A Multiple Server Scheme for Fingerprint Fuzzy Vaults

Jesse Hartloff[a]
hartloff@buffalo.edu

Matthew Morse[a]
mjmorse@buffalo.edu

Bingsheng Zhang[b]
b.zhang2009@gmail.com

Thomas Effland[a]
tom.effland@gmail.com

Jennifer Cordaro[a]
jacordar@buffalo.edu

Jim Schuler[a]
jcschule@buffalo.edu

Sergey Tulyakov[a]
tulyakov@buffalo.edu

Atri Rudra[a]
atri@buffalo.edu

Venu Govindaraju[a]
govind@buffalo.edu

[a]University at Buffalo, SUNY
Buffalo, NY, USA

[b]National and Kapodistrian University of Athens
Athens, Greece

Abstract

In this work, we present a multiple server fingerprint verification scheme that provides enhanced template security by eliminating several known vulnerabilities of the fuzzy vault scheme. We secure templates from adversarial attacks in honest-but-curious server scenarios by utilizing commutative encryption in which the raw fingerprint template is never used in matching or storage.

In this system, there is a matching server that performs the enrollment and matching functions on fingerprint data that has been encrypted by a separate encryption server. Since the encrypted template is stored at one server and the encryption key is on another server, an attacker would have to compromise both servers to decrypt the data. Even in this case, the templates are protected by the fuzzy vault scheme. Thus, this scheme limits an attacker's ability to attack active users even after compromising both servers providing multiple layers of template security.

1. Introduction

As the use of fingerprints as a means to verify identity increases, so does the importance of securing stored fingerprint templates. To enable matching, many schemes require storage of raw fingerprint templates, leaving them exposed to attackers who can compromise the database. Conventional measures such as firewalls and encryption may be suitable to prevent attacks in some applications, such as government uses. However, large-scale commercial deployment of fingerprint recognition schemes may not be realized until matching can be performed accurately without exposing the raw fingerprint data to anyone other than the user. This work makes progress towards this end.

We present a one-factor system that operates with an encryption server and a matching server in such a way that neither server has access to unencrypted template data as described in Section 5. While we present this as a two server system, each server can be naturally extended to be distributed over multiple servers for large scale applications. The encryption server stores and applies the user-specific encryption keys and the matching server stores a hash of the encrypted template using the fuzzy vault scheme [11]. We use fuzzy vault since the space we draw template points from is too small to prevent a brute force attack on a straightforward hash function. Since the template points are encrypted with a random key before constructing the vault, we avoid a common attack on the fuzzy vault where two vaults locked by the same template can be compared to find the genuine points [24]. We use n-gons as fingerprint features as presented in [18]. Most attacks on our system are ineffective due to the encryption of the templates, though even if both servers are compromised, the stored templates are still protected by the fuzzy vaults.

We achieve a ZeroFAR (FRR when the FAR is zero) of 12.6% on FVC2002-DB2 while a popular fuzzy vault scheme by Nandakumar et al. reports a ZeroFAR of 14% on the same dataset [19].

2. Related Work

While most proposed methods for secure storage of biometric templates assume that the templates are represented by fixed length binary or float value vectors, fuzzy vault [11] has been designed as a method to store biometric templates

represented as unordered set of values. This property makes it well suited for securing fingerprint templates, which are typically represented as unordered sets of minutia points. Multiple implementations of fingerprint fuzzy vault have been proposed [6, 29, 19, 15, 28] and it currently seems to be the most popular method of securing fingerprint data among researchers. The advantages of fingerprint fuzzy vault method over other methods of securing fingerprint data include the existence of a formal security argument based on the hardness of decoding underlying error correcting codes [9], relatively good matching performance, and the ability to use fuzzy vault in identification operating mode [3, 10].

At the same time, multiple possible modes of attack on fuzzy vaults have been reported. Scheirer and Boult describe [24] three possible attacks on privacy preserving biometric systems, all them are applicable to fingerprint fuzzy vaults. The first attack is via record multiplicity (also called cross matching, correlation, or collusion attack), in which the intruder is able to recover original biometric data given two fuzzy vaults of the same finger. The experiments confirming the computational resources of such attack have been performed as well [14, 22]. The second attack, surreptitious key-inversion, assumes that the intruder gains access to the fuzzy vault of the user as well as the the key which was used to lock the vault, and which should be recovered during matching; given these, the intruder can easily obtain original user's fingerprint data. The third attack, blended substitution, consists of adding fingerprint data of a second person to the existing fingerprint fuzzy vault. The possibilities to computationally recover hidden biometric data from fuzzy vaults have been reported as well [8, 17], especially given non-uniform distribution of minutia points in fingerprints [4], but such attacks can be mitigated by choosing proper fuzzy vault configurations. In the current paper, we propose an additional fuzzy vault specific encryption of minutia data strings, which foremost addresses record multiplicity attacks, as well as makes other types of attacks more difficult.

