Cryptanalysis of the Fuzzy Vault for Fingerprints: Vulnerabilities and Countermeasures

Dissertation
zur Erlangung des mathematisch-naturwissenschaftlichen
Doktorgrades
“Doctor rerum naturalium”
der Georg-August-Universität Göttingen

im Promotionsprogramm
“Ph.D. School of Mathematical Sciences (SMS)”
der Georg-August University School of Science (GAUSS)

vorgelegt von
Berend-Benjamin Tams
aus Göttingen

Göttingen 2012

Betreuungsausschuss

Prof. Dr. Preda Mihăilescu, Mathematisches Institut, Georg-August Universität Göttingen

Prof. Dr. Axel Munk, Institut für Mathematische Stochastik, Georg-August Universität Göttingen

Mitglieder der Prüfungskommission

Referent/in:	Prof. Dr. Preda Mihăilescu
Korreferent/in:	Prof. Dr. Axel Munk
ggf. 2. Korreferent/in:	—

Weitere Mitglieder der Prüfungskommission:

Prof. Dr. Carsten Damm

Prof. Dr. Dieter Hogrefe

Prof. Dr. Russell Luke

Prof. Dr. Max Wardetzky

Tag der mündlichen Prüfung: 5. Dezember 2012

Contents

1 Introduction **1**

1.1 Security and Biometry 1

1.2 Thesis Contribution 6

1.3 Thesis Outline 10

1.4 Acknowledgments 11

2 Basics **13**

2.1 Cryptography 13

2.2 Fingerprints 16

2.3 Error-Correcting Codes 27

2.4 The Fuzzy Commitment Scheme 32

3 The Fuzzy Fingerprint Vault **37**

3.1 The Fuzzy Vault Scheme 37

3.2 Implementations for Fingerprints 40

3.3 Alignment 45

3.4 Implementation 47

4 Vulnerabilities **55**

4.1 Brute-Force Attack 57

4.2 Auxiliary Alignment Data May Help to Improve Attacks 67

4.3 False-Accept Attack 69

4.4 Cross-Matching and the Correlation Attack 73

5 Cross-Matching Resistant Minutiae Fuzzy Vault 87

5.1 Basic Vault Construction 89
5.2 Modified Vault Construction without Chaff Points 96
5.3 Training 97
5.4 Randomized Decoder 101
5.5 Implementation 104
5.6 Performance Evaluation 106
5.7 Security Analysis and Evaluation 112
5.8 Comparison with other Implementations 121

6 Discussion 123

6.1 Conclusion 123
6.2 Outlook 124
6.3 Open Problems 124
6.4 Final Remarks 125

A Software Library 127

A.1 Programming Interface 127
A.2 Number Theory 128
A.3 Cryptographic Tools 137
A.4 Error-Correcting Codes 137
A.5 Fingerprints 139
A.6 Fuzzy Vault 141
A.7 Minutiae Fuzzy Vault 142
A.8 Cross-Matching Resistant Minutiae Fuzzy Vault 144
A.9 Miscellaneous 145

B Tables 147

Bibliography A

Index K

Curriculum Vitae O

1. Introduction

1.1 Security and Biometry

In societies in which computer-aided communication is the norm, the transfer of information is ubiquitous and easy. This raises questions about the intended use and source of any particular piece of information. For example, if it is meant to be private, how to keep it that way? Furthermore, how to reliably determine who actually created the information? The field of cryptography helps resolve these questions. It provides techniques for the encryption and decryption of information using keys. Moreover, protocols to exchange common keys through a possibly eavesdropped channel are proposed. Both of these are important methods to enable privacy. In addition, cryptography provides techniques that ensure confidence in authenticity. For example, an individual can be authorized based on a key that he provides (e.g., *personal identification number* (*PIN*) or password). Such a criterion for authenticating identities can be said to be based on *what they know*. However, a password can be forgotten or, if written down, can be stolen. To prevent these risks, many individuals attempt to create easily memorable passwords. Unfortunately, this often results in the individual choosing a weak password, typically constructed using personal information (e.g., a birthday or the name of a significant other), which increases the risk of others' guessing the password — and therefore, of theft. An alternate criterion for the verification of identity might be based on *what they possess* (e.g., a chip card). However, such a token can be lost or stolen too. Hybrid solutions in which both criteria — *what they possess* and *what they know* (e.g., a chip card in combination with a PIN) — are fused, provide better protection from theft. In this case, in order to be authorized, an intruder would have to steal the chip card and guess the correct PIN. But this hybrid still does not address the problem of passwords being forgotten by authorized individuals.

A popular alternative to *what they know* is *what they are criteria*. Here, *biometrics*[1], such as a fingerprint, the iris of the human eye, a palm, or the face serve as evidence for *what they are*: a person's true, unique identity. A digitized template is created that functions similar to a password. Authentication protocols that incorporate biometric templates do not have the disadvantage that they can be forgotten

---

[1] For a comprehensive overview of biometrics we refer to Jain et al. (2007).

or lost. Barring injuries, fingers are always with us; moreover, it is unlikely that fingerprints change significantly during our lives.

Whatever the criteria, secure re-identification of individuals requires us to store information that is related to them. The conventional system involves a party that stores information about authorized individuals, called the *server*, while we refer to the individuals as *users* (or *system users*).

In a password-based authentication scheme, user names along with their respective passwords are stored on a server-side database. In such a scenario, we usually cannot prevent the fact that some persons will have access to the content of the database. Such persons (for example, system administrators) are thus able to read the password information related to enrolled users. To prevent such persons from using password information to impersonate authorized users, a *non-invertible transformation*[2] (e.g., a one-way hash function) of each password is stored rather than the unprotected password. During authentication, the user sends his password to the service provider, which computes the password's non-invertible transformation. Next, the service provider compares the transformation that is stored on the database with the just-transformed password. If both agree, the authentication attempt is accepted; otherwise it is rejected. So a would-be thief cannot find the passwords in the database; he must either steal them from users or guess at them.

If a biometric-based authentication scheme is in place, a would-be thief proceeds a little differently: He might capture an authentic fingerprint, create a fake finger, and then create a fake biometric fingerprint template, thus impersonating the corresponding user. The situation is rather serious, because biometric templates may correspond to human beings with nearly unique precision compared to mere passwords. One consequence of this is that if the biometric templates are stored in clear on a database, any person who is able to read its content might be able to determine exact identities of corresponding human beings. Depending on the kind of services the server provides or the delicate nature of the data involved, the identity of a corresponding human being is especially worthy of protection. Thus, biometric templates have especially to be stored protected. In addition, ineffective protection of biometric templates has consequences beyond breach of privacy, as compared to protecting passwords: For example, if a password is corrupted (e.g., discovered by others) it can be replaced easily compared to replacing a biometric template. We only have ten fingers, while keys (such as a PIN or password) can be replaced nearly arbitrarily many times.

While the requirements for so-called *biometric template protection schemes* are similar to those used for protecting passwords, they are more difficult to achieve. With high confidence it must be efficiently verifiable whether a provided biometric template matches the template that is encrypted by the stored data; furthermore, it must be computationally infeasible to derive the unencrypted biometric template from the stored data. Here there is a great difference between password and biometric schemes: Accurate biometric authentication, contrary to password keys, necessarily deals with inexact data. Biometric templates extracted from the same individual are only reproducible within controlled error bounds, because each acquisition of the same biometric modality is (at least) slightly different.

---

[2] Non-invertible transformation of a password means that it is easy to transform the password, while on the contrary the derivation of a password with given transformation is computationally hard.

Overview of Biometric Template Protection Schemes

In this section, we offer a preliminary discussion of biometric template protection schemes and their respective function. For a comprehensive overview we refer to Cavoukian and Stoianov (2009).

There is no such thing as an exact biometric template; each one is merely an approximation of its source. Different measurements of the same biometric source will differ within some bounds while also having some reasonable similarity. Consequently, computerized matchers have to allow for these reasonable differences between biometric templates. These differences between two biometric templates of the same individual can be usefully conceptualized as deviations or errors. In this way, some proposals for biometric template protection schemes couple techniques known from traditional cryptography with techniques from the discipline of *error-correcting codes*.

The Fuzzy Commitment Scheme and the Fuzzy Vault Scheme

Probably the easiest scheme that combines traditional cryptography with error-correcting codes is the *fuzzy commitment scheme* of Juels and Wattenberg (1999). They assume that the biometric templates are encoded as vectors v of fixed length. Independently from the template, a secret key κ in terms of an error-correcting code is generated. The offset $\kappa+v$ as well as a non-invertible hash $h(\kappa)$ forms the protected template. Assuming randomness of the templates v and a large pool for choosing codewords κ it is infeasible to derive either the template v or the secret key κ only from $\kappa + v$. Upon authentication, a query template v' is used to "shift" the offset $\kappa + v$ back: If v and v' are of sufficient similarity (in terms of *Hamming distance*)[3], i.e. if v and v' were extracted from the same individual, then $\kappa + v - v'$ is close to the key κ and can be corrected to κ. Successful recovery of the key κ can be verified using $h(\kappa)$ in the same way as for password-based authentication.

For a reasonably large error-correcting code, and assuming the hash function h used to compute $h(\kappa)$ is non-invertible, then information-theoretic security of the fuzzy commitment scheme can be asserted if the data within biometric templates is both randomly as well as uniformly distributed. But of course, biometric templates are usually non-uniformly distributed; e.g., they look like fingerprints, not noise. This limits the amount of security that biometric template protection schemes are capable of providing, including the fuzzy commitment scheme. In any case, under ideal assumptions, the fuzzy commitment scheme, despite its similarity, has proven security and enables good authentication performance for biometric modalities whose instances can be easily encoded as fixed-length feature vectors (such as human irises).

However, fingerprint templates seem to be not well-suited for protection by the fuzzy commitment scheme. Fingerprint templates such as *minutiae templates*[4] usually are encoded as feature sets without a specific order, rather than as ordered feature vectors (see Figure 1.1). This is mainly due to the scanning process typically used to acquire fingerprints. It is perhaps unavoidable that, during each scan, a user will place different regions of his finger on the scanner's surface, causing different

---

[3] The Hamming distance is the number of positions in which two strings/vectors differ, i.e. $\mathrm{dist}(v_1, \ldots, v_n; w_1, \ldots, w_n) = \sum_{v_i \neq w_i} 1$

[4] Informally, a fingerprint minutia is a characteristic location on a fingerprint image.

features to be present in the resulting feature set. Moreover, fingerprint images are often of low quality due to noise, e.g., caused by dirt, skin conditions such as dry or wet fingers, etc. Therefore, important or ground truth features might be missing in the feature set. In addition, spurious fingerprint features are easily extracted. For these reasons, the similarity of fingerprint features better corresponds to a set difference measure rather than to a measure correlating with Hamming distance.

Later, Juels and Sudan (2002) proposed the so-called *fuzzy vault scheme*. Here the authors assume that the biometric templates are encoded as unordered sets v, which makes the fuzzy vault scheme a promising method to protect fingerprint templates. Furthermore a secret key κ is generated. Informally, again in terms of an error-correcting code, the key κ is bound to the elements of v in the sense that if sufficiently many elements from v are known, then the secret κ can be recovered. The features v as well as the key κ are protected by generating a large number of chaff features. Finally, the vault is the union of genuine features v constituted with additional information, i.e. κ, all dispersed within the chaff features. The security of the vault can be described as the difficulty of distinguishing genuine from chaff features. On authentication, a second template encoded as a set v' is provided. If v' overlaps with v substantially then the information that is linked to v can be used to recover the key κ and the entire template v. In this case, the authentication attempt is successful. Otherwise, if κ cannot successfully be recovered then the query is rejected. Again, a criterion to verify the correct κ can be enabled by storing a cryptographic hash value $h(\kappa)$ along with the vault.

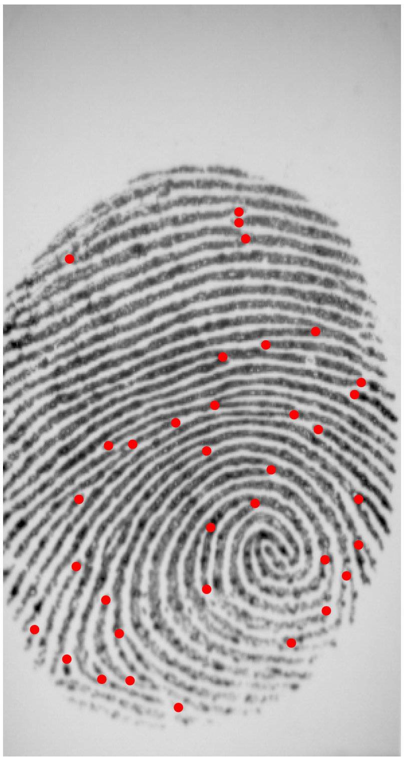


Figure 1.1: Locations of a Fingerprint's Minutiae

Juels and Sudan (2002) proved that information-theoretic security of the fuzzy vault scheme is possible using appropriate parameters.[5] In particular, for there to be high information-theoretic security, successful genuine authentication occurs only when the feature sets v and v' share quite a large number of common elements. Unfortunately, for two genuinely matching fingerprints, far fewer feature correspondences are able to be captured by the matcher than is required by theory to provide proven security. As a consequence, realistic implementations of the *fuzzy fingerprint vault*[6] usually do not have proven security as intended by Juels and Sudan (2002) (see Merkle et al. (2010a)). But this does not necessarily imply that the fuzzy fingerprint vault must be insecure, in particular for implementations where multiple

---

[5] Vault parameters comprise length size of the template, length of the key, and number of chaff features.

[6] Fuzzy fingerprint vaults are implementations of the fuzzy vault scheme that protect fingerprint features.

fingers are used. Under reasonable assumptions[7] that are commonly believed to be true, certain security capacities of fuzzy fingerprint vaults may be established for multiple fingers. But there are other risks that weaken effective protection of fingerprint templates via current implementations of the fuzzy fingerprint vault, and these cannot be solved by using multiple fingers. Implementations of the fuzzy fingerprint vault have to be made secure against these vulnerabilities before one begins designing multi-fingerprint vaults — which would require exhaustive evaluations on appropriate databases (e.g., see Ortega-Garcia et al. (2003)).

Early Approaches

The fuzzy commitment scheme and the fuzzy vault scheme were not the first approaches to implement biometric template protection.

Pioneer work goes back to Tomko et al. (1994). The authors proposed to protect an entire intensity image $f : \mathbb{R}^2 \to \mathbb{R}$, such as a fingerprint image. In this purpose, the authors encode a deterministic secret key as a two dimensional signal $s : \mathbb{R}^2 \to \mathbb{R}$. Now, if $F = \mathrm{FT}(f)$ and $S = \mathrm{FT}(s)$ denote the *two-dimensional Fourier transforms*[8] of f and s, respectively, then the filter function $H = S/F$ corresponds to the protected template. On authentication, if a query fingerprint image $g : \mathbb{R}^2 \to \mathbb{R}$ is provided and if $G = \mathrm{FT}(g)$ then let $s' = \mathrm{FT}^{-1}(H \cdot G)$ be the inverse transform of $H \cdot G$. If F and G are close, which is the case if f and g correlate well, then s' is close to the signal s, which can be reconstructed using certain properties of s.

The system is invariant to translation of f and g due to the incorporated Fourier transform. This is in particular useful for the biometric *fingerprints* in where fingers are likely placed on different regions of the scanner's surface.

At a first glance, it seems impossible to derive the template f or the signal s from the fraction $H = \mathrm{FT}(s)/\mathrm{FT}(f)$. However, due to the special shape of the signal s an adversary who has intercepted H has quite a good chance to successfully recover s and thus f. Moreover, the scheme is not very practical: It requires the enrolled image and the query image to correlate very well.

Even if the scheme proposed by Tomko et al. (1994) lacks of effective template protection it was the first attempt to protect biometrics. It had great influence on subsequent research on biometric template protection.

Probably, Davida et al. (1998) were the first who proposed a scheme that provides biometric template protection using a combination of traditional cryptography and error-correcting codes. They assumed, that biometric templates are encoded as binary fixed-length vectors. In this way, the difference of biometric templates correlates with the hamming distance of corresponding vectors, i.e. the number of vector positions in where two vectors differ. Roughly, the scheme works as follows. On enrollment, for an encoding of a biometric template v check bits r (in terms of an error-correcting code) are extracted from v. Furthermore, a signature $h(v)$, e.g, in terms of a cryptographic hash function, is computed. The public data is $(r, h(v))$ which corresponds to a protection of the biometric template encoded as v referred to as the *private template* by Davida et al. (1998). On authentication, a second template v' is provided. If v and v' are of sufficient similarity then a decoding attempt of v' combined with the check bits r would be successful and reveals the enrolled v.

---

[7] These comprise certain intractability assumptions (see Bleichenbacher and Nguyen (2000) and Guruswami and Vardy (2005)).

[8] Two-dimensional Fourier transform: $\mathrm{FT}(f)(u) = \int_{\mathbb{R}^2} f(x) \cdot \exp(2\pi i \cdot \langle x, u \rangle)\, \partial x$

In this case, the correctness of v can be verified using $h(v)$ and the authentication attempt is accepted; otherwise it is rejected.

The authors showed that their scheme provides effective template protection under the following two assumptions. First, it must be infeasible to produce a biometric template encoding v' of sufficient similarity to the enrolled v. Second, it must be infeasible to forge the signature $h(v)$.

Given an appropriate error-correcting code, the scheme of Davida et al. (1998) generates a private template solely from a biometric template while in the fuzzy commitment scheme it is bound to a randomly generated key.

Biometric Template Protection Schemes as a Building Block

In this thesis we focus on fuzzy fingerprint vaults. But we give a brief discussion on where to arrange them.

One may claim that biometric data, such as minutiae templates, is no private but public data: We leave the traces of fingerprints on every surface we touch; modern cameras are even able to capture irises of the human eye of sufficient quality from a reasonable distance such that the acquisition may be unseen. Thus, in a candid scenario one may omit the assumption that biometric data is private data. Furthermore, for authentication, the question whether the templates stem from living persons has to be resolved prior a deployed implementation of a biometric authentication scheme, i.e. *liveness detection*.[9] Sarier (2011) states about biometric template protection schemes, including fuzzy fingerprint vaults, the following.

> Although the security of some these systems is questionable when they are utilized alone, integration with other technologies such as digital signatures or Identity Based Encryption (IBE) results in cryptographically secure applications of biometrics.

This gives an idea that biometric template protection schemes play the role of a building block but do not solve the problem of secure biometric authentication alone. A comprehensive cryptographic discipline has been established over the last decade which also uses biometric template protection schemes as a building block. A nice source to get familiar with these topics may be the Ph.D. thesis of Neyire Deniz Sarier (Sarier (2011)) who, in particular, refers to Bringer et al. (2007a,b) and Sahai and Waters (2005).

In this thesis we face the challenge of a particular biometric, namely fingerprints, and the efficacy of the fuzzy vault scheme to protect them.

1.2 Thesis Contribution

This thesis focuses on the fuzzy vault scheme and examines its efficacy in protecting single fingerprint templates. We find inherent risks in current implementation of the fuzzy fingerprint vault — particularly, *correlation attack* — that are not resolved by moving to multiple fingerprint templates. And so we propose a novel minutiae fuzzy vault that is inherently resistant to correlation attacks.

---

[9] This is in fact a very delicate question. For some scenario, e.g., border control, it is assumed that the biometric template are acquired under supervision in where it is hard to provide a faked biometric source.

We first examine brute-force attacks, which reflects the worst case for the attacker and thus the best for system users. This is analyzed using vault parameters found in the literature and is found to pose a threat. In addition, we provide detailed analyses of different kinds of brute-force attacks against a hybrid implementation of Nagar et al. (2008, 2010). The hybrid implementation incorporates so-called *minutiae descriptors*[10] in addition to minutiae, with the purpose of improving vault security. These minutiae descriptors are protected via the fuzzy commitment scheme at a *feature-level fusion*.[11] The particular error-correcting code used in the fuzzy commitment scheme, however, has a very low *sphere-packing density*.[12] In attacks, a low sphere-packing density can be used to decouple the problem of attacking the instances of the fuzzy commitment scheme from the problem of attacking the instances of the fuzzy vault scheme. As a consequence, the partial securities essentially add up but do not multiply as it has been assumed in an analysis given by the authors. As a consequence, for codes of very low sphere-packing density the hybrid implementation does not have significant advantages over a standard implementation, which is already very vulnerable even against naive brute-force attacks.

Actually, the observation that a low sphere-packing density decreases the security of the hybrid implementation is not new. Nagar et al. (2010) noted that their analysis only holds in case a *perfect code*[13] is used. Furthermore, they proposed how to incorporate other error-correcting codes of higher sphere-packing density such that the overall security could be effectively improved. However, brute-force attacks are not the only risks, and a low sphere packing density was not taken into account to analyze a reasonable *false-accept attack*, which are attacks taking advantage out of the system's *false-acceptance rate*.

