Biometric Matching by Prehashing and Applications

Lea Grieder (328216)
Leila Sidjanski (330810)

Computer Science / Communication Systems
EPFL

March 2024

Responsible
Prof. Serge Vaudenay

Contents

1 Introduction

1.1 Background

Authentication is the process of confirming the validity of a claimed identity seeking access to a system or resource. Over decades, authentication mechanisms have evolved from basic password systems in the 1960s to advanced methods such as multifactor authentication by the late 2010s, driven by a persistent commitment to combat evolving security risks while enhancing user convenience[1]. Various methods such as password-based authentication, certificate-based authentication, one-time passwords, multifactor authentication, and biometric authentication are employed[3].

Biometric authentication, which involves analyzing unique physical characteristics, is often considered more secure than traditional authentication methods due to the difficulty in duplicating biometric traits. This encompasses technologies such as facial recognition, fingerprint recognition, eye recognition, and voice recognition[2]. However, despite the enhanced security of biometric authentication, it is not immune to exploitation. For instance, fingerprints left on surfaces can be copied, or hackers may obtain images of individuals online to deceive authentication systems. Choosing the right authentication mechanism requires careful consideration of various factors including the necessary security level, ease of use for users, cost implications for setup and ongoing upkeep, as well as the unique risks and vulnerabilities pertinent to the system or data in question. Typically, the requisite level of security steers the selection process; for example, platforms managing sensitive personal information might mandate the use of robust authentication methods, such as biometric verification. The inherent challenge in deploying such secure systems lies in achieving a delicate equilibrium between high security measures and user convenience. The goal is to create an authentication process that is both seamless and efficient, ensuring that access is granted swiftly and accurately to the rightful user without necessitating multiple attempts, thus maintaining a user-friendly experience while upholding the highest security standards.

While advanced biometric systems typically rely on externally visible physical attributes, finger-vein authentication focuses on internal anatomical features, adding a unique layer of security, as they are less prone to replication or theft compared to external characteristics. Nevertheless, it is important to note that finger-vein authentication does not completely eliminate challenges. Despite its emphasis on internal features, attackers can exploit inherent structures in finger veins, such as common patterns among individuals and predictability in acquired data, which poses risks to the authentication process.

In light of these considerations, this project is dedicated to authentication using finger-vein features. This involves utilizing a specialized scanner equipped with two infrared cameras to capture finger veins from different angles. The registration process involves capturing an image, termed the model image, while the authentication process involves capturing another image, known as the probe image. These images undergo processing through a pipeline designed to extract and align the finger-vein

patterns. The pipeline outputs a feature vector, which is essentially a bitstring, where 0's represent where there are no finger veins, and 1's show where veins are present. Following this process, the system evaluates whether the feature vector extracted from the probe image sufficiently matches the feature vector of the model image associated with the individual attempting authentication.

1.2 Extraction Pipeline

This project extends the work on optimizing a finger-vein recognition pipeline that has demonstrated the lowest Equal Error Rate (EER) by incorporating a novel hashing step to process the output of the pipeline. The purpose of integrating Hash Functions within this context is multifold, but before delving into hash functions, which are central to our project, it's essential to outline the foundation upon which we have built our advancements. This initial context will provide a clearer understanding of the starting point from which our developments began.

Simon Sommerhalder and Burcu Yildiz have both made significant contributions to the system. Simon introduced an innovative approach to the alignment of freshly captured images (probe images) with those stored in the system (model images), ensuring the hashing process (following the alignment of the finger) is based on the unique finger-vein pattern rather than how the finger is positioned on the scanner. This project aimed to find an alignment technique that could work effectively without accessing both the model and probe simultaneously, a significant challenge given the variability in how users may place their finger on the scanner.

Simon has developped a pipeline (see Figure 1.1) to align finger-vein images independently, enhancing security by eliminating the need to compare the model and probe images side by side. He organized the pipeline into six clear steps:

1. **Masking**: The first step of the pipeline isolates the finger area in the image. This involves creating a mask that outlines the finger, ensuring that subsequent processing focuses solely on the relevant part of the image.
2. **Prealignment**: This step involves adjusting the position and orientation of the finger within the image before extracting vein patterns. It's aimed at roughly aligning the image based on the finger's outline, helping to standardize the position of the finger across different scans.
3. **Histogram equalization**: To ensure the images have consistent lighting and contrast, this step adjusts the brightness levels. This makes the vein patterns more distinct and comparable across different images.
4. **Feature extraction**: Here, the actual vein patterns are identified and extracted from the image. The process converts the visual image into a digital format that represents the presence or absence of veins at specific locations.
5. **Postalignment**: After extracting the vein patterns, this step fine-tunes the alignment of the image. It's based on the vein patterns themselves, ensuring that the comparison between model and probe images is as accurate as possible.

6. **Distance Caclulation**: The final step involves comparing the feature vector of the probe image with that of the model image. This is done using a specific metric to quantify the similarity between the two, ultimately determining if they match closely enough for authentication to succeed.

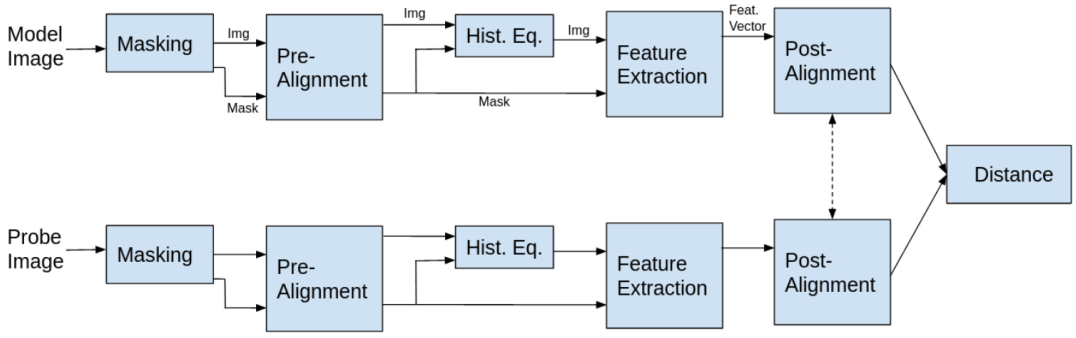

Figure 1.1: Simon's Extraction Pipeline. Compares a single probe image to a model image

Burcu's work on the finger vein authentication project built upon Simon's foundational pipeline, focusing on refining image processing for vein extraction and evaluating different distance calculation methods for authentication. She optimized preprocessing steps, investigated masking and prealignment issues, and tackled reference selection to enhance matching accuracy. A significant part of her contributions involved exploring fuzzy extractors Fuzzy Extractors to bolster security, conducting a thorough analysis of the dataset to identify and address challenges, and proposing solutions to improve the system's reliability and efficiency.

Simon and Burcu explored a variety of function combinations at different stages of the authentication pipeline, aiming to pinpoint the configuration that yields the most favorable outcomes. They utilized the Equal Error Rate (EER) as a benchmark to measure the pipeline's efficacy, focusing on achieving a balance between security and accessibility. This metric, representing the point where the rate of false acceptances(impostor incorrectly granted access) matches the rate of false rejections(legitimate user incorrectly denied), serves as an indicator of the system's reliability and accuracy. The configuration that demonstrated superior performance, leading to the lowest EER and thereby optimizing the process of analyzing and processing images, is showcased in the subsequent figure.

Table 1: EER values for the best pipeline: Edge masking, translation pre-alignment, no pre-processing, no post- processing, Miura matching as the post-alignment

Camera	EER
Camera 1	0.041
Camera 2	0.029
Sum of cameras	0.030

1.3 Project Overview

In the progression of our project, building upon the work of Simon and Burcu, we aim to address the inherent variability in biometric data, particularly with finger vein patterns, by considering hash functions. Our objective is to add a hashing step at the end of our established pipeline, prior to the post-alignment phase. The integration of this step serves to enhance security and overall functionality by allowing us to store a hash of each biometric image X in our database, instead of the raw biometric data. Upon receiving a new image Y, we compute hashes for all potential translations of Y, comparing these with the hash of X. Unfortunately, this process is computationally expensive because it requires computing the hash for every possible translation of the image. The purpose of employing a hashing process in this system is multifold:

- **Security**: Hash values can be stored instead of raw biometric data. In the event of a database breach, attackers would find it significantly more challenging to reconstruct the original biometric information from the hashed values due to the one-way nature of hash functions.
- **Consistency**: By focusing on the unique patterns of the biometric trait (like finger-vein patterns) and standardizing how this data is processed and hashed, the system aims to produce consistent hash values for the same individual across different scanning sessions. This is crucial for reliable authentication, ensuring that minor variations in finger placement do not affect the system's ability to recognize the user.
- **Performance**: Hashing biometric data into a compact, fixed-size format facilitates quicker comparison and verification processes. It's more efficient to compare hash values than to perform complex pattern recognition operations on raw biometric images.

Traditional hash functions, while pivotal in various data security contexts, generate a unique output for each unique input. This one-to-one mapping means even minor variations in the input — common in biometric data due to natural changes in biological traits or differences in scanning conditions — result in completely different hashes. This sensitivity to input variability poses a challenge in biometric authentication systems, where the goal is to accurately recognize and authenticate an individual despite these natural variations.

Fuzzy hashing stands as a sophisticated solution to this challenge. Unlike traditional hash functions, fuzzy hashing is designed to produce consistent cryptographic keys for inputs that are similar, but not identical. This is particularly advantageous in biometric authentication systems, where it's essential to recognize the same biometric trait across different instances, despite slight variations. The "fuzziness" of this approach allows the system to map these similar inputs to the same or closely related hash values, thereby ensuring that legitimate users are not incorrectly denied access due to minor discrepancies in their biometric data. Furthermore, the application of fuzzy hashing in our pipeline is instrumental in protecting user privacy. Since the hashed values, rather than raw biometric data, are stored and used for authentication, users' biometric information is safeguarded against potential breaches. Even if

hashed values were accessed without authorization, the complexity of fuzzy hashing algorithms makes it extremely challenging to reverse-engineer the original biometric data.

The process of our Fuzzy hashing algorithm, that we will detail in Section 3 begins with a biometric capture, a finger image, which goes through the already developped pipeline 1.1 to extract a bitstring. This bitstring undergoes a pre-hashing process, where a subset of significant bits (denoted as vein pixels) is selected based on a permutation keyed by a secure key, resulting in a tuple that significantly reduces the data's dimensionality while preserving its distinguishing features.

Upon generating the fuzzy hash, the next step involves further compressing this hash to prepare it for storage. This compression is achieved through a function, postHash, which maps the tuple to a bitstring of a defined length. The output, essentially a compressed fuzzy hash, exhibits nearly uniform distribution when both the input data and the key are random, enhancing security and storage efficiency.

To further bolster the security of the stored biometric data, our implementation incorporates Fuzzy Extractors. The fuzzy extractor framework ensures that even if the stored data (in the form of compressed fuzzy hashes) is compromised, reconstructing the original biometric data or compromising individual privacy remains computationally infeasible. This is accomplished by generating a secure, random key from the biometric input using a generation process (Gen) and allowing for the reliable reproduction of this key from an approximation of the original input using a reproduction process (Rep), without directly storing the biometric data itself.

In essence, we aim to store the output of the fuzzy extractor alongside the compressed fuzzy hash. This method ensures that the stored biometric data is not only compact and efficiently stored but also securely obfuscated, requiring the correct biometric input for any form of decryption or matching to occur.

2 Biometric Setting

This section provides a comprehensive overview of the mathematical and technical aspects involved in fingervein matching, which is essential for comprehending the subsequent discussions in this paper. Additionally, insights into the process of obtaining these equations and statements will be provided, evaluating their relevance and implications within the context of the research.

2.1 Biometric Data and Similarity Scores

We begin by delving into the representation of the biometric data. Finger images, designated as biometric templates and formated as 250x386, result in $n = 96'500$ pixels per image. To enhance processing efficiency, these templates are converted into vectors, departing from their original 2-dimension image structure.

In the biometric context, each finger serves as a biometric subject, with a corresponding biometric capture represented as a bitstring X of length n. This capture encapsulates the specific vein pixel information extracted from the finger image, while the biometric template serves as a reference mode derived from these images.

