Improving the Efficiency of Capture-resistant Biometric Authentication based on Set Intersection

Xunhua Wang, Philip D. Huff, Brett C. Tjaden
Department of Computer Science
James Madison University
Harrisonburg, VA 22807 USA
{wangxx,huffpd,tjadenbc}@jmu.edu

Abstract

Traditional biometric authentication systems store biometric reference templates in cleartext on an authentication server, making them vulnerable to theft. Fuzzy extractors allow an authentication server to store biometric verification data that are resistant to capture. It is hard to recover the reference templates from these biometric verification data, thus increasing the privacy of the reference templates. In this paper, we improve the efficiency of a set intersection-based fuzzy extractor in two ways. First, we speed up the computation of verifying a biometric sample under some parameter combinations through integrating a Reed-Solomon decoding algorithm. Second, we propose a new function to improve the storage efficiency of the fuzzy extractor. A prototype implementation is developed to validate our improvements and it shows that our first improvement could speed up computation as many as \(2.29 \times 10^6\) times.

1 Introduction

Entity authentication is a fundamental issue in information security and has been studied extensively in the past thirty years [1, 17, 25]. Earlier authentication research focused on password-based authentication and cryptographic key-based authentication. These two authentications actually authenticate either a password (remembered by a user) or a secret (stored on a token), which can be either shared or transferred. Biometric authentication, on the other hand, verifies characteristics from human beings and has received more research attention recently [8, 13, 24]. Unlike passwords, the space of biometrics is usually big enough to resist brute-force attacks. Different from cryptographic keys, biometrics are more intrinsic to the user and are very hard, if not impossible, to be transferred.

In addition to these benefits, biometric authentication exhibits two distinguishing characteristics. First, most biological biometrics do not change much over a long period of time, making their revocations and system recovery after compromise very hard. Second, unlike passwords or cryptographic keys, the comparison of a given biometric sample against a stored biometric template is not exact. Two consecutive readings of the same biometric are usually close but not exactly the same. As a result, comparisons of biometric samples are often threshold-based and two biometric samples are considered the same when their similarity is larger than a threshold \(t\) [19]. Consequently, biometric authentications are affected by two types of errors: false match (FM), in which two different biometrics are incorrectly considered the same, and false non-match (FNM), in which two samples of the same biometric are incorrectly deemed different. The bigger the similarity threshold \(t\), the bigger the FNM rate (FNMR) and the smaller the FM rate (FMR). Different applications tolerate different FMR/FNMR and thus choose different \(t\).

Similar to passwords, biometrics are typically used for authenticating a client only (i.e., client-side authentication). The client first enrolls his biometric sample (called reference template), from which the authentication server generates and stores related biometric verification data (BVD). Given a cryptographic hash function \(h\), a reference template \(A\) and a fresh biometric sample \(B\) to be authenticated, even when \(B\) is close to \(A\), their cryptographic hashes \(h(A)\) and \(h(B)\) are very different, making \(h(A)\) not appropriate for BVD. These days, the common practice is to use the reference template \(A\) itself as BVD. However, storing biometric reference templates on a server in cleartext has negative security and privacy implications. If the server were compromised, all biometric templates stored on it would be revealed and it would be hard to recover from this break.

To address this security issue, the concept of fuzzy extractor was developed [11, 16]. Given a reference template \(A\), a fuzzy extractor generates a value \(U\) and a uniformly
random secret $s$. $U$ leaks little information about $A$ or $s$, and can be made public. For a fresh biometric sample $B$ that is sufficiently close to $A$, one can use $U$ and $B$ to reproduce $s$. Thus, $(U, h(s))$ can be used as capture-resistant BVD: if $(U, h(s))$ were stolen due to a server compromise, the attacker still could not recover $A$ or $s$.

Depending on how biometrics are represented, there are different metrics to measure the closeness of biometric samples, including Hamming distance, set difference, edit distance, and set intersection. These representational differences call for different designs of fuzzy extractor. Dodis et al. [11] described several fuzzy extractors based on the set difference metric, where biometric samples are represented as a set of points and the difference of two biometric samples $A$ and $B$ is half of their symmetric difference (that is, $|A - B) \cup (B - A)|$ where $||$ denotes set size and $\cup$ denotes set union). These set difference-based fuzzy extractors are built upon error-correcting codes such as the Reed-Solomon code [30, 2, 21] and employ a difference threshold $t_d$. $B$ is considered close to $A$ only when their set difference is not greater than $t_d$. Since an error-correcting code works only when there are more correct elements than errors, these fuzzy extractors have a fundamental limit on parameter $t_d$: $t_d$ can not be greater than half of the number of elements in $A$.

Because of this limit, for some biometric authentications, the set intersection metric is more appropriate. For example, in many fingerprint authentications, two fingerprints should be considered the same as long as they share at least 12 minutiae points, which is called the 12-point guideline [26, 20]. Under this guideline, when $|A| = 36$, a fingerprint $B$ of size 36 is considered the same as $A$ as long as the size of their intersection (i.e., their common fingerprint minutia points) is larger than a similarity threshold $t = 12$, making set intersection an ideal metric. Under the set difference metric, it would require an error-correcting code to correct 24 errors in a 36-element set; however, no such error-correcting code exists, as there are more errors (i.e., 24) than correct elements (i.e., 12).