In any authentication system there is a non-zero chance that a wrong user can be falsely authenticated. This chance is referred to as the system's *false-acceptance rate*, and this rate can be exploited by an attacker. In a scenario in which a false-accept attack is conducted, we assume that the attacker has previously established a large database containing fingerprint templates that is accessible by him. Having intercepted a vault record, the attacker can iterate through all the templates in the database and simulate impostor authentication attempts. In accordance with the particular false acceptance rate, the attacker will succeed in breaking a vault record eventually. If the average time of a single impostor authentication attempt is known, then the expected time for the attacker to successfully break the vault record can be easily calculated. Therefore, we use the false-acceptance rate as well as the average time of an impostor authentication attempt determined for an own reference implementation to deduce the effort required (measured in time) for a successful false-accept attack. Moreover, we analyze the resistance of the hybrid implementation of Nagar et al. (2008, 2010), which mainly follows that of Nandakumar et al. (2007a), in an attack scenario in which the false-accept attack is combined with the information that is leaked by the sphere-packing density of the codes used for

---

[10] Informally, minutiae descriptors describe global features of the corresponding fingerprint around its corresponding minutia.

[11] On authentication, only if sufficiently many vault minutiae, i.e. encoded as vault features, can be extracted in combination with their respective minutiae descriptors (of sufficient similarity), can the correct secret key be successfully reconstructed.

[12] The sphere-packing density of an error-correcting code, informally, is the proportion of words that are correctable to a valid codeword in comparison to all words in the ambient space.

[13] Perfect codes are error-correcting codes with sphere-packing density equal to 100%.

protecting the minutiae descriptors.

We find that the minimal effort required to successfully accomplish false-accept attacks is highly alarming; such attacks can be conducted much faster than brute-force attacks. For example, against our reference implementation, a brute-force attack requires approximately 11–12 days, while a successful false-accept attack is likely to consume between 35 seconds and 18 minutes.[14] This is strong evidence supporting the contention that a single finger cannot provide reasonable cryptographic security via the fuzzy vault scheme, and that instead, multi-finger fuzzy vaults should be investigated. Unfortunately, there remain problems that cannot be resolved merely by switching to the use of multiple fingers.

Simply stated, in order to effectively protect two different fuzzy fingerprint vaults, it should not be possible to determine whether both vaults protect the same template. An intruder who has intercepted multiple applications' databases can trace user activities and determine identities that correspond. Given two matching instances of a minutiae fuzzy vault the intruder can correlate them; genuine minutiae tend to agree well in comparison to chaff minutiae, which are likely to be in disagreement. Thus, an intruder may reliably determine whether two vault records match or not, i.e. *cross-matching*. Even worse, via correlation an attacker can try to distinguish genuine minutiae from chaff minutiae. If in this way a set of vault minutiae can be extracted that contains a reasonable proportion of genuine minutiae, then the vault can efficiently be broken. Consequently, this attack is called a *correlation attack* which is an *attack via record multiplicity* (see Scheirer and Boult (2007) and Kholmatov and Yanikoglu (2008)). Current implementations of the fuzzy fingerprint vault are inherently vulnerable to these apparent security risks.

In this thesis we derive a criterion for cross-matching vault records via correlation, i.e. a criterion for determining genuine vault correspondences across different applications' databases. In combination with the correlation attack, this can pose a great vulnerability when an intruder has stolen the content of multiple applications' databases that he expects share common users. To investigate the practicability of our cross-matching criterion in combination with the correlation attack we conduct corresponding performance evaluations using our reference implementation. In particular, on an ordinary desktop computer, we are able to cross-match[15] two vaults within less than $0.05s$, where the chance for successfully finding a particular genuine vault correspondence is between 46% and 76%, depending on the security level. Given a genuine vault correspondence that has been positively cross-matched, we are able to break the corresponding vaults in virtually all cases. The average time for running a correlation attack was found to be between $1.15s$ and $222s$, again depending on the vault security level. Furthermore, we argue that the risk of cross-matching fingerprint fuzzy vaults can not be solved merely by switching to features different from minutiae. Consequently, a properly implemented fuzzy fingerprint vault must be resistant against the correlation attack. Preferably, mechanisms that protect fuzzy fingerprint vaults against the correlation attack should be conceived before multiple finger fuzzy vaults are implemented.

---

[14] The time interval for a successful false-accept corresponds to a *Clopper-Pearson confidence interval* at a confidence level of 95%.

[15] First, our cross-matcher checks whether the criterion for cross-matching is fulfilled, and if true, a subsequent correlation attack is conducted.

There has been a proposal for further improving the security of the single finger fuzzy vault using additional user passwords (see Nandakumar et al. (2007b)). For a successful authentication, the user must provide a matching fingerprint template as well as the correct user password. One consequence of the use of a password is that the correlation attack and cross-matching via correlation are effectively hardened. However, this proposal again would require mechanisms for keeping the password secret and reintroduces the problem of creating and remembering passwords introduced earlier, which were some of the very drawbacks that were meant to be solved by using biometrics.

In this thesis, a minutiae fuzzy vault is proposed that is inherently resistant against the correlation attack. The approach is based on the simple idea of rounding minutiae to a rigid grid. Each grid minutia to which a minutia is rounded encodes a genuine vault feature, while all the remaining vault features encode chaff features. In this way, chaff features do not have to be generated, because they are present. A consequence of this is that the vault features of two vault records are equal. Thus, an attacker cannot distinguish genuine features from chaff features by correlating two matching vaults. The correlation will always produce the grid, conveying no additional information.

During training tests we performed to achieve reasonable vault parameters, authentication performance at first looked promising. However, upon authentication the process for authorization of system users would be too time-consuming for a practical implementation following the ordinary decoding approach proposed in the literature, in which, in an intermediate step, a set containing candidates for genuine vault features is obtained. These sets usually contain too many chaff features for efficient genuine decoding. To deal with this impracticality we propose to modify the conventional iterative decoder that has been used for fuzzy fingerprint vault implementations throughout the literature. We propose to randomize the decoding procedure. In this way, the more iterations the decoder performs, the more likely it is that the correct key is output. The chance for a successful authentication also depends on the amount of common elements of the enrolled template and the query template. As well it depends on the number of decoding iterations: The more iterations are performed the higher is the genuine acceptance rate. Furthermore, the false acceptance rate converges to a fixed upper bound as the number of decoding iterations increases. This bound agrees with the false acceptance rate that would be achievable for the ordinary iterative decoder, although as we found, it would be impractical to perform. A higher number of decoding iterations increases the false acceptance rate but also the time needed to finish the authentication attempts. We give arguments advocating that the expected time needed to perform a successful false-accept attack is minimized when in each impostor authentication attempt only a single iteration is conducted. Thus, the implementation's vulnerability against the false-accept attack may be analyzed for a single decoding iteration. A higher number of decoding iterations is to improve the genuine acceptance rate. The only limiting factor may be the time that one wants to provide for authorizing users.

We report the results of a performance evaluation that we conducted for our implementation. The FVC 2002 DB2 fingerprint database that we used is the same that has been used for performance evaluations in virtually all fuzzy fingerprint vault implementations found in the literature (see Maio et al. (2002)). For most of these implementations, the authentication rates were calculated using only a subset of the database—those templates containing high-quality fingerprint images (which reflects the situation in which system users cooperate). In these cases, genuine acceptance rates were estimated using a sample size of only 100, compared to the entire database yielding 2800 genuine authentication attempts. As might be expected, the genuine acceptance rates were found to be higher than they would be if evaluated over the entire set of fingerprints; but in addition these results are statistically less significant than if the entire database is used. Therefore, we report authentication rates that we measured for both the entire database as well as its subset containing fingerprint images of high quality. The genuine acceptance rate as well as the false acceptance rate that our implementation delivers turns out to perform very well when compared to other implementations found in the literature. For example, our implementation is capable of providing a genuine acceptance rate of 94% with no false accepts observed. The implementation of Li et al. (2010), for which (to the best of our knowledge) the best authentication rates using the FVC 2002 DB2 had been achieved, reported a genuine acceptance rate of 92% given no observed false accepts. For the entire FVC 2002 DB2 database we obtain a genuine acceptance rate of approximately 85% while Li et al. (2010) achieved 72%.[16] From these tests we conclude that it is possible to implement fuzzy fingerprint vaults that are resistant against the correlation attack without loss of authentication performance.

Furthermore, we evaluated the security of our implementation against brute-force and false-accept attacks under ideal conditions for the attacker. While our implementation provides good security against brute-force attacks, it does not provide improved resistance, relative to other implementations, against false-accept attacks. It appears that some measurable risk of a false-accept is inherent to the technique of matching a single fingerprint, and can only be ameliorated by other means such as the use of multiple fingers. We did not directly investigate our implementation to assess its vulnerability against the correlation attack (and thus cross-matching via correlation), since it is inherently resistant against it. But there is one last point concerning cross-matching to consider.

A distinct benefit that arises from selecting vault features from a rigid feature set is that the *modified fuzzy vault construction* of Dodis et al. (2008) can be used to protect the templates. The modified fuzzy vault construction avoids storing chaff features in the vault (which can become very large in number), and instead only builds data of genuine vault features. This has the advantage that the storage size of the vault is of the same order as the template size. Furthermore, from the modified vault construction an instance of the ordinary vault construction can be obtained. As a consequence, authentication performance, security against brute-force attacks, and security against false-accept attacks remain the same. However, to some extent an intruder might be able to cross-match, and so we discuss this point. Furthermore, we propose a simple countermeasure, which hardens the aforementioned possibility to cross-match, that applies a random permutation process to the templates which is similar to the idea of Kelkboom et al. (2011) to prevent the *decodability attack* in the fuzzy commitment scheme.[17]

---

[16] It must be mentioned that we did not implement an automatic mechanism for aligning the query fingerprints to the vault. Rather we evaluated vault performance in a well-solved alignment framework. Probably, with an automatic alignment mechanism our genuine acceptance rate would be worse.

[17] we refer to Kelkboom et al. (2011) for details of the decodability attack.

1.3 Thesis Outline

In Chapter 2 we summarize the basic concepts used in the thesis concerning traditional cryptography. In particular we focus on cryptographic hash functions and how they can be used to provide secure password-based authentication. After an overview of the discipline of fingerprint recognition, we address the concept of error-correcting codes. Finally, we define the details of the function of the fuzzy commitment scheme and discuss an implementation proposed by Hao et al. (2006) that provides secure authentication based on the iris of the human eye.

In Chapter 3 we describe the general function of the fuzzy fingerprint vault and then give an overview of implementations found in the literature. Most of these implementations are based on fingerprint minutiae, which require an accurate alignment of the query fingerprints to the vault. Therefore, we give an overview of different proposals found in the literature for dealing with alignment in a minutiae fuzzy vault. Afterwards, we describe the details of our own reference implementation, which mainly follows the description of Nandakumar et al. (2007a). A major difference from the implementation of Nandakumar et al. (2007a) is that we did not implement an automatic alignment of query fingerprints to the vault due to the lack of satisfactory proposals given in the literature. Rather we conducted our experiments in a well-solved alignment framework in which an accurate alignment is presumed. Finally, we report the results of a performance evaluation that we conducted with our reference implementation.

Chapter 4 contains security analyses of the fuzzy fingerprint vault. We begin by establishing a simple brute-force attack as a reference and report expected timings needed to perform a successful attack against vault parameters of different implementations found in the literature. In particular, we analyze the hybrid implementation of Nagar et al. (2008, 2010) against the brute-force attack. In an intermediate section, we discuss how the additional data created to assist in vault alignment can actually be used to accelerate attacks. Subsequently, we formulate the details of the false-accept attack. To demonstrate its power we report results of an evaluation of the false-accept attack against our reference implementation. Also, we discuss how the false-accept attack in combination with a low sphere packing density of the code used to protect minutiae descriptors can be used to break the hybrid implementation of Nagar et al. (2008, 2010). Afterwards, we describe the correlation attack in detail. A related criterion for cross-matching vaults is derived and then, using our reference implementation as an example, we evaluate the performance of a cross-matcher using this criterion.

In Chapter 5 we develop a minutiae fuzzy vault that is inherently resistant against the correlation attack. Furthermore, a self-contained description of the modified fuzzy vault construction proposed by Dodis et al. (2008) is included in the chapter. We report results of a training that we conducted to obtain reasonable vault parameters. Moreover, we propose a randomized decoder with the purpose to make vault authentication more practical. Results of a performance evaluation that we conducted on a public domain fingerprint database are given. Analysis of the brute-force and false-accept attack are given as well. Furthermore, the chapter contains a discussion about remaining possibilities to perform cross-matching, in particular when the modified vault construction is used. Finally, a comparison of our implementations with other implementations found in the literature is given.

Chapter 6 contains a final discussion as well as our view of the problems remaining to be solved before an implementation of the minutiae fuzzy vault provides superior cryptographic security.

1.4 Acknowledgments

First I would like to express my gratitude to my supervisor Preda Mihăilescu for giving me the opportunity to work on the interesting topic of fingerprints and security. When I got confused by the vast amount of disciplines my research was concerned with, he always helped me out in clarifying my thoughts. I am very thankful for the freedom to work out the thesis' contents by myself. Also, I am deeply indebted to Axel Munk for his help. Without his support it would have been certainly not be possible for me to write this thesis. To Carsten Gottschlich I am much obliged for fruitful discussions on the thesis' topics and for sharing his experiences and his comprehensive knowledge with me. A large bucket of thanks go to Johannes Merkle. His enthusiasm and his outstanding understanding frequently inspired me and helped me in finding views on the topic from different perspectives. He helped me in establishing and proving Lemma 3.2.1. Furthermore, he advised me of the possibility to cross-match multiple templates that are protected via the modified fuzzy vault construction (see Section 5.7.4). I thank Thomas Hotz for helpful consultations. In particular, he showed me how one can make reasonable guesses of a fingerprint's orientation fields when only its singular points are given (see Figure 4.1). Furthermore, I really appreciate the competent assistance of Mark Voss in eliminating typos as well as spelling errors. I am most grateful to the *Deutsche Forschungsgemeinschaft "Identifikation in mathematischen Modellen: Synergie stochastischer und numerischer Methoden"* for their financial support. The annual workshops in Goslar within the mountain range Harz always have been highlights for me and I enjoyed them very much. Besides the interesting lectures, talks, as well as the social events I will keep the evening game *Werwölfe* in good memories. To my colleagues and office mates Hoang Duy Thai, Kwang-Rae Kim as well as to my former office mates Robert Schieweck, Rebecca von der Heide, and Sophie Frick I thank for the nice atmosphere they provided. Similarly, to the members of the Institute of Mathematical Stochastics, especially to Andrea Krajina, Anja Frakstein, Karola Kombrink, Steffi Greiner, Stephan Huckemann, and Thomas Rippl, I thank for their helpfulness, kindness, fulfilling company during lunch, and enjoyable coffee breaks. For their virtual omnipresence and support, I thank my parents, grandparents, as well as my sisters and brothers (Miriam, Maren, Philipp, and Michael). A special thank goes to my stepfather Ulrich. Moreover, I would like to thank the rest of my family — I truly appreciated and enjoyed their company for which I am very grateful. Last but certainly not least I express my deepest gratitude to my girlfriend Sjaan. She made my life more pleasant to live in for the last five years.

2. Basics

In this chapter, we establish basic concepts that are needed during this thesis. We start with an overview to cryptography and then give a summary of fingerprint recognition with a focus of fingerprint minutiae. The concept of error-correcting codes are reviewed afterwards which is an important tool to combine cryptography with biometric recognition. Finally, the fuzzy commitment scheme is described and an example implementation is discussed that is to protected templates of irises of the human eye.

2.1 Cryptography

Cryptography is the science of keeping information secret. In recent decades, the advent of computer-aided information exchange has greatly increased demands upon the field, requiring it to mature quickly and develop flexible yet robust practices. These practices address the following important objectives within cryptography:

1) **Confidentiality**: Information is made unreadable to unauthorized persons or processes. This is, in part, achievable by encrypting data (see Section 2.1.1);

2) **Authenticity**: Access to stored data is controlled by verifying potential readers. This is an example where *cryptographic hashing techniques* can be applied (see Section 2.1.2);

3) **Integrity**: Information is prevented from being changed during transmission or on storage. Integrity of data can be verified using *check-sums*, which are related to hashing techniques;

3) **Accountability**: Information is definitely linked to its creator. *Public-key cryptography* is a commonly used approach to implement such *digital signatures* (El Gamal (1985); Rivest et al. (1983));

Details on cryptographic techniques and its theory can be found in a variety of standard textbooks concerned or related with this topic (Buchmann (2003); Cormen et al. (2001); Shoup (2005); Joachim von zur Gathen, Jürgen Gerhard (2003)). For the purposes of this thesis, we give a brief summary of cryptographic encryption of data, which is concerned with the above mentioned confidentiality. Afterwards we review the concept of cryptographic hashing with its application of providing authenticity.

2.1.1 Encryption/Decryption

The cryptography provides useful tools to implement privacy by providing methods for encrypting and decrypting data, which are based on the mathematical concept of *cryptographic systems* or (for short) *cryptosystem*. A very important concept, is the concept of *block ciphers* which are a special kind of cryptosystems.

Block Ciphers

A *block cipher* consist of block texts

$$\mathbf{B} = \Sigma^n \tag{2.1}$$

for an alphabet Σ and a block length n. Furthermore, it contains a family of *encryption functions*

$$\mathbf{E} = \{\mathrm{enc}_\kappa : \mathbf{B} \to \mathbf{B}\}_{\kappa \in \mathbf{K}} \tag{2.2}$$

and *decryption functions*

$$\mathbf{D} = \{\mathrm{dec}_\kappa : \mathbf{B} \to \mathbf{B}\}_{\kappa \in \mathbf{K}} \tag{2.3}$$

where $\mathbf{K}$ is a universe of keys. Encryption and decryption functions have the property that for each *encryption key* $\kappa_e \in \mathbf{K}$ there exists a *decryption key* $\kappa_d \in \mathbf{K}$ such that for all texts $m \in \mathbf{B}$ the identity

$$\mathrm{dec}_{\kappa_d}(\mathrm{enc}_{\kappa_e}(m)) = m \tag{2.4}$$

holds. More informally, for each encryption key κ_e there exists a valid decryption key κ_d.

A simple cardinality argument shows that encryption and decryption functions for a fixed key κ are necessarily permutations of $\mathbf{B} = \Sigma^n$.

We will assume soundness of a cryptosystem throughout: It is infeasible to derive m from $\mathrm{enc}_{\kappa_e}(m)$ unless κ_d is known.

A *symmetric block cipher* is a block cipher in where each encryption key κ is even a valid decryption key. Otherwise, if encryption keys are different from their corresponding decryption keys and if the encryption keys can not be feasibly derived from a given decryption key then the block cipher is referred to as an *asymmetric block cipher*.

Asymmetric block ciphers are in particular useful for implementing digital certificates for providing accountability: Assume a user received unprotected data along with data that has been encrypted using an encryption key κ_e; if the decrypted data using κ_d agrees with the unprotected data he can be confident that it originated from a person who knows κ_e.

Examples

The *Rivest-Shamir-Adleman cryptosystem* (*RSA*) is an asymmetric block cipher (Rivest et al. (1983)) that achieves its robustness from the (assumed) hardness of factoring a product of two large prime numbers.

The *data encryption standard* (*DES*) (National Institute of Standards and Technology (1999)) is a symmetric block cipher of $\mathbf{B} = \{0,1\}^{64}$ that uses 56-bit keys, i.e. $\mathbf{K} = \{0,1\}^{56}$. DES is a variation of the *Feistel cipher* (Feistel (1973)).

The 56 bit keys of the DES have become too small to ensure resistance against cryptographic attacks using modern hardware. As a countermeasure the National Institute of Standards and Technology (2001) announced the *advanced encryption standard* (*AES*) which defines key-lengths of sufficient length to resist attack using current hardware. Again, AES is a symmetric block cipher. It works on blocks of 128-bit length. The key-lengths are 128, 192, or 256 bits. AES is a specialized version of the *Rijndael cipher* (Daemen and Rijmen (2002)).

2.1.2 Cryptographic Hash Functions

Encryption of data prevents the information for being read by an unauthorized person. But in times of computer-aided communication, how can one distinguish between authorized and unauthorized persons? One solution used by server/client applications requires the enrollment of a user with a password. To enable such an authentication scheme, user names along with their corresponding password, respectively, are stored on a database. Such a scheme is currently used by *email providers*, *on-line banking*, *on-line shops*, on-line auctions, *multi-user systems*, and so on. Storing passwords *in clear*[1], however, implies security issues: What happens if the content of the database is copied unnoticed? Will the theft be recognized? Who is even allowed to read the content stored on the database? What happens if the password is also used for another account? One approach helping in resolving the above security questions is provided by the concept of *cryptographic hash functions*.

Definition 2.1.1 (Hash Function)**.** *Let Σ be a non-empty alphabet. A map*

$$h : \Sigma^* \to \Sigma^n, \quad n \in \mathbb{N}$$

is called a hash function*.*

In the definition, Σ^* denotes the space of all strings over the alphabet Σ. Since $\Sigma^* \supset \Sigma^n$, hash functions are never injective. A hash function which is intended to prevent the input to be recovered only if its *hash value* is known must have the following additional property.

Definition 2.1.2 (One-Way Function)**.** *A function $f : X \to Y$ is called a* one-way function *if*

- *for any $x \in X$ the value $f(x)$ can be easily computed, but*
- *it is infeasible to find a* collision*, i.e. find $x' \in X$ such that $h(x') = h(x)$. This property is commonly referred to as* weak collision resistance*.*

---

[1] *In clear* means unprotected.