Each of the n bits of the biometric capture is designated as $X_1, \ldots, X_n$, with X_i set to 1 if the i-th bit corresponds to a vein and 0 otherwise. We define a random variable as the Hamming Weight of a feature vector X divided by the total number of bits in the vector $\left(\frac{\mathrm{HW}(X)}{n}\right)$. The parameter p is defined as the mean of this random variable.

$$p = \mathbb{E}\left(\frac{\mathrm{HW}(X)}{n}\right) = Pr\left(X_i = 1\right) \tag{2.1}$$

In the case where i is a uniformly distributed random index and X is randomly chosen, the mean (p)[1], along with the variance (σ^2) and standard deviation (σ) of the random variable $\left(\frac{\mathrm{HW}(X)}{n}\right)$ across all images are quantified as follows:

$$p^{\mathrm{obs}} = \mathbb{E}\left(\frac{\mathrm{HW}(X)}{n}\right) = 3.29\% \tag{2.2}$$

$$\sigma_p^{\mathrm{obs}} = \sqrt{\mathbb{E}\left[\left(\frac{\mathrm{HW}(X)}{n} - p^{\mathrm{obs}}\right)^2\right]} \approx 4.69 \times 10^{-3} \tag{2.3}$$

$$(\sigma_p^{\mathrm{obs}})^2 = \mathbb{V}\left(\frac{\mathrm{HW}(X)}{n}\right) \approx 2.20 \times 10^{-5} \tag{2.4}$$

[1]Throughout this report, experimentally derived variables are denoted with the superscript "obs" (e.g., $\mathrm{var}^{\mathrm{obs}}$), while theoretical variables are presented without any superscript.

where σ and σ^2 represent the standard deviation and variance of the random variable $\left(\frac{\mathrm{HW}(X)}{n}\right)$ respectively.

In biometric authentication and identification, the uniqueness of each biometric capture is encoded in its bits. The objective revolves around discerning the similarity between two biometric captures to verify or identify two individuals. Therefore, the scoring mechansim plays a crucial role in quantatively determining the similarity between biometric captures. This similarity score holds significance in verifying the identity of an individual (authentication) or identifying potential matches in a database (identification). A higher score indicates a greater ressemblance between captures, while a lower score suggests less similarity. This scoring mechanism is indispensable for ensuring the accuracy and reliability of biometric systems. The score of (X, Y) is computed as

$$\begin{aligned} Score(X,Y) &= \frac{HW(X \wedge Y)}{HW(X) + HW(Y)} \\ &= \frac{1}{2} - \frac{1}{2}\frac{d_H(X,Y)}{HW(X) + HW(Y)} \end{aligned} \tag{2.5}$$

where HW denotes the Hamming Weight and d_H the Hamming Distance.

In conjunction with the scoring mechanism, Miura matching emerges as a specialized technique for comparing biometric samples. This method entails determining an optimal offset translation, denoted as d_X and d_Y, between two biometric samples to align their features for comparison, thereby compensating for differences in positioning or orientation. The alignment process maximizes the similarity score, defined as:

$$Score = \mathrm{Score}(d_X \cdot X, d_Y \cdot Y)$$

Once the optimal offsets, d_X and d_Y, are determined, they are applied to the original samples for alignment. This results in the calculated aligned positions, denoted as $\bar{X}$ and $\bar{Y}$, such that:

$$\begin{aligned} \bar{X} &= d_X \cdot X \\ \bar{Y} &= d_Y \cdot Y \end{aligned}$$

Miura matching and the scoring mechanism are interconnected components, where Miura matching facilitates the computation of the score, thereby offering insights into the similarity between biometric captures and enhancing the reliability of matching algorithms.

The probability of the i-th pixel of two captures not being the same after applying the optimal offset translations depends on the distribution of $(\bar{X}, \bar{Y})$, and is denoted as δ. We define the random variable as the normalized Hamming distance between two feature vectors $\left(\frac{d_H(\bar{X},\bar{Y})}{n}\right)$. The parameter δ is defined as the mean of this random

variable, and we additionally define the standard deviation and the variance of this random variable, as shown in the following figures.

$$\delta = \mathbb{E}\left(\frac{d_H(\bar{X},\bar{Y})}{n}\right) \tag{2.6}$$

$$\sigma_\delta = \sigma\left(\frac{d_H(\bar{X},\bar{Y})}{n}\right) \tag{2.7}$$

$$(\sigma_\delta)^2 = Var\left(\frac{d_H(\bar{X},\bar{Y})}{n}\right) \tag{2.8}$$

Distinguishing between the types of distributions associated with the two captures is crucial. When both captures originate from the same biometric subject, it is referred to as δ_{same}. Conversely, if the captures are from different subjects, it is labeled as δ_{diff}. Additionally, when $\bar{X}$ and $\bar{Y}$ consist of $2n$ independent random bits with an expected value of p and no optimal offset is applied, it is classified as δ_{indep}. In this scenario, $\delta^{\text{obs}}_{\text{indep}} = 2p(1-p) = 6.4\%$. As developed in Section 2.3, the experimental results produce the following figures:

	δ^{obs}	$(\sigma_\delta{}^{\text{obs}})^2$	$\sigma^{\text{obs}}_\delta$
Same-Finger Distribution	4.55%	1.06×10^{-4}	1.03×10^{-2}
Different-Finger Distribution	5.99%	4.58×10^{-5}	6.76×10^{-3}

Table 2: Comparison of Distributions: Mean, Variance, and Standard Deviation of $\left(\frac{d_H(\bar{X},\bar{Y})}{n}\right)$ when both captures originate from the same finger and from different fingers

Utilizing p (2.1) and δ (2.6), the joint distribution for $(\bar{X}_j,\ \bar{Y}_j)$ across j random instances is derived:

	$\bar{Y}_i = 0$	$\bar{Y}_i = 1$
$\bar{X}_i = 0$	$1 - p - \frac{\delta}{2}$	$\frac{\delta}{2}$
$\bar{X}_i = 1$	$\frac{\delta}{2}$	$p - \frac{\delta}{2}$

Table 3: Joint Distribution of $(\bar{X}_j,\ \bar{Y}_j)$ for Random Instances

2.2 Experimental Derivation of the Probability p

In order to derive the probability that a distributed random pixel is a vein, p (2.1), a dataset of 20 unique individuals was utilized. Each individual contributed images of both right and left index/middle fingers, with 5 trials per finger, captured using 2 different cameras. This methodology resulted in a comprehensive dataset of 800

images. Following the approach in Section 1.2, Simon's optimal pipeline was implemented to extract the feature vectors from each image in the dataset (refer to Figure 1.1). Subsequently, statistical analysis was performed on these feature vectors to gain insights into vein patterns, building on the preliminary work by Burcu. It's important to note that the calculation of the probability p does not take into account any postalignment method, like Miura Matching, even though the code might seem like it does.

We have carried out this analysis by implementing an algorithm which processes the dataset's feature vectors, performing two main functions:

1. For each pixel position, it updates a *veins[i]* array, which accumulates detections of veins across all images for each camera $i = 1, 2$.
2. It maintains a count of the number of images analyzed per camera $i = 1, 2$ in a *image_count[i]* array.

Following the processing of all feature vectors, the algorithm computes the probability of a pixel being part of a vein by dividing the aggregated vein detections by the total number of images analyzed, across both cameras. Visual representations were generated to illustrate the distribution of detected vein pixels for each camera individually and for the combined data from both cameras.

1. **Camera 1 and Camera 2 distribution**: The following histograms showcase the frequency distribution of vein pixels detected in images captured by Camera 1 and 2 individually, with a Gaussian fit overlaid to highlight the data's normal distribution trend.

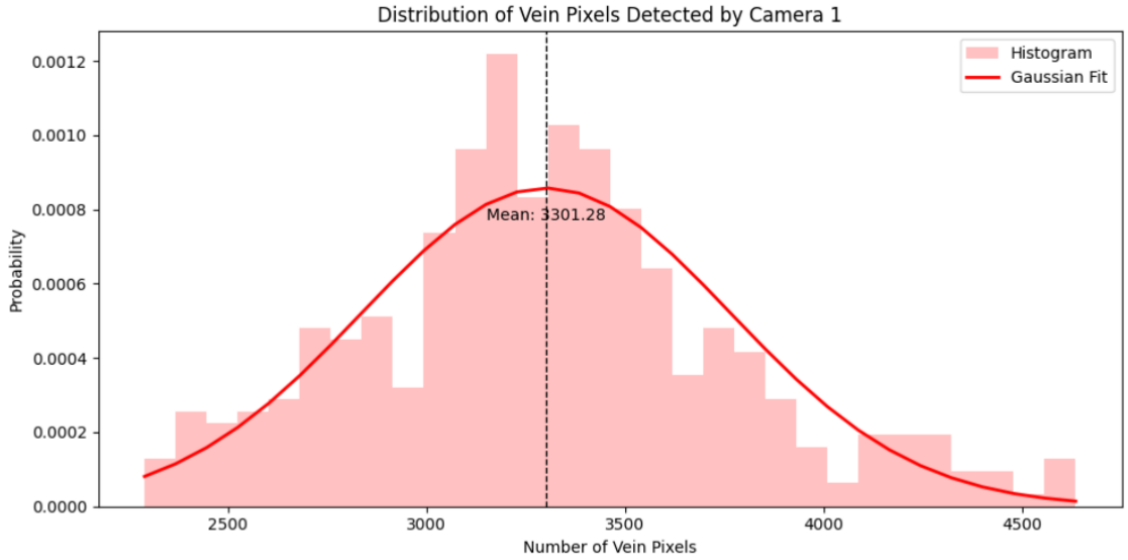

Figure 2.1: Vein Pixel Distribution Analysis Using Camera 1: Histogram Representation with Gaussian Fit Overlay with $X_i \sim \mathcal{N}(3301.28, 216433.17)$

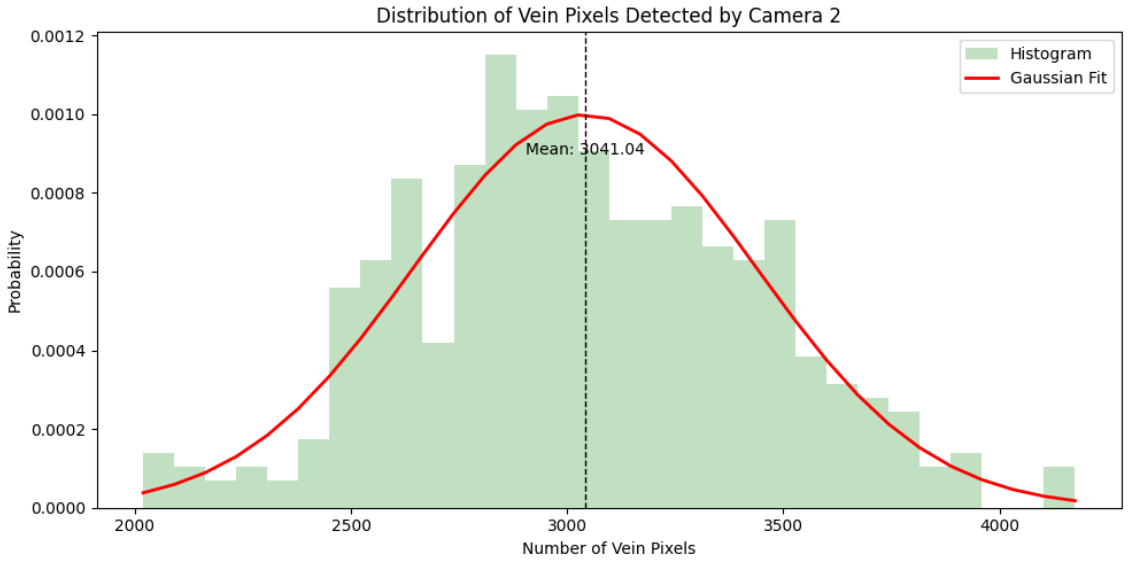

Figure 2.2: Vein Pixel Distribution Analysis Using Camera 2: Histogram Representation with Gaussian Fit Overlay $X_i \sim \mathcal{N}(3041.04, 159577.13)$

2. **Combined Cameras Distribution**: To understand the aggregate behavior of vein detection across both cameras, the data was merged, generating a comprehensive histogram with a Gaussian fit.

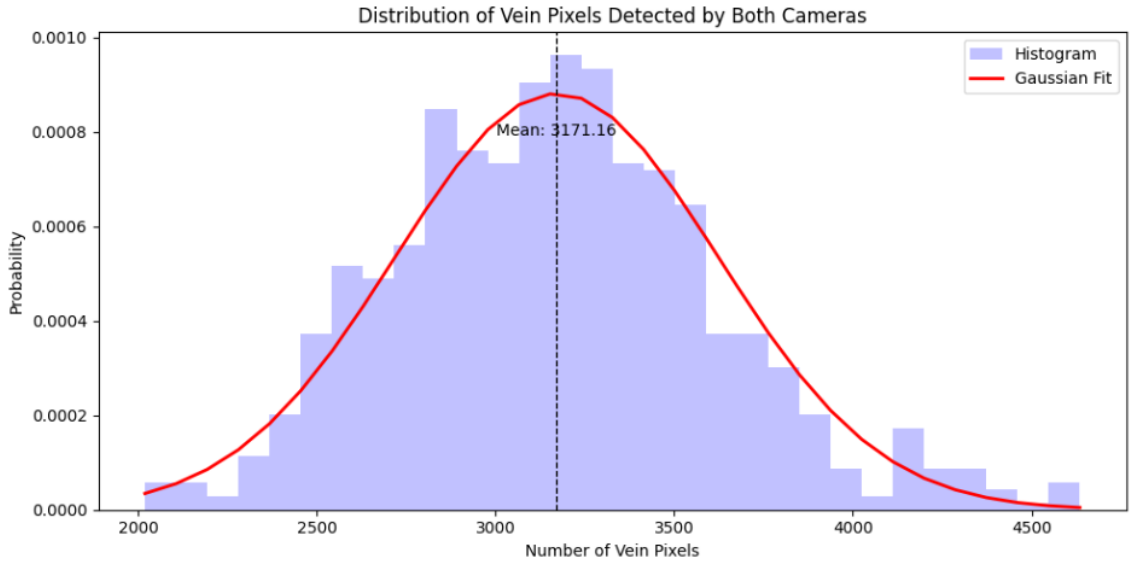

Figure 2.3: Aggregate Vein Pixel Distribution Analysis: Combined Histogram and Gaussian Fit from Both Cameras $X_i \sim \mathcal{N}(3171.16, 204935.79)$

Concluding the analysis of the vein pixel distribution across different camera captures and their aggregate dataset, the average probability p that a given pixel corresponds to a vein was computed. This involved summing up all vein-identifying pixels within the datasets for Camera 1, Camera 2, and the combined dataset. Subsequently, this sum was divided by the total number of pixels processed, multiplying the count of images by the total pixels per image, to ascertain the average likelihood p that any randomly selected pixel is part of a vein pattern.