Socck et al. [32] proposed a fuzzy extractor based on set intersection, which is called FESI hereafter. FESI is based on threshold secret sharing schemes [31], not error-correcting codes. In FESI, for a given reference template $A = \{a_1, a_2, \ldots, a_n\}$, where $n$ is an integer and $t \leq n$, a random secret $s$ is first chosen and its $t$-out-of-$n$ secret shares $(s_1, s_2, \ldots, s_n)$ are generated (see Section 3.1) [31]. Let $h$ be a cryptographic hash function and $f_A(x)$ be a discrete function such that $f_A(x) = s_i$ if $x = a_i$ and $f_A(x) = \hat{s}_i$ otherwise, where $\hat{s}_i$ is a random number (thus very likely $s_i \neq \hat{s}_i$). $\Gamma = (\mathcal{H}_A, y, F_A)$ is then stored on the server as $A$’s biometric verification data (BVD), where $\mathcal{H}_A = \{h(sa_1), h(sa_2), \ldots, h(sa_n)\}, y = h(s), F_A = f_A(x)$.

When a fresh biometric sample $B = \{b_1, b_2, \ldots, b_m\}$ (i.e., $|B| = m, t \leq m$) is presented, the server takes the following steps to verify its authenticity: for each $t$-subset $B_i$ of $B$, $1 \leq i \leq \binom{m}{t}$ where $\binom{m}{t}$ denotes the number of $t$-combinations of $m$, the server evaluates $f_A(B_i)$ to get $t$ values $\{\bar{s}_i^1, \bar{s}_i^2, \ldots, \bar{s}_i^t\}$, which are then used as shares to reconstruct a secret value $s_{B_i}$ [31]. Next, the server checks whether $h(s_{B_i}) = h$. If not, the next $B_i$ is tried (hereafter, the test of each $B_i$ is called a try); otherwise, the server calculates $\mathcal{H}_{B_i} = \{h(s_{B_i}b_1), h(s_{B_i}b_2), \ldots, h(s_{B_i}b_m)\}$ and $\Theta_{B_i} = \mathcal{H}_A \cap \mathcal{H}_{B_i}$, where $\cap$ denotes set intersection. If the cardinality of $\Theta_{B_i}, |\Theta_{B_i}|$, is not smaller than $t$, $B$ is considered close to $A$ and the client is authenticated. (After a successful authentication, the reconstructed secret $s$ can be used for other security purposes such as being used as an AES key.)

Let $\beta = |A \cap B|$, $\delta = \binom{\beta}{t}$, and $\lambda = \binom{A}{2}$. The expected number of tries to find a correct $B_i$ from $B$ is $e = \frac{\delta + 1}{A + 1}$ [32] (also see the Appendix of this paper). When $t$ is a small number (for example, $t = 12$), $e$ is not a big number. For example, when $t = 12, |B| = 80, \beta = 48$, we have $e = 865$.

The Problem However, some biometric authentication applications (such as the biometric authentication to a nuclear plant) tolerate little false matches and thus require a high threshold value $t$. For example, $(t = 40, |B| = 80, \beta = 60)$. In this case, $e$ would be a big number: $e = \frac{107507208733336176461620}{419184595805495} = 25646754.927203$, which is about $2.56 \times 10^5$.

Exhausting this number in the verification process would take an unreasonably long time. (Assume that each try takes 6.12 milliseconds, which is the amount of time to reconstruct a 128-bit secret with Shamir’s scheme on a 3.40 GHz Intel Pentium 4 processor running Linux 2.6.9 – 67.0.7, the FESI would cost 1.57 $\times 10^5$ seconds.)

Our Contributions In this paper, we improve FESI in two ways. The first improvement is computational and addresses the above problem, while the second one is related to storage efficiency.

First, we observe that in those biometric authentication applications that tolerate few false matches, biometrics typically have a high correct rate $\frac{\beta}{|B|}$, as these applications may employ more accurate biometric readers. Based on this observation, through integrating a Reed-Solomon decoding algorithm in the verification step, we can pick a much larger subset of $B$ in each try and as a result, significantly reduce the expected number of tries in verifying $B$. (We bring back error-correcting codes but for different purpose and in different ways.)

For the above example parameter combination $(t = 40, |B| = 80, \beta = 60)$, to verify a biometric sample $B$, the computational improvement requires only 1 try on average. This try takes just 0.0684 second on the aforesmen-
tioned 3.40 GHz Intel Pentium 4 processor.

Second, we develop a continuous function to replace the discrete function $f_A(x)$ of FEISI. This replacement saves storage space and makes FEISI more practical.

The remainder of this paper is organized as follows. In Section 2 we give related research. Section 3 describes the building blocks for our improvements, namely, Shamir secret sharing and Reed-Solomon decoding. In Section 4 we present our improvements and analyze their performances. Section 5 gives the details of a prototype implementation and its running results. Section 6 discusses some parameter selection issues. Concluding remarks are given in Section 7.

2 Related Work

Biometrics have been used for authentication and identification for decades. The Federal Bureau of Investigation (FBI) maintains a fingerprint database with 55 millions of records and is proposing to build a one-billion dollar computer database of biometrics including palm prints, scars, tattoos, iris patterns and facial shapes [33]. In addition to fingerprints, popular biometric identification/authentication techniques include speaker recognition [6] and facial authentication [34].

The risk of storing biometric information on a central server has been observed in several places [27, 28]. The problem has been brought out for privacy concerns as compromised biometrics lead to privacy breach.

Ratha et al. [29] developed the idea of cancelable biometrics and proposed to store distorted biometrics on a server for authentication. These constructions are typically restricted to a specific biometric type such as fingerprint.