The above requirements on a one-way function can be defined more theoretically in terms of *complexity theory* (Papadimitriou (1994)). However, even if a function is one-way in its asymptotic behavior, the computation of a collision might be feasible due to parameter selected: In an implementation n is fixed and does not grow towards infinity. For this reason we will keep our informal definition for a one-way function.

Definition 2.1.3 (Cryptographic Hash Function)**.** *A hash function* $h : \Sigma^* \to \Sigma^n$ *that in addition is one-way, is called a* cryptographic hash function*.*

For some applications, such as for digital signatures, a cryptographic hash function is required to have the stronger property of being *strongly collision resistant*, i.e. it is hard to find any (x, x') with $h(x) = h(x')$, not only if either x or x' is fixed.

Secure Hash Standard

The National Institute of Standards and Technology (1995) announced a standard for a cryptographic hash function which maps an input word to a 160-bit value. This standard is referred to as the *secure hash standard* (*SHS*). The SHS can be evaluated using the *secure hash algorithm* (*SHA-1*).

The National Institute of Standards and Technology (2002) announced additional secure hash standards, each collectively denoted as *SHA-2*, which map an input word to a 224 (*SHA-224*), 256 (*SHA-256*), 384 (*SHA-384*), and 512 (*SHA-512*) bit-value, respectively.

Recently in October 2012, there has been elected a winner of a competition to become the successor of the SHA-2 family, for being announced as *SHA-3*.[2]

Password Authentication using Hash Functions

In a server/client application, the password $x \in \{0,1\}^*$ in clear is not stored along with the user-name, but rather its hashed value $h(x) \in \{0,1\}^n$. Since $h : \{0,1\}^* \to \{0,1\}^n$ is required to be a cryptographic hash function, an unseen impostor who has intercepted a user's hashed password $h(x)$ will not be able to fake a login, since he will not be able to reproduce an $x' \in \{0,1\}^*$ with $h(x') = h(x)$ within a feasible amount of time.

Usually, the user passwords are *salted* before the hash values are computed: The passwords are concatenated with a string that is associated with the service provider (e.g., by the web address). This is to prevent an impostor from cross-matching between different databases of different providers. Moreover, salting passwords prevents the search of *rainbow-tables*[3] to quickly determine weak passwords.

2.2 Fingerprints

Even the most collision-resistant hash function becomes useless, when only weak passwords are used for authentication. On the other hand, passwords that users are able to keep in mind usually are of low entropy. Service providers as well

---

[2] `http://csrc.nist.gov/groups/ST/hash/sha-3/winner_sha-3.html`

[3] Rainbow tables are tables listing hash values along with weak or typical passwords.

as authorized users usually have an interest that no impostor is able to intrude into the system impersonating an authorized user. Currently, it is discussed to achieve high-entropy evidence for securing user-logins by the of use biometric sources rather than passwords. One biometric trait of a human being are his fingerprints. The security questions, however, remain similar: An intruder who has intercepted a humans fingerprint template might simulate an authorized user's login. In order to prevent an unencrypted biometric template from being read from the database, the templates have to be protected.[4] Cryptographic hash functions, however, are not appropriate for hashing biometric templates, since even small variations in the input will, with overwhelming probability, produce completely different hash values.

In order to better understand the design and implementation issues that a scheme for protecting fingerprint templates has to cope with, an overview of the particular biometric *fingerprints* is given in this section. For a comprehensive overview we refer the reader to Maltoni et al. (2009).

A fingerprint consists of traces left on a surface[5] by the ridges on a fingertip. It is very unlikely that the same traces could be left by the same finger a second time due to its *displacement*, *pressure on the surface*, *distortion* (skin elasticity), *skin condition* (dry/wet), and other reasons (such as *noise/dirt*).

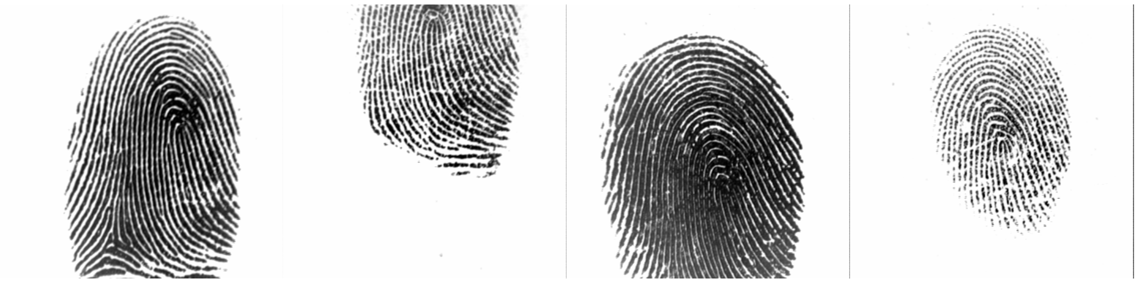

(a) normal traces of ridges (b) displaced (c) heavy pressure (d) light pressure

Figure 2.1: different scans of the same fingerprint

When one has to decide whether two fingerprints have been acquired from the same finger, the above effects have to be taken into account. One approach to perform comparison, i.e. *matching*, is to decide how well two fingerprints agree. For matching one has to tolerate for rotation, displacement, variation, and noise. These side conditions emphasize that it is unlikely that an algorithm exists which is able to positively match two fingerprints only if they have been extracted from the same finger.

2.2.1 Minutiae

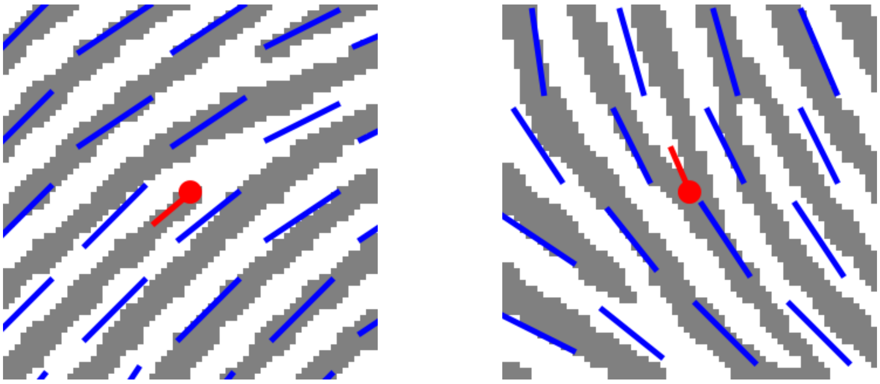
Prior to matching, it has become common to determine the positions where the ridges along their flow of a fingerprint end or bifurcate. These characteristic locations are known as *minutiae* and include *ridge endings* and *ridge bifurcations*.[6]

---

[4] Even if there are other risks how an adversary can gain knowledge about a user's fingerprint.

[5] For our purposes throughout, a fingerprint is acquired as a digital, grayscale, rastered image, e.g., by a scanner.

[6] Some references use further classes.



(a) Ridge ending directed to from where the ridge ends

(b) Ridge bifurcation directed to where a ridge bifurcates

Figure 2.2: Minutiae (red) which are directed endings and bifurcations of the fingerprint's ridge (gray); the directions correspond to the flow of the ridges (blue)

A ridge ending is the coordinate where the ridge flow ends abruptly while a ridge bifurcation is the coordinate where a ridge branches into two ridges. In addition to mere coordinate, a minutia is usually associated with a *minutia direction* or *angle*. The direction of a bifurcation can be defined as the angle parallel to the local ridge flow directed to where the ridge bifurcates. Then the direction of an ending is defined as the angle parallel to the local ridge flow directed oppositely to where the ridge ends. The set of all minutiae extracted from a fingerprint image is called a *minutiae template*.

Reliable extraction of minutiae is a challenging task. Usually, a preprocessing step enhancing the fingerprint image is performed before features are extracted. However, due to the side effects listed above, minutiae might be spurious or missing, even if the most reliable extractor available has been used. An approach to separate spurious minutiae from true minutiae is to associate a *quality index* (Chen et al. (2005)) with each minutia that is extracted. But the problem of spurious and missing minutiae is not solved completely: Minutiae might be of low quality although they are not spurious and of high quality although they are spurious.

2.2.2 Matching

Matching can be done on the basis of two fingerprint's minutiae templates. A simple approach to compare two minutiae templates $T = \{(a, b, \theta)\}$ and $T' = \{(a', b', \theta')\}$ is to sum up the scores of the n best matching minutiae correspondences.

For this purpose, the distance between two minutiae $\mathfrak{m} = (a, b, \theta)$ and $\mathfrak{m}' = (a', b', \theta')$ can be defined as

$$d(\mathfrak{m}, \mathfrak{m}') = \sqrt{(a-a')^2 + (b-b')^2} + w \cdot \min(\ |\theta - \theta'|\ ,\ |2\pi + \theta - \theta'|\ ). \tag{2.5}$$

$d(\cdot, \cdot)$ serves as a measure of similarity between two minutiae, where w is a weight controlling how significantly the minutiae angles are taken into account. For each correspondence of minutiae in T and T' the distance d is computed and stored in a list. The list is sorted increasingly and the first n distances in the list are summed up. This sum is denoted by $D(T, T')$. This expression is then referred to as the distance

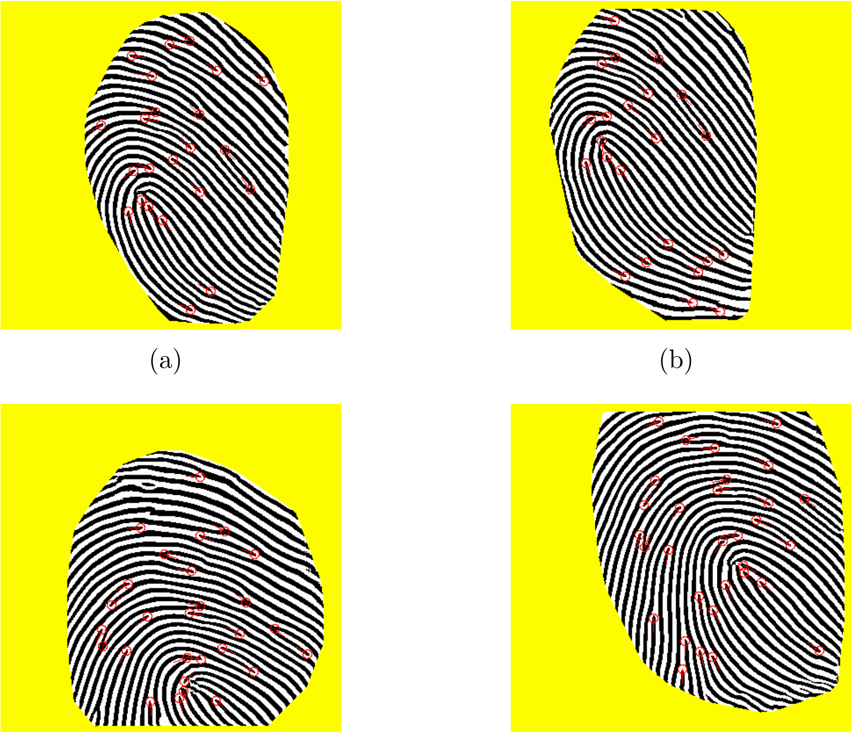

(a) (b) (c) (d)

Figure 2.3: Directed fingerprint minutiae (red) from the same finger

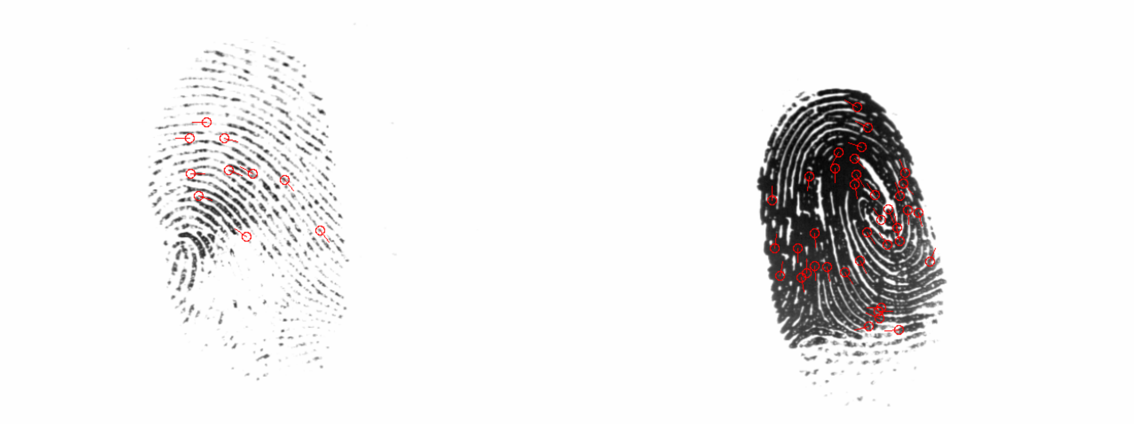

(a) missing minutiae (b) spurious minutiae

Figure 2.4: The effect of dry and wet finger on minutiae extraction

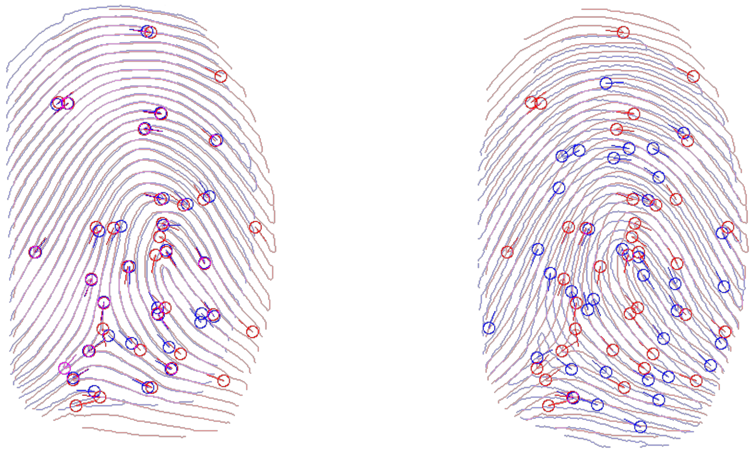
between T and T'. A global threshold t is consulted to decide whether T and T' are matching. If the sum is smaller than or equal to t, i.e.

$$D(T, T') \leq t, \tag{2.6}$$

the template T and T' are considered to match. Otherwise, if the sum is greater than t, i.e.

$$D(T, T') > t, \tag{2.7}$$

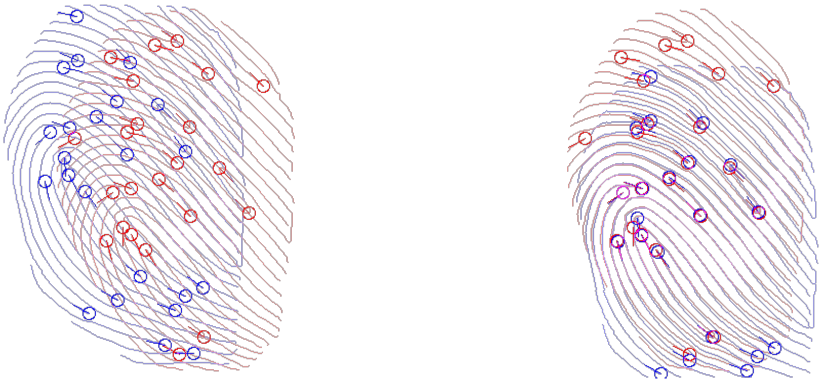
T and T' are considered not to match.



(a) (b)

Figure 2.5: Matching (a) and non-matching (b) minutiae templates

2.2.3 Alignment



(a) not aligned (b) aligned

Figure 2.6: Minutiae need to be aligned before evaluating how well they agree.

Even if two minutiae templates are extracted from the same finger, their minutiae might be in disagreement due to displacement, rotation, or partial overlap. In order to assert that matching minutiae are spatially in agreement, one of the template has to be rotated and moved. This task is performed in a preliminary step, referred to as the *alignment step*.

A straightforward approach is to iterate minutiae correspondences: For each *reference minutiae correspondence* $(a_0, b_0, \theta_0) \leftrightarrow (a'_0, b'_0, \theta'_0)$ the spatial movement

$$f : (x,y)^\top \mapsto \left(\begin{pmatrix} \cos(\theta_0 - \theta'_0) & -\sin(\theta_0 - \theta'_0) \\ \sin(\theta_0 - \theta'_0) & \cos(\theta_0 - \theta'_0) \end{pmatrix} \left(\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} a'_0 \\ b'_0 \end{pmatrix} \right) + \begin{pmatrix} a_0 \\ b_0 \end{pmatrix} \right)^\top , \quad (2.8)$$

which maps (a'_0, b'_0) to (a_0, b_0) and is a rotation by the angle $\theta_0 - \theta'_0$, is determined. The template T' is spatially moved using f attempting to align its element to T. More precisely, determine

$$f(T') := \{(\ f(a', b')\ ,\ \theta' + \theta_0 - \theta'_0) \mid (a', b', \theta') \in T'\}$$

A movement f resulting in a minimal $D(T, f(T'))$ is selected and $f(T')$ is then considered as the aligned minutiae template of T' to T. Further refinement steps might be applied to decrease $D(T, \cdot)$ (Eggert et al. (1997)).

2.2.4 Other Matching Methods

The above method has the disadvantage that it does not account for non-linear distortion, e.g., caused by varying pressure and skin elasticity. Jain et al. (1997) proposed a method that can, given an initial alignment, account for non-linear distortion by tracing minutiae matches and updating the alignment step by step. Furthermore, to compute the initial alignment, Jain et al. (1997) proposed to incorporate ridges associated with the reference minutiae correspondences rather than merely minutiae correspondences.

Matching methods that are based on an alignment step, are commonly referred to as *global matching methods*. In contrast to global matching methods there are approaches that use structures that are invariant to rotation and translations. Such methods are referred to as *local matching methods*, for examples the [7]*minutiae local structure* proposed by Jiang and Yau (2000).

Local matching methods have been shown to be more robust against non-linear distortion compared to global methods. Furthermore, local matching can be done much faster than global matching since there is no alignment step. However, local matching ignores the absolute positions of minutiae on a fingertip which add a lot of individuality to fingers. Hybrid methods, e.g., local matching followed by a *consolidation step*, have the potential to bring the best of both worlds together.

2.2.5 Identification Performance

There are several reasons why two fingerprint templates acquired from the same finger are different. As a consequence, fingerprint matchers are expected to make matching mistakes at some error rate. A fingerprint matcher makes two kind of mistakes: First, it might decide that two fingerprint templates belong to the same finger when they do not; second, it might decide that two fingerprint templates belong to different fingers when both have been extracted from the same finger.

To measure a fingerprint matcher's accuracy, two rates are commonly associated with it: the *genuine acceptance rate* (or *genuine match rate*) and the *false acceptance rate* (or *false match rate*).

Genuine Acceptance Rate and False Non-Match Rate

The *genuine acceptance rate* is defined to be the average percentage of correct positive matches. In other words, if the genuine acceptance rate is reported to be

---

[7] i.e. a feature vector built out of every minutia and its two nearest-neighbor minutiae

$\mathrm{GAR} \in [0,1]$ then within N *genuine authentication attempts*[8] the expected number of successful matches of two different fingerprint templates from the same finger is $\mathrm{GAR} \cdot N$. The error rate $1 - \mathrm{GAR}$ is called *false non-match rate*.

False Acceptance Rate

Analogously, the *false acceptance rate* is the average percentage of incorrect positive matches. A matcher of false acceptance rate $\mathrm{FAR} \in [0,1]$ is expected to wrongly report $\mathrm{FAR} \cdot N$ positive matches within N comparisons of two fingerprint templates from different fingers, i.e. within N *impostor authentication attempts*.

2.2.6 Confidence of Identification Performance

At this point it seems worthwhile to state some facts about the reliability of genuine and false acceptance rates as defined above. Without loss of generality, we can restrict our considerations to the false acceptance rate only.

Usually a system's false acceptance rate FAR is determined by a *point estimation*: Simulate a number, say N, of random independent impostor authentication attempts and count the number m of successful authentications. The false acceptance rate is then estimated as $\widehat{\mathrm{FAR}} = m/N$. A critical question is how reliable such an estimation can be.

Confidence Interval

Let FAR be the (unknown) false acceptance rate of a system. Assume that within a test of N independent simulated impostor authentication attempts exactly m false accepts occur. Assume the false acceptance rate is estimated as $\widehat{\mathrm{FAR}} = m/N$. The true false acceptance FAR rate might be smaller ($\mathrm{FAR} \leq \widehat{\mathrm{FAR}}$) or larger ($\mathrm{FAR} > \widehat{\mathrm{FAR}}$) than estimated. However, it is 100% certain that the true false acceptance rate FAR lies within $[0,1]$. Rather than the whole interval $[0,1]$ it would be useful to estimate a smaller range $[\mathrm{FAR}_0, \mathrm{FAR}_1]$ depending on the number of positive false accepts m such that $\mathrm{FAR} \in [\mathrm{FAR}_0, \mathrm{FAR}_1]$ for most cases. More precisely, for a fixed *confidence level* γ let FAR_0 and FAR_1 (depending on m) be such that $\mathrm{FAR} \in [\mathrm{FAR}_0, \mathrm{FAR}_1]$ is true in $100\gamma\%$ of all tests. The interval $[\mathrm{FAR}_0, \mathrm{FAR}_1]$ is called γ-*confidence interval*.