The findings from this analysis revealed the following probabilities:

1. For Camera 1, the probability $p^{obs} = \mathbb{E}\left(\frac{\mathrm{HW}(X)}{n}\right)$ was calculated to be 0.034, indicating a 3.42% chance that any given pixel in images from Camera 1 represents a vein.

2. For Camera 2, the probability $p^{obs} = \mathbb{E}\left(\frac{\mathrm{HW}(X)}{n}\right)$ was slightly lower at 0.032, translating to a 3.15% chance for vein representation in its images.

3. When considering the datasets from Both Cameras combined, the probability $p^{obs} = \mathbb{E}\left(\frac{\mathrm{HW}(X)}{n}\right)$ averaged out to 0.0329, suggesting a 3.29% likelihood of a pixel depicting a vein across the entire dataset. Moreover, the variance (σ_p^2) and the standard deviation (σ_p) of the random variable $\left(\frac{\mathrm{HW}(X)}{n}\right)$ provide insights into the the random variable's spread and consistency. The observed variance is low at approximately 2.20×10^{-5}, indicating minimal fluctuation

in the random variables' values among different feature vectors. This suggests that the feature extraction method is highly reliable and consistent across different images. Furthermore, the standard deviation, calculated as approximately 4.69×10^{-3}, reinforces the idea of minimal dispersion around the mean probability (p) of detecting a vein.

2.3 Experimental Derivation of the Probabilities $\delta_{\text{same}}, \delta_{\text{diff}}, \delta_{\text{indep}}$

To assess the probability that the i-th pixel of two captures diverges after applying the optimal offset translations d_X and d_Y (Equation (2.6)), we undertake additional experimental analysis. This investigation continues to utilize the same dataset comprising 20 distinct individuals. Here, the post-alignment process, specifically Miura Matching, is included in our examination as the calculation formula integrates this step. As detailed in Section 2.1, our objective is to evaluate three distinct delta values: $\delta_{\text{same}}, \delta_{\text{diff}}, \delta_{\text{indep}}$. The computation of δ_{indep} is relatively straightforward, having determined the value of p. In order to calculate the mean, the standard deviation, and the variance of the random variable $\left(\frac{d_H(\bar{X},\bar{Y})}{n}\right)$ when X and Y are made up of $2n$ independent random bits of expected value p and without applying the optimal offset to get $\bar{X}$ and $\bar{Y}$, we can do the following:

The observed value of p is given, so we compute the mean as:

$$\delta_{\text{indep}}^{obs} = 2p^{obs}(1-p^{obs}) = 2 \times 0.0329 \times (1 - 0.0329) \approx 6.36\%$$

Next, to find the variance and the standard deviation of the normalized Hamming distance we have:

$$(\sigma_{\delta_{indep}}^{\text{obs}})^2 = \mathbb{V}\left(\frac{d_H(\bar{X},\bar{Y})}{n}\right) = \frac{4p^{obs}(1-p^{obs})(1-2p^{obs}(1-p^{obs}))}{n} \approx 1.23 \times 10^{-6}$$

$$\sigma_{\delta_{indep}}^{\text{obs}} = \sqrt{\mathbb{E}\left[\left(\frac{d_H(\bar{X},\bar{Y})}{n} - \delta_{\text{indep}}\right)^2\right]} = \sqrt{\frac{4p^{obs}(1-p^{obs})(1-2p^{obs}(1-p^{obs}))}{n}} \approx 1.11 \times 10^{-3}$$

To accurately evaluate δ_{same} and δ_{diff}, a meticulous procedure was executed on our dataset, which involves 20 unique individuals, each having multiple biometric captures. Below is an outline of the methodology applied:

1. **Pairwise Comparison**: For each individual, we compared every biometric capture to every other capture within the dataset, applying the Hamming distance metric to compute the normalized distances.
2. **Statistical Analysis**: We organized the resulting distances into two distinct categories:
   - Intra-individual distances, where captures from the same individual (same finger images) were compared, forming the first group.
   - Inter-individual distances, comprising comparisons between biometric captures from different individuals, forming the second group.

3. **Probability Computation**:

   - For δ_{same}, we calculated the average normalized distances from intra-individual comparisons. This analysis provided the following results, quantifying the observed differences between captures from the same individual after alignment:

$$\delta_{\text{same}}^{obs} \approx 4.55\%$$
$$\sigma_{\delta_{same}}^{obs} \approx 1.03 \times 10^{-2}$$
$$(\sigma_{\delta_{same}}^{obs})^2 \approx 1.06 \times 10^{-4}$$

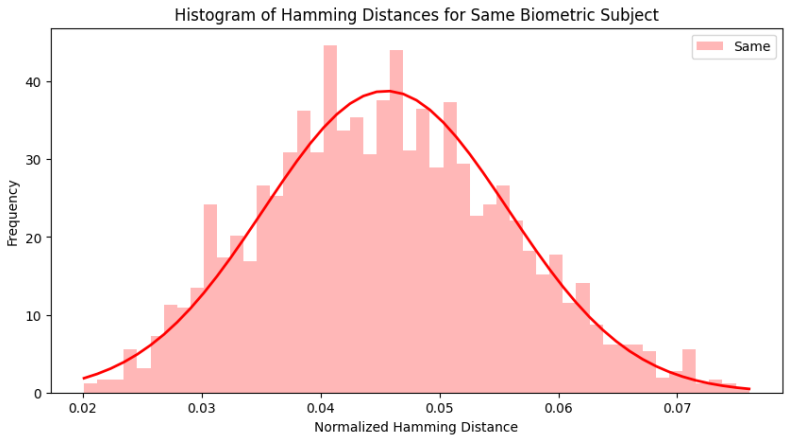

Figure 2.4: Frequency Distribution of Normalized Hamming Distances for Identical Biometric Samples with Alignment $\delta_{same} \sim \mathcal{N}(0.045, 0.000106)$

   - For δ_{diff}, we performed a similar averaging of the normalized distances from the inter-individual comparisons, which furnished us with the following results for observing a difference between captures from the different individuals after alignment:

$$\delta_{\text{diff}}^{obs} \approx 5.99\%$$
$$\sigma_{\delta_{diff}}^{obs} \approx 6.76 \times 10^{-3}$$
$$(\sigma_{\delta_{diff}}^{obs})^2 \approx 4.58 \times 10^{-5}$$

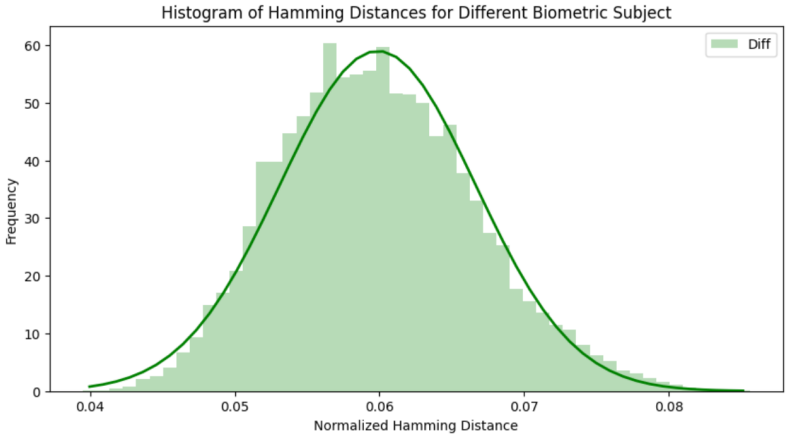

Figure 2.5: Frequency Distribution of Normalized Hamming Distances for Different Biometric Samples with Alignment $\delta_{diff} \sim \mathcal{N}(0.0599, 0.0000458)$

The joint probability distribution presented in Table 3 represents the relationship between the variables $\bar{X}_i$ and $\bar{Y}_i$, which compare the i-th bit of two biometric samples. The parameter p denotes the probability of a bit being '1', and by complement, $1 - p$ signifies the probability of a bit being '0'. The parameter δ encapsulates the probability that the two bits are not identical, which can occur due to various sources of error in biometric comparison.

In Table 3, we explain each entry:

- By utilizing the previously determined value of δ, we can infer the probabilities $P[\bar{X}_i = 1, \bar{Y}_i = 0]$ and $P[\bar{X}_i = 0, \bar{Y}_i = 1]$ as follows:

  $$\delta = P[\bar{X}_i \neq \bar{Y}_i] = P[\bar{X}_i = 1, \bar{Y}_i = 0] + P[\bar{X}_i = 0, \bar{Y}_i = 1]$$

  Hence we have:

  $$P[\bar{X}_i = 1, \bar{Y}_i = 0] = P[\bar{X}_i = 0, \bar{Y}_i = 1] = \frac{\delta}{2}$$

- To determine the probability$P[\bar{X}_i = 0, \bar{Y}_i = 0]$, we consider the following relationships and utilize the law of total probability:

  $$P[\bar{X}_i = 0] = 1 - p = P[\bar{X}_i = 0, \bar{Y}_i = 0] + P[\bar{X}_i = 0, \bar{Y}_i = 1]$$

  $$P[\bar{Y}_i = 0] = 1 - p = P[\bar{X}_i = 0, \bar{Y}_i = 0] + P[\bar{X}_i = 1, \bar{Y}_i = 0]$$

  By solving these equations, we can derive the probability of interest, $P[\bar{X}_i = 0, \bar{Y}_i = 0]$, and hence infer the previously obtained values for $P[\bar{X}_i = 1, \bar{Y}_i = 0]$ and $P[\bar{X}_i = 0, \bar{Y}_i = 1]$:

  $$P[\bar{X}_i = 0, \bar{Y}_i = 0] = 1 - p - \frac{\delta}{2}$$

- The probability $P[\bar{X}_i = 1, \bar{Y}_i = 1]$ is derived by

$$P[\bar{X}_i = 1, \bar{Y}_i = 1] = 1 - P[\bar{X}_i = 0, \bar{Y}_i = 0] = p - \frac{\delta}{2}$$

Utilizing δ^{obs}_{same}, δ^{obs}_{diff} and δ^{obs}_{indep}, we can determine the corresponding probabilities presented in Table 3. This gives us the following figures:

	$\bar{Y}_i = 0$	$\bar{Y}_i = 1$
$\bar{X}_i = 0$	$1 - p - \frac{\delta^{obs}_{indep}}{2} = 93.5\%$	$\frac{\delta^{obs}_{indep}}{2} = 3.2\%$
$\bar{X}_i = 1$	$\frac{\delta^{obs}_{indep}}{2} = 3.2\%$	$p - \frac{\delta^{obs}_{indep}}{2} = 0.1\%$

Table 4: Joint Distribution of $(\bar{X}_j, \bar{Y}_j)$ for X and Y made up of $2n$ independent random bits of expected value p

	$\bar{Y}_i = 0$	$\bar{Y}_i = 1$
$\bar{X}_i = 0$	$1 - p - \frac{\delta^{obs}_{same}}{2} = 94.4\%$	$\frac{\delta^{obs}_{same}}{2} = 2.3\%$
$\bar{X}_i = 1$	$\frac{\delta^{obs}_{same}}{2} = 2.3\%$	$p - \frac{\delta^{obs}_{same}}{2} = 1\%$

Table 5: Joint Distribution of $(\bar{X}_j, \bar{Y}_j)$ for X and Y Originating from the Same Biometric Subject

	$\bar{Y}_i = 0$	$\bar{Y}_i = 1$
$\bar{X}_i = 0$	$1 - p - \frac{\delta^{obs}_{diff}}{2} = 93.7\%$	$\frac{\delta^{obs}_{diff}}{2} = 3\%$
$\bar{X}_i = 1$	$\frac{\delta^{obs}_{diff}}{2} = 3\%$	$p - \frac{\delta^{obs}_{diff}}{2} = 0.3\%$

Table 6: Joint Distribution of $(\bar{X}_j, \bar{Y}_j)$ for X and Y Originating from Different Biometric Subject

3 Fuzzy Hashing

Fuzzy hashing, as opposed to traditional hashing, produces consistent cryptographic keys for similar but not identical inputs, enabling recognition of the same biometric trait across different instances despite slight variations. This approach ensures legitimate users are not incorrectly denied access due to minor discrepancies and protects user privacy by storing and using hashed values instead of raw biometric data, making it difficult to reverse-engineer the original data even if unauthorized access occurs. This section will discuss how we implemented the fuzzy hashing algorithm, its corresponding mathematical aspects and some experiments.