2.1 Other fuzzy extractors

Juels and Wattenberg [16] described an elegant fuzzy extractor scheme under the Hamming distance metric, where a biometric sample is represented as a fixed-size and fixed-order binary string. This scheme, called JW99 hereafter, is based on error-correcting codes (ECC). To generate biometric verification data from a biometric sample $x$, the server first picks a random codeword $C$ and calculates $v = (x \oplus C)$ and $y = h(C)$, where $\oplus$ denotes bitwise exclusive OR and $h$ is a crypto hash function like SHA-192. $(v, y)$ is the biometric verification data and is stored on the server. To verify a given biometric sample $x'$, the server first calculates $u = v \oplus x'$ and then applies the decoding function of the error-correcting code on $u$. If $x$ and $x'$ are close enough under the Hamming distance metric, $u$ can be corrected to $C$, whose correctness can be verified by checking $h(u) \equiv y$. (The essence of this and other ECC-based schemes is to use the error-correcting capability of ECC to tolerate small differences between biometric samples.)

In the JW99 scheme, if the server were compromised and $(v, y)$ were stolen, since $C$ is randomly picked, an attacker would not be able to recover $x$ or a close biometric sample. This scheme is also secure against the multiple-use attack (called chosen perturbation attack in [4]): a client may use his biometrics in several applications with each server storing a set of $(v_i, y_i)$; compromising multiple such servers to obtain $(v_i, y_i)$ does not give the attacker more useful information about the client’s biometric $x$.

Although JW99 is conceptually simple, real-world biometric samples are rarely fixed-size and fixed-order binary string.

Juels and Sudan [14] developed a fuzzy vault scheme based on the set difference metric. Dodis et al. [11] improved it, developed the concepts of secure sketch and fuzzy extractor, and proposed schemes under the Hamming distance metric, the aforementioned set difference metric, and the edit distance metric, where biometric samples are represented as variable-size binary strings. In the edit distance metric, the distance of two biometric samples is the number of insertions/deletions needed to transform one to the other.

Other research along this line include [9, 10, 5].

3 Building blocks

Two building blocks, Shamir secret sharing and Reed-Solomon decoding, are used in our improvements. Shamir secret sharing is used in generating biometric verification data $\Gamma$ while Reed-Solomon decoding is used in verifying a fresh biometric sample $B$ against $\Gamma$.

3.1 Shamir secret sharing

Shamir secret sharing allows a secret $s$ to be divided into many $(n)$ shares [31]. A threshold number $t$ or more such shares can be used to reconstruct $s$, while any $(t - 1)$ or fewer shares cannot. Indeed, any $(t - 1)$ or fewer shares leak no information about $s$.

- In the Shamir secret share generation algorithm, $(t - 1)$ random elements, $r_1, r_2, \ldots, r_{t-1}$, are first picked from a finite field $F_q$, where $q$ is a prime, $q > \max(n, s)$. Let $\varphi(x) = s + r_1 x + r_2 x^2 + \ldots + r_{t-1} x^{t-1} \mod q$. (Thus, the secret $s$ is encoded as the free term of $\varphi(x)$.)

Next, shares $s_i$ are calculated as $s_i = \varphi(ID_i), 1 \leq i \leq n$, where $ID_i$ is the identity of $s_i$; for example, $ID_i = i$ for $1 \leq i \leq n$. This share generation step is denoted as $s \xrightarrow{(t,n)} (s_1, s_2, \ldots, s_n) \mod q$. 

142
Let \( \{ ID_{i1}, ID_{i2}, \ldots, ID_{id} \} \) be the identities of shares \((s'_{i1}, s'_{i2}, \ldots, s'_{id})\) that will participate in the reconstruction, where \( d \geq t; 1 \leq i_j \leq n \) for \( 1 \leq j \leq d \). For \( 1 \leq j \leq d \), define \( c_{i_j} = \prod_{1 \leq k \leq d, k \neq j} \frac{ID_{ik}}{ID_{ij} - ID_{ij}} \mod q \). The shared secret can be reconstructed as \( s' = \sum_{j=1}^{d} c_{i_j} \times s'_{i_j} \mod q \).

When all participating shares \((s'_{i1}, s'_{i2}, \ldots, s'_{id})\) are correct (that is, \( s'_{i_j} = s_{ij} \), for all \( 1 \leq j \leq d \)), it can be proved that \( s' = s \). This reconstruction step is denoted as \( s \equiv (s_{i1}, s_{i2}, \ldots, s_{id}) \mod q \).

### 3.2 Reed-Solomon decoding

In the Shamir secret reconstruction step, if one or more participating shares have an incorrect value, the reconstructed secret \( s' \) will not be the same as the original secret \( s \).

As noted by [23], Shamir secret sharing is closely related to the Reed-Solomon code [30] and the set of participating shares, \( \Omega = (s'_{i1}, s'_{i2}, \ldots, s'_{id}) \), can be viewed as a codeword. As a result, if there are some error shares but enough number of correct shares in \( \Omega \), the original secret \( s \) can still be reconstructed from \( \Omega \) with the Reed-Solomon decoding function.

Several Reed-Solomon decoding functions have been developed [2, 22]. For simplicity, we use the algorithm given in [12], which is rephrased for our application below. For all polynomials of the following algorithm, their coefficients are computed in \( F_q \).

1. Define \( g_0(x) = \prod_{j=1}^{d} (x - ID_{ij}) \)
2. Find through polynomial interpolation \( g_1(x) \) of degree \( \leq (d - 1) \) such that \( g_1(ID_{ij}) = s'_{ij} \) for \( 1 \leq j \leq d \).
3. Find a polynomial \( g(x) \) of degree \( \leq \frac{d-t}{2} \) through the extended Euclidean algorithm such that \( u(x) \times g_0(x) + v(x) \times g_1(x) = g(x) \).
4. Divide \( g(x) \) by \( v(x) \) to get \( g(x) = \varphi_1(x) \times v(x) + r(x) \). If \( r(x) \neq 0 \), output “decoding error.” Otherwise output \( s = \varphi_1(0) \).