Clopper-Pearson Interval

Clopper and Pearson (1934) derived a method for computing an exact γ-confidence interval when m positive false accepts within N impostor authentication attempts have been experienced.

The probability of m successful impostor authentication attempts can be modeled by the probability mass function of the binomial distribution, i.e.

$$B(\ m \mid \mathrm{FAR}, N\ ) = \binom{N}{m} \cdot \mathrm{FAR}^m \cdot (1 - \mathrm{FAR})^{N-m}. \tag{2.9}$$

---

[8] A genuine authentication attempt is a scenario where two templates extracted from the same finger are compared.

Consequently, write the *binomial distribution function*

$$P(\ x \mid \mathrm{FAR}, N\ ) = \sum_{i=0}^{\lfloor x \rfloor} B(\ i \mid \mathrm{FAR}, N\ ). \tag{2.10}$$

Assume that $0 < m < N$ and let $\alpha = (1-\gamma)/2$. Under the condition that m positive false accepts within N impostor authentication attempts have been observed let FAR_1 be such that

$$P(\ m \mid \mathrm{FAR}_1, N\ ) = \alpha. \tag{2.11}$$

As a consequence, when FAR_1 is chosen as above in all but $100\alpha\%$ of the tests one can exclude the possibility that $\mathrm{FAR} > \mathrm{FAR}_1$. Analogously, let FAR_0 be such that

$$1 - P(\ m-1 \mid \mathrm{FAR}_0, N\ ) = \alpha, \tag{2.12}$$

i.e. in all but $100\alpha\%$ of the tests it can be excluded that $\mathrm{FAR} < \mathrm{FAR}_0$. The interval $[\mathrm{FAR}_0, \mathrm{FAR}_1]$ is called *Clopper-Pearson interval.*

Special cases occur if $m = 0$ or $m = N$. If $m = 0$ then $\mathrm{FAR}_0 = 0$, but to still obtain a confidence interval of confidence level γ the upper limit FAR_1 must be chosen such that $\mathrm{FAR} \geq \mathrm{FAR}_1$ in all but $100(1-\gamma)\%$ of the tests rather than $\alpha = (1-\gamma)/2$. Conversely, if $m = N$ then $\mathrm{FAR}_1 = 1$ but $\mathrm{FAR} \leq \mathrm{FAR}_0$ in all but $100(1-\gamma)\%$ of the tests.

To summarize, for $\gamma \in (0,1)$ assume that m false accepts have been observed within N impostor authentication attempts. The limits of the Clopper-Pearson interval FAR_0 and FAR_1 are such that

$$\begin{aligned} P(\ m-1 \mid \mathrm{FAR}_0, N\ ) &= \begin{cases} 0 & \text{if } m = 0 \\ \gamma & \text{if } m = N \\ \frac{1+\gamma}{2} & \text{else} \end{cases} \\ P(\ m \mid \mathrm{FAR}_1, N\ ) &= \begin{cases} 1-\gamma & \text{if } m = 0 \\ 1 & \text{if } m = N\ . \\ \frac{1-\gamma}{2} & \text{else} \end{cases} \end{aligned} \tag{2.13}$$

Rule of Three

In the case where no false accepts ($m = 0$) have been observed and if a 95%-confidence interval is sought (which is a very commonly used confidence level), there is a simpler way to compute a 95%-confidence interval (Hanley and Lippman-Hand (1983)). The *rule of three* states that

$$[0, 3/N] \tag{2.14}$$

is a confidence interval of confidence level at least 95% for the false acceptance rate when no false accepts have been observed within N impostor authentication attempts. For a derivation as well as a discussion of the rule of three we refer to Jovanovic and Levy (1997).

Example

For example, if no false accepts occurred within a test of $10,000$ impostor authentication attempts, for a confidence level of 95% with the help of the corresponding Clopper-Pearson interval we can state that the false acceptance rate will be less than 0.03%. Furthermore, the rule of three leads to an upper bound of 0.03% of the false acceptance rate (with a confidence of 95%).

In Table 2.1, for varying confidence levels, upper bounds of the false acceptance rate are listed.

Table 2.1: Upper bounds of the false acceptance rate for given confidence when no false accepts have been observed within $10,000$ impostor authentication attempts

confidence γ	$= 1\%$	$= 50\%$	$= 90\%$	$= 95\%$	$= 99\%$
FAR	$< 1.01 \cdot 10^{-6}$	$< 7 \cdot 10^{-5}$	$< 0.024\%$	0.03%	$< 0.047\%$

Discussion

In a test to determine the false acceptance rate of an authentication system one can estimate it by a point estimation. However, when a false acceptance rate originating from a point estimation is reported one should understand that the true false acceptance rate varies from the estimated value with high probability. But for a specific confidence the range of the true false acceptance rate can be described in terms of a confidence interval.

In the above example we have considered upper bounds of the false acceptance rate at different levels of confidence when no false accepts have been observed within $10,000$ impostor authentication attempts. We stress that no occurred false accepts does not imply a zero false acceptance rate. Furthermore, the observation of zero false accepts does not assume that the false acceptance rate to be negligible: When $10,000$ impostor authentication attempts are considered then the false acceptance rate will be at most $1.01 \cdot 10^{-6}$ with a confidence of 1%; thus, if no additional assumptions can be made one must conclude that with a confidence of 99% the true false acceptance rate will be even larger than $1.01 \cdot 10^{-6}$.

On the other hand, usually a system's matcher makes its decision depending on a threshold (see (2.6) and (2.7)). The smaller the threshold the more agreement between two biometric templates is required for the matcher to output a successful match. Now if for some threshold no false accepts have been observed then the true false acceptance rate will be non-zero with high probability. However, it is often safe to assume that the true false acceptance rate for even a lower threshold will decrease. Although, it will be still non-zero with high probability. Unfortunately, to the best of my knowledge there is no substantiated method to bound the false acceptance rate along with a similarity threshold.

The above discussion emphasizes that a measured false acceptance rate alone is not sufficient to verify a system's security against impostor authentication attempts.

Another way to furthermore decrease a matcher's false acceptance rate is to fuse different features. In the following we give an overview to further fingerprint features different to minutiae.

2.2.7 Further Features

Fingerprints contain more information than the established minutiae. Additional information can be incorporated to support matching of fingerprints, e.g., in order to increase low genuine acceptance rate or to decrease high false acceptance rates of a matcher. We outline the most commonly used among them.

Ridge Orientation

The *local ridge orientation* at the coordinate (x, y) of a fingerprint's image foreground is the angle that a line running parallel to the ridge flow at (x, y) forms with the x-axis. The ridge orientation has no direction, i.e. it does not affect the ridge orientation whether the lines parallel to the ridge orientation run in the opposite direction. Thus, the ridge orientation is an angle value in the range $[0, 180)$, $[0, \pi)$, etc., depending on what system for representing angles is chosen.

A pixel's ridge orientation can be estimated by different approaches, e.g., based on gradients (Kass and Witkin (1987)), in the frequency domain (Kamei and Mizoguchi (1995)), or global methods like using the *line sensor* (Mieloch et al. (2008)) as proposed by Gottschlich et al. (2009).

Ridge Frequency

The *local ridge frequency* is the inverse of the local average *inter-ridge distance* perpendicular to the ridge orientation. Thus, since a local ridge frequency is defined with the help of the local ridge orientation a good approximation to the inter-ridge distance necessarily requires a good approximation of the local ridge frequency.

A first approach to estimate a pixel's ridge frequency (Hong et al. (1998)) was to count the number of pixels between two consecutive mean intensity peaks in the rows of a window oriented along the local ridge orientation. Unfortunately, this approach lacks of robustness in regions where the ridge flow is strongly bending (i.e. where the *ridge curvature* is high). In order to achieve more robustness in regions of high ridge curvature, Gottschlich (2011) extended this approach by using curved windows that are bending along the ridge orientation.

Orientation/Frequency Field

A fingerprint image's *orientation field* (*frequency field*) consists of an estimation of all its ridge orientations (ridge frequencies) on the foreground.

Good estimations of the orientation field as well as of the frequency field are critical to fingerprint enhancement steps that are based, for instance, on the *Gabor filter* (Hong et al. (1998)).

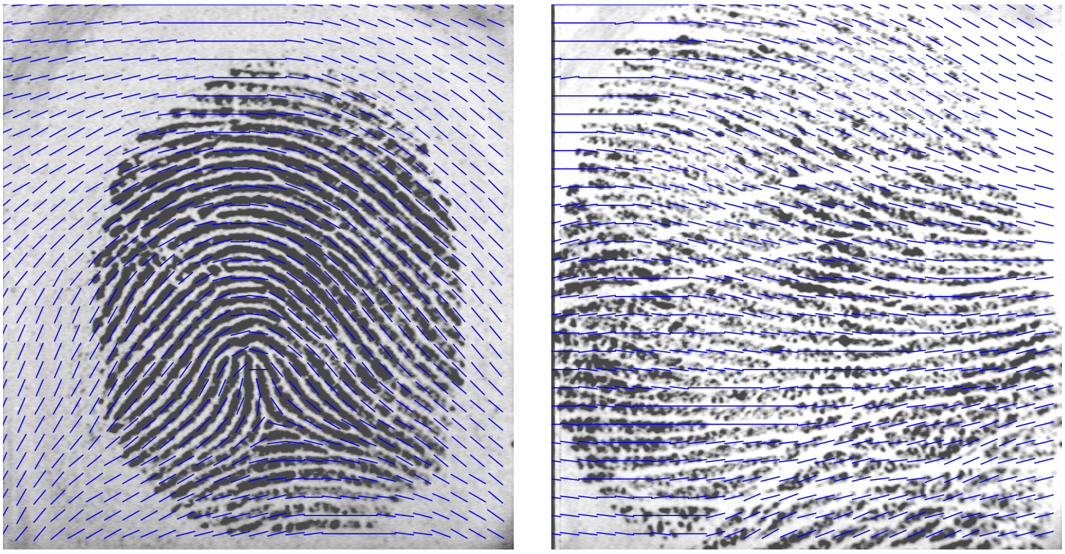
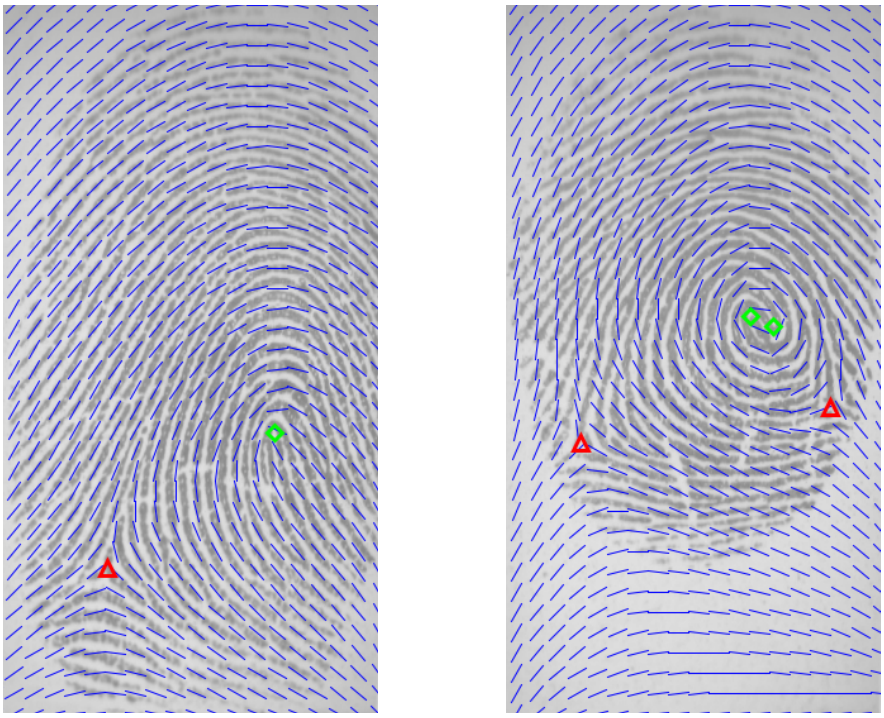


Figure 2.7: Orientation field estimation using the *line sensor* (see Gottschlich et al. (2009); Mieloch et al. (2008))



(a) right loop (b) whorl

Figure 2.8: Core (green diamond), delta (red triangle), and orientation field (blue)

Core/Delta

A ridge orientation does not have to exist for every location on the foreground of a fingerprint image, even if the acquired image is of good quality. On most fingerprints, there are points with no unique local ridge orientation, which are called *singular points*. These can be classified into two types: *core* and *delta*. A *core* is a point around which a ridge makes a half-turn, and a *delta* is a point where three ridges meet. The location as well as the type of singular points can be derived from the ridge orientations surrounding the singular points (Kawagoe and Tojo (1984)) (e.g., using the *Poincaré index*).

To some extent, the converse, i.e. an orientation field is given by the location and type of the singular points, also holds when some additional parameters are given and further assumptions are made (Hotz (2007); Huckemann et al. (2008); Sherlock and Monro (1993)). In Figure 2.8 orientation fields are extrapolated using a model based on the *quadratic differential* approach by Hotz (2007); Huckemann et al. (2008). The extrapolations fit well, albeit not everywhere.

Finger Classes

With the help of singular points, a fingerprint can be classified into types including *arch*, *right/left loop*, *double loop*, and *whorl*. An arch is a finger whose fingerprints contain no singular points; a right loop has a core at right to a delta (a left loop is analogous); a *double loop* contains two deltas and two cores; and finally a *whorl* is a double loop where two cores lie close together.

2.2.8 Minutiae Descriptor

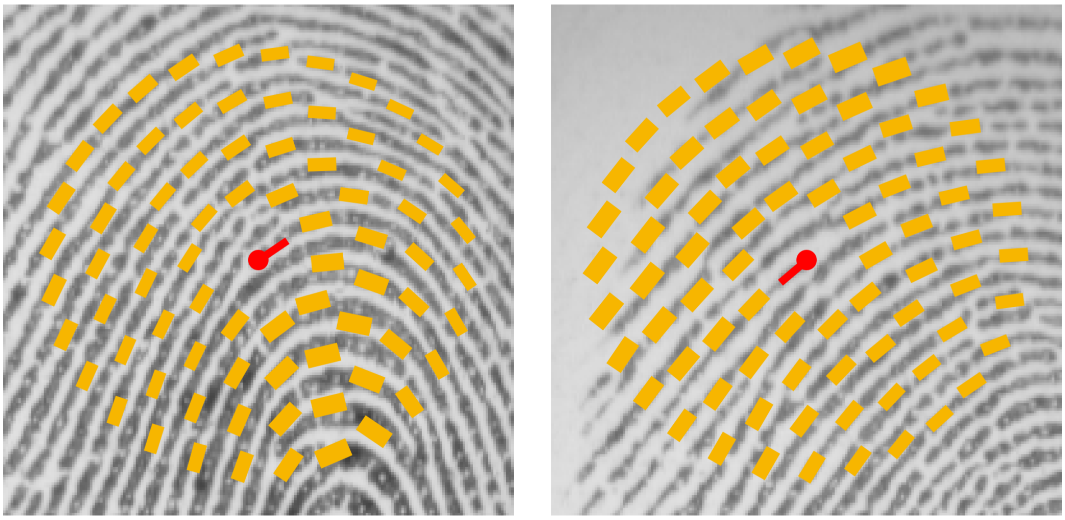


Figure 2.9: *Minutiae descriptors* — thickness and orientation of yellow lines correspond to frequency descriptor and orientation descriptor, respectively.

Orientation Descriptor

Along each minutia, Tico and Kuosmanen (2003) proposed to extract ridge orientations on adjacent locations to support fingerprint matching. They proposed to extract the ridge orientations from equidistant points uniformly spaced on circles arranged around the corresponding minutia. The authors tested the following sampling point configurations:

1) 14, 20, and 26 equidistant points spaced on circles of radii 42, 60, and 78 pixels, respectively;

2) 14, 20, 26, and 32 equidistant points spaced on circles of radii 42, 60, 78, and 96 pixels, respectively;

3) 10, 16, 22, and 28 equidistant points spaced on circles of radii 27, 45, 63, and 81 pixels, respectively.

The third configuration tested produced the best result on the test database used by Tico and Kuosmanen (2003).

A descriptor's sampling point configuration can be arranged around its reference minutia depending on the minutia's direction. In such a way, the sampling point configuration is independent of translation and ration (i.e. alignment). Moreover, the ridge orientations can be sampled with respect to the minutia's direction. Hence, the orientation descriptor is independent of the fingerprint alignment.

Frequency Descriptor

Other global features, independent of translation and rotation with respect to the reference minutia, can be sampled along the ridge orientation as well. One outstanding feature is the local ridge frequency (see Section 2.2.7).

In this way, finding true minutiae correspondences can be performed more reliably.

2.3 Error-Correcting Codes

It may seem contradictory to couple techniques from cryptography with biometrics to protect biometric templates. Classical cryptography relies on exact matches, while biometrics are inexact and at best produce statistical matches as has been emphasized with the example of fingerprints.

A first approach to deal with the inexactness of fingerprint features is to coarsely quantize a minutia's position and angle. On authentication, due to coarsely quantized minutiae of the query and reference fingerprint, the chance might be increased that two fingerprint templates can be reproduced exactly. This would allow cryptographic hash functions to be evaluated on a concatenation of quantized minutiae directly and fingerprint based authentication reduces to password based authentication.

However, as demonstrated in the last section, minutiae may be missing or spurious due to low-quality areas caused by skin conditions, noise, dirt, and other factors, which would cause gross errors and therefore mismatches. For these reasons it seems very unlikely that quantization alone will be sufficient to enable a secure hash based authentication scheme with fingerprint minutiae.

The *fuzzy commitment scheme* and the *fuzzy vault scheme* are biometric cryptosystems that have been designed to tolerate errors while protecting biometric templates. Both schemes make use of *error-correcting codes* to tolerate errors. An overview of the mathematical discipline of error-correcting codes is given in this section. For details, comprehensive textbooks are available in the literature (Berlekamp (1984); Blahut (2003); Roth (2006)).

2.3.1 Preliminaries

An *error-correcting code* is a subset $\mathbf{C} \subset \mathbf{F}^n$ where $\mathbf{F}$ is a finite field. The vector space $\mathbf{F}^n$ is called *ambient space of* $\mathbf{C}$ while the elements $r \in \mathbf{F}^n$ are called *ambient vector*. The amount of errors a code is capable of correcting is commonly determined by its *minimal distance*, i.e. the maximal d^* such that $d^* \leq \text{weight}(v - w)$ for all different $v, w \in \mathbf{F}^n$. By weight$(\cdot)$ we refer to the *hamming weight*.[9] Analogously, $\text{dist}(v, w) = \text{weight}(v - w)$ refers to the *hamming distance*.

2.3.2 Linear Codes

If $\mathbf{C} \subset \mathbf{F}^n$ is a k-dimensional vector space then $\mathbf{C}$ is called a *linear error-correcting code* and is commonly denoted as an (n, k)-code. If the minimal distance d^* for $\mathbf{C}$ is known, then $\mathbf{C}$ can also be denoted as an (n, k, d^*)-code.

The fraction k/n is called the *rate* of $\mathbf{C}$ and the difference $n - k$ is called its *redundancy*. Linear codes can be defined with the help of a generating matrix: Let $g_1, \dots, g_k \in \mathbf{F}^n$ be a basis for $\mathbf{C}$; then the matrix $G = (g_1 \cdots g_k) \in \mathbf{F}^{k \times n}$, whose columns successively consist of $g_1, \dots, g_k$, is called a *generator matrix*. Analogously, a linear code can be defined by a *control matrix*, i.e. a nonzero $H \in \mathbf{F}^{(n-k) \times n}$ such that $H \cdot c = 0$ for all $c \in \mathbf{C}$. The matrix H is also referred to as a *parity check matrix*. Generator matrix G and control matrix H of a linear error-correcting code fulfill $H \cdot G = 0$.

Encoding

With the help of a generator matrix of a linear error-correcting code its encoding-procedure can be described.

Let $G \in \mathbf{F}^{k \times n}$ be a generator matrix of an (n, k)-code $\mathbf{C} \subset \mathbf{F}^n$. A message vector $m \in \mathbf{F}^k$ is encoded by $\mathbf{C}$ as follows. Redundancy is added to m by mapping m to its corresponding codeword using G, i.e. $m \mapsto Gm$. In such a way, the codeword $c = Gm$ uniquely encodes a message m that can be re-obtained from c.

Decoding

Let $\mathbf{C} \subset \mathbf{F}^n$ be an error-correcting code of minimal distance d^* and let $r \in \mathbf{F}^n$. If there is a unique codeword $c \in \mathbf{C}$ minimizing $\text{dist}(c, r)$ then r is said to be *correctable* to c. The search for a unique codeword nearest to the received vector r is called *maximum likelihood decoding* (*MLD*) of r.

It can not be guaranteed for all error-correcting codes of length n and for all $r \in \mathbf{F}^n$ that a unique codeword closest to r exists. However, if for $r \in \mathbf{F}^n$ there exists a codeword $c \in \mathbf{C}$ with $\text{dist}(r, c) \leq \lfloor (d^* - 1)/2 \rfloor$ then r can be uniquely corrected to c. The bound $\nu^* = \lfloor (d^* - 1)/2 \rfloor$ is called *guaranteed error-correction bound* or *error-correction bound* of $\mathbf{C}$.