3.1 PreHashing Algorithm

The *PreHash* algorithm is the first step in the fuzzy hashing process, designed to manipulate biometric templates extracted from finger vein patterns. It operates on a bitstring X, representing the presence (1) or absence (0) of vein pixels across n pixels, where $n = 96'500$.

Algorithm Inputs and Outputs:

1. **Inputs**: As illustrated in the pseudocode 1, the algorithm takes three main inputs:
   - **A parameter m**: the number of indices to find
   - **A bitstring X**: the feature-extracted vein patterns of a biometric capture
   - **A key**: used to initialize a Pseudorandom Number Generator (PRNG)
2. **Output**: The algorithm outputs a tuple $(i_1, ..., i_m)$ consisting of the m smallest indices i_j such that $1 \leq i_1 < ... < i_m$ and the pixel at $PRNGkey(i_j)$ in X is identified as a vein pixel.

Detailed Process of *PreHash*:

- **Initiallization**: Utilizing the provided key, the algorithm initializes a PRNG. This PRNG is based on the Advanced Encryption Standard (AES) in Counter (CTR) mode, ensuring the generation of uniform and independent pseudorandom sequences.
- **Nonce Generation**: A nonce in CTR mode encryption is initialized to zero to maintain simplicity and security. We opted against using a keyed hash function to generate the nonce, as it would tie the nonce to the secret key. Such a dependency would mean that both the nonce and the Pseudo-Random Number Generator (PRNG) would rely on the same key, creating a security risk by concentrating security on a single element. To avoid this, we keep the generation of the nonce separate from the key.
- **Pseudorandom Sequence Generation**: Upon initialization with the specified parameters—key and nonce—the PRNG operates in Counter (CTR) mode

to generate a sequence of pseudo-random numbers. As illustrated in Figure 3.1, the process involves encrypting an incrementing counter combined with the nonce using the specified key. The output from this block cipher encryption is used to generate a unique stream of pseudo-random bits. To optimize the utilization of the cryptographic output and reduce operational overhead, the PRNG is designed to generate a full 128-bit block of pseudo-random data at a time, even though only a portion of these bits are used immediately. When a new block is generated, the algorithm extracts the lowest 17 bits to obtain the pseudo-random number for the current operation. This choice is aligned with the project's requirement for representing image sizes, where a maximum of $96'500$ pixels can be represented, necessitating exactly 17 bits per operation ($\lceil \log_2(96'500) \rceil = 17$). These extracted bits serve as the pseudo-random number. Following extraction, the algorithm discards the used bits from the current block by right-shifting the block by 17 bits. This operation updates the state of the PRNG, ensuring that the consumed bits are no longer considered in subsequent operations. In the case where there are no longer 17 bits available after extraction, the algorithm generates a new pseudo-random number using the PRNG and repeats the process. This mechanism guarantees the efficient utilization of all bits produced by the block cipher, enhancing efficiency and reducing operational overhead. The predictability and reproducibility of the sequence are entirely dictated by the chosen key. Employing the same key across sessions guarantees the generation of an identical sequence of pseudo-random numbers.

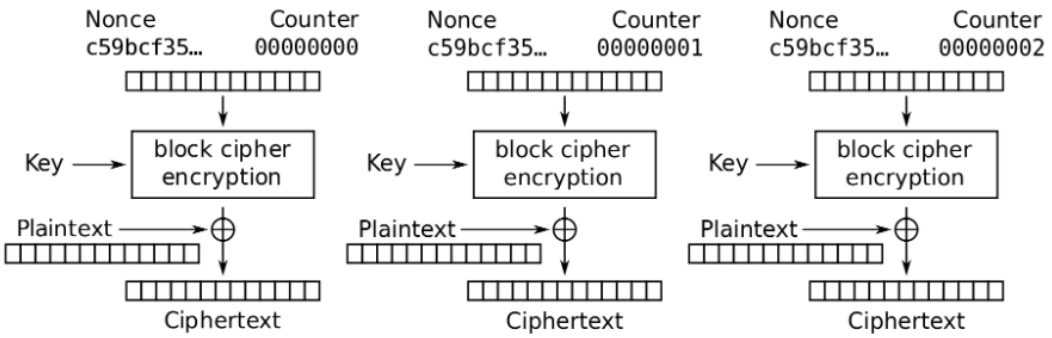

Figure 3.1: Counter (CTR) mode encryption

- **Selection of Indices**: The algorithm iterates through the generated pseudo-random sequence, selecting the first m indices corresponding to vein pixels in the biometric template X. This selection process involves a careful mechanism to ensure the uniqueness and proper ordering of indices.
- **Corner Cases**: In scenarios where there are no veins present in the image, the algorithm incorporates a built-in mechanism to address such situations. Prior to initiating the prehash process, it assesses whether at least one vein is present in the image. If no veins are detected, the algorithm suspends further execution, preventing an infinite loop.

Algorithm 1 *preHash* Algorithm

function $\text{preHash}^{m}_{\text{KEY}}(X)$
    ▷ $L \leftarrow \lceil \log_2(n) \rceil$
    Initialize PRNG using key for L bits
    $i \leftarrow 0$
    for $j \leftarrow 1$ to m **do**
        repeat
            repeat
                $k \leftarrow$ PRNG
            until $1 \leq k \leq n$
            $i \leftarrow i + 1$
        until X_k is a vein pixel
        $i_j \leftarrow k$
    end for
    return $i_1, \ldots, i_m$
end function

The algorithm, preHash, illustrated in Figure 1 generates m smallest indices, denoted by i_j, such that $j \in [1, m]$ and $1 \leq i_1 < ... < i_m$, where each i_j correponds to an index such that $X_{PRNG_{key}(i_j)} = 1$. It achieves this by rigorously verifying that the numbers produced by PRNG (PRNG[i]) stay within the specified bounds PRNG$[i] \in (0, n]$ for $i \in [0, m]$.

3.2 Assessing Similarity of Biometric Inputs After PreHash Application

After the finger images are processed through the pipeline described in Pipeline 1.1 to extract their feature vectors, and the *preHash* algorithm is applied, the outcome is a set of indices that fall within the inclusive range PRNG$[i] \in (0, n]$ for $i \in [0, m]$, effectively mapping each selected feature to a unique index within the feature vector's length.

In the context of a simplified scenario where the hash length parameter (m) is set to 1, implying the generation of a single-index hash, and assuming a randomly chosen key for the *preHash* algorithm, along with k representing a uniformly distributed random index, the probability that the *preHash* operation yields the same index for two different fixed inputs X and Y can be mathematically delineated as follows:

$$
\begin{aligned}
Pr[preHash^1_{key}(X) = preHash^1_{key}(Y)] &= \sum_{i>0} Pr[preHash^1_{key}(X) = preHash^1_{key}(Y) = i] \\
&= \sum_{i>0} Pr[X_k = Y_k = 0]^{i-1} Pr[X_k = Y_k = 1] \\
&= \frac{Pr[X_k = Y_k = 1]}{1 - Pr[X_k = Y_k = 0]} \\
&= \frac{HW(X \wedge Y)}{HW(X) + HW(Y) - HW(X \wedge Y)} \\
&= \frac{1}{\frac{1}{Score(X,Y)} - 1}
\end{aligned}
\tag{3.1}
$$

This equation encapsulates the likelihood of two images, X and Y, having their singular hash index coincide, based on the presence of matching features identified by the algorithm. The final form of the equation relates the probability to the scoring function between X and Y, inversely proportional to the score minus one.

It is noticed that there is a direct link with the Miura matching score that is of interest. The direct link between the *preHash* algorithm's outcomes and the Miura matching score lies in their shared foundation of evaluating biometric similarities. Specifically, both methodologies utilize Hamming weight and bitwise operations to assess the overlap between biometric samples, such as finger vein patterns. The *preHash* algorithm, through its probabilistic formula, quantifies the likelihood of matching indices based on feature presence, closely paralleling the Miura score's approach of comparing binary patterns to derive a similarity score. The above computation can also be expressed as follows:

$$
\begin{aligned}
Pr[preHash^1_{key}(\bar{X}) = preHash^1_{key}(\bar{Y})] &= \frac{Pr[\bar{X}_k = \bar{Y}_k = 1]}{1 - Pr[\bar{X}_k = \bar{Y}_k = 0]} \\
&= \frac{\frac{Pr[X_k=1]+Pr[Y_k=1]}{2} - \frac{1}{2}Pr[\bar{X}_k \neq \bar{Y}_k]}{\frac{Pr[X_k=1]+Pr[Y_k=1]}{2} + \frac{1}{2}Pr[\bar{X}_k \neq \bar{Y}_k]}
\end{aligned}
\tag{3.2}
$$

The following approximations are made, inspired by equations p (2.1) and δ (2.6):

$$
\mathbb{E}\left(\frac{\frac{Pr[X_k=1]+Pr[Y_k=1]}{2} - \frac{1}{2}Pr[\bar{X}_k \neq \bar{Y}_k]}{\frac{Pr[X_k=1]+Pr[Y_k=1]}{2} + \frac{1}{2}Pr[\bar{X}_k \neq \bar{Y}_k]}\right) \approx \frac{p - \frac{\delta}{2}}{p + \frac{\delta}{2}} \tag{3.3}
$$

The core of this approximation revolves around the expectation formula, which integrates probabilities of feature presence $Pr[X_k = 1] + Pr[Y_k = 1]$ and the likelihood of discrepancies between X and Y, $Pr[\bar{X}_k \neq \bar{Y}_k]$. This formula essentially aims to quantify the similarity between two biometric samples by considering both the concurrence of features and the instances where they diverge.

To achieve a more accurate approximation of the expected value of equation 3.2, we define the following parameters and employ a Taylor series expansion of the function $f(A,B,C) = \frac{A+B-C}{A+B+C}$ around the point (p,p,δ). This method allows us to incorporate not only the means but also the variances and covariances of the variables involved.

$$A = \frac{HW(\bar{X})}{n}, \quad B = \frac{HW(\bar{Y})}{n}, \quad C = d_H(\bar{X},\bar{Y})$$

where $HW(\cdot)$ denotes the Hamming weight and $d_H(\cdot,\cdot)$ represents the Hamming distance. Given these definitions, we know the expected values and variances from Section 2 :

$$\mathbb{E}(A) = \mathbb{E}(B) = p^{\text{obs}}, \quad \mathbb{E}(C) = \delta^{\text{obs}}, \quad \mathbb{V}(A) = \mathbb{V}(B) = (\sigma_p^{\text{obs}})^2, \quad \mathbb{V}(C) = (\sigma_\delta^{\text{obs}})^2$$

Next, we perform a Taylor series expansion of $f(A,B,C)$ up to the second order around the point (p,p,δ). The first-order partial derivatives are calculated as follows:

$$f'_A = \frac{2C}{(A+B+C)^2}, \quad f'_B = \frac{2C}{(A+B+C)^2}, \quad f'_C = \frac{-2(A+B)}{(A+B+C)^2}$$

The second-order partial derivatives are:

$$f''_{AA} = f''_{BB} = f''_{AB} = \frac{-4C}{(A+B+C)^3}, \quad f''_{CC} = \frac{4(A+B)}{(A+B+C)^3}, \quad f''_{AC} = f''_{BC} = \frac{-2(A+B-C)}{(A+B+C)^3}$$

By substituting $A = p$, $B = p$, and $C = \delta$ into these second order derivatives, we obtain:

$$f''_{AA} = f''_{BB} = f''_{AB} = \frac{-4\delta}{(2p+\delta)^3}, \quad f''_{CC} = \frac{8p}{(2p+\delta)^3}, \quad f''_{AC} = f''_{BC} = \frac{2(2p-\delta)}{(2p+\delta)^3}$$

Finally, the second order Taylor expansion of $f(A,B,C)$ around (p,p,δ) is given by:

$$f(A,B,C) \approx f(p,p,\delta) + f'_A(p,p,\delta)(A-p) + f'_B(p,p,\delta)(B-p) + f'_C(p,p,\delta)(C-\delta)$$

$$+\frac{1}{2}\left[f''_{AA}(p,p,\delta)(A-p)^2 + f''_{BB}(p,p,\delta)(B-p)^2 + f''_{CC}(p,p,\delta)(C-\delta)^2\right.$$

$$+2f''_{AB}(p,p,\delta)(A-p)(B-p)+2f''_{AC}(p,p,\delta)(A-p)(C-\delta)+2f''_{BC}(p,p,\delta)(B-p)(C-\delta)$$

