

## REPORT

### BIOMETRIC TEMPLATE BASE ON BLOOM FILTER

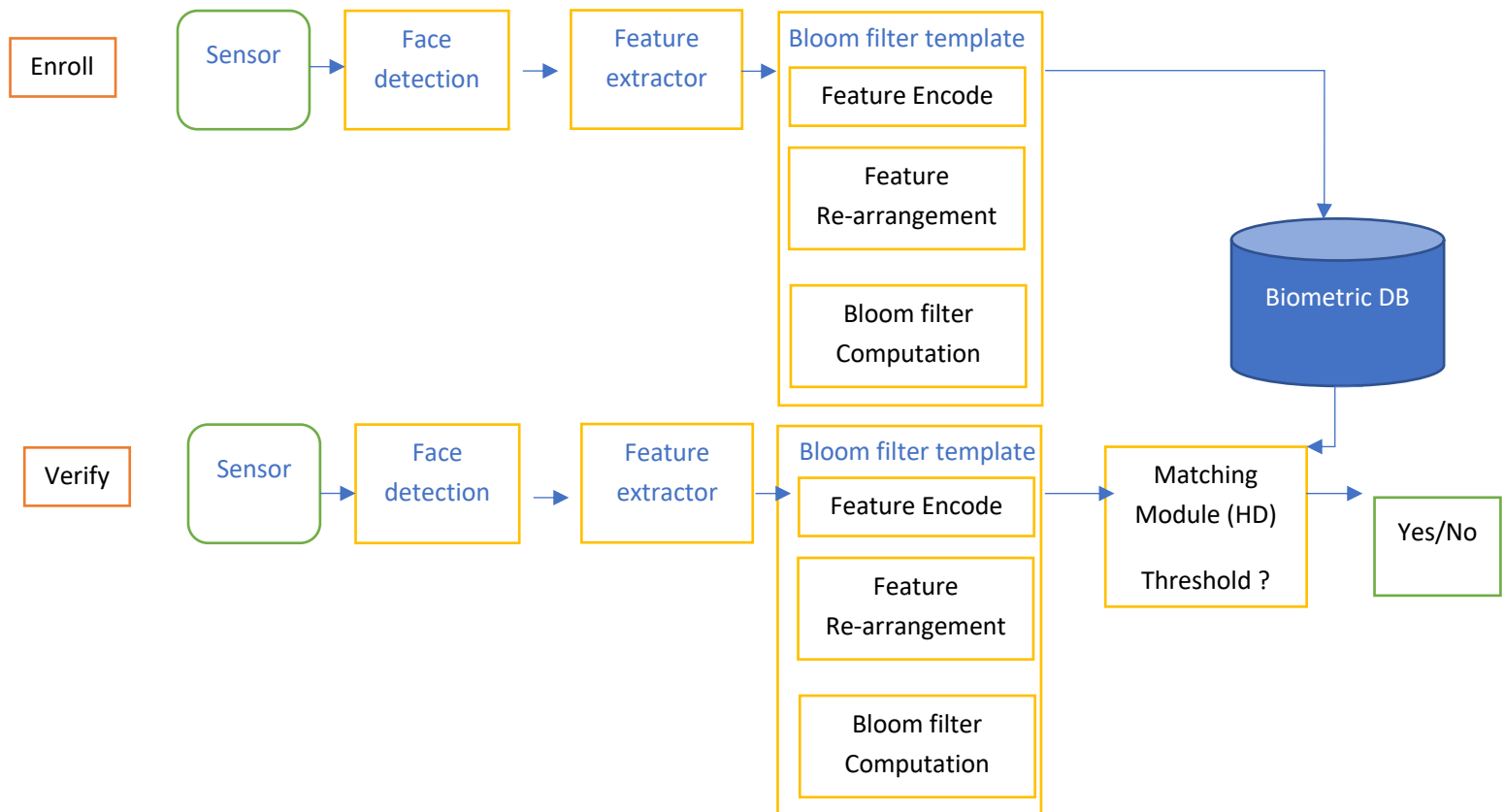
#### I. INTRUCTIONS

Traditional biometric authentication systems store biometric templates together with the data identifying an individual in a database for later comparison. In order to authenticate an individual the biometric data presented is looked up in the database. If a record is found with biometric data that is sufficiently close to that presented, the person is identified. However, the storage of biometric data leads to considerable risks for the authentication system and raises serious concerns regarding data protection. Privacy sensitive personal data that is potentially threatened by internal or external attacks on the database.