We have not yet given an explicit algorithm for decoding an particular error-correcting code but only mentioned under what circumstances it can be guaranteed that a vector can be uniquely rounded to its closest codeword. In fact, the construction of efficient decoding algorithms is very challenging and efficient decoders for a given error-correcting code may not even exist, even in case of linear codes.

---

[9] The hamming weight of a vector is the number of its nonzero entries, i.e. $\text{weight}(v_1, \dots, v_n) = \sum_{v_i \neq 0} 1$.

One approach to correct $r \in \mathbf{F}^n$ is to compute its *syndrome* first, i.e. compute $s = Hr$ where H is a control matrix for $\mathbf{C}$. If $s = 0$ then $r \in \mathbf{C}$ is the codeword to which r is corrected. Otherwise, one aims to find a pattern $e \in \mathbf{F}^n$ of minimal hamming weight, such that $He = s$: For the nearest codeword c we have $r = c + e$ and thus $s = Hr = H(c + e) = Hc + He = He$. As a consequence, it suffices to consider the syndrome for maximum likelihood decoding. However, it can be shown that maximum likelihood decoding of a general linear code is NP-hard (Berlekamp et al. (1978)) and thus it is also difficult in general to find an error pattern e of minimal hamming weight.

BCH codes

A real breakthrough has been the discovery of a special kind of error-correcting codes, the so-called *Bose-Chaudhuri-Hocquenghem codes* (*BCH code*) (Bose and Ray-Chaudhuri (1960); Hocquenghem (1959)). There exist efficient decoding algorithms that successfully decode BCH codes, provided that no more than $\lfloor (d^* - 1)/2 \rfloor$ errors have occurred. The most notable are the *Berlekamp-Massey algorithm* (Berlekamp (1966); Massey (1969)) and the *Peterson-Gorenstein-Zierler algorithm* (Gorenstein et al. (1960)). However, these algorithms are not successful if the received vector varies from its nearest codeword in more than $\lfloor (d^* - 1)/2 \rfloor$ positions even if a unique nearest codeword exists.

It is widely believed that there is no efficient algorithm performing maximum likelihood decoding for BCH codes.

2.3.3 Perfect Codes

There are codes for that every ambient vector is of hamming distance at most $\lfloor (d^* - 1)/2 \rfloor$ to a codeword.

A linear (n, k, d^*)-code $\mathbf{C} \subset \mathbf{F}^n$ is said to be *perfect*, if for all $r \in \mathbf{F}^n$ there exists a codeword $c \in \mathbf{C}$ such that $\operatorname{dist}(r, c) \leq \lfloor (d^* - 1)/2 \rfloor$. If $q = \#\mathbf{F}$ then for any codeword $c \in \mathbf{C}$ there is a total of

$$\mathcal{B}(\mathbf{C}) = \sum_{\ell=0}^{\lfloor (d^*-1)/2 \rfloor} (q-1)^\ell \binom{n}{\ell} \tag{2.15}$$

ambient vectors of hamming distance at most $\lfloor (d^* - 1)/ \rfloor$ to c and thus correctable to c. Hence, there are

$$\mathcal{S}(\mathbf{C}) = q^k \cdot \mathcal{B}(\mathbf{C}) \tag{2.16}$$

ambient vectors that are guaranteed to be correctable to a codeword. Thus, the code $\mathbf{C}$ is perfect, if and only if the inequality, called *sphere packing bound*,

$$q^n \geq \mathcal{S}(\mathbf{C}) \tag{2.17}$$

is sharp. Furthermore, the fraction

$$\frac{\mathcal{S}(\mathbf{C})}{q^n} \tag{2.18}$$

is called *sphere packing density*, which is equals 1 if and only if the code $\mathbf{C}$ is perfect. Consequently, the perfectness of a linear code depends only on its parameters (n, k, d^*) and its finite field size q.

Unfortunately, there are only a few perfect codes. Any perfect linear code is of either the following parameters:

- Trivial codes, i.e. $\mathbf{C} = \{0\}$ or $\mathbf{C} = \mathbf{F}^n$ which are of parameters $(n, 0, n)$ or $(n, n, 1)$;
- binary *repetition codes* of odd code-word length, i.e. of parameters $(2m + 1, 1, 2m + 1)$ with $q = 2$;
- *hamming codes*, i.e. $((q^m - 1)/(q - 1), n - m, 3)$-codes;
- *binary Golay code*, i.e. $(23, 12, 7)$ with $q = 2$;
- *ternary Golay code*, i.e. $(11, 6, 5)$ with $q = 3$.

Any linear code of parameters different than above is imperfect. Depending on the code's size and error-correction capability the number of words that can be decoded can become negligible when comparing the number of words that can not be decoded. An impression of the effect is given by Table 2.2.

Table 2.2: Sphere packing densities — i.e. proportion of ambient vectors that can be corrected to a codeword of an (n, k, d^*)-code

	n	k	d^*	% **correctable words**
binary golay code	23	12	7	$= 100$ %
ternary golay code	11	6	5	$= 100$ %
binary BCH code	15	5	7	$= 56.25$ %
"	31	6	15	≈ 10.65 %
"	63	7	31	≈ 0.24 %
"	127	8	63	$\approx 8.1 \cdot 10^{-5}$ %
"	255	9	127	$\approx 6.2 \cdot 10^{-12}$ %
"	511	10	255	$\approx 2.5 \cdot 10^{-26}$ %
"	511	19	239	$\approx 1.3 \cdot 10^{-27}$ %

Golay Codes

Outstanding perfect codes are the *Golay codes* (Golay (1949)). They are the only perfect non-trivial, non-repetition codes that are capable in correcting more than 1 error.

For the binary Golay code a 23-bit vector can be decoded efficiently to its nearest Golay codeword by searching its closest element in a table of size $2^{12} = 4096$. Accelerations are possible using a *low-complexity scheme* (Ching-Lung et al. (2006)) to avoid exhaustive iterations through the codeword table.

2.3.4 Reed-Solomon Codes

Reed-Solomon codes are the class of codes that are of direct importance to the fuzzy vault scheme.

We give the definition of *Reed-Solomon codes* in their *original view* analogous to Reed and Solomon (1960). For the view of *Reed-Solomon codes* as BCH codes (*classic view*) and other relations, we refer to Roth (2006).

Definition 2.3.1 (Reed-Solomon Code)**.** *Let* $\mathbf{F}$ *be a finite field,* $\alpha_1, \ldots, \alpha_n \in \mathbf{F}$ *be distinct, and* $k \leq n$ *an integer. The set*

$$\mathbf{C} = \{(f(\alpha_1), \ldots, f(\alpha_n)) \mid f \in \mathbf{F}[X] \text{ where } \deg f < k \}. \tag{2.19}$$

is called a Reed-Solomon code*. The elements* $\alpha_1, \ldots, \alpha_n$ *are called* code locators*.*

A Reed-Solomon code $\mathbf{C} \subset \mathbf{F}^n$ is a linear (n,k)-code. The remarkable property of a Reed-Solomon code, is that its minimal distance is optimal in the sense that it sharply meets the *singleton bound*[10]:

$$d^* = n - k + 1. \tag{2.20}$$

Encoding of Reed-Solomon Codes

Assume one wants to transmit a message $f \in \mathbf{F}^k$. The vector $f = (f_0, \ldots, f_{k-1})$ is interpreted as the coefficient vector of a polynomial in the indeterminate X with coefficients in $\mathbf{F}$ for that we write $f(X) = f_0 + f_1 X + \ldots + f_{k-1} X^{k-1}$. The corresponding codeword $c \in \mathbf{C}$ is defined to consist of the evaluations on the locators $\alpha_1, \ldots, \alpha_n$, i.e. $c = (f(\alpha_1), \ldots, f(\alpha_n))$.

To re-obtain the original message it suffices to find a polynomial $f(X)$ interpolating the first pairs $(\alpha_1, f(\alpha_1)), \ldots, (\alpha_k, f(\alpha_k))$ and output its coefficient vector. The remaining $n-k$ pairs add redundancy by over-determining the polynomial f.

Decoding of Reed-Solomon Codes

When a vector $b = (\beta_1, \ldots, \beta_n)$ has been received that differs from a Reed-Solomon codeword in $\mathbf{C}$ in at most $\lfloor (n-k)/2 \rfloor$ positions, efficient algorithms have been discovered to correct for these errors, i.e. they find the nearest codeword c to b. Among them, we mention the *Berlekamp-Massey algorithm* (Berlekamp (1966); Massey (1969)), the *Berlekamp-Welch algorithm* (Welch and Berlekamp (1983)), and the algorithm proposed by Gao (2002) that benefits from fast finite field and polynomial arithmetic.

The Berlekamp-Massey algorithm runs in $O(n^2)$, while the Berlekamp-Welch algorithm runs in $O(n^3)$. The algorithm of Gao (2002) can be implemented such that it runs in $O(n \cdot \log^2 n)$ which becomes faster for large n ($n > 1000$).

---

[10] The singleton bound, which holds for every linear error-correcting code, is the following: $d^* \leq n - k + 1$

Decoding of Reed-Solomon Codes beyond the Error-Correction Bound

As we will see later, the fuzzy vault for fingerprint provides its security via the conjecture that maximum likelihood decoding of Reed-Solomon code is hard (Guruswami and Vardy (2005)).[11] However, there are cases where maximum likelihood decoding of Reed-Solomon codes is easy.

One obvious case where maximum likelihood decoding is easy is where the received vector has a hamming distance to a Reed-Solomon codeword of at most $\lfloor (n-k)/2 \rfloor$ and thus can be decoded using one of the efficient algorithms from above.

Sudan (1997) discovered a polynomial time algorithm that outputs all Reed-Solomon code words differing in approximately at most $n-\sqrt{2nd}$ positions from the received word, where $d=k-1$. This algorithm is thus able to correct errors beyond the error-correction bound, provided the code's rate is bounded by approximately 1/3.

Later Guruswami and Sudan (1998) generalized the *Sudan algorithm* to an algorithm that can correct up to $n-\sqrt{nd}$ errors. In its original description, the algorithm has a running time of $O(n^{15})$ in the worst case. However, the running time can be reduced by implementing the main steps of the algorithm using a variety of approaches (Alekhnovich (2002); Cohn and Heninger (2011); Gao et al. (1999); Koetter and Vardy (2003a); McEliece, R. (2003); Roth and Ruckenstein (2000); Trifonov (2010)). Furthermore, the worst case running time only holds if one wants to tolerate precisely $\lceil n-\sqrt{nd}-1 \rceil$ errors. If only fewer errors are to be tolerated, the Guruswami-Sudan algorithm can be implemented such that it may be feasible.

2.4 The Fuzzy Commitment Scheme

The *fuzzy commitment scheme* by Juels and Wattenberg (1999) is a simple and elegant mechanism to couple the non-exactness of biometrics with well-known techniques from cryptography. The scheme makes use of error-correcting codes (see Section 2.3).

Throughout this section, let $\mathbf{F}$ be a finite field and $\mathbf{C} \subset \mathbf{F}^n$ be an error-correcting code of minimal distance d^* and of error-correction capability ν^*.

2.4.1 Enrollment

Assume a reference biometric template is encoded as a fixed-length vector $v \in \mathbf{F}^n$. A codeword $c \in \mathbf{C}$ is selected randomly (or encodes some secret). The offset $c+v$ is called a *fuzzy commitment of* v. Optionally, to allow safe recovery, a cryptographic strong hash value $h(c)$ of the secret is published along with $c+v$. In this case the pair $(c+v, h(c))$ is referred to as a *fuzzy commitment of* v *and* c.

2.4.2 Authentication

On authentication, a fresh query biometric template $v' \in \mathbf{F}^n$ of the (alleged) genuine user is provided to the server. The server attempts to use v' to eliminate most of the errors in $c+v$, i.e. the server computes $r = c+v-v = c+(v-v')$.

---

[11] NP-hardness has been proven for large field sizes, but it is believed to be true also for arbitrary field sizes.

Now, if $\text{weight}(v - v') \leq \nu^*$, the server will succeed in correcting r to the codeword c using a decoder that can correct ν^* errors. Otherwise, two cases are possible: First, the decoder outputs a different codeword $c' \neq c$, or second, the decoder fails to discover any codeword c'. Note that the second case is the more likely case for most error-correcting codes (see Section 2.3.3). However, if $h(c)$ is published along with the commitment $v + c$ and if the decoder outputs a codeword c', the server can verify with high probability whether the authentication attempt is successful by checking whether $h(c) = h(c')$.

2.4.3 Security

To simplify analysis, assume that v was chosen uniformly at random from $\mathbf{F}^n$. Juels and Wattenberg (1999) define in their security analysis a fuzzy commitment $F(c, v) = (c + v, h(c))$ to be *strongly binding*[12] if it is infeasible to produce a *witness collision*. A *witness collision* is a fuzzy commitment $(c + v, h(c))$ and a pair $v_1 \neq v_2$ such that v_1 and v_2 both *decommit*[13] $(c + v, h(c))$ but $\text{dist}(v_1, v_2) \geq d^*$. With this definition they observe the following.

Proposition 2.4.1 (Juels and Wattenberg (1999), Claim 1)**.** *If an attacker is able to find a witness collision then he can find a collision of the hash function h.*

Proof. Let $(c + v, h(c)), v_1, v_2$ be a witness collision. Since $\text{dist}(v_1, v_2) \geq d^*$ then also $\text{dist}(c + v - v_1, c + v - v_2) \geq d^*$. Thus, $c + v - v_1$ and $c + v - v_2$ correct to different codewords c_1 and c_2, respectively. Since c_1 and c_2 are decommitments, $h(c_1) = h(c_2)$. In particular, (c_1, c_2) is a collision of the hash function h. □

Effective security of a fuzzy commitment scheme is thus realized under the assumption that 1) the templates encoded as $v \in \mathbf{F}^n$ have been chosen uniformly at random and 2) the hash function h is *strongly collision resistant*.[14] While the second assumption has been shown up to be realistic, the first assumption is not so clear, in particular for biometric templates $v \in \mathbf{F}^n$. We will emphasize this with the example of iris.

2.4.4 Protection of Iris Templates

As an example we carry out the implementation of Hao et al. (2006), which can be considered as a two-layer fuzzy commitment scheme operating on a human's eye iris encoded as the famous 2048-bit *iris code* (Daugman (2004)).

Enrollment

The construction proposed by Hao et al. (2006) can be considered as a two-layer fuzzy commitment scheme where the secret key is encoded in two stages.

First, for a fixed integer m, a $7 \cdot m$-bit key κ is selected and divided into 7-bit blocks $\kappa = (\kappa_1, \ldots, \kappa_m)$. Each block is next encoded as a $(64, 7, 32)$-Hadamard

---

[12] This is a generalization of the concept of *binding* in a more general *commitment scheme* (Brassard et al. (1988)).

[13] Here, v' is said to *decommit* $c+v$ if $c+v-v'$ can be corrected to a codeword c' and $h(c') = h(c)$.

[14] *Strongly collision resistant* means it is infeasible to find any collision $h(c_1) = h(c_2)$ — not just for an either fixed c_1 or c_2.

codeword $r_i \in \mathbf{F} = \mathbb{F}_{2^{64}}$. Second, $r = (r_1, \ldots, r_m)$ is encoded as a $(32, m)$-Reed-Solomon codeword over the field $\mathbf{F}$, i.e. $c \in \mathbf{F}^{32}$. Note, that c can be interpreted as a 2048-bit vector.

The user's iris is provided and encoded as the famous 2048-bit iris code, i.e. $v \in \{1, 0\}^{2048}$. The offset $c + v$ as well as a cryptographic hash value $h(c)$ forms the fuzzy commitment.

Authentication

The decoding, on authentication, also has to be performed in two stages.

A query iris template $v' \in \{0, 1\}^{2048}$ is used to determine $c + v - v'$, which is divided into 64-bit blocks $b_1, \ldots, b_{32}$. Attempts are made to correct each block b_i to its nearest $(64, 7, 32)$-Hadamard codeword r'_i, respectively, to correct small local bit-errors. In such a way a vector $r' = (r'_1, \ldots, r'_{32}) \in \mathbf{F}^{32}$ where $\mathbf{F} = \mathbb{F}_{2^{64}}$ is obtained. To correct global errors, r' is attempted to be corrected to its nearest $(32, m)$-Reed-Solomon codeword $c' \in \mathbf{F}^{32}$. If no such c' can be discovered, the authentication attempt is rejected. Otherwise, if c' comes out with $h(c') = h(c)$ the authentication is accepted. Or if $h(c') \neq h(c)$ then it is rejected.

Note, the commitment scheme proposed by Hao et al. (2006) differs slightly from the fuzzy commitment scheme because the error-correction is more dynamic than simply tolerating a fixed number of errors.

The probability that a random r'_i 64-bit words is placed within a $(64, 7, 32)$-Hadamard codeword's decoding are is approximately 99.84% (see Section 2.3.3). There are two ways to deal with decoding failures for some r'_i.

First, the decoding of a $(64, 7, 32)$-Hadamard code can be performed by exhaustive search, i.e. maximum likelihood decoding: For each block r'_i it is feasible to iterate through all $2^7 = 128$ codewords to find a codeword closest to r'_i. This is the way Hao et al. (2006) follow.

Another valid way would be to decode the subsequent Reed-Solomon codeword by accounting for erasures: Assume it was not possible to correct e blocks r'_i; then one can decode a $(32 - e, m)$-Reed-Solomon code by simply omitting the code locators and the positions r'_i.

Authentication Performance

Hao et al. (2006) report authentication performances for $m = 6, \ldots, 32$ which corresponds to a key of bit-length $42, \ldots, 224$, respectively, encoded as a 2048-bit vector. For instance, if $m = 16$ then a genuine acceptance rate of 99.85% at a false acceptance rate of 0.02% was measured.

Security

If $m = 16$, the key-length corresponds to 112 bits. However, the vulnerability against a simple false accept attack is estimated as $-\log(0.02\%) \approx 13$ bits only. But if the difficulty in guessing a matching template $v' \in \{0, 1\}^{2048}$ was 112 bits, the chance of a false accept should be much lower.

The genuine acceptance rate for $m = 20$ was measured as 99.53% at a false acceptance rate[15] of 0%. Hao et al. (2006) estimate the security for $m = 20$ as

---

[15] There have been no false accepts within $241,300$ impostor authentication attempts.

≈ 44 bits in the worst case, where the attacker has perfect knowledge about each individuals iris statistics. However, the security against the false accept attack is likely to be much lower (≈ 16 bits with the rule of three; see Section 2.2.6).

Moreover, even if the security level of the implementation were 44 bits, for some purposes this is not sufficient. To prevent an attack from successfully breaking the implementation within $\approx 2^{44}$ iterations, Hao et al. (2006) propose to artificially increase the computational cost for verifying whether $h(c') = h(c)$ by letting h be a hash function evaluating their inputs, e.g., a million times. The purpose was to add approximately 20 bits of security against brute-force attacks. Consequently, the overall security would be 64 bits. Hao et al. (2006) do not consider the case where r' can not be corrected to a valid Reed-Solomon codeword c'. However, the chance that a random input $r' \in \mathbf{F}^{32}$ can be corrected to a $(32, 20, 13)$-Reed-Solomon codeword using traditional decoders (see Section 2.3.4) is at most $2.29 \cdot 10^{-110}$ which is negligible (see Section 2.3.3). This makes repeated hashing obsolete.

We finish this section with the remark that the authentication rates of the above implementation to protect iris templates is quite good when compared to other biometrics. For instance, the rates that are achievable for fingerprints are currently much worse. The problems arising when one attempts to protect fingerprints are treated through the rest of this thesis.

3. The Fuzzy Fingerprint Vault

3.1 The Fuzzy Vault Scheme

Juels and Sudan (2002) proposed a cryptographic construction called *fuzzy vault scheme*. The fuzzy vault scheme and the fuzzy commitment scheme (see Section 2.4) are related. Both work by binding a secret key to the template that one wants to protect. However, while the fuzzy commitment scheme requires the biometric templates to be encoded as an ordered vector v with entries in a finite field $\mathbf{F}$, the fuzzy vault scheme differs in that: The encodings are unordered sets $v \subset \mathbf{F}$. This makes the fuzzy vault scheme more appropriate for biometric modalities where matching templates may overlap. An example for such biometrics are fingerprints: Minutiae might be extracted from different regions of the finger, where the finger touches the scanner's surface.

Before we discuss implementations of the fuzzy vault to fingerprint minutiae templates, we describe its function in general terms.