To approximate the expected value $E(f(A,B,C))$, we use the following relations:

$$\mathbb{E}(A-p) = 0, \quad \mathbb{E}(B-p) = 0, \quad \mathbb{E}(C-\delta) = 0$$

$$\mathbb{V}(A) = \mathbb{V}(B) = \mathbb{E}((A-p)^2), \quad \mathbb{V}(C) = \mathbb{E}((C-\delta)^2)$$

$$\text{Cov}(A,B) = \mathbb{E}((A-p)(B-p)), \quad \text{Cov}(B,C) = \mathbb{E}((B-p)(C-\delta))$$

$$\mathrm{Cov}(A, C) = \mathbb{E}((A - p)(C - \delta))$$

And we hence have:

$$\mathbb{E}(f(A, B, C)) \approx \frac{p - \frac{\delta}{2}}{p + \frac{\delta}{2}} + \frac{1}{2}\left[f''_{AA}(p, p, \delta)\mathbb{V}(A) + f''_{BB}(p, p, \delta)\mathbb{V}(B) + f''_{CC}(p, p, \delta)\mathbb{V}(C)\right.$$
$$\left. + 2f''_{AB}(p, p, \delta)\mathrm{Cov}(A, B) + 2f''_{AC}(p, p, \delta)\mathrm{Cov}(A, C) + 2f''_{BC}(p, p, \delta)\mathrm{Cov}(B, C)\right]$$

Simplifying with our values and notations we get:

$$\begin{aligned}\mathbb{E}(f(A, B, C)) \approx \frac{p - \frac{\delta}{2}}{p + \frac{\delta}{2}} + \frac{1}{2}\Bigg[&\frac{-4\delta}{(2p + \delta)^3} \cdot (\sigma_p^{\mathrm{obs}})^2 + \frac{-4\delta}{(2p + \delta)^3} \cdot (\sigma_p^{\mathrm{obs}})^2 \\ &+ \frac{8p}{(2p + \delta)^3} \cdot (\sigma_\delta^{\mathrm{obs}})^2 + 2 \cdot \frac{-4\delta}{(2p + \delta)^3} \cdot \mathrm{Cov}(A, B) \\ &+ 2 \cdot \frac{2(2p - \delta)}{(2p + \delta)^3} \cdot \mathrm{Cov}(A, C) + 2 \cdot \frac{2(2p - \delta)}{(2p + \delta)^3} \cdot \mathrm{Cov}(B, C)\Bigg]\end{aligned} \tag{3.4}$$

Hence for (X, Y) random,

$$Pr[preHash^1_{key}(offset_X * X) = preHash^1_{key}(offset_Y * Y)] \leq \mathbb{E}(f(A, B, C)) \tag{3.5}$$

where equality is reached for the optimal offset translations.

Depending on the distribution of (X, Y), it is denoted

$$\mu = \mathbb{E}(f(A, B, C)) \tag{3.6}$$

In order to assess the theoretical approximations of μ_{same}, μ_{diff}, and the value of μ_{indep}, we can easily replace the different values in Equation 3.4 with the corresponding values calculated in the previous sections. We demonstrate how this is done for $\mu^{obs}_{\mathrm{indep}}$.

Once we have determined the value of p^{obs}, $\delta^{obs}_{\mathrm{indep}}$, and their respective variances, we aim to calculate $\mathbb{E}(f(A, B, C))$. This calculation is based on X and Y comprising $2n$ independent random bits with an expected value of p, without applying the optimal offset to obtain $\bar{X}$ and $\bar{Y}$. The expectation $\mathbb{E}(f(A, B, C))$ is simplified because the covariances between the different random variables are zero in Equation 3.4. For clarity, note that in the following equation, p corresponds to p^{obs}, δ corresponds to $\delta^{obs}_{\mathrm{indep}}$, σ_p corresponds to σ_p^{obs}, and σ_δ corresponds to $\sigma^{\mathrm{obs}}_{\delta_{\mathrm{indep}}}$. This allows us to express $\mu^{obs}_{\mathrm{indep}}$ as follows:

$$\mu_{\text{indep}}^{obs} = \mathbb{E}(f(A,B,C)) \approx \frac{p-\frac{\delta}{2}}{p+\frac{\delta}{2}} + \frac{1}{2}\left[\frac{-4\delta}{(2p+\delta)^3}\cdot(\sigma_p)^2 + \frac{-4\delta}{(2p+\delta)^3}\cdot(\sigma_p)^2 + \frac{8p}{(2p+\delta)^3}\cdot(\sigma_\delta)^2\right] \tag{3.7}$$

$$\approx 0.01449 = 1.45\%$$

We proceed in a similar way to assess the theoretical values of μ_{same}, μ_{diff}, except in this case the covariances are not 0 and have been calculated by observing the correlation between the different random variables (A, B, C) in the experimental results obtained in Section 2. That is, we calculated the covariance matrix between the following random variables:

$$A = \frac{HW(\bar{X})}{n}, \quad B = \frac{HW(\bar{Y})}{n}, \quad C = d_H(\bar{X},\bar{Y})$$

We obtained the following covariance matrix:

The following theoretical approximations are provided:

μ_{same}	μ_{diff}	μ_{indep}^{obs}
18.8%	4.6%	1.45%

Table 7: Comparison of Distributions: μ_{same}, μ_{diff}, and μ_{indep}

Finally, it is observed that:

$$Pr[preHash_{key}^{m}(offset_X * X) = preHash_{key}^{m}(offset_Y * Y)] \leq \mu^m \tag{3.8}$$

where equality is reached for the optimal offset translations.

3.3 Experimental Derivation of the Probabilities $\mu_{\text{same}}^{\text{obs}}, \mu_{\text{diff}}^{\text{obs}}, \mu_{\text{indep}}^{\text{obs}}$

In Section 3.2, we have laid out the mathematical framework that defines the upper bound μ. This probability is crucial in our fuzzy hashing approach, as it quantifies the matching likelihood after applying the optimal offset translations to the biometric captures. The application of these optimal offsets is an important preprocessing step before hashing the dataset, ensuring that the biometric features are accurately aligned for comparison.

We will proceed to evaluate these theoretical results experimentally, leveraging the same comprehensive database that has been utilized in our prior research, as explained in Section 2.3. This experimental assessment will enable us to substantiate the theoretical predictions of μ_{same} and μ_{diff}.

In the experimental setup, μ_{same} refers to the probability that a hash function, when applied to two biometric captures of the same finger, will yield the same index. This is the measure of success in correctly identifying matches within the same individual's biometric captures. Conversely, μ_{diff} denotes the probability that the same hash function will produce different indices for biometric captures from different individuals, representing the ability to distinguish between different people's biometric data. Lastly, μ_{indep} indicates the probability of a match in the case where the biometric captures are completely independent, serving as a baseline for random chance.

For μ_{same}^{obs} and μ_{diff}^{obs}, our experimental protocol adhered to the established optimal pipeline (refer to Figure 1.1) to process the images in our dataset. Following the alignment of images through the computation of optimal offsets (referred to as Miura Matching), our fuzzy hashing function was applied to the extraced features. In alignment with the premise that μ is computed under the condition where the hash function's output is a single index ($m = 1$).

1. **Pairwise Comparison**: We performed a pairwise comparison for each image pair within the dataset after applying the optimal offsets to ensure proper alignment. The fuzzy hashing function, tailored to output a single index per image, was utilized to hash the images.
2. **Hamming Distance Computation**: We calculated the Hamming distance between the indices obtained from the fuzzy hashing. The distances were binary:
    - A Hamming distance of 0 indicates the indices are identical, suggesting a match.
    - A Hamming distance of 1 indicates differing indices, suggesting a non-match.
3. **Statistical Analysis and Probability Computation**:
    - $\mu_{\textbf{same}}^{obs}$: We computed μ_{same}^{obs} by averaging the number of Hamming distances that resulted in a match (Hamming distances of 0) of the intra-individual comparisons, which furnished us with the following probability:

$$\mu_{\text{same}}^{obs} \approx 0.22047 = 22.05\%$$

    - $\mu_{\textbf{diff}}^{obs}$: Similarly, μ_{diff}^{obs} was determined by averaging the number of Hamming distances that resulted in a match (Hamming distances of 0) of the inter-individual comparisons, which furnished us with the following probability:

$$\mu_{\text{diff}}^{obs} \approx 0.08249 = 8.25\%$$

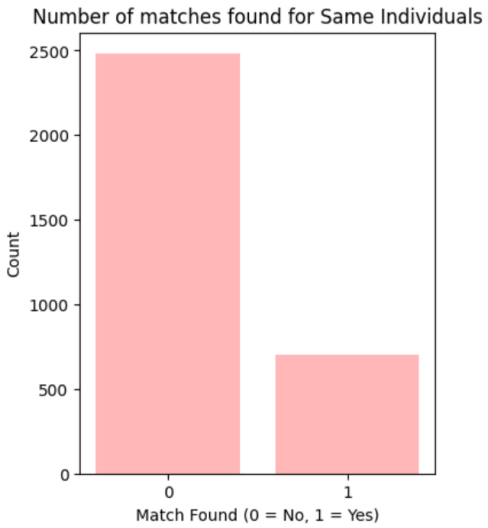

Figure 3.2: Count of the number of matches for Same, Aligned Biometric Samples with Single Index PreHashing

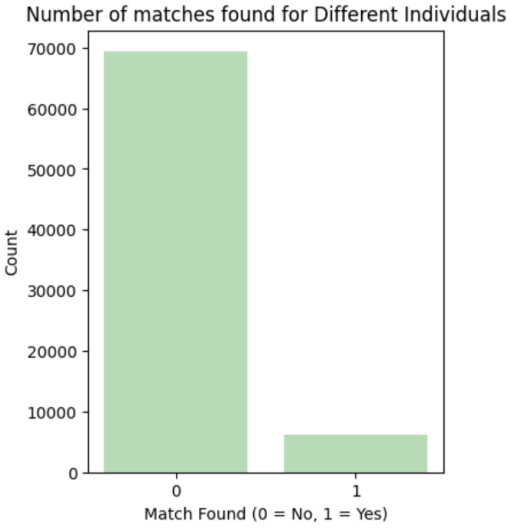

Figure 3.3: Count of the number of matches for Different, Aligned Biometric Samples with Single Index PreHashing

In our analysis of the dataset at hand featuring biometric data from 20 unique individuals, we have quantified the performance of the preHash function in distinguishing between identical and different biometric subjects. Specifically, when evaluating samples from the same individual, the probability that the preHash function will produce the same index is approximately 22.05%. Conversely, when the function is applied to samples from different individuals, the probability of a match in the indices drops to roughly 8.25%. This indicates that the likelihood of mistakenly identifying different individuals as the same is about three times lower than correctly matching samples from the same individual, demonstrating the preHash function's relative efficacy in biometric differentiation within this specific dataset.

It would be valuable to conduct an experimental investigation to determine the effect of omitting the Miura Matching step (finding optimal offsets before comparing images), which serves as a post-alignment phase in our pipeline, on the coincidence probabilities of the indices. Our interest in conducting this verification stems from the contents of Equation 3.5. Should some other non-optimal offsets be used, or should the Miura Matching phase be excluded entirely from our process, it is anticipated that the probabilities of matching indices should be less than or equal to the values determined for μ_{same} and μ_{diff}, respectively.

4 Compressed Fuzzy Hashing

Compressed fuzzy hashing generates concise hashes of data, capturing key features while allowing for slight variations - a concept referred to as "fuzziness". Unlike conventional cryptographic hashing, which generates fixed-length outputs and is highly sensitive to even minimal changes in input, compressed fuzzy hashing maintains a robust link to the original content despite alterations. This section delves into the mechanics of the compressed fuzzy hashing algorithm, called the *PostHash* procedure. We will explore the mathematical concepts that enable its effectiveness and discuss experimental results that demonstrate its capabilities and performance on our dataset.

4.1 PostHashing Algorithm

The *PostHash* algorithm, constituting the second and final step in the fuzzy hashing process, generates compact hashes based on the indices derived from the *PreHash* algorithm (see section 3). This algorithm is designed to transform a tuple of indices into a concise hash format, $h_1||\ldots||h_m$, where each h_i, for $i \in [1, m]$, is an integer within the range $[0, \ldots, D-1]$.The parameter D, defined as 2^d, denotes the total number of possible values each h_i can assume, with d representing the bit depth used per index.

Algorithm Inputs and Outputs:

1. **Inputs**: As illustrated in Figure 2, the algorithm takes as input:
   - **A tuple of indices**: $(i_1, \ldots, i_m)$, which is the output of the *PreHash* algorithm
2. **Output**: Converted indices, forming a compact hash $h_1||\ldots||h_m$, where h_i $i \in [1, m]$, is an integer in $[0, \ldots, D-1]$

Detailed Process of *PostHash*:

1. **Subroutine**: For each index provided by the *PreHash*, subroutine T checks if the index is within the acceptable range. If an index is out of range, T returns 0, otherwise, it proceeds with a table lookup. The subroutine maps each valid index i to an integer nearly uniformly distributed between $[0, D-1]$, ensuring that all potential hash values are equally probable. This mapping is important for the security and effectiveness of the fuzzy hashing system.