Some approaches to address the vulnerabilities of fingerprint fuzzy vaults have been previously proposed. One type of approach considers the transformation of local minutia information before encoding this information into the vault [20, 16]. In these approaches, the user has to remember the key, which is used for transformations, and the transformations are determined based on the positions of minutia, their local neighborhoods and the key. Note that although some matching results of enhanced fuzzy vaults appear to be better than the original ones, these can be caused by implicit incorporation of the user specific key into the matching decision [23]. As recommended by [23] we measure the matching performance of our system under the assumption of a stolen user key. We also present matching numbers for our two-factor system without a compromised key, though these results come from the unlikeliness of two keys (rather than two templates) matching.

The biometric database cross matching attacks exist for other privacy protections methods, and the need to perform additional key based transformation of data have been stressed as well. Kelkboom et al. [13] address the problem of cross matching biometric templates protected by fuzzy commitment scheme. The proposed random permutation of data before the fuzzy commitment application makes cross matching attack infeasible. That method could be considered advantageous to ours since the random permutations are assumed to be public, but it is not directly applicable to fuzzy vaults. The weaknesses in secure sketches and fuzzy vaults from cross matching attacks have been described by other researchers as well [27, 2].

Alternative methods include [12] which has good security properties, though we cannot compare matching accuracy with this system since the work does not provide an implementation. For general template security in biometrics, the work of [26] provides a theoretical framework for analysis.

3. Fuzzy Vault

In this section we briefly describe how the fuzzy vault [11] is locked and unlocked given a template representing a fingerprint as a set of integers $T = \{t_0, \ldots, t_{n-1}\}$.

3.1. Locking the Vault

The locking function first generates a random secret polynomial, p_s which will be 'locked' in the vault by T. The genuine vault points are then evaluated and stored as $\{(t_i, p_s(t_i))\}$. To obfuscate the genuine data, chaff points are added to the vault as (t, γ) where t and γ are generated uniformly at random such that $t \notin T$ and $\gamma \neq p_s(t)$. A hash of the secret polynomial $h(p_s)$ is stored along with the vault and is used during the unlocking process for verification.

3.2. Unlocking the Vault

To unlock a vault V given a set T', the algorithm first extracts the subset of vault points $P \subseteq V$ by matching the values of T' with V as follows: $(t, \gamma) \in P$ if and only if $t \in T'$. Note that we use γ here because it is unknown which elements of P are genuine and which are chaff. Input P into the Welch-Berlekamp decoder [1] and compare the hash of the output with the stored $h(p_s)$ for verification.

4. Commutative Encryption

We utilize a commutative encryption scheme as one building block of our system. For notational simplicity, we denote $e(m)$ as the encryption of a message m under the

encryption key e; similarly, we denote $d(c)$ as decryption of the ciphertext c under the decryption key d. An encryption scheme is called commutative if for any combination of message m and keys e_1 and e_2 we have

$$e_1(e_2(m)) = e_2(e_1(m)) \ .$$

Hence, by decrypting $e_2(e_1(m))$ with the inner decryption key d_1, we are able to obtain $e_2(m)$. This feature enables our system to perform so-called *blind encryption* on the users' fingerprint template points. That is, our encryption server can encrypt the users' fingerprint template points without knowing those template points. (See Section 5.)

4.1. Instantiation Over Elliptic Curve

RSA is a well-known candidate for such a commutative encryption scheme; however, it is terribly inefficient, as it is recommended to use 2000-bit RSA modules in order to achieve reasonable security guarantee. RSA also needs to be modified to a private key scheme to gain the commutative property. For efficiency, we employ the Pohlig-Hellman exponentiation cipher [21] with the same modulus and instantiate it in elliptic curve groups over $\mathbb{F}_p$. Let $\mathsf{Param} := (p, a, b, g, q, \zeta)$ be the elliptic curve domain parameters, consisting of a prime p that specifies the finite field $\mathbb{F}_p$, two elements $a, b \in \mathbb{F}_p$ that specifies an elliptic curve $E(\mathbb{F}_p)$ defined by the equation:

$$E: \quad y^2 = x^3 + ax + b \pmod{p} \ ,$$

a base point $g = (x_g, y_g)$ on $E(\mathbb{F}_p)$, a prime q which is the order of g, and an integer ζ which is the cofactor. We denote the cyclic group generated by g as $\mathbb{G}$, and it is assumed that the discrete logarithm assumption holds over $\mathbb{G}$, i.e. for all probabilistic polynomial time adversary $\mathcal{A}$, the advantage

$$\mathsf{Adv}^{DL}_{\mathbb{G}}(\mathcal{A}) = \Pr_{x \leftarrow \mathbb{Z}_q}[\mathcal{A}(\mathsf{Param}, g^x) = x]$$

is negligible.