3.1.1 Encoding

Let $\mathbf{F}$ be a finite field. Assume a template, encoded as $v \subset \mathbf{F}$ of cardinality t, is given on enrollment. To protect the template v, a secret key encoded as a polynomial $f \in \mathbf{F}[X]$ of degree $< k$ is selected, either randomly or by encoding a secret password. A *genuine set* $\mathbf{G}$ is built that binds the template v to the secret f, i.e.

$$\mathbf{G} = \{(\alpha, f(\alpha)) \mid \alpha \in v\}. \tag{3.1}$$

Next, if $t \geq k$ then $\mathbf{G}$ uniquely encodes the template v (on the abscissa) as well as the secret polynomial f which interpolates any k distinct points from $\mathbf{G}$. In order to protect both the template v and the key f both given by $\mathbf{G}$, a set of *chaff points* $\mathbf{C}$ of size r is generated at random, i.e.

$$\mathbf{C} = \{(\alpha, \beta) |\ x \notin \mathbf{G},\ \ f(\alpha) \neq \beta\}. \tag{3.2}$$

Finally, the union of size $n = t + r$ builds the vault, i.e.

$$\mathbf{V} = \mathbf{G} \cup \mathbf{C}. \tag{3.3}$$

To simplify notation, write

$$\mathbf{V} = \{(\alpha_1, \beta_1), \ldots, (\alpha_n, \beta_n)\} \tag{3.4}$$

The vault $\mathbf{V}$ constructed this way can be considered as a received vector $(\beta_1, \ldots, \beta_n)$ of an (n, k)-Reed-Solomon code with code locators $\alpha_1, \ldots, \alpha_n$. The (now unknown) positions i where $(\alpha_i, \beta_i) \in \mathbf{C}$ are errors in the received vector. We have already mentioned that if $t \leq \sqrt{n \cdot (k-1)}$ finding the correct polynomial becomes infeasible and is in general even an NP-hard problem (see Section 2.3.4). This reduces the problem of recovering the secret f and, equivalently, the template v to the NP-hard problem of the *polynomial reconstruction* (see Bleichenbacher and Nguyen (2000)).

3.1.2 Decoding

On authentication, a second template encoded as $v' \subset \mathbf{F}$ of cardinality (say) s is reproduced by an individual who is attempting to gain access to the system. Using v' the *unlocking set* $\mathbf{U}$ is extracted from the vault $\mathbf{V}$, i.e.

$$\mathbf{U} = \{(x, y) \in \mathbf{V} \mid x \in v'\}. \tag{3.5}$$

Again, to simplify notation, write

$$\mathbf{U} = \{(x_1, y_1), \ldots, (x_m, y_m)\} \tag{3.6}$$

where $m \leq s$. As above, $\mathbf{U}$ can be considered as a received vector $(y_1, \ldots, y_m)$ of the (m, k)-Reed-Solomon code with locators $x_1, \ldots, x_m$. Now, if v' overlaps with the enrolled template v substantially, the polynomial f and equivalently the template v, can be recovered using an algorithm for decoding Reed-Solomon codes. Successful recovery can be guaranteed if $|v' \cap v| > (m+k)/2$.

As with the fuzzy commitment scheme, on enrollment, it can be convenient to store a cryptographic hash value $h(f)$ together with the vault points $\mathbf{V}$. This enables safe recovery of the genuine polynomial.

3.1.3 Security

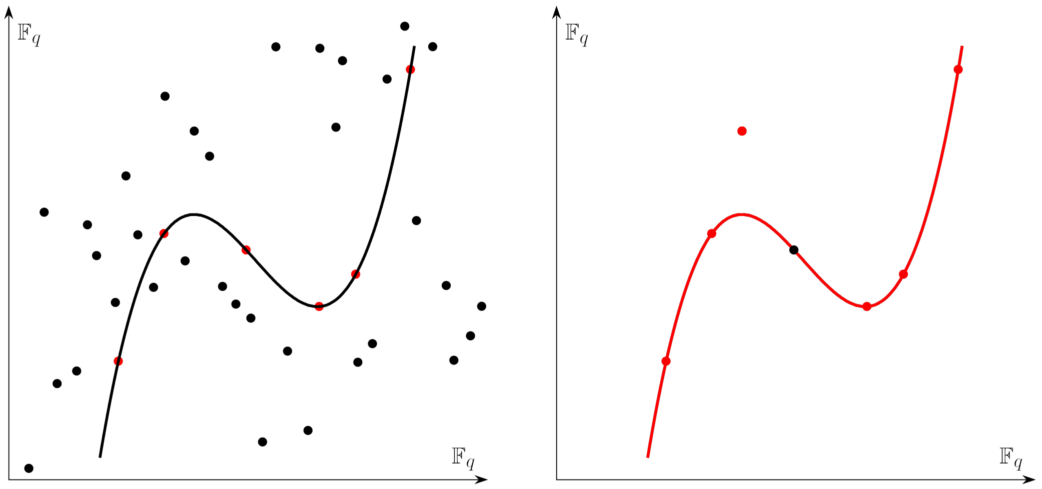
We have already mentioned above that the problem of discovering a polynomial f in $\mathbf{V}$ of degree $< k$ interpolating exactly t vault points reduces to an NP-hard problem. This, however, has only been proven for finite fields of sufficiently large cardinality (see Guruswami and Vardy (2005)). Nonetheless, even for small finite fields, no efficient algorithm for recovering the hidden polynomial in a vault $\mathbf{V}$ is known for general parameters.

In fact, Juels and Sudan (2002) do not base their security analysis on the hardness of the polynomial reconstruction problem but on the following information-theoretic result whose proof can be found in (Juels and Sudan (2002)).

Lemma 3.1.1 (Juels and Sudan (2002)). *Let* $\mathbf{V} \subset \mathbf{F} \times \mathbf{F}$ *be a random vault instance of size* n *and let* $q = |\mathbf{F}|$ *be the finite field's cardinality. For every* $\mu > 0$ *with probability at least* $1 - \mu$ *there exist at least*

$$\frac{\mu}{3} q^{k-t} (n/t)^t \tag{3.7}$$

polynomials $g \in \mathbf{F}[X]$ *of degree* $< k$ *such that* $\mathbf{V}$ *contains exactly* t *points of the form* $(x, g(x)) \in \mathbf{F} \times \mathbf{F}$.



(a) Polynomial encoded by genuine points (red) which are hidden by chaff points (black)

(b) Erroneous unlocking points (red) and reconstructed polynomial (red). The correctness of the polynomial is confirmed by an additional vault point (black).

Figure 3.1: Encoding and decoding in the fuzzy vault scheme

For example, if $q \approx 10^4$, $n \approx q$, $t = 24$, and $k = 18$ and a random vault $\mathbf{V}$ with probability at least $\approx 1 - 2^{-64}$, there are $\approx 2^{64}$ polynomials of degree < 18 interpolating exactly 24 points from $\mathbf{V}$.

Roughly speaking, the security (of ≈ 64 bits) is not achieved from the difficulty in finding a polynomial of degree $< k$ interpolating at least t vault points, but via the information-theoretic observation that if such a polynomial is found, the probability that it is not the genuine secret polynomial is overwhelming.

Let us consider another example. Let $q = 2^{16}$, $n = 224$, $t = 24$, and $k = 9$. Then, the information-theoretic argument given by Lemma 3.1.1 cannot be used to make a statement about the vault's security, since the outcome of (3.7) is for all μ negligible small. In such cases, the security of the vault can still be asserted from the assumed hardness of finding the genuine polynomial hidden in $\mathbf{V}$. This problem is believed to be hard, even for arbitrary finite fields. Vault security is then conservatively estimated as the computational cost of a simple **brute-force attack**: *1) Try the interpolating polynomial f^* of k randomly selected vault points; 2) if f^* interpolates at least t vault points, output f^*; 3) otherwise continue with step 1).* Analogously to Clancy et al. (2003), the chance for successful recovery within a single iteration is $1/\mathbf{bf}(n,t,k)$ where

$$\mathbf{bf}(n,t,k) := \binom{n}{k} \cdot \binom{t}{k}^{-1}, \tag{3.8}$$

which is approximately 2^{31} in the example.

From a mathematical view, a security estimation that is asserted by Lemma 3.1.1 should be preferred to a security given by the size $\mathbf{bf}(n,t,k)$. It is not yet clear whether brute-force attacks can be substituted for by significantly more efficient attacks if Lemma 3.1.1 can not guarantee sufficient security. However, under assumptions widely believed to be true, the polynomial reconstruction problem can be considered to be hard, even for small finite fields.

3.2 Implementations for Fingerprints

In this section, we will discuss implementations of the fuzzy vault to fingerprints proposed by the literature. But first, we describe the function of minutiae fuzzy vault in more general terms, without excluding modifications.

3.2.1 General Description of a Minutiae Fuzzy Vault

A basic way to build a fuzzy vault from fingerprint minutiae data works as follows. Assume a user (say Alice) wishes to be identified by a server by her minutiae template $T = \{\mathfrak{m}\}$, but does not want that it is stored publicly on the server. She additionally chooses a secret encoded as a polynomial $f \in \mathbf{F}[X]$ of degree $< k$. On enrollment, $t \geq k$ minutiae are selected from T and encoded by elements in $\mathbf{F}$ using some convention, i.e. using an encoding map

$$E : \text{``set of all minutiae''} \to \mathbf{F},\ \mathfrak{m} \mapsto x. \tag{3.9}$$

Analogously to our general description of the fuzzy vault scheme, the genuine set

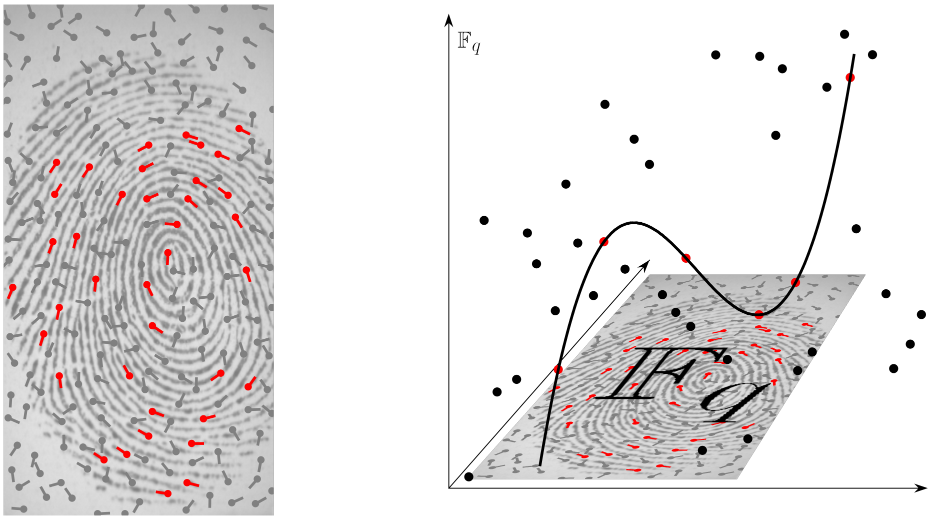
$$\mathbf{G} = \{(x, f(x)) \mid x = E(\mathfrak{m}),\ \mathfrak{m} \in T\} \tag{3.10}$$

of cardinality t is built. To hide $\mathbf{G}$ (and thus f as well as T) a chaff set $\mathbf{C} = \{(x, y) \mid y \neq f(x)\}$ of cardinality r is generated at random and unified with the elements of $\mathbf{G}$ to build the vault $\mathbf{V} = \mathbf{G} \cup \mathbf{C}$.

On authentication, a user who is claiming to be Alice provides a fresh query template $T' = \{\mathfrak{m}'\}$ aligned to the vault $\mathbf{V}$. The authentication procedure is slightly different from the general vault work, since it may comprise an extraction step in contrast to mere exact vault point identification: That is, each vault point's abscissa $x \in \mathbf{F}$ encodes a minutiae $\mathfrak{m}^{\text{vault}}$, chaff or genuine. Using this observation, those elements of $\mathbf{V}$ are extracted corresponding to elements in $\{\mathfrak{m}^{\text{vault}}\}$ that are well approximated by elements from T' to form the unlocking set $\mathbf{U} \subset \mathbf{V}$. The remaining work in the same manner as above: If $\mathbf{U}$ overlaps with $\mathbf{G}$ substantially, then the secret polynomial f can be recovered and the user will be positively authenticated as Alice.

3.2.2 Limitations of the Minutiae Fuzzy Vault

The more chaff minutiae are generated the higher is the corresponding vault's resistance against the brute-force attack. To enable a genuine user to safely extract genuine vault points, minutiae corresponding to chaff points should keep sufficient distance to genuine vault minutiae. As a consequence, each vault minutiae, chaff or genuine, need to be sufficiently distant from genuine vault minutiae. As a consequence, each vault minutia, chaff or genuine, must be a sufficient distance from another (say at least $\delta > 0$). Otherwise chaff minutiae could be easily separated from genuine vault minutiae. A further consequence is that the number of chaff points can not be arbitrarily large. Otherwise genuine minutiae would appear isolated from chaff minutiae (see Figure 3.3). A conclusion drawn from this is that pushing vault



(a) Genuine minutiae (red) and chaff minutiae (gray)

(b) Chaff and genuine minutiae, are encoded on the abscissa

Figure 3.2: Minutiae Fuzzy Vault

parameters into extremes is not possible, which requires the vault security to be limited due to the limitations of chaff points.

Clancy et al. (2003) deduced "optimal" parameters for a single-finger minutiae fuzzy vault for the case where only the locations of minutiae (and no angle) are used for vault construction. These are

$$\begin{aligned} n &= 313, \\ k &= 15, \\ t &= 38, \text{ and} \\ \delta &= 10.7 \end{aligned} \tag{3.11}$$

with the notation from above.

3.2.3 Security of a Minutiae Fuzzy Vault

For a fuzzy fingerprint vault with parameters providing a usable fuzzy fingerprint vault, Lemma 3.1.1 can not be used to assert provable security (see Merkle et al. (2010a)). In fact, something of a converse of Lemma 3.1.1 holds: It is very unlikely for a second polynomial of degree $< k$ to interpolate t vault points, which is shown in the following.

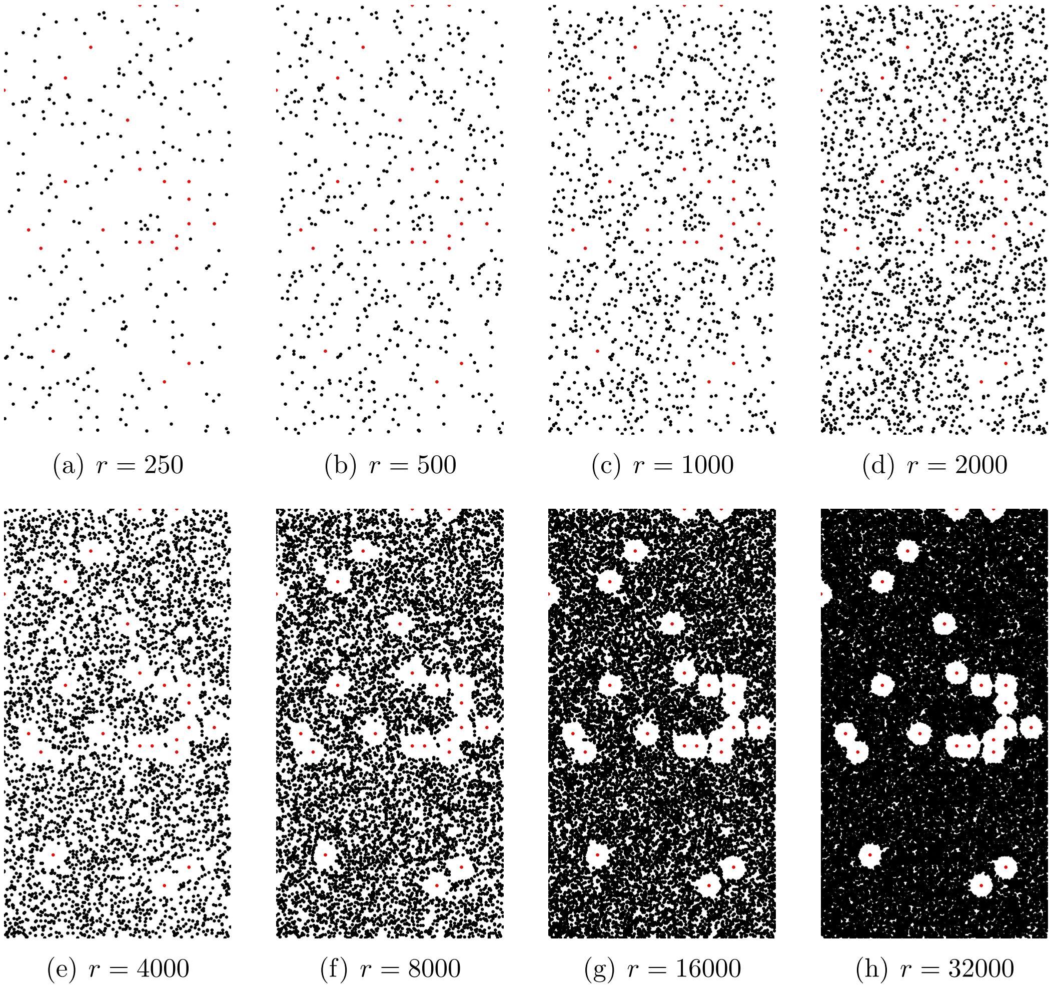
Let $\mathbf{V} = \mathbf{G} \cup \mathbf{C}$ be a random vault of cardinality n and genuine points

$$\mathbf{G} = \{(x_1, f(x_1)), \ldots, (x_t, f(x_t))\} \tag{3.12}$$

for a polynomial f of degree $< k$.

Lemma 3.2.1. *For simplicity, allow that* $\mathbf{C}$ *contains elements that lay on the graph of* f*. The expected number of spurious vault polynomials, i.e. polynomials of degree* $< k$ *interpolating exactly* t *vault points, is*

$$\sum_{s=0}^{k-1} \binom{t}{s} \cdot q^{k-s} \cdot \binom{n-t}{t-s} \cdot (1/q)^{t-s} \cdot (1-1/q)^{n-t-s} \tag{3.13}$$


(a) $r = 250$ (b) $r = 500$ (c) $r = 1000$ (d) $r = 2000$

(e) $r = 4000$ (f) $r = 8000$ (g) $r = 16000$ (h) $r = 32000$

Figure 3.3: Visualization of the effect of generating a different number r of vault chaff minutiae locations: Genuine locations (red) appear isolated

Proof. Let f^* be a polynomial of degree $< k$. Assume f^* interpolates exactly s genuine points (note, $s < k$ otherwise f^* would interpolate at least k genuine vault points and thus $f^* = f$). For f^* to be spurious, it is necessary to interpolate exactly $t - s$ chaff points. The probability that f^* interpolates exactly $t - s$ chaff points is

$$\binom{n-t}{t-s} \cdot (1/q)^{t-s} \cdot (1-1/q)^{n-t-s}. \tag{3.14}$$

Furthermore, for a fixed s there are

$$\binom{t}{s} \cdot q^{k-s} \tag{3.15}$$

polynomials of degree $< k$ that interpolate exactly s genuine points. The expected number of spurious vault polynomials thus is as claimed. □

For the "optimal" parameters deduced by Clancy et al. (2003) the expected number of spurious vault polynomials is approximately 2^{-28} using the above result. As a consequence, one can not expect vault security as intended by Juels and Sudan

(2002). However, under reasonable assumptions that are commonly believed to be true[1], security of the fuzzy fingerprint vault can be achieved via the difficulty in finding a polynomial of degree $< k$ that interpolates exactly t vault points. Furthermore, using Lemma 3.2.1 if a polynomial of degree $< k$ has been found interpolating exactly t vault points it is very likely that this polynomial will be the genuine polynomial. This gives a criterion for identifying the genuine polynomial, even if an optional hash value is not stored publicly along with the vault.

3.2.4 Minutiae Fuzzy Vault Implementations

Uludag et al. (2005) implemented the fuzzy vault scheme protecting the locations of a single fingerprint minutiae template. The authors achieved the minutiae as well as the alignment manually to make their report of vault performance independent from other error sources.

In their original description of the fuzzy vault scheme, Juels and Sudan (2002) proposed the decoding to be based upon a Reed-Solomon decoder which can be implemented to run very efficiently (see Section 2.3.4). For a conventional Reed-Solomon decoder to successfully output the correct polynomial it is necessary that more than 50% of the elements of the unlocking set consist of genuine points. However, to obtain useful genuine acceptance rates in a minutiae fuzzy vault a decoder must be able to successfully decode even if 25%–33% of the unlocking points are genuine points.

To meet this requirement, Uludag et al. (2005) proposed to iterate through all combinations of k distinct unlocking points selected from the unlocking set $\mathbf{U}$ of size t. Using a criterion for identifying the correct polynomial, only in the case that at least one combination fully consists of k genuine points will the decoding be successful. This approach to decode the unlocking set requires to iterate through up to

$$\binom{t}{k} = \frac{t!}{k! \cdot (t-k)!} \tag{3.16}$$

candidates for the secret polynomials. As a consequence, if decoding the unlocking set is sought to be feasible t and k have to be chosen carefully.

The correct polynomial can be identified in several ways. For example it is possible to test how many vault points are interpolated by the candidate polynomial: If the candidate polynomial interpolates exactly t vault points, then for reasonable vault parameters it will be the correct polynomial with very high probability (see Lemma 3.2.1). Evaluating the candidate polynomial on all vault points, however, can be very time consuming.

Thus, Uludag et al. (2005) proposed to identify the correct polynomial using a 16-bit *cyclic redundancy check*. Uludag et al. (2005) report authentication performances for vaults of size $n = 218$ hiding a polynomial of degree $< k = 9$ where $t = 18$ minutiae encode genuine vault points. The genuine acceptance rate and false acceptance rate have been measured as GAR = 79% and FAR = 0%, respectively. Using Equation (3.8) the security can be estimated as approximately 2^{36}.