When the number of error shares in \( \Omega \) is not larger than \( \left\lfloor \frac{(d-t)}{2} \right\rfloor \), the above algorithm will return the original \( s \).

The above Reed-Solomon decoding algorithm involves polynomial computations and thus is relatively more computation intensive than the Shamir secret reconstruction step. This is not unexpected as one needs to pay to correct those error shares.

### 4 Improvements

In this section, we shall present our computational and storage improvements. For simplicity, some notations are introduced first.

#### 4.1 Notation

Throughout this paper, \( \mathcal{A} \) is a set representing a biometric reference template, from which biometric verification data \( \Gamma \) is derived and stored on an authentication server. \( \Gamma \) often contains a strong secret \( s \) locked by \( \mathcal{A} \). \( \mathcal{B} \) represents a fresh biometric sample set to be verified by the authentication server against \( \Gamma \).

Given a set \( \mathcal{A}, |\mathcal{A}| \) denotes the number of elements in \( \mathcal{A} \), \( \mathcal{A} \cap \mathcal{B} \) denotes the intersection of sets \( \mathcal{A} \) and \( \mathcal{B} \). Unless state otherwise, \( \mathcal{A} \) always has \( n \) elements \( \{ a_1, a_2, \ldots, a_n \} \); \( \mathcal{B} \) always has \( m \) elements \( \{ b_1, b_2, \ldots, b_m \} \), where \( m \) and \( n \) are two integers.

Given two integers \( x_1 \) and \( x_2, x_1 \geq x_2, \left( \frac{x_1}{x_2} \right) \) denotes the number of \( x_2 \)-combinations out of \( x_1 \). \( \min(x_1, x_2) \) returns the minimum of \( x_1 \) and \( x_2 \).

Given a real number \( z \), \( \lfloor z \rfloor \) denotes the smallest integer not less than \( z \); \( |z| \) denotes the greatest integer not larger than \( z \).

\( q \) represents a prime number larger than \( s \) and \( |q| \) denotes the binary length of \( q \). \( \varphi \) always represents a polynomial of degree at most \( (t - 1) \) for Shamir secret share generation (see Section 3.1). \( h \) represents a cryptographic hash function like \( \text{SHA-512} \).

#### 4.2 Computational improvement: reducing the number of tries

The computational improvement does not require any changes to the BVD \( \Gamma \) of the FESTI scheme. That is, \( \Gamma \) for \( \mathcal{A} = \{ a_1, a_2, \ldots, a_n \} \) remains as \( \langle \mathcal{H}_{\mathcal{A}}, y, F_{\mathcal{A}} \rangle \), where \( \mathcal{H}_{\mathcal{A}} = \{ h(sa_1), h(sa_2), \ldots, h(sa_n) \} \), \( y = h(s), F_{\mathcal{A}} = f_{\mathcal{A}}(x) \), \( s \) is the secret committed by \( \mathcal{A} \) and it can be either a 128-bit AES key or a 1024-bit RSA private key (see Section 6). Let \( \{ s_1, s_2, \ldots, s_n \} \) be a \( t \)-out-of-\( n \) secret sharing of \( s \), \( s \rightarrow_{(t,n)} \{ s_1, s_2, \ldots, s_n \} \).

\( f_{\mathcal{A}}(x) \) is a discrete function, \( f_{\mathcal{A}}(x) = s_i \) if \( x = a_i \) and \( \hat{s}_i \) otherwise, where \( \hat{s}_i \) is a random number (thus very likely \( s_i \neq \hat{s}_i \)).

The computational improvement lies in how a fresh biometric sample \( \mathcal{B} = \{ b_1, b_2, \ldots, b_m \} \) (i.e., \( |\mathcal{B}| = m; t \leq m \)) is verified. We introduce an extra integer parameter \( \sigma, t \leq \sigma \leq m \). How \( \sigma \) is chosen will be discussed shortly.

The server takes the following steps to verify \( \mathcal{B} \): for each \( \sigma \)-subset \( \mathcal{B}_i \) of \( \mathcal{B} \), \( 1 \leq i \leq \binom{m}{\sigma} \), the server
1. evaluates \( f_A(B_i) \) to get \( \sigma \) values \( \{s_1^\sigma, s_2^\sigma, \ldots, s_t^\sigma\} \), which are then used by the Reed-Solomon decoding algorithm of Section 3.2 to reconstruct a value \( s_{B_i} \).

If the decoding algorithm returns an error, the next \( B_i \) is tried.

2. When the decoding algorithm does return a value, the server then proceeds to check whether \( h(s_{B_i}) = y \). If not, the next \( B_i \) is tried.

If \( h(s_{B_i}) = y \) does hold, the server calculates
\[
\mathcal{H}_{B_i} = \{h(s_{B_i}b_1), h(s_{B_i}b_2), \ldots, h(s_{B_i}b_m)\}
\]
and \( \Theta_{B_i} = \mathcal{H}_A \cap \mathcal{H}_{B_i} \).

If \( |\Theta_{B_i}| \geq t \), \( B \) is considered close to \( A \) and the verification stops. Otherwise, the next \( B_i \) is tried.

In the worst case, the above algorithm will run \( \binom{m}{\alpha} \) tries. However, as shown below, its expected performance is much better under certain conditions.

### 4.2.1 The selection of \( \sigma \) and performance analysis

In the above algorithm, for the Reed-Solomon decoding algorithm to return the correct secret \( s \), \( \sigma \geq t \) must hold and the number of correct shares in \( B_i \) should not be smaller than \( \alpha = (t + \lceil \frac{\sigma - 1}{\alpha} \rceil) \).