Encryption in this scheme is performed by first choosing a random encryption key $e \leftarrow \mathbb{Z}_q$ and setting the decryption key $d = e^{-1} \pmod{q}$. The encryption of a fingerprint point $t \in \{0,1\}^*$ with key e_1 is denoted as $e_1(t)$ and computed as $(g^{H(t)})^{e_1} = g^{H(t)\cdot e_1}$, where $H(\cdot)$ is a cryptographic hash function, say SHA1. Encrypting the point again with another key e_2 is computed as $(e_1(t))^{e_2} = g^{H(t)\cdot e_1 \cdot e_2}$, and the original encryption can be removed using $d_1$by computing $(g^{H(t)\cdot e_1 \cdot e_2})^{d_1} = g^{H(t)\cdot e_2}$.[1]

The benchmark is tested with the 192-bit elliptic curve domain parameters recommended by NIST p192, where $p = 2^{192} - 2^{64} - 1$, which gives about 96-bit security level. The elliptic curve domain parameters Param are hardwired. We also used the standard point compression technique: a point on $E(\mathbb{F}_p)$ is represented by its x coordinate together with the least significant bit of its y coordinate, and thus each ciphertext is only 193 bit long. The code is implemented in C++, using *Multi-precision Integer and Rational Arithmetic C/C++ Library* (MIRACL) crypto SDK.

[1] $g^{H(t)}$ is an efficient encoding that maps the fingerprint template point t to a point on the curve; however, our system does not require the existence of an efficient decoding. The decoding problem is known as the discrete logarithm problem, which increases the difficulty of an attacker who is trying to recover the original fingerprint template.

5. Encryption Server Scheme

The scheme consist of clients, an encryption server (S_E) storing user-specific encryption keys, and a matching server (S_M) implementing fuzzy vault on encrypted templates. In this scheme neither of the servers gain any information about the underlying fingerprint as it is encrypted before it leaves the client. This is accomplished by the client using a temporary pair of random encryption keys. A diagram illustrating the overview of the scheme is presented in Figure 5 and below we discuss each step of the system in detail.

For each enrollment or verification, the client generates a random key pair, sends an encrypted template to S_E, sends the decryption key to S_M, then deletes the key pair. It is important to note that the user is never required to remember these keys and a new pair is generated each time a user utilizes the system. Then S_E reencrypts the template with a user-specific key which is stored and stays constant for each user. This twice encrypted template is then sent to S_M where the temporary user encryption is removed and a fuzzy vault is locked, or unlocked for enrollment or verification respectively.

5.1. Client

Each client can submit a template for either enrollment or testing under their unique identifier, ID. The client executes the same steps for enrollment or verification.

Step 1: Generate a random temporary encryption key pair (e_u, d_u) using the commutative encryption scheme as described in Section 4.1.

Step 2: Send (d_u, ID) to S_M.

Step 3: Compute and send $(e_u(H(T)), ID)$, where $H(\cdot)$ is a cryptographic hash function, to S_E as follows:

(a) Convert a fingerprint reading f into a set of translation invariant and rotation variant integers $T = \{t_0, t_1, \ldots, t_{n-1}\}$ via a template extraction process as described in Section 8.

(b) Compute the encryption of the template as $e_u(H(T)) = \{e_u(H(t_0)), \ldots, e_u(H(t_{n-1}))\}$ and send it to S_E along with ID.

(c) Delete e_u and d_u.

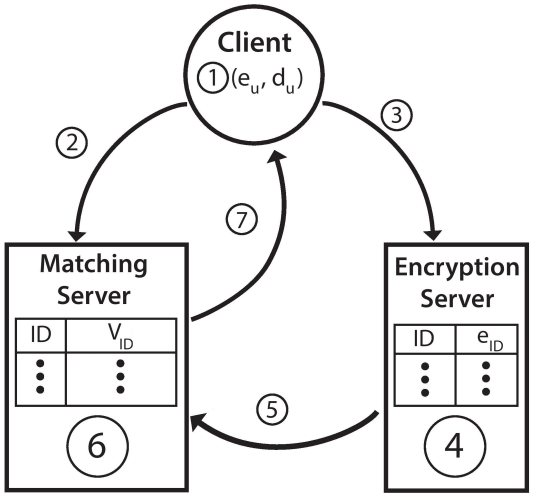

Figure 1. This illustration shows the steps involved in the multi-server system. At Step 1, the client, claiming to be user ID, generates a commutative encryption/decryption key pair (e_u, d_u) and at step 2 sends (d_u, ID) to S_M. At Step 3, the client hashes and encrypts her template T with her temporary key e_u and sends $(e_u(H(T)), ID)$ to S_E. At Step 4, S_E reencrypts $e_u(H(T))$ with e_{ID} and then, at step 5, sends $(e_{ID}(e_u(H(T))), ID)$ to S_M. For Step 6, S_M decrypts $e_{ID}(e_u(H(T)))$ with d_u and by the commutative property of the encryptions, produces $e_{ID}(H(T))$. If the system is being used for enrollment, S_M will generate a secret polynomial and lock the template $e_{ID}(H(T))$ in the fuzzy vault V_{ID}. If the system is being used for verification, S_M will attempt to unlock V_{ID} using $e_{ID}(H(T))$. At step 7, if V_{ID} unlocks and produces the correct secret, S_M returns back a "Verified" decision to the client. The full details of all steps are described in the corresponding "steps" in this section.

5.2. Encryption Server (S_E)

The encryption server is responsible for generating, storing, and applying the user-specific encryption keys. We present S_E as a single server, but it can be generalized to any number of servers if more encryptions are desired. If an attacker can recover the encrypted fingerprint by unlocking the fuzzy vault, they must then recover the keys stored at all of these servers to decrypt the stored data offline.