Later Uludag and Jain (2006) proposed an automatic mechanism for dealing with the alignment of query fingerprints to the vault by storing points of high ridge

---

[1] We assume that $P \subsetneq NP$ and that maximum-likelihood decoding of Reed-Solomon codes remains NP-hard for arbitrary finite fields.

curvature publicly along with the vault (see Section 3.3). The secret polynomial's degree is bounded by $< k = 9$ evaluated on $t = 24$ genuine points encoded by genuine minutiae locations. The genuine points are hidden within a vault of size $n = 224$. The authors report a genuine acceptance rate of GAR $= 72.6\%$ where the false acceptance rate has been measured as FAR $= 0\%$. According to Equation (3.8) the security of this implementation is conservatively estimated as 2^{31}.

Nandakumar et al. (2007a) refined some of the details of the implementation of Uludag and Jain (2006), in particular those concerning automatic vault alignment. A major difference is that the authors also use minutiae directions. On enrollment, only those template minutiae are used for vault construction that are well-separated from all the other minutiae (with respect to Equation (2.5) on page 19 with $w = 11.46$)[2]. Furthermore, at most the first $t = 24$ minutiae of best *minutiae quality* (see Chen et al. (2005)) are selected. While the incorporation of minutiae direction would allow the generation of more chaff points, the authors nonetheless only report results where the vault again has a size of $n = 224$. For authentication, the authors proposed the use of a minutiae matcher (see Jain et al. (1997)) for extracting minutiae of a of unlocking minutiae that are coarsely approximated by query minutiae. Nandakumar et al. (2007a) report results for different polynomial degrees bounded by $k = 8, 9, 11$. The genuine acceptance rates are GAR $= 91\%, 91\%, 86\%$, respectively, while the false acceptance rates vary between FAR $= 0.13\%, 0.01\%, 0\%$, respectively. While the security against a naive brute-force attack is estimated as (only) 2^{27}, 2^{31}, and 2^{39}, respectively, it must be mentioned that the incorporation of minutiae directions leak even more information about the protected template: Spatially close minutiae are unlikely to have inconsistent direction, but random generation of chaff points does not account for this.

Nagar et al. (2008, 2010) proposed to incorporate minutiae descriptors (see Section 2.2.8) as additional fingerprint features. The main purpose for this was to improve the security of the fuzzy vault vault scheme against both brute-force attack as well as false-accept attack. As in (Nandakumar et al. (2007a)) each minutia $\mathfrak{m}$ is mapped to a vault point $(x, y) \in \mathbf{F} \times \mathbf{F}$, where x encodes the minutia and y associates the minutia with the secret polynomial. A minutiae descriptor associated with $\mathfrak{m}$ is encoded as a 511-bit vector v. The ordinate value y is encoded as a codeword of the $(511, 19, 2 \cdot 119 + 1)$-BCH code. The fuzzy commitment $c + v$ is published instead of the clear value y. In this way, for each minutia an uncertainty of ≈ 2 bits[3] is added. If the $(511, 19)$-BCH code were perfect then the security against the brute-force attack would be increased by $\approx k \cdot 2$ bits. For $t = 24$, $n = 224$, and $k = 9$ the security would increase from 2^{31} to 2^{49} at a genuine acceptance rate of GAR $\approx 88\%$ where FAR $\approx 0\%$. However, the $(511, 19)$-BCH code is far from perfect. Nagar et al. (2008, 2010) mentioned the imperfectness of the $(511, 19)$-BCH code and in fact concluded that the security estimate of 2^{49} is not valid. The authors proposed countermeasures to repair for the minor security gain if the $(511, 19)$-BCH-code is used by replacing it with the $(31, 6)$ and $(15, 5)$-BCH-code, respectively. This requires dimension reductions of the minutiae descriptors. The effect of the imperfectness of these codes then is lessened but the authentication performances again approaches

---

[2] In (Nandakumar et al. (2007a)) the weight w affects minutiae distances with respect to the angular measure such that the weight w is 0.2 there.

[3] This corresponds to the difficulty in guessing a random minutiae descriptor encoded as a 511-bit vector in all but 119 bits (see Nagar et al. (2010)).

the performances of Nandakumar et al. (2007a). As a consequence, the improvement of the implementation's security or authentication performance is not as significant as one might think from a cursory look at the corresponding paper.

Merkle et al. (2010b) investigated performance of a fuzzy vault implementation based on using multiple fingers rather than a single finger. The implementation's resistance against the brute-force attack (see Section 3.1.3) is promising. The authors reported that up to 92% (on average) genuine minutiae were successfully re-identified on vault authentication in a test where their implementation and their privately collected database was used. If $t = 90$ genuine minutiae from three fingers were used for vault enrollment and authentication dispersed in a vault of size $n = 351$ hiding a polynomial of degree $< k = 42$ then Equation (3.8) evaluates to $\approx 2^{96}$. While a successful identification of 92% genuine minutiae promises a reasonable genuine acceptance rate, a complete investigation of authentication performance is still missing.

3.3 Alignment

Given a fuzzy fingerprint vault protecting a fingerprint's minutiae (with or without direction) it is necessary to align a query fingerprint to the vault, i.e. to the genuine vault minutiae. In contrast to aligning two minutiae templates in clear (see Section 2.2), the problem of aligning a query template to a fingerprint template that is encrypted by a fuzzy vault scheme is much more challenging.

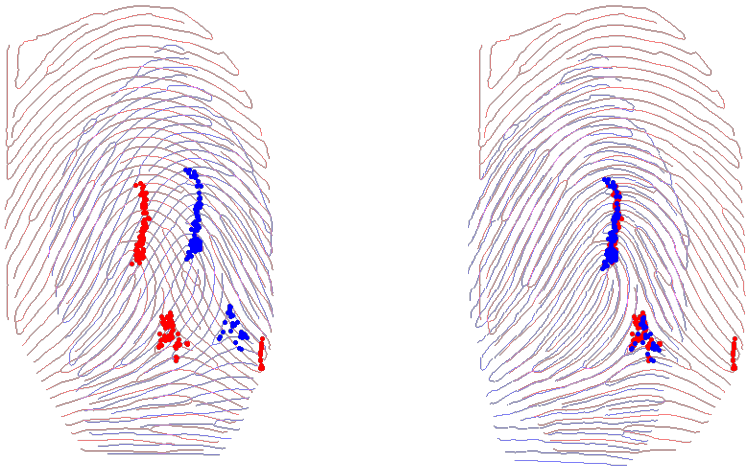
In the following, we give an overview of proposals for dealing with the alignment in a fuzzy fingerprint vault, which is one of the most critical factors that influences the authentication performance of a minutiae fuzzy vault.

Yang and Verbaudwhede (2005) proposed to extract some reliable reference minutiae from an enrolled minutiae template and publish them publicly along with the vault. To filter out reliable reference minutiae, multiple scans are needed. For each scan, a matcher is utilized to determine reliable minutiae correspondences. The most reliable minutiae are stored publicly along with the vault. On authentication, each query template's minutiae is aligned to each of the vault's reference minutiae, each giving a candidate for alignment. If one of the attempted alignments opens the vault, the authentication attempt is accepted, and is rejected otherwise.

Uludag and Jain (2006) proposed to store points corresponding to high curvature in the fingerprint's orientation field publicly along with the vault. These points, called *helper data*[4] in the corresponding paper, are obtained from *orientation field flow curves* (Dass and Jain (2004)). In such a way, the problem of aligning the query to the vault is translated to the problem of registering point clouds (see Figure 3.4). The registration point clouds is implemented using the *iterative closest point* algorithm (Besl and McKay (1992)) where for each point of high ridge curvature its corresponding curvature is used as an alignment-invariant feature (Sharp et al. (2002)). The method of Uludag and Jain (2006) for vault alignment has been refined by Nandakumar et al. (2007a). A major difference from the alignment implementation of Uludag and Jain (2006) is that the authors use the *trimmed iterative*

---

[4] The concept of helper data here is not to be confused with the concept of *helper data systems*, which is frequently used as a synonym to *biometric template protection schemes* in the literature.



(a) not aligned (b) aligned

Figure 3.4: query fingerprint (blue) aligned to reference fingerprint (red) using high curvature points; the high curvature points (red) are stored publicly along with the vault

closest point algorithm (Chetverikov et al. (2002)). As a consequence, the alignment is more robust to outliers.[5]

Jeffers and Arakala (2007) investigated three different structures for their capability to assist vault alignment. All of these structures can be derived from the corresponding minutiae template.

The first structure is the *five nearest neighbor structure*, which consists of the reference minutiae and the five minutiae closest to it. Similarly, the *Voronoi neighbor structure* has been investigated, which, as the five nearest neighbor structure, consist of the reference minutiae and all its Voronoi neighbors (the Voronoi cells are given by the template's minutiae locations). The third structure investigated is referred to as *triangle based structure*, which are triangles of where each vertex (each constituted with the local ridge orientation) corresponds to a minutia's location in the template. Note, that each vertex thus corresponds to an *undirected minutia*, i.e. a minutia with an undirected orientation.

For each structure type and for a different number of structure instances, Jeffers and Arakala (2007) investigated the enabled accuracy of alignment.[6] Among other results, it was reported that nine triangle based minutiae structures, respectively, suffice to align all but 5% of the query templates with an accuracy of 5 pixels.

Actually, according to the description of Jeffers and Arakala (2007), the authors did not store these structures publicly but created multiple fuzzy vaults for each structure instance. Each structure defines a coordinate system which can be reconstructed on authentication given a ground-truth matching structure can be derived from the query. However, excepting the different generation of chaff points in each vault instance, this is equivalent to storing the structures publicly along a single vault.

---

[5] Outliers might be caused by errors during the extraction or partial overlap of the fingerprints.

[6] Alignment accuracy of two fingerprints was defined as the distance between its two core points by Jeffers and Arakala (2007).

Li et al. (2008) proposed to store topological information along with the vault. This *topo-structure* is determined with the help of all minutiae that are of pre-defined spatial distance to the fingerprint's core, i.e. with the help of minutiae that beside in the *sampling area*. Furthermore, ridges associated with these minutiae are used for the construction of the topo-structure. The structure is constructed such that it does not reveal the actual minutiae locations around the core. But it still contains enough information such that a query fingerprint can be rotated with sufficient accuracy. The translation is determined by mapping the query's core point estimation to the core point estimation of the enrolled finger.

Merkle et al. (2010b) proposed a method for pre-aligning fingerprints for enrollment and for authentication. The coordinate system's origin is shifted to the fingerprint's foreground image center. Using the shifted coordinate system, the foreground image is divided into four quadrants. If the area of the upper left and lower right quadrants of the foreground image exceeds the area of the upper right and lower left quadrants, the fingerprint image is rotated clockwise by 1°. Otherwise, the fingerprint is rotated counterclockwise by 1°. The iteration stops if a subsequent test would cause a change in the rotation. Using this approach, the fingerprints are coarsely pre-aligned. In order to refine the alignment on authentication using the pre-aligned query fingerprint, an isometry is determined which maximizes the number of matches between minutiae in the query and the vault minutiae. To prevent the isometry from completely dis-aligning the pre-aligned query fingerprint, the isometry must have rotation and translation within pre-configured limits.

To avoid problems arising from the need to implement fingerprint alignment under security aspects, Li et al. (2010) proposed a fingerprint fuzzy vault cryptosystem that uses features that are independent of spatial movement. These features are given by *minutiae descriptors* (see Tico and Kuosmanen (2003) and Section 2.2.8) and *minutiae local structures* (Jiang and Yau (2000)). The authentication rates the authors reported are rather good when compared to other implementations: The genuine acceptance rate was reported as $\mathrm{GAR} = 92\%$ at a false acceptance rate of $\mathrm{FAR} = 0\%$.

3.4 Implementation

Investigations of the fuzzy fingerprint vault should be based on an implementation. We implemented the fuzzy fingerprint vault following different ideas found in the literature. Our vault construction mainly follows the approach of Nandakumar et al. (2007a), while the way we perform the vault unlocking is somewhat simpler.

3.4.1 Minutiae Selection

Given a minutiae template $T_{\text{in}} = \{\mathfrak{m}\}$ only those minutiae are selected as candidates for vault construction that keep a reasonable distance from other minutiae in the template. More precisely, for a bound $\delta_1 > 0$ a minutia $\mathfrak{m} \in T_{\text{in}}$ is selected only if $d(\mathfrak{m}, \mathfrak{m}') \geq \delta_1$ for all $\mathfrak{m}' \in T_{\text{in}} \setminus \{\mathfrak{m}\}$. Here $d(\mathfrak{m}, \mathfrak{m}')$ measures the distance between $\mathfrak{m}$ and $\mathfrak{m}'$ as in Equation (2.5) on page 19 with some weight w incorporating minutiae angles. For the set of well-separated minutiae we write

$$T_{\text{sep}} = \{\ \mathfrak{m} \in T \mid d(\mathfrak{m}, \mathfrak{m}') \geq \delta_1 \text{ for all } \mathfrak{m}' \in T_{\text{in}} \setminus \{\mathfrak{m}\}\ \}. \tag{3.17}$$

Moreover, we assume that T_{sep} is given as a list sorted by the minutiae's quality (Chen et al. (2005)). If T_{sep} contains more than t minutiae then only the first t minutiae of best quality are selected. Otherwise, all well-separated minutiae are selected. For the set of selected minutiae we write T_{select}.

3.4.2 Enrollment

Given a minutiae template T_{enroll} genuine minutiae T_{gen} are selected from it as described in Section 3.4.1.

In case it is not possible to extract at least $t_{\min}$ minutiae this way, the enrollment is counted as a *failure to capture*. Otherwise if T_{gen} contains between $t_{\min}$ and t minutiae the construction of the vault continues as follows.

A set of well-separated chaff minutiae T_{chaff} of sufficient distance to genuine minutiae are generated at random such that T_{chaff} reaches a size of $n - t'$ where $t' = |T_{\text{gen}}|$. More precisely, for all chaff minutiae $\mathfrak{m} \in T_{\text{chaff}}$ the inequality $d(\mathfrak{m}, \mathfrak{m}') \leq \delta_1$ for all $T_{\text{gen}} \cup T_{\text{chaff}} \setminus \{\mathfrak{m}\}$ holds. In this way, an adversary is not able to distinguish genuine from chaff minutiae simply by considering the distances to their closest neighbors.

Nandakumar et al. (2007a) proposed to generate a fixed number of chaff minutiae, whereas in our implementation the number of chaff minutiae depends on the number of genuine minutiae such that there is a total of n vault minutiae. Vault minutiae consist of both genuine minutiae and chaff minutiae, i.e.

$$T_{\text{vault}} = T_{\text{gen}} \cup T_{\text{chaff}}. \tag{3.18}$$

Note that when T_{vault} is represented as a list containing minutiae one has to assert that it is not possible to distinguish genuine minutiae from chaff minutiae simply from their respective list positions. A valid way to hide genuine points within T_{vault} is to sort its elements by some convention, e.g., the lexicographical order.

In contrast to the implementation of Nandakumar et al. (2007a) we do not encode the elements of T_{vault} as elements of a finite field. Rather we encode only the list positions in T_{vault} as elements of a finite field as follows. Write

$$T_{\text{vault}} = \{\mathfrak{m}_1, \ldots, \mathfrak{m}_n\}. \tag{3.19}$$

Then the finite field element associated with $\mathfrak{m}_i$ depends on i only. An advantage of this approach is that the minutiae do not need to be quantized furthermore, which would cause loss of information. Moreover, there is more freedom in the choice of the finite field (see Merkle et al. (2010a)).

The finite field is chosen as $\mathbf{F} = \mathbb{F}_{2^{16}}$ although a smaller finite field would be sufficient. By

$$E : T_{\text{vault}} \to \mathbf{F} \tag{3.20}$$

we denote an injective map encoding the list positions of vault minutiae as elements of the finite field as outlined above. Furthermore, a polynomial $f \in \mathbf{F}[X]$ in the indeterminate X and of degree $< k$ is chosen as the secret key.

The genuine vault points are

$$\mathbf{G} = \{\ (x, f(x)) \mid \mathfrak{m} \in T_{\text{gen}},\ x = E(\mathfrak{m})\ \} \tag{3.21}$$

where the chaff points are generated randomly as

$$\mathbf{C} = \{\ (x, y) \mid \mathfrak{m} \in T_{\text{chaff}},\ x = E(\mathfrak{m}),\ y \in_R \mathbf{F} \setminus \{f(x)\}\ \}. \tag{3.22}$$

The set of vault points

$$\mathbf{V} = \mathbf{G} \cup \mathbf{C} \tag{3.23}$$

consist of both genuine and chaff points.

To enable safe recovery of the secret polynomial, a hash value $h(f)$ is computed from a concatenation of the k coefficients of the polynomial f and stored along with the vault. Although there exists more secure algorithms to compute cryptographic hashes (see National Institute of Standards and Technology (2002)) we claim that the *secure hash algorithm* (*SHA-1*) (see National Institute of Standards and Technology (1995) and Section 2.1.1) is sufficient for our evaluation purposes.

Finally, the public vault data is

$$(\mathbf{V}, T_{\text{vault}}, h(f)). \tag{3.24}$$

The construction of the vault depends on the following parameters:

- w controls how significant angles are accounted for measuring the distance of minutiae;
- δ_1 controls the mutual distances between vault minutiae;
- t is an upper bound on the number of genuine minutiae selected from the enrollment template;
- $t_{\min}$ is a lower bound on the number of genuine minutiae selected from the enrollment template. If only fewer than $t_{\min}$ minutiae can be selected, the enrollment is aborted and counted as a *failure to capture*;
- n denotes the number of vault points. If $t' = t_{\min}, \ldots, t$ genuine minutiae have been selected then $r = n - t'$ chaff points are generated such that there is a total of n vault points;
- k is the number of coefficients of the secret polynomial f;

3.4.3 Authentication

On authentication, a vault instance $(\mathbf{V}, T_{\text{vault}}, h(f))$ and a minutiae template T_{query} aligned to the vault are given. Using T_{query} the aim is to reconstruct the secret f. A subset T_{access} is selected from T_{query} in the same way as T_{gen} has been selected from T_{enroll} on enrollment (see Section 3.4.1).

For all $\mathfrak{m} \in T_{\text{access}}$ a minutiae $\text{nearest}(\mathfrak{m}) \in T_{\text{vault}}$ that best approximates $\mathfrak{m}$ is determined. In case $\text{nearest}(\mathfrak{m})$ is of distance to $\mathfrak{m}$ below a threshold δ_2 then $\text{nearest}(\mathfrak{m})$ is chosen to unlock the vault. In this way the set of unlocking minutiae $T_{\text{unlock}} \subset T_{\text{vault}}$ is extracted. The vault points corresponding to the elements of T_{unlock} yield the unlocking set

$$\mathbf{U} = \{\ (x,y) \in \mathbf{V} \mid \mathfrak{m} \in T_{\text{access}},\ x = E(\text{nearest}(\mathfrak{m})),\ d(\mathfrak{m},\mathfrak{m}) < \delta_2\ \}. \tag{3.25}$$

Decoding

Analogous to most implementations (see Section 3.2.4), for all combinations of k unlocking points from the unlocking set of size $|\mathbf{U}| = |T_{\text{access}}| \leq t$, its unique interpolation polynomial f^* of degree $< k$ is computed. If $h(f^*) = h(f)$ then $f^* = f$ with very high probability and f^* is output as the correct secret polynomial, in which case the authentication is considered to be successful. This will be the case if and only if the unlocking set $\mathbf{U}$ contains at least k genuine points from $\mathbf{G}$. Otherwise if no f^* with $h(f^*) = h(f)$ can be found, then the query is rejected.

3.4.4 Alignment

On decoding, we assume that the query fingerprints are aligned to the vaults. To simulate our evaluations in a well-solved alignment framework, we decouple the alignment work from the vault. Consequently, if an authentication of a genuine user is simulated, the alignment is obtained by aligning both minutiae templates in clear.

We follow an alignment approach similar to the description of Section 2.2.2. Every minutia of the query template is matched against every minutia of the enrolled template. Incorporating the minutiae angles, an isometry is obtained (see Equation (2.8) on page 19). The quality of the alignment is evaluated by scoring how well the 5 best ridge segment correspondences agree. Finally, the best of such alignments is refined by minimizing the sum of the squared distances of matching points Eggert et al. (1997). The point correspondences are obtained from the 5 best matching segments. The valuation of the quality of an alignment using ridge segments has been adopted from Jain et al. (1997).

The reasons why we decouple the alignment work from the vault are the following. Many minutiae fuzzy vault implementations propose a mechanism enabling vault alignment using auxiliary alignment helper data (see, for example, Nandakumar et al. (2007a); Uludag and Jain (2006); Yang and Verbaudwhede (2005)). Furthermore, there are publications that propose alignment helper data structures to assist alignment for an arbitrary minutiae fuzzy vault implementation (see Jeffers and Arakala (2007); Li et al. (2008)). For a review of fuzzy vault alignment schemes see Section 3.4.4. If our fuzzy vault implementation would use such additional auxiliary alignment helper data then the vault performance would also depend on the performance of such an alignment scheme. Moreover, information leaked by the alignment helper data about the finger affects overall vault security. In the literature, the security of a minutiae fuzzy vault is often related to the amount of information that is leaked about the minutiae template. Even if alignment data would leak no information about minutiae in the template, it does, however, leak information about the finger's individuality. This may help an intruder to cross-match vaults only by matching the alignment helper data (see Chapter 4 for the concept of cross-matching). The amount of leaked information from the alignment helper data as proposed by Li et al. (2008); Nandakumar et al. (2007a); Uludag and Jain (2006), for instance, have not been estimated.