Let \( \beta \) be the number of correct elements in \( B \) (i.e., \( \beta = |A \cap B| \)). Define \( \delta'_{\sigma} = \binom{m}{\sigma} \tau \), where \( \tau = \min(\beta - \alpha, \sigma - \alpha) \), and \( \lambda'_{\sigma} = \sum_{i=0}^{\tau} \binom{\beta}{i} \times \binom{m-\beta}{\sigma-i} \). \( \delta'_{\sigma} \) is the number of ways to pick a \( \sigma \)-subset from \( B \). \( \lambda'_{\sigma} \) is the number of ways to pick a \( \sigma \)-subset from \( B \) such that the subset has at least \( \alpha \) correct elements.

**Proposition 1** The performance of our improved algorithm can be summarized as follows:

1. If \( \beta < \alpha \), no \( B_i \) can be used to recover \( s \) and thus \( \binom{m}{\alpha} \) tries are required.

2. If \( \alpha \leq \beta \leq (m - \lceil \frac{\sigma - 1}{\alpha} \rceil) \), the expected number of tries of our improved algorithm select a \( B_i \) of size \( \sigma \) to recover \( s \) and pass the check is \( e'_{\sigma} = \frac{\delta'_{\sigma} + 1}{\lambda'_{\sigma} + 1} \).

3. If \( \beta > (m - \lceil \frac{\sigma - 1}{\alpha} \rceil) \), only one try is needed.

The first case is straightforward as there are no appropriate \( B_i \) and the algorithm will brute-force all \( \binom{m}{\alpha} \) combinations. In the third case, there are so many correct elements in \( B \) such that any \( B_i \) can be used to recover \( s \) and thus the algorithm needs only one try.

The proof of the second case can be found in the Appendix of this paper.

Our goal is to choose \( \sigma \) to make \( e'_{\sigma} \ll e \) of the FESI scheme. Given \( t, \beta \), and \( |B| \), one can brute-force to find the best \( e'_{\sigma} \) and the corresponding \( \sigma \).

### 4.2.2 Examples

To compare the number of tries in FESI and the computational improvement, we choose several parameter combinations and calculate their numbers of tries as Table 1. (See Section 6.1 for discussions on value \( \beta \).)

In Table 1, the improvement ratio is the ratio of the number of tries in FESI to the number of tries in the computational improvement. The bigger the improvement ratio, the more efficient our improvement is.

For example, when \( \langle |B| = 80, t = 40 \rangle, \beta = 0.75 \rangle \), our improvement has \( \langle \sigma = 80, \alpha = 60 \rangle \) and needs only one try on average while FESI requires 25646754.93 tries. Similarly, when \( \langle |B| = 80, t = 35, \beta = 0.725 \rangle \), our improvement needs only one try on average (with \( \langle \sigma = 79, \alpha = 57 \rangle \)) while FESI requires 6580194.07 tries.

Thus, when \( \beta \) is reasonably large, which is the case of many biometric authentications, our improvement needs much fewer tries and is much more efficient.

### 4.3 Storage improvement: a new \( f_A(x) \)

As described in Section 1, in FESI [32], \( f_A(x) \) is defined as a discrete function and it returns \( s_i \) when \( x = a_i \).

\( f_A(x) \) returns a random \( \hat{s}_i \) when \( x \neq a_i \) (\( s_i \) are secret shares generated from secret \( s \) with Shamir secret share function \( \varphi \)). Details are missing on how this discrete function is stored in FESI.

Both \( a_i \) and \( s_i \) should be kept secret. One natural way to hide \( \langle a_i, s_i \rangle \) is to create some chaff points \( \langle \hat{a}_j, \hat{s}_j \rangle \), where \( \hat{a}_j \neq a_i, \hat{s}_j \neq s_i \), and store them together [14, 15]. (Note that \( a_i, s_i \) are points on \( \varphi \) while \( a_j, s_j \) are not.) In this way, when the server is compromised, the attacker will still not be able to find \( a_i \) or \( s_i \).

According to Lemma 4 of [15], for \( 0 < \mu < 1 \), with probability \( (1 - \mu) \), \( \zeta \) chaff points will result in at least
\[
\eta = \mu^{(\zeta + |A|)} q^{(\zeta - (|A| + \zeta))(q - 1)} \text{ polynomials to hide } \varphi.
\]

The bigger \( \eta \), the better that \( \varphi \) is hidden by the chaff points and thus the better the secret embedded in \( \varphi \) is hidden. It is worth noting that this security is information-theoretic and unconditional in that its security does not depend on an attacker’s computational power.

With fixed parameters \( \langle \mu, q, t, |A| \rangle \), increasing \( \zeta \) leads to bigger \( \eta \) and thus higher security. However, bigger \( \zeta \) requires bigger storage space for the chaff points. Thus, a balance point must be found between high-level security and efficient storage.

The size of \( q \) plays a significant role on the size of \( \zeta \) for \( \eta \) with reasonable security. For example, when \( q \) is a 129-bit prime (which is needed to share a 128-bit secret such as an AES-128 key), for parameters \( \langle |A| = 36, t = 12, \mu = 2^{-43} \rangle \), to have 80-bit security, we will need more than \( 5 \times 10^{12} \) chaff points. This huge number of chaff points makes...
the chaff-point scheme impractical. (Similarly, when $q$ is a 120-bit prime, for parameters $(|A| = 80, t = 30, \mu = 2^{-43})$, to have 80-bit security, we will need far more than $5 \times 10^{10}$ chaff points.)