Step 4: Compute and send $(e_{ID}(e_u(H(T))), ID)$ to S_M as follows:

(a) Wait for a message from the client containing $(e_u(H(T)), ID)$.

(b) Enrollment only: Generate an random encryption key e_{ID} using the same commutative encryption scheme used for e_u and d_u and store (ID, e_{ID}). The decryption key d_{ID} is not computed.

(c) Encrypt $e_u(H(T))$ using e_{ID} to get $e_{ID}(e_u(H(T)))$.

(d) Delete $(e_u(H(T)), ID)$.

Step 5: Send $(e_{ID}(e_u(H(T))), ID)$ to S_M and delete it.

5.3. Matching Server (S_M)

The matching server locks, unlocks, and stores each enrolled user's fuzzy vault. We present S_M as a single server, though it can easily be extended to multiple servers to distribute the computation and storage requirements.

Step 6: Enroll or match templates as follows:

(a) Wait for communication from S_E containing $(e_{ID}(e_u(H(T))), ID)$ and from the client containing (d_u, ID).

(b) Compute $d_u(e_{ID}(e_u(H(T)))) = e_{ID}(H(T))$. Store $h(e_{ID}(H(T)))|_\ell$ in the database, where ℓ is the size of the original template points and $h(\cdot)|\ell$ stands for the t left-most bits of the hash digest.[2]

(c) Enrollment: Construct and store a fuzzy vault V_{ID} using the set $e_{ID}(H(T))$ as described in Section 3.1.

(d) Verification: Use the set $e_{ID}(H(T))$ to attempt to unlock V_{ID} as described in Section 3.2.

Delete (d_u, ID), $(e_{ID}(e_u(H(T))), ID)$, and $e_{ID}(H(T))$.

Step 7: If the vault unlocks successfully, the client gains access to the system.

6. Adversarial Scenarios

In this section, we give an overview of the security of our system for all the combinations of servers being compromised. We use secure communication and assume that all the communication channels are protected using authenticated encryption, e.g. SSL/SSH. Other modern methods, such as RSA, can be used to further secure the channel communication. Thus, we only consider attacks on the servers themselves.

6.1. No Servers Compromised

In every key-less secure biometric system an attacker can submit fingerprints using a specific user's ID until one of them accepts. Each one of the random fingerprints is expected to be successful with probability equal to the FAR of the system. For this reason, we only report the ZeroFAR (the FRR where FAR is zero) to measure matching performance [25]. Since we test with 4950 impostor matches with a FAR of 0, an attacker would be expected to submit more than 4950 random fingerprints to find a match. The system would be able detect this many failed attempts on the same ID and lock the user's account.

[2] $h(\cdot)$ is implemented by using SHA256 in our prototype.

6.2. Matching Server Compromised

Offline security: If S_E is offline, the stored templates are protected by the encryption scheme. Note that all the information S_M uses for matching is encrypted fingerprint template points. First of all, similar to RSA, $f_e(x) = x^e$ is conjectured to be a pseudo-random permutation indexed by e under the *discrete logarithm* assumption. In addition, our encryption scheme maps a fingerprint template point t to an group element on the elliptic curve as $g^{H(t)}$. When $H(\cdot)$ is viewed as a random oracle, no efficient adversary can distinguish the correlation between $H(t_1)$ and $H(t_2)$ for any correlated t_1 and t_2; otherwise, she can break the random oracle assumption. Hence, for any t_1 and t_2, the ciphertexts of them, $g^{H(t_1)\cdot e}$ and $g^{H(t_2)\cdot e}$ appear to be random to any computationally bounded adversary, when e is drawn from $\mathbb{Z}_q$ uniformly at random. Therefore, the confidentiality of the users' fingerprint template is preserved. In Theorem 1, we show that if $f_e(x)$ is a pseudo-random permutation then $h(f_e(x))|_\ell$ is indistinguishable from a uniformly random ℓ bit string, where $h(f_e(x))|_\ell$ stands for the leftmost ℓ bit truncation of $h(f_e(x))$ and $h(\cdot), H(\cdot)$ are some cryptographic secure hash functions.

Online security: If the attacker has control of S_M and S_E is still online, they can pretend to be a particular client in an attempt to build a mapping of e_{ID}. The attacker can generate a set of template points as a client to send to S_E, then observe the encryptions of these points received at S_M. As discussed in Section 7.2, the attacker cannot recover the encryption key e_{ID} by querying the encryption oracle (S_E). However, by doing this repeatedly the attacker can determine the mapping of all the encryptions of template points since the space of possible template points is fairly small (experimentally about 2^{30}). In this way, the attacker can decrypt the data without attacking S_E directly, though the repeated submission of templates with the same ID may be detectable at S_E, which can take action to limit this attack.