Biometric templates should hence be protected and impersonation with stolen templates must be prevented, while preserving system's performance.

#### II. PROPOSED SYSTEM ARCHITECTUTE

In that context, Biometric template protection (BTP) technologies offer solutions to privacy preserving biometric authentication. We propose a general BTP scheme applied on face templates using Bloom filter based protected templates.



System modules:

- ✓ Face detection (Enroll & verification).
- ✓ Feature extraction
- ✓ Generate protected biometric templates (bloom filter template)
- ✓ Matching

### III. REVIEW WORKS

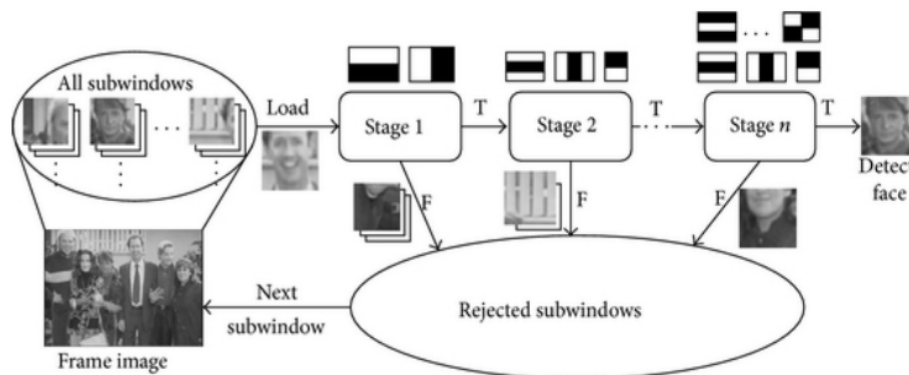
#### 1. Face detection

Face detection is a computer technology that determines the locations and sizes of human faces in arbitrary(digital) image. The face detection framework, introduced by Viola and Jones in 2001 in the field of real-time object detection, this framework is based on Haar Cascades. We used Haar-like filters in order to face processing because of its real-time capability, high accuracy, and availability as open-source software under the [Open Computer Vision Library\(OpenCV\)](#). In this work we have tested cascade for the frontal face detection (haarcascade\_frontalface\_alt.xml)

##### ➤ Haar Features

- ✓ Each feature is a single value obtained by subtracting
- ✓ Sum of pixels under the white rectangle from sum of pixels under the black rectangle
- ✓ Each feature is related to a special location in the sub-window
- ✓ All faces share some common properties: The eyes region is darker than the upper cheeks , The nose bridge region is brighter than the eyes

##### ➤ AdaBoost Feature Selection



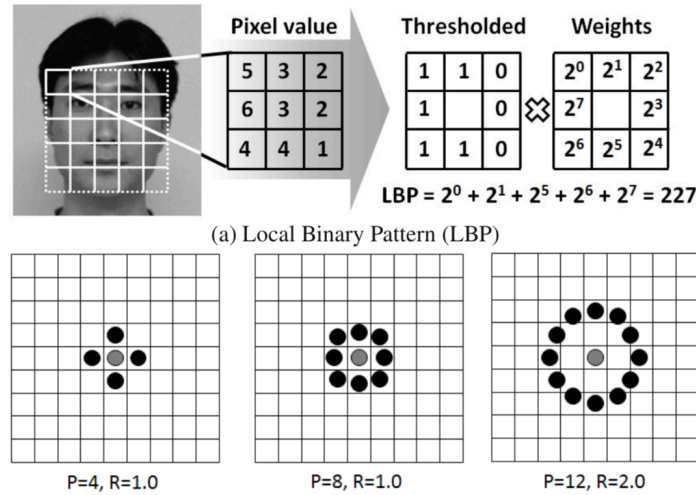
#### 2. Feature extraction

Local Binary Pattern (LBP) used as a local texture descriptor in general object recognition as well as in face recognition.