Fortunately, the information-theoretic nature of the chaff-point scheme allows us to reduce the number of chaff points significantly by having small $q$. Given a 128-bit secret $s$, we can first split it into small pieces, for example, each with at most 12 bits. For each such piece, we can then choose a 12-bit prime $q_i$. When applying this smaller $q_i$ to the chaff-point scheme, we need much smaller number of chaff points $c_i$. For example, when $q_i$ is a 12-bit prime, for parameters $(|A| = 36, t = 12, \mu = 2^{-43})$, to have 12-bit security, we will need just about 10700 chaff points. Together, for a 128-bit secret, we will need at most $10700 \times 11 = 117700$ chaff points.

This is far better but still uses much storage space. One natural improvement over this discrete function is to find and store a continuous function, just as done in [11]. Following this idea, we propose the following improvement on $f_A(x)$ for better storage efficiency.

Unlike the computational improvement of Section 4.2, the storage efficiency improvement requires a minor change in $\Gamma$. We assume that a constant $\omega$ exists such that it will not appear in any biometric sample set. For example, $\omega$ can be a big number in $F_q$.

To generate $\Gamma$ for a given $A = \{a_1, a_2, \ldots, a_n\}$, in the step of generating shares for secret $s$, we do not embed $s$ in the traditional way as the free term of $\varphi(x)$ (see Section 3.1). Instead, after picking random values $r_1, r_2, \ldots, r_{t-1}$, we calculate $r_0$ such that, for $\varphi'(x) = r_0 + r_1x + r_2x^2 + \ldots + r_{t-1}x^{t-1} \mod q, \varphi'(\omega) = s$. That is, we embed $s$ as the value corresponding to $\omega$. Next, shares are generated as $s_i = \varphi'(ID_i)$ and $f_A(x)$ is constructed as follows.

- Find a polynomial $\pi(x) = x^n + c_{n-1}x^{n-1} + \ldots + c_1x + c_0 \mod q$ such that $\pi(a_i) = s_i, 1 \leq i \leq n$. The values of $c_{n-1}, c_{n-2}, \ldots, c_2, c_1, c_0$ can be found by solving the $n$ equations.

We then define $f_A(x) = \pi(x)$. In this improvement, the storage of $f_A(x)$, as part of $\Gamma$, is equivalent to the storage of $n$ values $(c_{n-1}, c_{n-2}, \ldots, c_2, c_1, c_0)$, which for typical cases like $n = 80$ needs far less storage space than 117700 chaff points. This improvement can be as high as $\frac{117700}{80} = 2942$ times. The generations of the other parts of $\Gamma$ remain the same as $FESI$.

Similar to [11], $f_A(x)$ constructed this way does not leak $s_i$ or $s$. An attacker who has compromised the server and stolen $f_A(x)$ needs to solve the noise interpolation problem to recover $s_i$, which is considered hard [3].

Under this improvement, the process to verify a given biometric sample $B$ needs to be changed accordingly. In step 4 of the Reed-Solomon decoding algorithm of Section 3.2, $s$ should be computed as $s = \varphi_1(\omega)$.

It should be noted that our storage improvement does not negatively impact the computational improvement of Section 4.2 and they coexist very well.

## 5 Prototype Implementation

Our computational improvement reduces the number of tries but increases the computation time of a single try. To compare the overall computing time of $FESI$ and our improvement, we developed a prototype implementation.

### 5.1 Implementation details

In $FESI$, to verify a given biometric sample $B = (b_1, b_2, \ldots, b_n)$, in each try the server picks one $t$-subset $B_i$ and checks whether $s$ can be recovered and whether $|H_A \cap H_{B_i}|$ is larger than or equal to $t$. The most computation-intensive step of such a try is the reconstruction of $s_{B_i}$ from $\{s_i, s_i^1, \ldots, s_i^t\}$, which are obtained by evaluating $f_A(B_i)$. We implemented this step as a $t$-out-of-$n$ Shamir secret reconstruction [31]. (Correspondingly, the $t$-out-of-$n$ Shamir share generation is used in generating the biometric verification data.) We implemented both the Shamir share generation and Shamir secret reconstruction in C++, using the computational number library

| $|B|$ | $t$ | $\beta (|B|)$ | Number of Tries | Improvement Ratio |
|------|----|-------------|----------------|------------------|
| 80   | 40 | 60 (0.75)   | 25646754.93    | 1 (60, 80)       | 25646754.93      |
| 80   | 35 | 58 (0.725)  | 6580194.07     | 1 (57, 79)       | 6580194.07       |
| 80   | 30 | 55 (0.6875) | 2874867.40     | 1 (55, 80)       | 2874867.40       |
| 60   | 40 | 50 (0.83)   | 408073.5       | 1 (50, 60)       | 408073.5         |
| 80   | 25 | 53 (0.6625) | 402034.73      | 1 (52, 79)       | 402034.73        |
| 60   | 30 | 45 (0.75)   | 342927.67      | 1 (45, 60)       | 342927.67        |
| 60   | 35 | 48 (0.8)    | 269091.95      | 1 (47, 59)       | 269091.95        |
| 60   | 25 | 43 (0.717)  | 85336.84       | 1 (42, 59)       | 85336.84         |

Table 1. Number of tries in $FESI$ and our improvement
Improvement provides the subroutines for manipulating $(0.6875) \times \times$ and number of tries and the running time of a single secret reconstruction. When $\sigma$ is the product of the $s$ is a constant. In reality, for a specific biometric application, $\beta$ can be obtained by statistics on moderate number of biometric samples in that application. The value of $\beta$ affects the values of $\sigma$ and $c'_r$. Figure 1 describes the impact of $\beta$ for parameter combination ($t = 40, |A| = 80$). (It should be noted that the y-axis is in the log scale.)