   The subroutine function, T, works as follows:

   - **Inputs**:
     - **Shifted Index**: Each index from the *PreHash* output tuple $(i_1, \ldots, i_m)$ is adjusted before conversion into a hash component. Specifically, the index i is shifted by subtracting a previously determined index i', resulting in a transformed index $i - i'$. This shifting process normalizes the indices, maintaining a consistent relative positioning within

the dataset, which helps in achieving a more uniform distribution of hash values across the range $[0, D-1]$.

- **d**: Represents the bit length used for each hash index h_i. This value determines $D = 2^d$, the total number of distinct values each h_i can take, effectively setting the hash resolution.

- **Output**: h_i, the mapped integer

Algorithm 2 *postHash* Algorithm

```
function (postHash ∘ preHash^m_KEY)(X)
    hash = []
    i' ← 0
    for i ∈ indices do
        h_i ← Subroutine(i − i', d)
        hash.append(h_i)
        i' ← i
    end for
    return h_1, ..., h_m
end function
```

Algorithm 3 *Subroutine* Algorithm

```
function Subroutine(i, d)
    p = 0.0329
    h_i = ⌊2^d (1 − p)^i⌋
    return h_i
end function
```

4.2 Assessing Similarity of Biometric Inputs After PostHash Application

After processing finger images through the pipeline referenced in Pipeline 1.1 to extract their feature vectors, the resulting data undergo the *preHash* and subsequently, the *postHash* algorithms. The final output from *postHash* consists of a set of integers h_i, each bounded within the inclusive range $[0, D-1]$. This process effectively assigns each feature vector index to an integer within this range.

In our methodology, we model these integers h_i as following a geometric distribution. This assumption is based on the output relationship established by $Hash^m_{key}(X) = \text{postHash}(\text{preHash}^m_{key}(X))$. We assume the same conditions as discussed in Section 3: the key is randomly chosen, and k represents a uniformly distributed random index. Consequently, the probability of the *Hash* operation yielding the same index for two different inputs X and Y can be mathematically characterized as follows:

$$Pr[Hash^m_{key}(d_X) = Hash^m_{key}(d_Y)] \leq \mu^m(1 - \frac{1}{D}) + \frac{1}{D}$$

where equality is reached for the optimal offset translations.

The overall probability q of a matching hash index under random inputs X and Y, reflecting their distribution, is denoted by:

$$q = \mu^m(1 - \frac{1}{D}) + \frac{1}{D}$$

Based on the range of possible values for d, we have computed several specific instances of q:

	q_{same}	q_{diff}	q_{indep}
$m = 1$ $d = 1$	61.5%	53.9%	50.7%
$m = 1$ $d = 2$	42.3%	30.8%%	26.1%
$m = 1$ $d = 3$	32.6%	19.2%	13.7%
$m = 1$ $d = 4$	27.8%	13.5%	7.6%

Table 8: Comparison of Distributions: q_{same}, q_{diff}, and q_{indep}

4.3 Experimental Derivation of the probabilities q^{obs}_{same}, q^{obs}_{diff}, and q^{obs}_{indep} for different m, d parameter combinations

In the subsequent section, we explore enhancements to data compression techniques within our fuzzy hashing framework, emphasizing the use of various m and d parameter combinations. We initiate our analysis by conducting experiments to determine q^{obs}_{same}, q^{obs}_{diff}, and q^{obs}_{indep}.

Building on the foundational calculations for q_{indep}, which we derived using Formula 4.2 from the diverse $\mu_{\text{indep}^{obs}}$ values previously identified in Section 3, our exploration extended to $q_{\text{same}^{obs}}$ and q^{obs}_{diff}. To evaluate these parameters, we initiated a series of experiments. Integral to this implementation and the following ones, is the creation of the lookup table, to convert indices to their compressed counterparts. The generation of this table and all subsequent ones, is governed by a geometrically-derived formula that accurately encapsulates the transformation of index values. The specific formula is presented as:

$$PostHash(i) = \left\lfloor 2^d \left(1 - p\right)^i \right\rfloor$$

Leveraging the above formula, we have developed lookup tables that convert a single index ($m = 1$) into various bit representations ($d = 1$, $d = 2$, $d = 3$, and $d = 4$). Each entry in these tables represents an integer, denoted by i, and includes a corresponding hash output and the probability of that output occurring.

Binary Hash Output Distribution Table ($d = 1$)

$$PostHash(i) = \begin{cases} 1 & \text{if } 1 \leq i \leq 20 \quad (\Pr = 48.78\%) \\ 0 & \text{if } 21 \leq i \quad (\Pr = 51.22\%) \end{cases}$$

Quaternary Hash Output Distribution Table ($d = 2$):

$$PostHash(i) = \begin{cases} 3 & \text{if } 1 \leq i \leq 8 \quad (\Pr = 23.48\%) \\ 2 & \text{if } 9 \leq i \leq 20 \quad (\Pr = 25.3\%) \\ 1 & \text{if } 21 \leq i \leq 41 \quad (\Pr = 25.85\%) \\ 0 & \text{if } 42 \leq i \quad (\Pr = 25.37\%) \end{cases}$$

Octal Hash Output Distribution Table ($d = 3$):

$$PostHash(i) = \begin{cases} 7 & \text{if } 1 \leq i \leq 3 \quad (\Pr = 9.55\%) \\ 6 & \text{if } 4 \leq i \leq 8 \quad (\Pr = 13.93\%) \\ 5 & \text{if } 9 \leq i \leq 14 \quad (\Pr = 13.92\%) \\ 4 & \text{if } 15 \leq i \leq 20 \quad (\Pr = 11.39\%) \\ 3 & \text{if } 21 \leq i \leq 29 \quad (\Pr = 13.32\%) \\ 2 & \text{if } 30 \leq i \leq 41 \quad (\Pr = 12.53\%) \\ 1 & \text{if } 42 \leq i \leq 62 \quad (\Pr = 12.8\%) \\ 0 & \text{if } 63 \leq i \quad (\Pr = 12.57\%) \end{cases}$$

Hexadecimal Hash Output Distribution Table $d = 4$

$$PostHash(i) = \begin{cases} 15 & \text{if } i = 1 \quad (\Pr = 3.29\%), \\ 14 & \text{if } 2 \leq i \leq 3 \quad (\Pr = 6.26\%) \\ 13 & \text{if } 4 \leq i \leq 6 \quad (\Pr = 8.64\%) \\ 12 & \text{if } 7 \leq i \leq 8 \quad (\Pr = 5.3\%) \\ 11 & \text{if } 9 \leq i \leq 11 \quad (\Pr = 7.31\%) \\ 10 & \text{if } 12 \leq i \leq 14 \quad (\Pr = 6.61\%) \\ 9 & \text{if } 15 \leq i \leq 17 \quad (\Pr = 6\%) \\ 8 & \text{if } 18 \leq i \leq 20 \quad (\Pr = 5.41\%) \\ 7 & \text{if } 21 \leq i \leq 24 \quad (\Pr = 6.41\%) \\ 6 & \text{if } 25 \leq i \leq 29 \quad (\Pr = 6.9\%) \\ 5 & \text{if } 30 \leq i \leq 34 \quad (\Pr = 5.83\%) \\ 4 & \text{if } 35 \leq i \leq 41 \quad (\Pr = 6.7\%) \\ 3 & \text{if } 42 \leq i \leq 50 \quad (\Pr = 6.6\%) \\ 2 & \text{if } 51 \leq i \leq 62 \quad (\Pr = 6.2\%) \\ 1 & \text{if } 63 \leq i \leq 82 \quad (\Pr = 6.13\%) \\ 0 & \text{if } 83 \leq i \leq n \quad (\Pr = 6.43\%) \end{cases}$$

In our current implementation of PostHash output tables, each partition does not distribute probabilities equally among the bins. This uneven distribution, observed

from binary to hexadecimal outputs, poses some challenges, particularly in our system, where hash functions are employed to ensure the confidentiality of biometric data. The non-uniform distribution can lead to predictable patterns, potentially compromising the security by making the system more vulnerable to attacks such as hash collisions. These predictable patterns can be exploited by attackers to reverse-engineer or guess hash values, thereby breaching the confidentiality of biometric information and undermining the integrity of our security measures.

In the context of compressed fuzzy hashing, domain scrambling aims to reorder the indices in a way that the resulting hash values are "nearly" uniformly distributed. This task shares similarities with a well known problem in computer science called the "Partition Problem". This problem involves deciding whether a given multiset of positive integers can be partitioned into two or more subsets such that the sum of the numbers in each subset is equal. This problem is a specific instance of the subset sum problem, where the target sum is half of the total sum of all elements in the set. The partition problem is classified as NP-complete, a categorization in computational complexity theory used to describe decision problems. A problem is in NP (Non-deterministic Polynomial time) if a solution for the problem can be verified in polynomial time given a candidate solution. moreover, a problem is NP-complete if it is as hard as the hardest problems in NP and if every problem in NP can be reduced to it in polynomial time. This implies two things for NP-complete problems:

1. Verification is Feasible: Given a "solution," we can verify whether it is correct in polynomial time.
2. There is no known algorithm that can find an optimal solution to NP-complete problems in polynomial time for all general cases.

Given the NP-completeness of the partition problem, programming an optimal solution for domain scrambling — which is akin to a continuous partitioning challenge — is impractical. Theoretically, while it is feasible to verify if a given distribution of indices across hash buckets is optimal, finding that distribution algorithmically cannot be guaranteed to be efficient for all possible cases. However, we did try to implement this algorithm and observe the results for the case where the hash depth is $d = 4$, resulting in $D = 16$ possible hash values. The program written is designed to distribute a sequence of integers (1 to 96'500) across predefined buckets (0 to 15) to achieve a target probability distribution based on our geometric decay model 4.3. This process begins by mapping each integer to a bucket using a calculated index derived from the geometric formula, adjusted by a scaling factor. Initial bucket assignments aim to approximate a uniform distribution of probabilities. The program then iteratively adjusts these assignments: integers are moved between buckets to better align with the target probabilities, ensuring that the sum of probabilities within each bucket remains close to the desired uniform distribution. This adjustment process continues until the distribution of probabilities across all buckets falls within an acceptable error margin, or until no further beneficial adjustments can be made. The following lookup table illustrates the distribution of indices across each bucket, including the count of indices falling within each bucket and the range of

indices encompassed by it.

$$\text{Bucket Assignments} = \begin{cases} 0 & \text{if } i \in [1, 96500] \quad (\text{Count} = 88216, \text{Pr} = 6.269\%), \\ 1 & \text{if } i \in [2, 8287] \quad (\text{Count} = 2075, \text{Pr} = 6.274\%), \\ 2 & \text{if } i \in [3, 6215] \quad (\text{Count} = 1216, \text{Pr} = 6.298\%), \\ 3 & \text{if } i \in [4, 5003] \quad (\text{Count} = 864, \text{Pr} = 6.292\%), \\ 4 & \text{if } i \in [5, 4143] \quad (\text{Count} = 671, \text{Pr} = 6.289\%), \\ 5 & \text{if } i \in [6, 3476] \quad (\text{Count} = 549, \text{Pr} = 6.253\%), \\ 6 & \text{if } i \in [7, 2931] \quad (\text{Count} = 465, \text{Pr} = 6.298\%), \\ 7 & \text{if } i \in [8, 2471] \quad (\text{Count} = 405, \text{Pr} = 6.289\%), \\ 8 & \text{if } i \in [9, 2071] \quad (\text{Count} = 358, \text{Pr} = 6.298\%), \\ 9 & \text{if } i \in [10, 1719] \quad (\text{Count} = 321, \text{Pr} = 6.290\%), \\ 10 & \text{if } i \in [11, 1404] \quad (\text{Count} = 291, \text{Pr} = 6.297\%), \\ 11 & \text{if } i \in [12, 1120] \quad (\text{Count} = 269, \text{Pr} = 6.298\%), \\ 12 & \text{if } i \in [13, 859] \quad (\text{Count} = 247, \text{Pr} = 6.231\%), \\ 13 & \text{if } i \in [14, 620] \quad (\text{Count} = 230, \text{Pr} = 6.092\%), \\ 14 & \text{if } i \in [15, 399] \quad (\text{Count} = 215, \text{Pr} = 5.981\%), \\ 15 & \text{if } i \in [80, 192] \quad (\text{Count} = 108, \text{Pr} = 6.252\%) \end{cases}$$

The distribution of probabilities and counts across buckets demonstrates a commendable effort towards achieving uniformity. However, the variations observed, particularly in buckets 13 and 14 with their notably lower probabilities of 5.981% and 6.092, indicate potential areas for further optimization. To address this, we have decided to try implementing a close to optimal solution by manually scrambling the domain to achieve a desirable uniform distribution. This manual method involved:

- Assigning indices to buckets (hash values) initially based on heuristic insights and preliminary analysis to ensure a rough balance.
- Iteratively adjusting the distribution by swapping indices between buckets to refine the balance, based on observed distributions and targeted uniformity.