The basic LBP operator compares the 8 neighboring pixel intensity values to the intensity value of the central pixel in the region and represents the result as a 8-bit

binary string. After LBP encoding of each pixel, the face image is divided into several smaller windows and the histogram of local binary patterns in each window is computed. The number of bins in the histogram is 8 and  $2^p$  for the basic LBP. A global feature vector is then generated by concatenating histograms of all the individual windows and normalizing the final vector.

Finally, two face images can be matched by computing the similarity (or distance) between their feature vectors.



The face verification system that served as baseline for the proposed approach is an implementation of the LGBPHS algorithm [1]. In order to extract features from a captured biometric sample (a set of face images enrolled, verify)

LGBPHS: face image is preprocessed and Gabor-based features are extracted.

### 3. Generate protected biometric templates

In order to generate protected biometric templates, the method proposed in will be used. In the present scheme, this sub-system can receive two different inputs:

(i): a biometrics sample at authentication time

(ii): a binary feature vector at database generation step 3 (figure 1), which can comprise real biometric information of  $i$ th subject ( $T_i^1$ ), or random binary vector ( $T_i^j$ , with  $j: 2, \dots, k+1$ ), instead of a biometric image. In both case, the scheme comprises three case steps

#### Step 1:

Feature extraction and encoding: in the first step, if the input is a biometric sample, a two-dimensional binary feature vector is extracted from it. Then, resulting vector (or input random binary vector for the database generation step,  $T_i^j$ , is divided into nBlocks blocks of size nBits  $\times$  nWords bits.

## Step 2:

Structure-preserving feature re-arrangement: in order to achieve unlinkable templates, we need to dissipate the information of the feature vectors among different blocks, while preserving verification performance. To that end, we first re-group  $n\text{Blocks}$  blocks into  $G$  groups consisting of  $B$

Blocks ( $n\text{Blocks} = G \times B$ ), and then re-arrange the features in a structure-preserving manner into consecutive sub-steps:

Row-wise permutation (perm): for each of the  $G$  sets, the rows of the vertical concatenation of all  $B$  blocks are permuted. This way, information is diffused between blocks and block-based attacks prevented.

Word-wise shift (shift): in this second sub-step, circular vertical shifts are applied to words independently for each block. Since vertical shifts are not performed equally for each word in a block, different words may result from identical ones and vice versa, further concealing the number of different words of each original block.

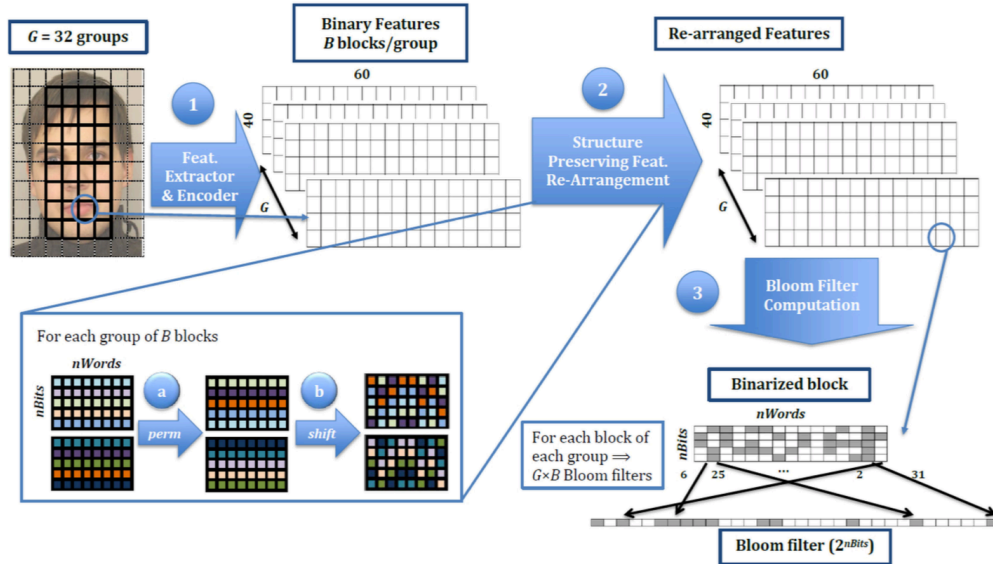


Figure 1

## Step 3: Bloom filter computation:

One Bloom filter is computed from each of the  $n\text{Blocks}$  blocks, such that the final protected template PI consists of  $n\text{Blocks}$  Bloom filters of size  $2^{n\text{Bits}}$