Our system itself limits the effectiveness of this attack by hashing the encrypted data at S_M using the fuzzy vault scheme. After the attacker decrypts the fuzzy vault points, they will still have to unlock the vault. This limits the attacker either attempting to crack the fuzzy vault, or perform online attacks on live data. As a live attack, they can wait for a user to submit a template for enrollment or testing. This template will be encrypted, but not protected by a fuzzy vault so the attacker could potentially compromise the template using the attack mentioned earlier to find the mapping of e_{ID}. In this way, an attacker compromising S_M can potentially recover templates from active users, but cannot easily recover the template of every enrolled user.

Protection from specific attacks: We discuss three specific attacks on fuzzy vaults in Section 2 that would normally be executed by an attacker compromising S_M. We show that our system can protect against these attacks.

With a standard fuzzy vault, an attacker recovering two vaults locked using the same template can perform the record multiplicity attack [14, 22]. Since our fuzzy vaults are all encrypted with different random encryption keys, no such offline attack is possible. The attack can somewhat be executed in the online scenario by slowly mapping the encryption of e_{ID}, though this would be detectable and difficult to perform.

The second attack occurs when an intruder obtains a fuzzy vault as well as the secret polynomial used to lock that vault. Given these, it is easy to determine which vault points are genuine. The secret polynomial could only be recovered by observing the unlocking process on a successful unlocking of the vault or cracking the vault directly since we don't use or store this information for any other purpose. Even after recovering the genuine template points, the intruder still needs to map the encryptions of e_{ID} to discover the underlying fingerprint data.

The final attack we discuss here is that of blended substitution. With access to a locked fuzzy vault, an intruder can add or replace points using a known template and lock an additional secret in the vault. This allows the intruder undetectable access since the legitimate user will still have access as well. We limit this by comparing the unlocked secret to a hash of the locking secret. It would be computationally infeasible for an attacker to generate a secret with the same hash value due to the size of the random secrets used.

6.3. Encryption Server Compromised

The encryption server only stores the encryption keys and it only receives ciphertexts of the users' fingerprint template points and performs re-encryption. Note that users' fingerprint template points $t_i \in \{0,1\}^\ell$ may be correlated; nonetheless, as we will see it does not cause any security problem. Assuming the cryptographically secure hash function $H : \{0,1\}^\ell \mapsto \{0,1\}^\kappa$ realizes a random oracle, by definition, $H(\cdot)$ has the *uniform difference property*: $\forall t_1, t_2 \in \{0,1\}^\ell, t_1 \neq t_2$, $H(t_1) - H(t_2) \pmod{2^\kappa}$ is uniformly distributed in $\{0, \ldots, 2^\kappa - 1\}$. Therefore, given any two correlated t_1, t_2, $H(t_1)$ and $H(t_2)$ should appear to be independent to all computationally bounded adversaries. So that their corresponding group elements, $g^{H(t_1)}$ and $g^{H(t_2)}$ is computationally indistinguishable from two random group elements. Considering that $(g^{H(t_i)})^{e_u}$ are pseudo-random permutations of a set of independent random group elements, no efficient adversary can obtain any information from the user's ciphertexts; otherwise the random oracle assumption of the underlying cryptographic hash function does not hold. We can even upgrade our system to achieve unconditional security by requiring the client to use a fresh random encryption key for each fingerprint point. With a one-time key, $e_u^{(i)}(t_i)$ information theoreti-

cally hides t_i when the encryption key $e_u^{(i)} \leftarrow \mathbb{Z}_q$ is picked uniformly at random. This is because the distribution of $H(t_i) \cdot e_u^{(i)}$ is uniform distribution over $\mathbb{Z}_q$, and thus $e_u^{(i)}(t_i)$ is a random group element.

6.4. Matching Server and Encryption Server Compromised

If both servers are comprimised all, attacks on the templates stored at S_M can be performed offline, though the genuine points are still hidden among the chaff points of the fuzzy vault. Mapping the encryption of the keys stored at S_E can be computed offline since the attacker has access to all the e_{ID} so we assume that there is no encryption and the data is only protected by the fuzzy vault.

With both servers compromised, online attacks are straight-forward. Since S_E receives $e_u(T)$ and S_M receives d_u, the encryption can be mapped and T can be recovered. This is computationally expensive since recovering $(g^{H(t)})^{e_u}$ given e_u and g is still the discrete-log problem. However, since the space the template points are drawn from is relatively small, the mapping of the encryption can be found by brute forcing every possible template point.

7. Security

Our system is secure against semi-honest (a.k.a. honest but curious) non-colluding adversarial servers. In this setting, both servers follow the protocol description, but they might seek to recover the users' fingerprints from all the available information. This reflects the scenario where both servers are honest but one of them is compromised by the adversary.

7.1. Fuzzy Vault

To measure the security of the fuzzy vault, we use the security parameter defined by [9]

$$\lambda = \sqrt{cz} - g, \tag{1}$$

where z is the number of terms in the polynomial, c is the total number of points in the vault, and g is the number of genuine points in the vault. This parameter is analogous to the bits of security of the vault. We note that this parameter gives an upper bound on the security assuming the template locking set is uniformly distributed. This is the case since the vault is locked using encrypted data, though some security is lost when the adversary can decrypt the vault points. The exact amount of security lost when the encryption is compromised is difficult to analyze, though the attacker would be able to perform attacks that take advantage of the distribution of fingerprint templates.