![Figure 1. The impact of $\beta$ for ($t = 40, |B| = 80$)](image)

From Figure 1, we can see that our computational improvement outperforms FESI on most $\beta$ values: when $\beta \geq 60$, our scheme needs one try on average; when $50 \leq \beta < 60$, our scheme takes more than 1 try but still needs far fewer tries than FESI. In a few extreme cases

| $|B|$ | $t$ | $\beta$ ($\frac{\beta}{|B|}$) | Computation Time (seconds) | Improvement Ratio |
|---|---|---|---|---|
| 80 | 40 | 60 (0.75) | $25646754.93 \times 0.00612 = 156958.14$ | 0.0684 | $2.29 \times 10^6$ |
| 80 | 35 | 58 (0.725) | $6580194.07 \times 0.00458 = 30137.29$ | 0.0661 | $4.56 \times 10^3$ |
| 80 | 30 | 55 (0.6875) | $2874867.40 \times 0.00344 = 9889.54$ | 0.0682 | $1.45 \times 10^3$ |
| 60 | 40 | 50 (0.83) | $408073.5 \times 0.00611 = 2493.33$ | 0.0349 | $7.14 \times 10^2$ |
| 80 | 25 | 53 (0.6625) | $402034.73 \times 0.00231 = 928.70$ | 0.0667 | $1.39 \times 10^2$ |
| 60 | 30 | 45 (0.75) | $342927.67 \times 0.00343 = 1176.24$ | 0.0354 | $3.32 \times 10^1$ |
| 60 | 35 | 48 (0.8) | $269091.95 \times 0.00476 = 1280.88$ | 0.0335 | $3.82 \times 10^1$ |
| 60 | 25 | 43 (0.717) | $85336.84 \times 0.00237 = 202.25$ | 0.0348 | $5.81 \times 10^3$ |

Table 2. Computation time for representative parameters

LiDIA [18]. We chose $s$ as a 128-bit random secret and $q$ is a 129-bit prime number (in hexadecimal format, it is $0x12:cb639444cfb09183261b9def68e68b$).

To measure the computation time of Shamir secret reconstruction, we ran the reconstruction method a thousand times and calculated their average.

In our improvement, in each try, the most computation-intensive step is the reconstruction of $s$ through the Reed-Solomon decoding algorithm. We implemented the Reed-Solomon decoding algorithm of Section 3.2 with the LiDIA package. LiDIA provides the subroutines for manipulating polynomial functions over a finite field $F_q$ needed by the Reed-Solomon decoding algorithm.

To benchmark the Reed-Solomon decoding algorithm, we first generated $n$ secret shares from $s$ with Shamir secret share generation. Next, we randomly picked $(1 - \frac{\beta}{|B|}) \times |B|$ secret shares and modified their values. (As a result, these shares have incorrect values.) Following the improvement scheme, we then randomly picked $\sigma$ shares and sent them to the Reed-Solomon decoding algorithm. This subroutine was run 100 times and its average was calculated.

5.2 Results

We ran both implementations on a computer with a 3.40 GHz Intel 4 processor and 1 G bytes of memory to get their computation times. Table 2 gives the computation time for FESI and our improvement for some representative parameters. The computation time of FESI is the product of the *expected* number of tries and the running time of a single Shamir secret reconstruction. The computation time of our improvement is the product of the improved expected number of tries and the running time of a single secret reconstruction with the Reed-Solomon decoding algorithm.

The improvement ratio is the ratio of FESI’s computation time to the computation time of our improvement.

From Table 2, we can see that for these representative parameters, our improvement significantly enhances the performance.

6 Further Discussions

We shall discuss several related issues in this section.

6.1 The impact of $\beta$

In Table 1, we gave some example parameter combinations where $\beta$ is a constant. In reality, for a specific biometric application, $\beta$ can be obtained by statistics on moderate number of biometric samples in that application.

The value of $\beta$ affects the values of $\sigma$ and $c'_r$. Figure 1 describes the impact of $\beta$ for parameter combination ($t = 40, |A| = 80$). (It should be noted that the y-axis is in the log scale.)

![Figure 1. The impact of $\beta$ for ($t = 40, |B| = 80$)](image)
(such as $\beta = t$ and $\beta = |\mathcal{B}|$), our computational improvement needs the same number of tries as FESI.

It is worth noting that our computational improvement over FESI uses error-correcting codes but in a way different from the set difference-based fuzzy extractors. Those set difference-based fuzzy extractors focus on the number of errors and use all elements of $\mathcal{B}$ in the authentication step. In contrast, our computational improvement is based on the set intersection metric and in the authentication step, it uses just a subset of $\mathcal{B}$. Instead of depending on the specific number of errors in $\mathcal{B}$, our first improvement works best when the percentage of correct elements $\beta_{|\mathcal{B}|}$ is sufficiently large.

6.2 Two threshold verifications in FESI

In FESI [7], there are two checks related to threshold $t$:
- the check of $h(s_{\mathcal{B}_i}) \approx y$ and the check of $|\Theta(\mathcal{B}_i)| \geq t$.

First, when all the elements of a $t$-subset $\mathcal{B}_i$ are correct, the reconstructed value $s_{\mathcal{B}_i}$ will be equal to $s$ and the check of $h(s_{\mathcal{B}_i}) \approx y$ will succeed. This threshold check is probabilistic and if $\mathcal{B}_i$ contains an error element, depending on the design of $f_A(x)$, $s_{\mathcal{B}_i}$ will be different from $s$ with very high probability. Second, after $\mathcal{B}_i$ passes the first check, $H(\mathcal{B}_i)$ and $\Theta(\mathcal{B}_i)$ are calculated and $|\Theta(\mathcal{B}_i)| \geq t$ is checked. This is an exact check.