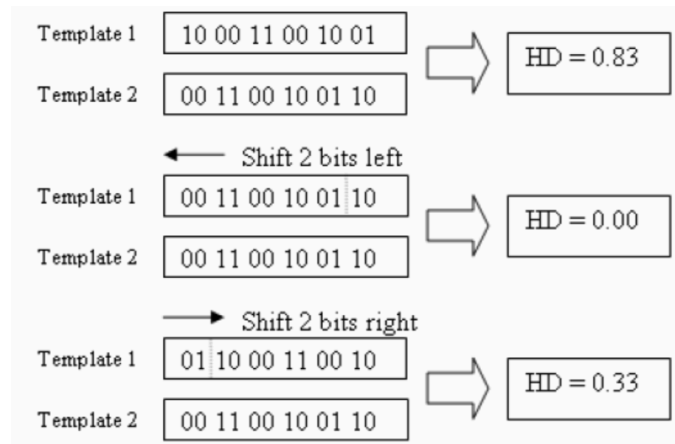
The distance between two bloom-filter templates, PI and PI\*, is defined as average distance of pairwise bloom filters  $b_i$  and  $b_j$  with  $i = 1, \dots, n\text{Blocks}$ . In order to compute such pairwise distances, since Bloom filters comprise a variable number of ones, their dissimilarity can be efficiently computed as **Hamming distance (HD)**,

Let  $|b|$  denote the amount of bits within a Bloom filter  $b$  set to 1. Then the dissimilarity score  $DS$  between two Bloom filters  $b_i$  and  $b_j$  is defined as

$$DS(b_i, b_j) = \frac{HD(b_i, b_j)}{|b_i| + |b_j|}$$

Therefore their dissimilarity  $DS(PI, PI^*)$  is estimated as

$$DS(PI, PI^*) = \frac{1}{nBlocks} \sum_{i=1}^{nBlocks} \frac{b_i \text{ XOR } b_j}{|b_i| + |b_j|}$$



#### IV. IMPLEMENTATION

- Install Pycharm for Programing: Python
- Library: Open CV
- Install the Bob packages that you need in that environment:
- We offer pre-compiled binary installations of Bob using [conda](#)

##### 1. Face detection.

Code module face detection.

##### Enroll phase:

Using file detection-face.py -> run -> detect & capture face images for enroll -> store in to path: `db/enrollment/' + username/images` (directory where the face templates - origin will be stored)

*Command: python detection-face.py \$1 enroll (ex: python detection-face.py long enroll)*

##### Verify phase:

Using file detection-face.py -> run -> detect & capture face images for enroll -> store in to path: `db/ verification/' + username/images` (directory where the face templates - origin will be stored)

*Command: python detection-face.py \$1 verify (ex: python detection-face.py tinh verify)*

##### 2. Feature extraction

Input: Folders containing the `db/enrollment/' + username/images/` or `db/ verification/' + username/images`

Output: Extracts unprotected template from image file for each user

- Extracting LGBPHS features for user when enroll (unprotect)