7.2. Chosen Plaintext Key Recovery Resistance

The chosen plaintext key recovery attack is modeled as the following game:

- **Setup phase:** The challenge $\mathcal{C}$ picks a random key $e \leftarrow \mathbb{Z}_q$.
- **Query phase:** For $i = 1, 2, \ldots$:
  - The adversary $\mathcal{A}$ sends $\mathcal{C}$ query $x_i \in \mathbb{G}$.
  - $\mathcal{C}$ replies $c_i := x_i^e$ to $\mathcal{A}$.
- **Guess phase:** $\mathcal{A}$ outputs e^*

We say that the adversary wins if and only if $e = e^*$.

Our commutative encryption scheme is chosen plaintext key recovery resistant if the discrete logarithm problem is hard. (See [21] for more discussion.)

7.3. Pseudo-randomness of the Truncations

In this section, we show that the left-most ℓ-bit truncations of $h(g^{H(t)\cdot e})$ is indistinguishable from U_ℓ, assuming that $f_e(x) := x^e$ is a pseudo-random permutation and $h(\cdot)$ realizes a random oracle, where U_ℓ denotes the uniform distribution over $\{0,1\}^\ell$. In our scheme, we map each t to a unique group element $x = g^{H(t)}$. Without loss of generality, we will examine the property of $h(f_e(x))|_\ell$, as one can always set $x = g^{H(t)}$ for our scheme.

Theorem 1. *Let $\lambda \in \mathbb{N}$ be the security parameter, and let $\mathbb{G}$ be a cyclic group with a λ-bit prime order q, where the discrete logarithm problem is hard. Let $h(\cdot)$ be a random oracle. Assuming $\mathcal{F}_\lambda = \{f_e(x) = x^e | \forall e \in \mathbb{Z}_q\}$ is a pseudo-random permutation family $\mathbb{G} \mapsto \mathbb{G}$, for all interger $0 < \ell < \lambda/2$, the probability that a $\mathsf{poly}(\lambda)$ running time adversary can distinguish $h(f_e(x))|_\ell$ from U_ℓ is negligible i.e., $< \frac{1}{\mathsf{poly}(\lambda)}$, where e is chosen uniformly at random from $\mathbb{Z}_q$.*

Proof. Given that $f_e(x)$ is a pseudo-random permutation, by definition, no $\mathsf{poly}(\lambda)$-time adversary can distinguish $f_e(x)$ from a true random permutation $\pi : \mathbb{G} \mapsto \mathbb{G}$. In addition, the $\mathsf{poly}(\lambda)$-time adversary is not able to distinguish a true random permutation $\pi : \mathbb{G} \mapsto \mathbb{G}$ from a true random function $r : \mathbb{G} \mapsto \mathbb{G}$ with more than negligible probability. It is due to the fact that the only event where the adversary is able to distinguish a random permutation from a random function is when the outputs of the random function have a collision. Since a $\mathsf{poly}(\lambda)$-time adversary can only query at most $\mathsf{poly}(\lambda)$ queries to the random function, the probability to have a collision is bounded by $\frac{\mathsf{poly}(\lambda)}{|\mathbb{G}|} < \frac{1}{\mathsf{poly}(\lambda)}$. Therefore, we can switch the output of $f_e(x)$ with a uniformly random group element in $\mathbb{G}$, and the adversary has

negligible probability to distinguish such a switch. Moreover, when $h(\cdot)$ realizes a random oracle, $h_\ell(x) := h(x)|_\ell$ is a universal hash function $\mathbb{G} \mapsto \{0,1\}^\ell$, i.e.

$$\forall x, y \in \mathbb{G}, x \neq y : \Pr[h_\ell(x) = h_\ell(y)] \leq 2^{-\ell}.$$

By the leftover hash lemma, the computational distance between $h_\ell(f_e(x))$ and U_ℓ is at most $2^{-\frac{\lambda-\ell-1}{2}} \leq 2^{-\lambda/4}$. Hence, the probability that a poly(λ)-time adversary can distinguish $h(f_e(x))|_\ell$ from U_ℓ is negligible, where e is chosen uniformly at random from $\mathbb{Z}_q$. □

8. Experimental Results

In this section, we analyze the experimental performance of our system in terms of matching and timing. We conducted the experiments on the first fingerprint database from the Second International Fingerprint Verification Competition (FVC2002/DB2). The minutiae points were extracted by finding large curvature points on the contours of the binarized fingerprint images [7]. Additionally, to improve the performance of minutia extraction, the fingerprints were enhanced by the short time Fourier transform algorithm [5]. We also remove suspect spurious minutia points by removing all points within a euclidean distance of 3 from each other. A standard testing protocol for calculating all possible 2800 genuine and 4950 impostor matches was used. For the hash functions $h(\cdot)$ and $H(\cdot)$ we use SHA256.

