialam, and T. Lakshman. Incremental bloom filters. In *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*, pages 1067–1075. IEEE, 2008.

[14] E. J. Kindt. Privacy and data protection issues of biometric applications. In *A Comparative Legal Analysis*, volume 12. Springer, 2013.

[15] R. Kumar Pandey, Y. Zhou, B. Urala Kota, and V. Govindaraju. Deep secure encoding for face template protection. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 9–15, 2016.

[16] K. Lai, O. Kanich, M. Dvořák, M. Drahanskỳ, S. Yanushkevich, and V. Shmerko. Biometric-enabled watchlists technology. *Iet Biometrics*, 7(2):163–172, 2018.

[17] L. C. Noll. Fnv hash. `http://www.isthe.com/chongo/tech/comp/fnv/`, 1994. (Accessed on 23/02/2023).

[18] R. K. Pandey, Y. Zhou, B. U. Kota, and V. Govindaraju. Learning representations for cryptographic hash based face template protection. *Deep learning for biometrics*, pages 259–285, 2017.

[19] P. J. Phillips, P. J. Flynn, T. Scruggs, K. W. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min, and W. Worek. Overview of the face recognition grand challenge. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 947–954. IEEE, 2005.

[20] B. Rahul, K. Kuppusamy, and A. Senthilrajan. Chaos-based audio encryption algorithm using biometric image and sha-256 hash algorithm. *Multimedia Tools and Applications*, 82(28):43729–43758, 2023.

[21] T. Ramu and T. Arivoli. Biometric template security: an overview. In *Proceedings of International Conference on Electronics*, volume 65, 2012.

[22] K. Ricanek and T. Tesafaye. Morph: A longitudinal image database of normal adult age-progression. In *7th international conference on automatic face and gesture recognition (FGR06)*, pages 341–345. IEEE, 2006.

[23] S. Sawhney, K. Kacker, S. Jain, S. N. Singh, and R. Garg. Real-time smart attendance system using face recognition techniques. In *2019 9th international conference on cloud computing, data science & engineering (Confluence)*, pages 522–525. IEEE, 2019.

[24] S. Sharma, K. Shanmugasundaram, and S. K. Ramasamy. Farec—cnn based efficient face recognition technique using dlib. In *2016 international conference on advanced communication control and computing technologies (ICACCCT)*, pages 192–195. IEEE, 2016.

[25] S. Shomaji, F. Ganji, D. Woodard, and D. Forte. Hierarchical bloom filter framework for security, space-efficiency, and rapid query handling in biometric systems. In *2019 IEEE 10th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, pages 1–8. IEEE, 2019.

[26] L. Spacek. Collection of facial images: Faces94. *Computer Vision Science and Research Projects, University of Essex, United Kingdom, http://cswww. essex. ac. uk/mv/allfaces/faces94. html*, 2007.

[27] S. N. Yanushkevich, K. W. Sundberg, N. W. Twyman, R. M. Guest, and V. P. Shmerko. Cognitive checkpoint: Emerging technologies for biometric-enabled watchlist screening. *Computers & Security*, 85:372–385, 2019.