This manual approach allowed us to control the distribution directly and ensure a quasi-even spread across hash outputs, which is crucial for maintaining the efficacy and reliability of the compressed fuzzy hashing system.

$$PostHash(i) = \begin{cases} 15 & \text{if } i \in \{1\} \cup \{4\} & (\Pr = 6.27\%), \\ 14 & \text{if } i \in \{2,3\} & (\Pr = 6.26\%), \\ 13 & \text{if } i \in \{5,6\} \cup \{53\} & (\Pr = 6.24\%), \\ 12 & \text{if } i \in \{7,8\} \cup \{38\} & (\Pr = 6.25\%), \\ 11 & \text{if } i \in \{9,10\} \cup \{29\} & (\Pr = 6.24\%), \\ 10 & \text{if } i \in \{12,13\} \cup \{83\} \cup \{85,\ldots,92\} \cup \{99\} \cup \{170\} & (\Pr = 6.24\%), \\ 9 & \text{if } i \in \{15,\ldots,17\} \cup \{78\} & (\Pr = 6.23\%), \\ 8 & \text{if } i \in \{18,\ldots,20\} \cup \{42\} & (\Pr = 6.24\%), \\ 7 & \text{if } i \in \{21,\ldots,23\} \cup \{63,64\} \cup \{71\} \cup \{100\} \cup \{102\} & (\Pr = 6.25\%), \\ 6 & \text{if } i \in \{25,\ldots,28\} \cup \{61\} \cup \{84\} & (\Pr = 6.26\%), \\ 5 & \text{if } i \in \{30,\ldots,34\} \cup \{62\} & (\Pr = 6.27\%), \\ 4 & \text{if } i \in \{35,\ldots,37\} \cup \{39,\ldots,41\} \cup \{57\} & (\Pr = 6.25\%), \\ 3 & \text{if } i \in \{43,\ldots,50\} \cup \{59\} & (\Pr = 6.23\%), \\ 2 & \text{if } i \in \{11\} \cup \{51,52\} \cup \{54,\ldots,56\} \cup \{58\} \cup \{60\} \cup\{98\} & (\Pr = 6.27\%), \\ 1 & \text{if } i \in \{24\} \cup \{65,\ldots,70\} \cup \{72,\ldots,77\} \cup\{79,\ldots,82\} & (\Pr = 6.27\%), \\ 0 & \text{if } i \in \{14\} \cup \{93,\ldots,97\} \cup \{101\} \cup \{103,\ldots,169\} \cup\{171,\ldots,n\} & (\Pr = 6.24\%) \end{cases}$$

Utilizing these derived lookup tables, alongside our established PreHash function(1) and the newly implemented PostHash function (2), we are equipped to conduct experiments on our dataset to assess q_{same} and q_{diff} for the 4 different combinations of the parameters d and m. The modification in our pipeline involves passing the output from the PreHash function directly into the PostHash function. Subsequently, we calculate the Hamming distance between each image pair at the conclusion of both the PreHash and PostHash stages. This method allows us to measure the similarity of the biometric inputs after the complete application of our hashing process.

1. $d = 1$ and $m = 1$: Single-index outputs ($m = 1$) and binary PostHash indices ($d = 1$).
   - $q_{same}^{obs} = 65.7\%$
   - $q_{diff}^{obs} = 56.4\%$

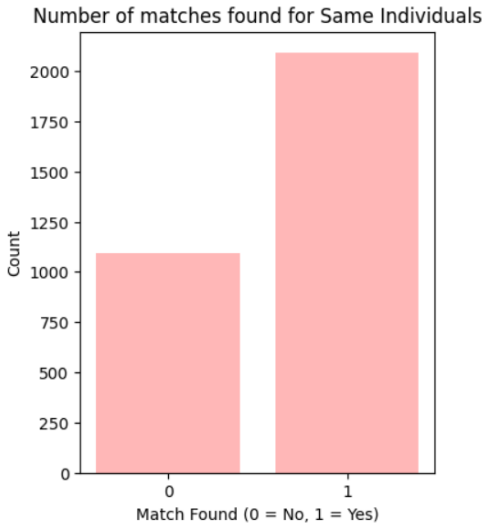

Figure 4.1: Count of the number of matches for Same, Aligned Biometric Samples with Single Index Hashing and Binary PostHash Indices

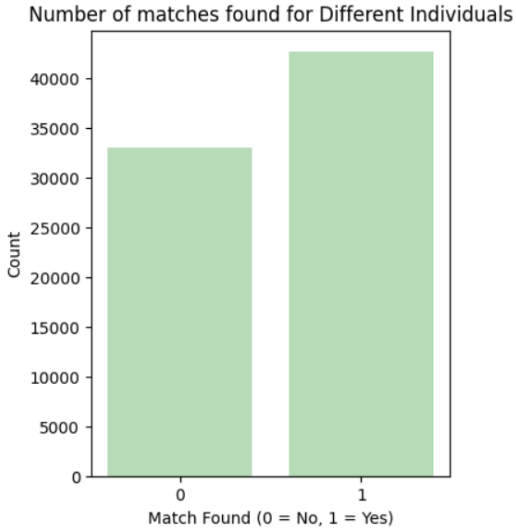

Figure 4.2: Count of the number of matches for Different, Aligned Biometric Samples with Single Index Hashing and Binary PostHash Indices

2. $d = 2$ and $m = 1$: Single-index outputs ($m = 1$) and Quaternary PostHash indices ($d = 2$).
   - $q_{same}^{obs} = 46.8\%$
   - $q_{diff}^{obs} = 30.6\%$

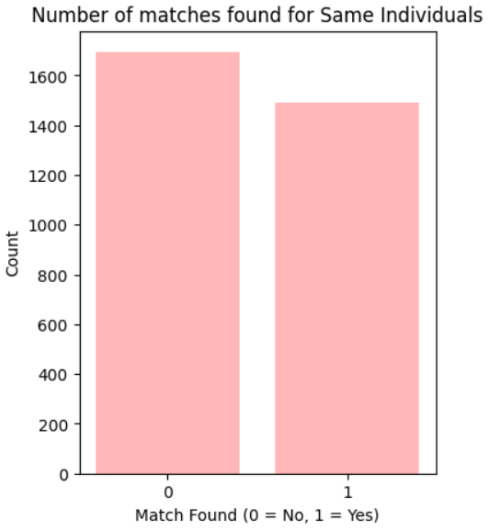

Figure 4.3: Count of the number of matches for Same, Aligned Biometric Samples with Single Index Hashing and Quaternary PostHash Indices

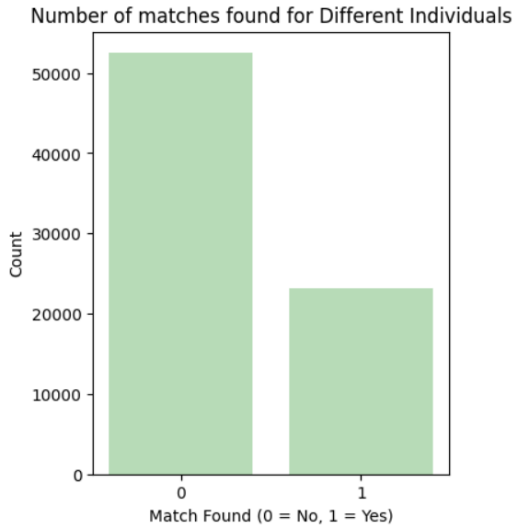

Figure 4.4: Count of the number of matches for Different, Aligned Biometric Samples with Single Index Hashing and Quaternary PostHash Indices

3. $d = 3$ and $m = 1$: Single-index outputs ($m = 1$) and Octal PostHash indices

$(d = 3)$.

- $q^{obs}_{same} = 34.8\%$
- $q^{obs}_{diff} = 18\%$

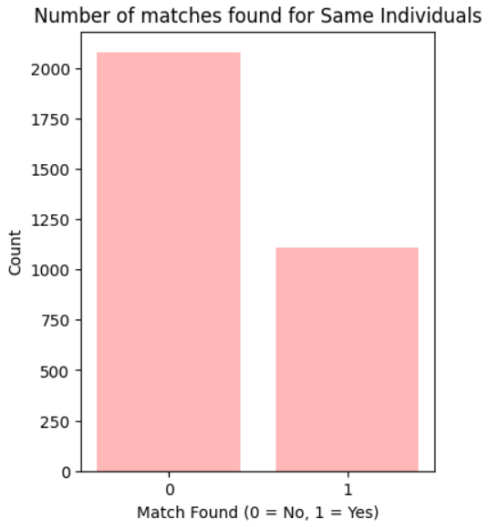

Figure 4.5: Count of the number of matches for Same, Aligned Biometric Samples with Single Index Hashing and Octal PostHash Indices

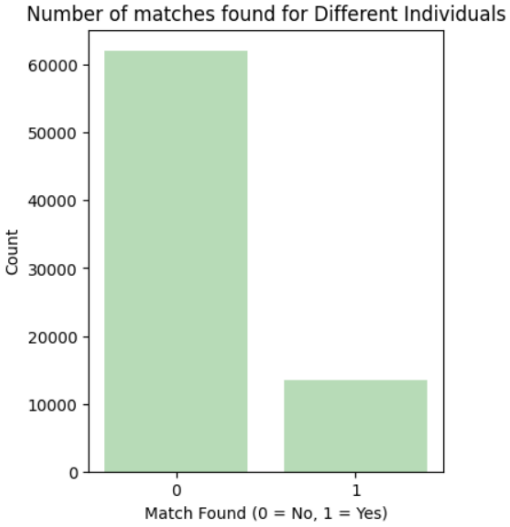

Figure 4.6: Count of the number of matches for Different, Aligned Biometric Samples with Single Index Hashing and Octal PostHash Indices

4. $d = 4$ and $m = 1$: Single-index outputs $(m = 1)$ and Hexa-Decimal PostHash indices $(d = 4)$.
   - $q^{obs}_{same} = 28.7\%$
   - $q^{obs}_{diff} = 11.8\%$

Figure 4.7: Count of the number of matches for Same, Aligned Biometric Samples with Single Index Hashing and Hexa-Decimal PostHash Indices

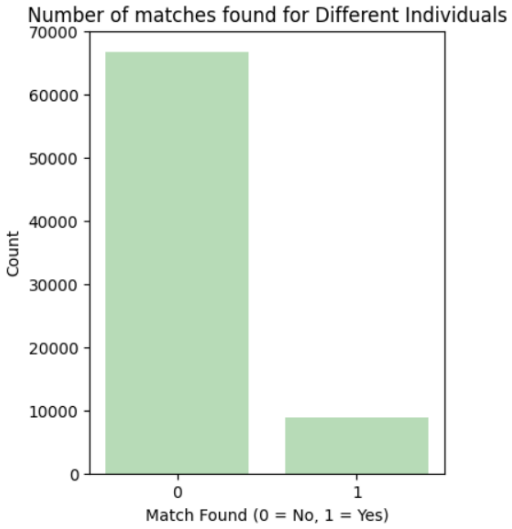

Figure 4.8: Count of the number of matches for Different, Aligned Biometric Samples with Single Index Hashing and Hexa-Decimal PostHash Indices

5 Application: Private and Compact Biometric Matching

This section delves into the practical application of fuzzy hashing within the realm of biometric matching. Employing the Hamming distance for biometric matching offers a systematic approach by iteratively generating l iterations of the *PreHash* function, defined as:

$$\begin{aligned} Hash^m_{\text{key}} &= Hash^m_{\text{key}_1,\ldots,\text{key}_l}(X) \\ &= (PreHash^m_{\text{key}_1}(X), \ldots, PreHash^m_{\text{key}_l}(X)) \end{aligned} \tag{5.1}$$

Subsequently, the Hamming distance between the resulting hash values of two biometric samples X and Y is calculated as:

$$d_H(Hash_{key}(X), Hash_{key}(Y)) = \#\{i : PreHash_{key_i}(X) \neq PreHash_{key_i}(Y)\} \tag{5.2}$$

This expression quantifies the instances "i" where the outputs of the *PreHash* function differ between samples X and Y.

One notable advantage of this approach is the reduction in size of the stored biometric template. Rather than storing n pixels, ml integers are stored. Additionally, the key renders the reference *PreHash* less privacy-sensitive compared to a biometric template. Specifically, if the key is known, each integer in the hash discloses about $\frac{1}{p}$ pixels, revealing $\frac{ml}{p}$ pixels at worst. This disclosure occurs because, with the keys, the hash values can potentially be reverse-engineered to reveal characteristics of the original biometric pattern. The term p reflects the information entropy associated with each bit being a vein ($'1'$) and thus quantifies the average information content each disclosed pixel conveys when the hash is decoded.

Similarly, when employing the Hamming distance for biometric matching through the iterative generation of l iterations of the *PostHash* function, analogous advantages arise. Here, the stored biometric template is condensed to mld integers instead of n pixels. Furthermore, the key diminishes the sensitivity of the reference *PostHash* in terms of privacy, exposing $\frac{mld}{p}$ pixels at most if known. Additionally, *PostHash* contributes to leakage reduction.

It's crucial to note that for both scenarios, additional privacy safeguards can be implemented, for instance a restricted access to the key. Hence, the intricacies of the biometric infrastructure must be addressed on a case-by-case basis.