Command: `python feature_extractor.py $user enroll`

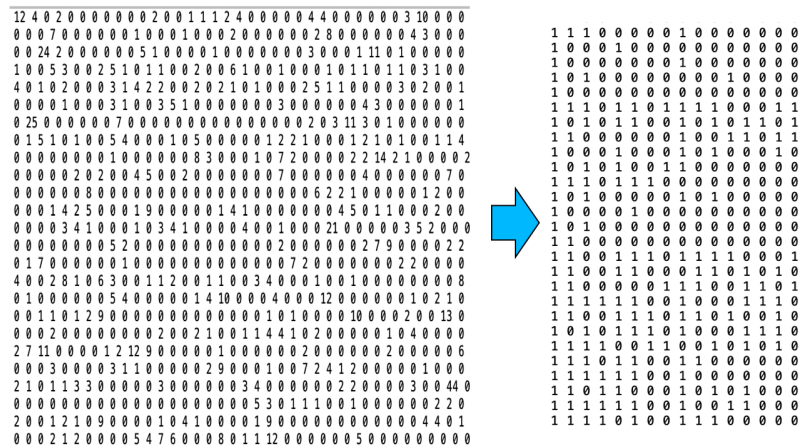
*Db/enrollment/\$username/features/ori/ : directory where the unprotected face templates will be stored when enroll*

*Db/verification/\$username/\$user\_ori.txt : directory where the unprotected face templates will be store after verify*

- Extracting LGBPHS features for user when verify (unprotect)

Command: `python feature_extractor.py $user verify`

(`feature_extractor.extract_verify(username)`)



### 3. Generate protected biometric templates (bloom filter template)

Input: Folders containing the unprotected face templates

Output: Bloom filter protected template

- Extracting LGBPHS features and BF templates for user when enroll (protected BF)

Command: `python feature_extractor.py $user enroll`

*Db/enrollment/\$username/features/bf/ : directory where the protected face templates will be stored when enroll*

- Extracting LGBPHS features and BF templates for user after verify (protected BF)

Command: `python feature_extractor.py $user verify`

*Db/verification/\$username/\$user\_bf.txt/ : directory where the protected face templates will be stored when verify*

### 4. Matching:

Computes the normalised Hamming distance between two Bloom filter templates

```
def getBFDistance(X,Y):
```

```
    return hamming_distance(X,Y)
```

in order to call function hamming\_distance -> using file compute.py in directory root

Input: folders with the unprotected and protected templates

Output: mated and non-mated scores, stored in text files with one score per row  
(Figure 1.3)

## V. EVALUTION

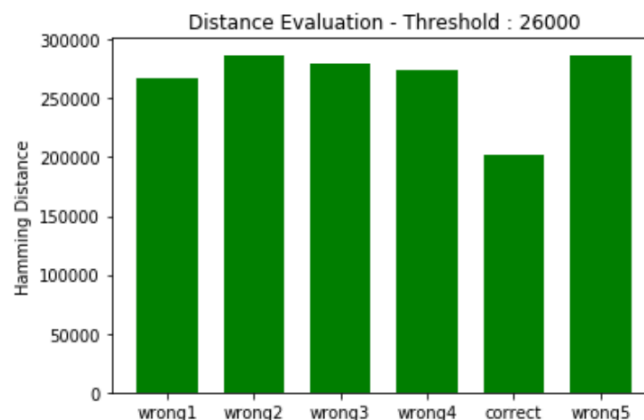
False Match Rate is equal with non - bloom filter. See Figure 1.5

+ Using non-bloom filter, with threshold 260000, using square error, the chart illustrates at 1.1 and 1.2

+ With bloom filter, with threshold 0.4, using hamming distance, the chart illustrates at 1.3 and 1.4

Distances among correct patterns	Distance between correct & wrong patterns
('The original distance is: ', 242521)	('The original distance is: ', 275136)
('The original distance is: ', 159136)	('The original distance is: ', 292921)
('The original distance is: ', 212613)	('The original distance is: ', 274161)
('The original distance is: ', 149234)	('The original distance is: ', 271475)
('The original distance is: ', 227260)	('The original distance is: ', 276398)
('The original distance is: ', 140073)	('The original distance is: ', 273667)
('The original distance is: ', 172852)	('The original distance is: ', 272370)
('The original distance is: ', 151466)	('The original distance is: ', 274012)
('The original distance is: ', 195335)	('The original distance is: ', 262309)
('The original distance is: ', 0)	('The original distance is: ', 273494)