Here we briefly present matching results via the ZeroFAR which is the False Reject Rate (FRR) at a fuzzy vault operating threshold that achieves a False Accept Rate(FAR) $= 0$. The matching scheme used in this system is n-gons as developed in [18]. In this matching scheme a template is represented by m minutia points and each minutia point is represented as an ordered triple (x, y, θ). For each minutia point in the template, the closest k neighboring minutia are located, where $k > n-1$, and subsets of these minutia of size n are used to form the quantized n-gon feature as follows. Each point in the feature is labeled $p_0, p_1, ..., p_{n-1}$ where $x_0 \leq x_1 \leq ... \leq x_{n-1}$, i.e., in order from left to right with ties broken by y value. Each point is then translated toward the origin by p_0 in x and y but not θ, i.e, $p_i = (x_i - x_0, y_i - y_0, \theta_i)$ for every $i \in 0, 1, ..., n-1$. Then each resulting (x, y, θ) triple is quantized into 8 bins and concatenated together to form the final n-gon feature $t = \theta_0 \circ x_1 \circ y_1 \circ \theta_1 \circ ... \circ x_{n-1} \circ y_{n-1} \circ \theta_{n-1}$. After doing this for each minutia point in the template, we have a template $T = t_1, t_2, ..., t_q$, where q is the resulting size of the template. By design, this scheme is invariant in translation, but variant in rotation. During verification, we generate a larger template of all rotations by rotating each n-gon over a fixed interval and quantizing it at each rotation to achieve a form of rotation invariance for the test fingerprints. n-gons all rotations (AR) locks the entire template of rotations inside the fuzzy vault and also uses creates the same template for verification. We summarize our matching performance in Table 1. The average enrollment and verification times were 3.844 and 4.057 seconds. The tests were run using three Macbook Pro's with 2.0-2.9Ghz intel i7's and 16Gb of memory connected via a gigabit switch.

Method	# Chaff	g	z	λ	ZeroFAR
n-gons	38,000	395	6	84.97	24.3%
n-gon AR	200,000	1474	12	80.89	12.6%

Table 1. Secure matching performance for the multi-server system where g is the average number of genuine points in the fuzzy vaults.

We note that the n-gons all rotations method introduces a large amount of correlation between genuine points in the template. In order to ameliorate this issue, we hash the template points using a cryptographic hash function to remove any correlation before encryption. However if an adversary were able to recover an encryption key and it's matching fuzzy vault, she could potentially recognize this correlation. Using n-gons single enrollment addresses this issue by only enrolling under a single rotation, but at the cost of matching performance, as shown in Table 1.

9. User Encrypted Scheme

In this section, we briefly describe how our system can be converted into a two-factor system that provides a significant increase in both security and matching accuracy. In this version of the system, each user must maintain a user-specific encryption key which is applied to each of the template points before they are sent to the server where a fuzzy vault will be constructed. These templates are sent directly to the matching server since the user is performing the function of the encryption server. For enrollment, the genuine points are encrypted with this key and sent to the matching server to create a fuzzy vault; for verification, the genuine template points are encrypted with this same key and used to attempt to unlock the fuzzy vault. If the server is compromised, the user-specific encryption that is unknown to the adversary is still protecting the template points. The adversary must now overcome the encryption mapping before they can unlock the fuzzy vault. The increase in matching accuracy comes from the fact that impostor tests are performed using a different encryption key than was used during enrollment making all the imposter test scores equal to 0. By adjusting our parameters such that all genuine scores are at least 1, we achieve a ZeroFAR of 0.

We note that we do not need commutative encryption for this version of our system and could use a wide range of encryption methods. We still use commutative encryption in our implementation since it is secure with small key sizes and is efficient in practice.

It may be unrealistic for a user to memorize a lengthy encryption key, but we note that this can be achieved through smart cards, near-field communication (NFC), and other recently developed technologies. Unfortunately, user-specific keys can be compromised in practice so there is much desire to have a keyless system with similar security guarantees.

10. Acknowledgments

This research is supported by NSF grant TC 1115670 and the Center for Identification Technology Research (CITeR).

References

[1] E. R. Berlekamp and L. R. Welch. Error correction for algebraic block codes, Dec. 30 1986. US Patent 4,633,470.

[2] M. Blanton and M. Aliasgari. Analysis of reusability of secure sketches and fuzzy extractors. *Information Forensics and Security, IEEE Transactions on*, 8(9):1433–1445, 2013.

[3] J. Bringer, H. Chabanne, and M. Favre. Fuzzy vault for multiple users. In A. Mitrokotsa and S. Vaudenay, editors, *Progress in Cryptology - AFRICACRYPT 2012*, volume 7374 of *Lecture Notes in Computer Science*, pages 67–81. Springer Berlin Heidelberg, 2012.

[4] E.-C. Chang, R. Shen, and F. W. Teo. Finding the original point set hidden among chaff. In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, ASIACCS '06, pages 182–188, New York, NY, USA, 2006. ACM.

[5] S. Chikkerur, A. N. Cartwright, and V. Govindaraju. Fingerprint enhancement using STFT analysis. *Pattern Recognition*, 40(1):198–211, 2007.

[6] T. Clancy, D. Lin, and N. Kiyavash. Secure smartcard-based fingerprint authentication. In *ACM Workshop on Biometric Methods and Applications (WBMA 2003)*, Berkeley, CA, USA, 2003.