If the design of $f_A(x)$ can guarantee that any $t$-subset $\mathcal{B}_i$ with one or more wrong elements will result in $s_{\mathcal{B}_i} \neq s$, the second check will be unnecessary.

6.3 Bigger secrets

In our prototype implementation, $s$ is a 128-bit secret. For those applications where $s$ is a large number (for example, a 1024-bit or 2048-bit RSA private key), it is not desirable to directly apply such $s$ to the scheme, as it would slow down the computations of Shamir share generation, secret reconstruction, and the Reed-Solomon decoding algorithm.

For those cases, we can first generate a 128-bit random value $s'$ and encrypt $s$ as $c = E_{s'}(s)$, where $E$ is AES-128. Next, we apply the 128-bit $s'$ to the improved fuzzy extractor. $c$ should be stored as part of the biometric verification data.

When a close biometric sample $\mathcal{B}$ is presented and checked, $s'$ is first reconstructed with the improved fuzzy extractor and can be used to decrypt $c$ to recover $s$.

6.4 Other parameters and future work

In this paper, we developed a method to improve the computation of FESI for some $(t, |\mathcal{B}|, \beta)$ combinations.

It is worth noting that there exist $(t, |\mathcal{B}|, \beta)$ combinations, where both the original FESI scheme and our computational improvement are rather slow. For example, as shown in Figure 1, when $(t, |\mathcal{B}|, \beta) = (40, 80, 42)$, the FESI requires $1.247 \times 10^{20}$ tries and the computational improvement requires $6.095 \times 10^{19}$ tries with $\sigma = 42$, both of which are computation prohibitive. The improvement of these situations constitutes our future research.

7 Conclusions

Fuzzy extractor schemes allow a biometric authentication server to store biometric verification data resistant to capture, thus improving the privacy of biometrics.

In this paper, we improve the performance of FESI, a fuzzy extractor based on the set intersection metric, in two ways. First, we speed up the verification step of this fuzzy extractor under some parameter combinations, which is especially important for those biometric authentications that tolerate fewer false matches. This improvement is accomplished through the integration of a Reed-Solomon decoding algorithm. Our prototype implementation has shown that this improvement speeds up biometric verification as much as $2.29 \times 10^6$ times.

Secondly, by introducing a new continuous function $f_A(x)$, we increase the fuzzy extractor’s storage efficiency. Compared to password authentication, biometric authentications are secure against brute-force attacks and can strengthen the security of web authentication. The development of fuzzy extractor schemes, including the improvements of this paper, enhances the privacy of biometrics and thus will speed up the adoption of biometric authentication on the web.

Acknowledgement

The authors wish to thank the anonymous reviewers for their constructive comments, Christopher H. Howe for contributing his C++ implementation of the Reed-Solomon decoding algorithm, and Jennifer Leahy for reviewing the earlier draft of this paper.

References

In the FEST, the expected number of tries to find a correct $B_i$ from $B$ is $e = \frac{\delta + 1}{\lambda + 1}$ [32], where $\beta = |A \cap B|$, $\delta = |B'|$, and $\lambda = \binom{n}{\lambda}$.

In our first improvement, the expected number of tries to select a $B_i$ to recover $s$ and pass the check is $e' = \frac{\delta + 1}{\lambda + 1}$, where $\beta = |A \cap B|$, $\alpha = t + \left\lfloor \frac{\sigma - 1}{2} \right\rfloor$, $\delta' = \binom{B'}{\alpha}$, $\tau = \min(\beta - \alpha, \sigma - \alpha)$, and $\lambda' = \sum_{i=0}^{\tau} \binom{\beta}{\alpha+i} \times \binom{B'-\beta}{\sigma-\alpha-i}$.

Both formulae can be derived from the following fact: given an urn with $c$ red balls and $b$ black balls (the total number of balls is $(c + b)$), if we randomly choose balls from it without replacement, the expected number of tries until we get a red ball is $\frac{c+b+1}{b+1}$. The reasoning is as follows: let $E(i, j)$ be the expected number of tries to get a red ball when the urn has $i$ red balls and $j$ black balls. If we condition on the outcome of the first draw, we have
\[ E(c, b) = 1 \times \frac{c}{c+b} + (1 + E(c, b - 1)) \times \frac{b}{c+b}. \] Since \( E(c, 0) = 1 \), we get \( E(c, b) = \frac{(c+b)+1}{c+b} \).

For FESI, the number of red balls corresponds to the number of ways to pick a correct \( t \)-subset, which is \( \lambda = \binom{t}{\alpha} \), and the total number of balls corresponds to the number of ways to pick \( t \)-subset from \( B \), which is \( \delta = \binom{|B|}{t} \). Thus, we have \( e = \frac{\delta + 1}{\lambda + 1} \).

For our improved scheme, the number of red balls corresponds to the number of ways to pick a \( \sigma \)-subset that contains at least \( \alpha \) correct elements, which is \( \lambda'_\sigma = \sum_{i=0}^{\sigma} \binom{\beta}{\alpha + i} \times \binom{|B| - \beta}{\sigma - \alpha - i} \). The total number of balls corresponds to the number of ways to pick a \( \sigma \)-subset from \( B \), which is \( \delta'_\sigma = \binom{|B|}{\sigma} \). Thus, we have \( e'_\sigma = \frac{\delta'_\sigma + 1}{\lambda'_\sigma + 1} \).