5.1 Theoretical Foundations of FPR and FNR within Fuzzy Hashing Systems

Transforming biometric data into a hash, using methods like *PreHash* or its more compact version, *PostHash*, plays a key role in enhancing the system's efficiency

and security. This process helps protect privacy, which in turn influences important aspects of the system's performance, such as the False Negative Rate (FNR) and the False Positive Rate (FPR). By converting detailed biometric data into a simpler, hashed format, the system not only uses storage space more efficiently but also reduces the chances of unauthorized access to sensitive information. This transformation is crucial for maintaining the integrity of the data and provides a strong line of defense against potential security threats. Additionally, choosing the right techniques for generating these hashes can greatly improve the system's ability to distinguish between authorized and unauthorized users. This means fewer mistakes in the form of false rejections or acceptances, leading to a more accurate and dependable biometric verification process.

We establish a threshold t to evaluate the match between two biometric samples, X and Y, by analyzing the Hamming distance between their hash values. We define that a match is confirmed if the difference between l (the total iterations) and the Hamming distance is equal to or exceeds the threshold t, expressed as:

$$l - d_H(Hash_{\text{key}}(X), Hash_{\text{key}}(Y)) \geq t$$

In contrast, we define there being no match if the following equation holds:

$$l - d_H(Hash_{\text{key}}(X), Hash_{\text{key}}(Y)) < t$$

To further refine our understanding, we use statistical methods to approximate this comparison to a normal distribution, allowing us to more accurately calculate the False Negative Rate (FNR). This measure helps us assess how often the system incorrectly fails to recognize a match between the biometric samples when there actually is one. We define:

$$FNR = \Phi\left(\frac{t - l\mu^m_{\text{same}}}{\sqrt{l\mu^m_{\text{same}}}}\right) \tag{5.3}$$

Here, Φ denotes the Cumulative Distribution Function (CDF) of the standard normal distribution, denoted as $\mathcal{N}(0, 1)$.

In contrast, the False Positive Rate (FPR), the proportion of non-matching biometric samples incorrectly identified as matches by the system, is defined as:

$$FPR = \Phi\left(-\frac{t - l\mu^m_{\text{diff}}}{\sqrt{l\mu^m_{\text{diff}}}}\right) \tag{5.4}$$

These formulations allow for the evaluation of false match rates based on the standard deviation and mean of the distributions for same and different samples, respectively. The upper bound μ_{same} and μ_{diff} are defined in Equation 3.6.

For instance, employing $\Phi(-2.33) \approx 1\%$ as a benchmark, we calculate the threshold (t) from parameters m and l to achieve an FNR of 1% and an FPR $\leq 2^{-36}$. The theoretical resulting set of parameters is as follows:

m	l	t	$l\mu^m_{\text{same}}$	$l\mu^m_{\text{diff}}$	FNR	FPR
1	637	118	146.5	49	1.0%	2^{-74}
2	961	34	50.8	5.7	1.0%	2^{-106}
3	2569	18	31.3	1.2	1.0%	2^{-179}
4	8481	12	23.7	0.3	1.0%	2^{-377}
5	32 999	11	21.2	0.1	1.0%	2^{-967}
6	140 090	10	20.7	0.0	1.0%	$2^{-\infty}$
7	568 315	9	19.4	0.0	1.0%	$2^{-\infty}$
8	2 841 573	11	22.3	0.0	1.0%	$2^{-\infty}$
1	217	33	50	16.8	$\leq$1%	2^{-14}

Table 9: Theoretical Parameterization Results for FNR and FPR Calculation, using l Iterations of PreHash

Using compression techniques implemented through *PostHash*, specifically compressing to $D = 16$ ($d = 4$), we calculate the threshold (t) from parameters $m = 1$ and l to achieve an FNR of 1% and an FPR $\leq 2^{-36}$. The introduction of compression slightly modifies the equations for FNR (5.3) and FPR (5.4), necessitating the use of q in place of μ^m. The theoretical resulting set of parameters is as follows:

m	d	l	t	lq^m_{same}	lq^m_{diff}	FNR	FPR
1	4	1107	276	317.7	130.6	1.0%	2^{-121}
1	4	347	76	99.6	40.9	$\leq$1.0%	2^{-14}
1	1	3597	957	1032.3	424.4	$\leq$1.0%	2^{-488}

Table 10: Theoretical Parameterization Results for FNR and FPR Calculation, using l Iterations of PostHash

5.2 Experimental Derivation of the FNR and FPR for different (m, l) Parameter Configurations

To experimentally determine the False Negative Rate (FNR) and the False Positive Rate (FPR) for various (m, l) parameter configurations, the initial step is to compute the Hamming distance between $Hash_{\text{key}}(X)$ and $Hash_{\text{key}}(Y)$. The $Hash_{\text{key}}$ of an extracted feature vector is generated using l iterations of *PreHash*. We designed an algorithm that accepts a list of l keys (since each *PreHash* iteration requires a new key) and the parameter m, which indicates the number of indices each call to *PreHash* should generate per key. For testing purposes, the l keys are simply integers from 0 to $l-1$, converted into a two-byte binary format. We then calculate the Hamming distance between the resulting hash values for pairs of images. The Hamming distance between $Hash_{\text{key}}(X)$ and $Hash_{\text{key}}(Y)$ is defined as the number of positions at which the corresponding iterations of *PreHash* are different, as defined in Equation 5.2. In this formula, the expression $\#\{i : PreHash_{\text{key}_i}(X) \neq PreHash_{\text{key}_i}(Y)\}$

counts the number of iterations i where the pre-hash values of X and Y differ. Each comparison results in a 1 if the iterations differ and a 0 if they are the same. Summing these comparison results gives the total number of differing positions, thereby computing the Hamming distance between the hash values of the iterations.

Each experiment we conducted uses a different row from Table **??** or uses configuration **??**. In each experiment, we generate l *PreHash* iterations for all image pairs and then determine if two images, X and Y, are a match, defined as $l - d_H(Hash_{\text{key}}(X), Hash_{\text{key}}(Y)) \geq t$. We perform two types of comparisons on our data:

1. **Same Person, Same Finger Comparisons**: This involves comparing different trials of images from the same person and the same finger. These comparisons should theoretically result in a match. If two such images do not match (i.e., $l - d_H(Hash_{\text{key}}(X), Hash_{\text{key}}(Y)) < t$), this contributes to the FNR.
2. **Different People or Fingers Comparisons**: This involves comparing images from different people and/or different fingers. These comparisons should theoretically not result in a match. If two such images do match, this contributes to the FPR.

To calculate the False Negative Rate (FNR), we count the instances where images that should match do not, and then compute the mean of these instances. Similarly, for the False Positive Rate (FPR), we count the instances where images that should not match do, and then compute the mean of these instances.

For the experiments from Table **??**, we obtain the following results:

1. **m = 1, l = 217 and t = 33**
   - FPR:
   - FNR:
2. **m = 1, l = 637 and t = 118**
   - FPR:
   - FNR:
3. **m = 2, l = 961 and t = 34**
   - FPR:
   - FNR:
4. **m = 3, l = 2569 and t = 18**
   - FPR:
   - FNR:
5. **m = 4, l = 8481 and t = 12**
   - FPR:

- FNR:

6. **m = 5, l = 32'999 and t = 11**
   - FPR:
   - FNR:
7. **m = 6, l = 140'090 and t = 10**
   - FPR:
   - FNR:
8. **m = 7, l = 568'315 and t = 9**
   - FPR:
   - FNR:
9. **m = 8, l = 2'841'573 and t = 11**
   - FPR:
   - FNR:

Summary of the FNR and FPR results obtained, using l iterations of PreHash

m	l	t	$l\mu^m_{\text{same}}$	$l\mu^m_{\text{diff}}$	FNR	FPR
1	637	118	146.5	49	?%	?
2	961	34	50.8	5.7	?%	?
3	2569	18	31.3	1.2	?%	?
4	8481	12	23.7	0.3	?%	?
5	32 999	11	21.2	0.1	?%	?
6	140 090	10	20.7	0.0	?%	?
7	568 315	9	19.4	0.0	?%	?
8	2 841 573	11	22.3	0.0	?%	?

Table 11: Experimental Parameterization Results for FNR and FPR Calculation

5.3 Experimental Derivation of the FNR and FPR for different (m, l, d) Parameter Configurations with Compression

In this subsection, we will experimentally evaluate the False Negative Rate (FNR) and False Positive Rate (FPR) with compression. We designed an algorithm that uses the previously defined method to generate l iterations of *PreHash* and then compresses each hash value generated depending on the value of $D = 2^d$. For an image, the algorithm generates a list of the l iterations of *PreHash* and then applies the *PostHash* algorithm to each *PreHash* output.

After both images pass through this process, each iteration of each image is compared, resulting in a 1 if the iterations differ and a 0 if they are the same. Summing these comparison results gives the total number of ones, effectively computing the Hamming distance between the compressed and hashed images.

1. **d = 4, m = 1, l = 1107 and t = 276**
   - FPR:
   - FNR:
2. **d = 4, m = 1, l = 347 and t = 76**
   - FPR:
   - FNR:
3. **d = 1, m = 1, l = 3'597 and t = 957**
   - FPR:
   - FNR:

Summary of the FNR and FPR results obtained, using l iterations of PostHash

m	d	l	t	lq^m_{same}	lq^m_{diff}	*FNR*	*FPR*
1	4	1107	276	317.7	130.6	1.0%	2^{-121}
1	4	347	76	99.6	40.9	$\leq$1.0%	2^{-14}
1	1	3597	957	1032.3	424.4	$\leq$1.0%	2^{-488}

Table 12: Theoretical Parameterization Results for FNR and FPR Calculation, using l Iterations of PostHash

6 [1:N] Matching and System Evaluation

6.1 Implementation of [1:N] Matching

6.2 System Performance Evaluation

7 Conclusion

7.1 Future Directions and Enhancement

8 Definitions

Equal Error Rate (EER) Metric used to evaluate the performance of a system. It represents the point at which the system's false acceptance rate (FAR) equals its false rejection rate (FRR). A lower EER indicates a more accurate and reliable system as it signifies a balanced trade-off between security (minimizing FAR) and usability (minimizing FRR)

False Positive Rate (FPR) This is the probability of incorrectly accepting an unauthorized user

False Negative Rate (FNR) This is the the probability of incorrectly rejecting an authorized user

Hash Function A hash function is an algorithm that converts input data of any size to a smaller fixed-size string of characters, which typically acts as a data fingerprint. The output, known as a hash, is unique for different inputs in ideal cases, making hash functions crucial for cryptography, data integrity, and indexing in databases.

Fuzzy Extractors Fuzzy extractors are cryptographic tools designed to reliably and securely generate a consistent, reproducible cryptographic key from biometric data or other noisy inputs that are inherently inconsistent. They enable the extraction of a stable key from an input that may vary slightly over different measurements, ensuring that even with minor variations, the same key can be reliably regenerated. This process typically involves two main components: a generator that produces a stable key and some public data from an initial input, and a reproducer that can regenerate the original key from a similar but not identical input using the public data.

Uniform Distribution A uniform distribution signifies that each outcome within a set has an equal chance of occurring. In the context of finger vein patterns, it means any bit i in the biometric capture, representing either the presence or absence of a vein, is equally likely to be selected for analysis.

Hamming Distance The number of positions at which two biometric strings of equal length differ. It measures the similarity between the strings, with a lower distance indicating higher similarity.

Hamming Weight The number of 1's in a biometric string, indicating the presence of veins.

AES in CTR mode AES in Counter (CTR) mode is an encryption method that transforms a block cipher into a stream cipher. It achieves this by encrypting sequential counter values using AES. Each counter value is unique for each block of data, typically starting from an initial nonce (number used once) and incremented for subsequent blocks. This encryption method is traditionally used to encrypt data by generating a sequence of encrypted counters, which are then XORed with plaintext to produce ciphertext, allowing for encryption of arbitrary-sized data without padding. However, in the context of fuzzy hashing for biometric matching, AES-CTR is repurposed to generate a pseudorandom sequence, not for encrypting data but for selecting specific indices from a biometric template. This application leverages AES-CTR's cryptographic strength to ensure unpredictability and determinism in the selection process.

SHA-256 Hash SHA-256 is a cryptographic hash function in the SHA-2 family that produces a fixed-size 256-bit (32-byte) hash value from an input of arbitrary length. It has three fundamental properties: pre-image resistance, making it computationally infeasible to reverse the hash to find the original input; second pre-image resistance, preventing the discovery of a different input producing the same hash as a given input; and collision resistance, making it highly unlikely to find two different inputs that yield the same hash output. These properties ensure data integrity and security across various digital applications.

References

[1] asee. *History of Authentication: From Zero to Hero*. 2022. URL: https://cybersecurity.asee.io/blog/history-of-authentication/.

[2] LoginTC. *Biometric Authentication*. 2024. URL: https://www.logintc.com/types-of-authentication/biometric-authentication.

[3] Microsoft. *What is authentication?* 2024. URL: https://www.microsoft.com/en-us/security/business/security-101/what-is-authentication.