[7] V. Govindaraju, Z. Shi, and J. Schneider. Feature extraction using a chaincoded contour representation of fingerprint images. In *4th international conference on Audio- and video-based biometric person authentication*, volume 2688, pages 268–275, Guildford, UK, 2003. Springer-Verlag.

[8] X. Q. Guo and A. Q. Hu. The automatic fuzzy fingerprint vault based on geometric hashing: Vulnerability analysis and security enhancement. *Int. Conference on Multimedia Information Networking and Security*, 1:62–67, 2009.

[9] J. Hartloff, M. Bileschi, S. Tulyakov, J. Dobler, A. Rudra, and V. Govindaraju. Security analysis for fingerprint fuzzy vaults. In *Proc. SPIE*, volume 8712, pages 871204–871204–12, 2013.

[10] J. Hartloff, J. Dobler, S. Tulyakov, A. Rudra, and V. Govindaraju. Towards fingerprints as strings: Secure indexing for fingerprint matching. In *ICB*, pages 1–6, 2013.

[11] A. Juels and M. Sudan. A fuzzy vault scheme. *Des. Codes Cryptography*, 38(2):237–257, 2006.

[12] S. Kanade, D. Petrovska-Delacrétaz, and B. Dorizzi. Multi-biometrics based crypto-biometric session key generation and sharing protocol. In *MMSec*, pages 109–114, New York, NY, USA, 2011. ACM.

[13] E. J. C. Kelkboom, J. Breebaart, T. A. M. Kevenaar, I. Buhan, and R. N. J. Veldhuis. Preventing the decodability attack based cross-matching in a fuzzy commitment scheme. *Information Forensics and Security, IEEE Transactions on*, 6(1):107–121, 2011.

[14] A. Kholmatov and B. Yanikoglu. Realization of correlation attack against the fuzzy vault scheme. In *Security, Forensics, Steganography, and Watermarking of Multimedia Contents X*, volume SPIE 6819, pages 68190O–68190O–7, 2008.

[15] S. Lee, D. Moon, S. Jung, and Y. Chung. Protecting secret keys with fuzzy fingerprint vault based on a 3d geometric hash table. In *Adaptive and Natural Computing Algorithms*, volume 4432 of *LNCS*, pages 432–439. 2007.

[16] P. Li, X. Yang, K. Cao, P. Shi, and J. Tian. Security-enhanced fuzzy fingerprint vault based on minutiae's local ridge information. In M. Tistarelli and M. S. Nixon, editors, *International Conference on Biometrics*, volume 5558 of *Lecture Notes in Computer Science*, pages 930–939. Springer Berlin Heidelberg, 2009.

[17] P. Mihailescu, A. Munk, and B. Tams. The fuzzy vault for fingerprints is vulnerable to brute force attack. In *BIOSIG*, pages 43–54, 2009.

[18] M. Morse, J. Hartloff, T. Effland, J. Schuler, J. Cordaro, S. Tulyakov, A. Rudra, and V. Govindaraju. Secure fingerprint matching with generic local structures. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, June 2014.

[19] K. Nandakumar, A. K. Jain, and S. Pankanti. Fingerprint-based fuzzy vault: Implementation and performance. *IEEE Transactions on Information Forensics and Security*, 2(4):744–757, 2007.

[20] K. Nandakumar, A. Nagar, and A. Jain. Hardening fingerprint fuzzy vault using password. In *ICB*, pages 927–937, 2007.

[21] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, IT-24(1):106–110, 1978.

[22] H. T. Poon and A. Miri. A collusion attack on the fuzzy vault scheme. *ISeCure, The ISC International Journal of Information Security*, 1(1):27–34, 2009.

[23] C. Rathgeb and A. Uhl. Two-factor authentication or how to potentially counterfeit experimental results in biometric systems. In *Image Analysis and Recognition*, volume 6112 of *Lecture Notes in Computer Science*, pages 296–305. Springer Berlin Heidelberg, 2010.

[24] W. Scheirer and T. Boult. Cracking fuzzy vaults and biometric encryption. In *Biometrics Symposium, 2007*, pages 1 –6, sept. 2007.

[25] W. Scheirer and T. Boult. Bipartite biotokens: Definition, implementation, and analysis. In M. Tistarelli and M. Nixon, editors, *Advances in Biometrics*, volume 5558 of *Lecture Notes in Computer Science*, pages 775–785. Springer Berlin Heidelberg, 2009.

[26] K. Simoens, J. Bringer, H. Chabanne, and S. Seys. A framework for analyzing template security and privacy in biometric authentication systems. *Information Forensics and Security, IEEE Transactions on*, 7(2):833–841, April 2012.

[27] K. Simoens, P. Tuyls, and B. Preneel. Privacy weaknesses in biometric sketches. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 188–203, 2009.

[28] B. Tams. Absolute fingerprint pre-alignment in minutiae-based cryptosystems. In *BIOSIG'13*, pages 75–86, 2013.

[29] S. Yang and I. Verbauwhede. Automatic secure fingerprint verification system based on fuzzy vault scheme. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, volume 5, pages v/609–v/612 Vol. 5, 2005.