**Figure 1.1: Distance among patterns with non-bloom filter**



**Figure 1.2: non-bloom filter chart between correct & wrong patterns**



Distances among correct patterns	Distance between correct & wrong patterns
('The BF distance is: ', 0.38970080631916143)	('The BF distance is: ', 0.41220438749040605)
('The BF distance is: ', 0.3143623164185072)	('The BF distance is: ', 0.4003123146411181)
('The BF distance is: ', 0.3667957077077448)	('The BF distance is: ', 0.40505608421705663)
('The BF distance is: ', 0.30082767230385987)	('The BF distance is: ', 0.3961764924574617)
('The BF distance is: ', 0.37465418196622624)	('The BF distance is: ', 0.395613929462832)
('The BF distance is: ', 0.2943413108823498)	('The BF distance is: ', 0.3902126120932668)
('The BF distance is: ', 0.32183990340861895)	('The BF distance is: ', 0.40097733626266124)
('The BF distance is: ', 0.30430960413627955)	('The BF distance is: ', 0.4021919473686512)
('The BF distance is: ', 0.34606146859772946)	('The BF distance is: ', 0.4025057229846155)
('The BF distance is: ', 0.0)	('The BF distance is: ', 0.3984244240702267)

Figure 1.3: Distance between patterns with bloom filter

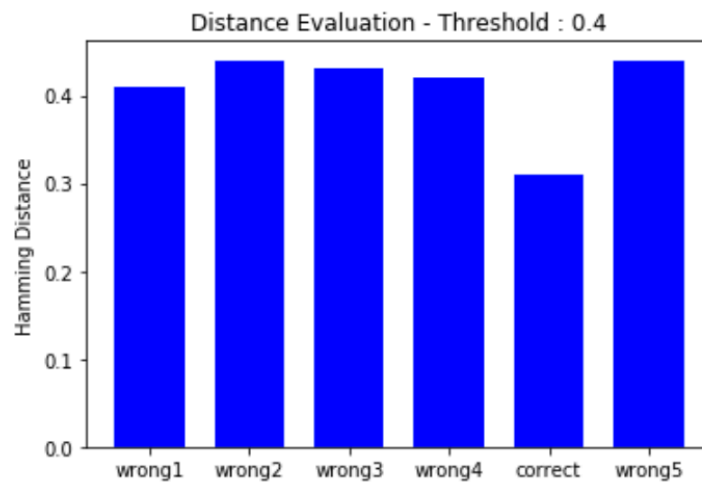


Figure 1.4: non-bloom filter chart between correct & wrong patterns

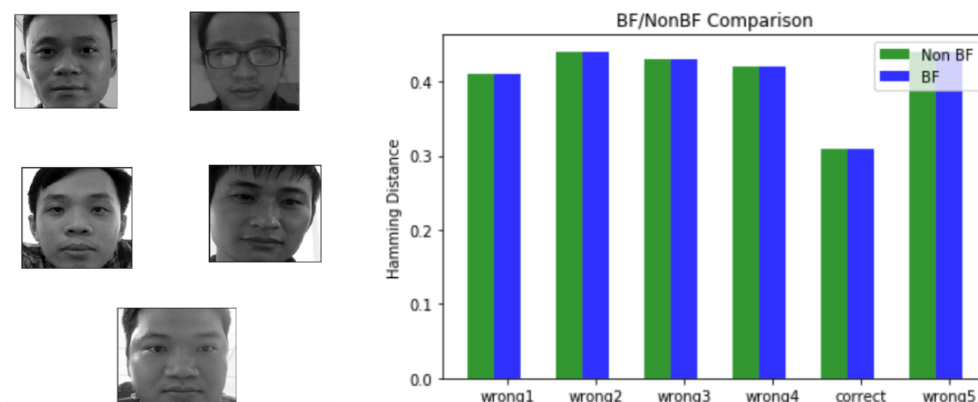


Figure 1.5: non-bloom/ bloom filter comparison with Scaling Threshold: 1/650,000

## VI. CONLUTIONS

Bloom filters is proposed, in order to grant privacy protection to the enrolled subject and detect the use of stolen templates.

The performance evaluation showed that verification accuracy was preserved with respect to the original unprotected system.