For these reasons, it seems better for us to investigate vault performance and security under a well-solved alignment framework assuming nothing is leaked about the fingerprint. Vault alignment performance (or ideally, pre-alignment performance) as well as the amount of leaked information about the corresponding finger can be analyzed independently from the remaining vault work.

3.4.5 Evaluation Database

We conducted performance evaluation of our vault implementation on the FVC 2002 DB2-A database (Maio et al. (2002)). The FVC 2002 DB2-A database consist of a total of 800 scans from 100 different fingers (index $1, \ldots, 100$). Eight scans of each finger are contained in the database.

For vault encoding and decoding, minutiae templates are required where each minutiae template is assumed to be given as a vector containing minutiae sorted in descending order of their qualities. For the performance evaluation the extraction of minutiae and the extraction of their respective qualities was accomplished with the help of a commercial extractor.[7]

3.4.6 Configuration

For evaluation of the vault implementation on the FVC 2002 DB-A database we adopted the configuration proposed by Nandakumar et al. (2007a), which is given by the following parameters:[8]

$$\begin{aligned} w &= 11.46, \\ \delta_1 &= 25, \\ \delta_2 &= 30, \\ t &= 24, \\ t_{\min} &= 18, \text{ and} \\ n &= 224; \end{aligned} \tag{3.26}$$

the size of the secret polynomial k corresponds to a similarity threshold and authentication performances were determined for varying k in our experiments.

3.4.7 Evaluation Protocol

The genuine acceptance rate as well as the false acceptance rate have been determined according to the FVC protocol (Maio et al. (2002)).

To determine the genuine acceptance rate, each template labeled $j = 1, \ldots, 7$ of the ith finger ($i = 1, \ldots, 100$) was used to build a vault instance of configuration as above. If a *failure to capture* was encountered, a vault for the next finger is constructed. Otherwise for each vault instance successfully built this way and for each $j' = j+1, \ldots, 8$ the j'th template of the ith finger aligned to the vault was used for an authentication attempt. If successful, the event was counted as a successful genuine accept. The fraction of successful genuine accepts to the overall number of genuine authentication attempts yielded the genuine acceptance rate. Note that if no failure to capture has been encountered the number of genuine authentication attempts was $100 \cdot (7 \cdot 8)/2 = 2800$.

The false acceptance rate was determined by attempting to open all vault instances for the ith finger being successfully built from the first templates, where $i = 1, \ldots, 99$, using the first templates of all i'th finger with $i' = i + 1, \ldots, 100$,

---

[7] We used Verifinger SDK 5.0 (Neurotechnology Ltd (2006)) for minutiae feature extraction.

[8] The weight w has been chosen as 0.2 by Nandakumar et al. (2007a) with respect to the angular measure. This choice corresponds to $w = 11.46$ in radian.

respectively. If one of these authentication attempts was successful, this was counted as a false accept. The number of successful false accepts within the total number of impostor authentication attempts yielded the false acceptance rate. Again note, if no failure to capture has been encountered there were a total of $99 \cdot 100/2 = 4950$ impostor authentication attempts.

3.4.8 Sub-Database of Good Quality

Within the literature, for the majority of fuzzy fingerprint vault implementations, authentication performances on a subset of FVC 2002 DB2-A have been reported but not on the complete database (see (Nagar et al. (2010); Nandakumar et al. (2007a,b); Uludag and Jain (2006))). This subset consists of only the first two impressions of each of the 100 fingers. If the sub-database is used to determine the genuine acceptance rate according to the FVC protocol, then there is only one authentication attempt per finger: The first scan is used to construct the vault and the second scan is used to open the vault. As a consequence, for the majority of fuzzy fingerprint vault implementations in the literature the estimation of the genuine acceptance rate has been obtained from a test set of size at most 100 only (assuming no failure to capture occurred).

On the other hand, the first two impressions of the database have a bias to be of better quality. Furthermore, there is less transformation between the first two impressions of the same finger. Therefore, one expects to observe higher genuine acceptance rates.

For the majority of fuzzy fingerprint vault implementations the evaluation of the false acceptance rate has been performed differently from the FVC protocol. Instead of using each i-th fingerprint for vault construction and then, correspondingly, each j-th finger where $j = i + 1, \ldots, 100$ as a query, they have used each j-th finger with $j = 1, \ldots, 100$ but $j \neq i$ as a query. As a result, the number of impostor authentication attempts is increased from 4950 to 9900 (barring failure to captures). However, for vault implementations, even if there is a difference between using a template S for enrollment and another template T as query ans using T for enrollment and S as a query both observations are not independent. For this reason, we will report our false acceptance rates analogous to the FVC protocol.

3.4.9 Performance Evaluation

We determined authentication performance for both the whole FVC 2002 DB2-A database as well as for the sub-database of good quality. The *genuine acceptance rate*, *false acceptance rate*, and *failure to capture rate* on the entire FVC 2002 DB2-A database are denoted by GAR, FAR, and FTCR, respectively. Consequently, on the sub-database of good quality the genuine acceptance and the failure to capture rate are denoted by sub-GAR and sub-FTCR, respectively. Moreover, the *average genuine decoding time* and the *average impostor decoding time* are denoted by GDT and IDT, respectively.

For example, when the FVC protocol on the whole database was applied with a secret polynomial length of $k = 9$ then

$$\begin{aligned} \mathrm{GAR} &\approx 62.51\%, \\ \mathrm{FAR} &\approx 0.33\%, \text{ and} \\ \mathrm{FTCR} &\approx 2.43\% \end{aligned} \tag{3.27}$$

Table 3.1: Authentication and computational performance measured on the FVC 2002 DB2 (Section 3.4.7) and on the subset of good quality (Section 3.4.8). The rates measured on the subset of good quality are indicated in brackets. The genuine acceptance rates correspond to the observation of 2761 genuine authentication attempts, whereas the false acceptance rate is based on 4856 impostor authentication attempts. Furthermore, 99 genuine authentication attempts have been observed for the genuine acceptance rates of the sub-database of good quality. The timings have been measured on a single core of an `AMD Phenom(tm) II X4 955` desktop processor.

length of secret polynomial	genuine acceptance rate GAR (sub-GAR)	false acceptance rate FAR	average genuine decoding time GDT	average impostor decoding time IDT	failure to capture rate FTCR (sub-FTCR)
$k = 7$	$\approx 77.04\%$ ($\approx 94\%$)	$\approx 3.81\%$	$\approx 0.056s$	$\approx 0.09s$	$\approx 2.43\%$ ($= 1\%$)
$k = 8$	$\approx 70.26\%$ ($\approx 90\%$)	$\approx 1.69\%$	$\approx 0.132s$	$\approx 0.178s$	"
$k = 9$	$\approx 62.51\%$ ($\approx 87\%$)	$\approx 0.33\%$	$\approx 0.26s$	$\approx 0.247s$	"
$k = 10$	$\approx 55.31\%$ ($\approx 85\%$)	$\approx 0.21\%$	$\approx 0.418s$	$\approx 0.291s$	"
$k = 11$	$\approx 47.74\%$ ($\approx 82\%$)	$\approx 0.04\%$	$\approx 0.58s$	$\approx 0.305s$	"
$k = 12$	$\approx 39.95\%$ ($\approx 73\%$)	$= 0\%$	$\approx 0.677s$	$\approx 0.245s$	"

where the *genuine acceptance rate* and the *failure to capture rate* on the sub-database were determined as

$$\begin{aligned} \text{sub-GAR} &\approx 87\% \text{ and} \\ \text{sub-FTCR} &= 1\% \end{aligned} \tag{3.28}$$

respectively. Moreover, on a single core of an `AMD Phenom(tm) II X4 955` the average time for a genuine user to be either authenticated or rejected were measured as approximately

$$\text{GDT} = 260\ ms \tag{3.29}$$

where the average time for an impostor was experimentally observed as approximately

$$\text{IDT} = 247\ ms. \tag{3.30}$$

For different secret polynomials length k corresponding rates can be found in Table 3.1.

3.4.10 Discussion

Our implementation differs from the implementation of Nandakumar et al. (2007a) in the following aspects:

First, minutiae locations and angles are not rounded to a coarser representation system. Rather, we encode each minutia by its index in the list of vault points. The reason why Nandakumar et al. (2007a) performed coarse minutiae quantization is that it allows a minutia to be encoded as a 16-bit field element. Applying the simple and elegant idea of encoding minutiae by its list position (Merkle et al. (2010a)), the quantization step becomes obsolete.

Second, for vault decoding we assume that the queries are aligned to the vault.

Nandakumar et al. (2007a) proposed a mechanism enabling automatic fingerprint alignment. This is achieved by storing points of high ridge curvature publicly along with the vault as auxiliary alignment helper data. However, to report vault performance independently of alignment performance, we decouple the alignment (also see Section 3.4.4).

Third, our method to decode the vault is somewhat simpler than the method used by Nandakumar et al. (2007a). Nandakumar et al. (2007a) make use of a minutiae matcher (Jain et al. (1997)) while our implementation works by extracting those minutiae that are closest to a selected query minutiae. Although the genuine acceptance rate seems to be unaffected, the false acceptance rate increases if our simpler decoder is used.

Finally, we note that the decoding for our implementation is faster. The average decoding timings of Nandakumar et al. (2007a) are reported as 8 s. Our implementation, however, requires an average genuine decoding time of only 677 ms, if $k = 12$. The processor that was used by Nandakumar et al. (2007a) has a clock rate of 3.4 `GHz` while we performed our tests on a 3.2 `GHz` processor. Therefore, one would expect that the timings of Nandakumar et al. (2007a) substantially agree with our timings. The reason that this is not the case can be explained by the fact that the implementation of Nandakumar et al. (2007a) is based on `Matlab` (Gilat (2011)) while our implementation is based on `C++` (Stroustrup (2000)).

4. Vulnerabilities

Authentication systems require that a server stores information about enrolled users. In password based authentication schemes this information usually consists of cryptographic hash values of user passwords stored along with their corresponding user names. Similar assumptions can be made for biometric authentication systems that are based on biometric template protection schemes, such as the fuzzy vault for fingerprints: Vault records are stored along with their corresponding user names. The aim for both cryptographic hash functions and biometric template protection schemes are that a user to successfully authenticate must provide evidence for that he is authorized. A password that differs from the one used to construct the hash value will produce (with overwhelming probability) a different hash value. Similarly, a fingerprint template differing too much from the one used on enrollment will not succeed in opening the vault. Furthermore, it should be infeasible for any person to find out a user's password/template only from the data that is stored on the server's database.

Note that even if the database's content is encrypted using techniques from traditional cryptography there are persons who are able to read the decrypted data, such as system administrators. Even worse, system administrators may even read the database's content such that the theft will not be recognized until user accounts get broken eventually. For these reasons, the protection of biometric templates (passwords) via biometric template protection schemes (hash functions) is necessary.

The critical question is how difficult it is to reconstruct templates (passwords) from vault records (hash values). Usually, the difficulty of performing such an attack is expressed in terms of the number of operations that are needed to successfully perform the reconstruction. A more intuitive way to express the difficulty of an attack is to report the time it consumes on a particular computer.

For example, if a hash value $h(p)$ of a user password has been intercepted by an intruder, he might try to find a collision, i.e. find a second password p^* such that $h(p) = h(p^*)$. In this way, the attacker can use the password p^* to gain user access to the system. If the SHA-1 hash function was used, finding such a password via brute-force is expected to require approximately 2^{80} iterations in where a random password p^* is chosen as a candidate for a collision. This corresponds to an effort for that one commonly agrees that this is infeasible to perform for the attacker.

However, for password based authentication schemes, as above, usually another approach is more efficient to find a collision. Usually, the attacker can assume that the passwords are chosen from a much smaller pool, say of size $10,000$ in the case of a four digit pin. Now, the attacker only needs to iterate through at most $10,000$ passwords p^* until $p = p^*$ and thus $h(p) = h(p^*)$. This corresponds to a security of approximately 2^{13}, which is an affordable amount of time for the attacker. We see that in the example the weakest link of the above authentication scheme is the small pool of passwords and not the security that the hash functions would be able to provide.

Similar observations can be made when one considers biometric authentication schemes, such as one that is based on, say, the implementation described in Section 3.4. Assuming minutiae are distributed independently and uniformly at random, guessing a minutiae template that is successful in opening a given vault record is approximately as hard as running the naive brute-force attack introduced on page 39. Thus, if the configuration of Section 3.4.6 and a secret length $k = 9$ is used, the difficulty to find an authenticating template would be approximately 2^{31}.

However, as experimentally observed in Section 3.4.9 the probability that another minutiae template successfully opens the vault is $\approx 0.33\%$. Thus, if the attacker iterates through a database containing a large amount of preliminarily collected minutiae templates extracted from real fingers he can expect to successfully open the vault after $\log(0.5)/\log(1 - 0.33\%) \approx 2^8$ iterations, which is much faster. We already referred to such kinds of attacks as *false-accept attacks*.

In Section 4.1 we analyze the fuzzy vault for fingerprints against the brute-force attack. In particular, we perform experiments against fuzzy vaults of parameters that come from implementations found in the literature. Furthermore, variations of brute-force attacks are analyzed against the hybrid implementation proposed by Nagar et al. (2008, 2010), which incorporates minutiae descriptors protected via the fuzzy commitment scheme. The error-correcting codes used for the fuzzy commitment scheme are imperfect. This, however, can be used to attack the fuzzy commitments before running the actual brute-force attack. As a consequence, there is less security gain as it were possible when the error-correcting codes were perfect. Nagar et al. (2010) considered the choice of other error-correcting codes of higher sphere packing density (see Section 2.3.3) to preserve some of the potential security. As a consequence there is a significant security gain against brute-force attacks but the genuine acceptance rate drops correspondingly. Moreover, brute-force attacks are just one of the risks of the fuzzy fingerprint vault.

In Section 4.3 we analyze how secure our implementation from Section 3.4 can be against the false-accept attack. Although a single iteration in the false-accept attack may require more computer-time than an iteration of the brute-force (since it comprises the decoding of a perturbed unlocking set while the iterations in the brute-force attack consist of a single polynomial interpolation) there remains a significant gain for the attacker when he uses the false-accept attack (to compare the efficiency of attacks, we compare their respective overall computing time they consume on an `AMD Phenom(tm) II X4 955` processor, which consist of four processor cores with 3.2`Ghz`). Furthermore, we show how the hybrid implementation of Nagar et al. (2008, 2010) can be attacked using the false-accept attack while the fuzzy commitment scheme instances it incorporates are decoupled previously as it is possible in a brute-force attack. We conclude that the security gain is not as significant as one may think from a cursory look in (Nagar et al. (2008, 2010)).

Fuzzy vaults to single-finger are currently too vulnerable against brute-force attacks and even much more vulnerable against false-accept attacks to provide cryptographic security. Consequently, there should be more investigations of the fuzzy vault applied to multi-finger (see Merkle et al. (2010b)). However, there is another difficulty with fuzzy fingerprint vaults that can not be solved merely by using multi-finger.

A big security risk of the fuzzy fingerprint vault is its high vulnerability against an *attack via record multiplicity* (see Scheirer and Boult (2007)): Experiments have shown that an adversary has a chance to break two vault records with probability $\approx 59\%$ (see Scheirer and Boult (2007) and Kholmatov and Yanikoglu (2008)) given both the vaults protect minutiae templates from the same finger using an attack that makes use of correlating vault minutiae. This chance for a successful *attack via record multiplicity* (Scheirer and Boult (2007)) is apparently too high to claim that the fuzzy fingerprint vault is safe against attacks via record multiplicity. Furthermore, using correlation the fuzzy vault for fingerprints is vulnerable against *cross-matching*: A thief who has intercepted the database content of two or more applications that contain fuzzy vault records can consider the correlation of vault minutiae to filter out genuine vault correspondences. While mere cross-matching is already a security issue (an intruder might attempt to trace a particular user's activity) the ability for an attacker to find genuine vault correspondences across different databases may be a major help when he is attempting to attack vaults via record multiplicity.

In Section 4.4 we derive a criterion for cross-matching via correlation and thus a criterion for when an attacker may decide to run the corresponding attack via record multiplicity. Furthermore, the cross-matching criterion we give is independent from which fingerprint features are used for the vault. It apparently applies to every vault implementation where genuine feature in two vault records agree well in comparison to non-genuine vault features. Thus, merely substituting minutiae features by other features (such as alignment-free features as proposed by Li et al. (2008)) will not suffice to secure a fuzzy fingerprint vault against cross-matching or the correlation attack via record multiplicity. After we define the attacker's approach to perform cross-matching using the derived criterion and possibly a subsequent correlation attack, we simulate it in an own experiment against our implementation of Section 3.4. The resulting *cross-matching rates* and *successful correlation attack rates* clearly confirm that fuzzy vaults are highly vulnerable against cross-matching and an attack via record multiplicity, i.e. the *correlation attack*, which requires a solution (see Scheirer and Boult (2007) and Kholmatov and Yanikoglu (2008)) .

4.1 Brute-Force Attack

The estimate of a brute-force attack against a fuzzy fingerprint vault implementation is an irrefutable upper bound for the security of that implementation. For this reason, we give an overview of the efficiency of brute-force attacks against different implementations from the literature. Moreover, at the end of this section we formulate two kinds of brute-force attacks against an implementation of Nagar et al. (2008, 2010). This implementation incorporates *minutiae descriptors* (see Section 2.2.8) to improve security and to drop the false acceptance rate. An important ingredient of the implementation is the *fuzzy commitment scheme* (see Section

2.4), which is based on error-correcting codes (see Section 2.3). Our attacks make use of information leaked from the imperfectness of the error-correcting codes (see Section 2.3.3), i.e. a low sphere packing density of the code. In this way, attacking the fuzzy commitment schemes can be decoupled from attack the remaining vault to some extent. Furthermore, the decoupling can also be combined with the *false-accept attack* that we will discuss later in Section 4.3. An impact from this is that under some circumstances the false-accept attack against the implementation with minutiae descriptor is almost as efficient as the false-accept attack without minutiae descriptors.

4.1.1 The Ordinary Brute-Force Attack

Let $\mathbf{V}$ be a vault of size n with t genuine points interpolated by a secret polynomial of degree $< k$. An attacker might run the following brute-force attack as we have it already seen in Section 3.1.3:

Algorithm 4.1.1 (Brute-Force Attack)**.**

Input: vault instance $\mathbf{V}$ hiding a polynomial of degree k interpolating t genuine points;
a criterion for identifying the correct polynomial;
Output: a polynomial interpolating t vault points;

```
1: select $k$ different vault points at random;
2: determine its interpolation polynomial $f^*$;
3: if $f$ fulfills the criterion for being correct, output $f^*$; otherwise, continue with Step 1.
```

Depending on the vault implementation, there are multiple ways to identify f^* as the correct polynomial f.

Criterion for Identifying the Correct Polynomial

If a cryptographically strong hash value of the correct polynomial is stored along with the vault, a criterion for deciding whether f^* in Step 3 of the brute-force attack is correct is to check whether $h(f^*) = h(f)$. The evaluation of hash functions and comparison of hash values can be done very efficiently and eases the unlocking work on an authentication attempt. However, the storing the correct polynomial's hash value enables the attacker also to run a faster brute-force attack.

If the vault does not store additional information about the correct polynomial, a criterion, which always works for a fuzzy fingerprint vault, is to check how many vault points f^* interpolates. With high probability f^* is the correct polynomial if t vault points are interpolated by f^* (see Lemma 3.2.1 on page 41).

For instance, Li et al. (2010) propose to use the SHA-2 hash function (see Section 2.1.2).

In (Nagar et al. (2008, 2010); Nandakumar et al. (2007a); Uludag and Jain (2006); Uludag et al. (2005)) the correct polynomial is constituted with information for a cyclic redundancy check. The idea is similar to the idea of storing a cryptographic hash value along with the vault: Once the polynomial f^* is seen, it can be efficiently verified whether f^* carries the cyclic redundancy check information.

Efficiency of the Ordinary Brute-Force Attack

For cryptanalysis it is interesting to compute the average number of iterations an attacker has to run before he succeeds with breaking a random vault. The chance that the interpolation polynomial of k randomly selected vault points will be the correct polynomial is

$$\mathbf{bf}(\mathbf{V}) = \mathbf{bf}(n,t,k) = \binom{n}{k} \cdot \binom{t}{k}^{-1}. \tag{4.1}$$

As a consequence, the probability for an attacker to succeed after $\mathcal{I}$ iterations is

$$1 - (1 - \mathbf{bf}(n,t,k)^{-1})^{\mathcal{I}}. \tag{4.2}$$

The average number of iterations an attacker needs for successfully breaking the vault coincides with the minimal number of iterations $\mathcal{I}$ such that

$$1 - (1 - \mathbf{bf}(n,t,k)^{-1})^{\mathcal{I}} \geq 0.5. \tag{4.3}$$

Thus, the expected number of iterations for a success is estimated by

$$\mathcal{I} = \left\lceil \frac{\log(0.5)}{\log(1 - \mathbf{bf}(n,t,k)^{-1})} \right\rceil \tag{4.4}$$

For example, consider the implementation of Uludag and Jain (2006) where $t = 24$ minutiae are hidden within a vault of size $n = 224$ interpolating a polynomial of degree $< k = 9$. According to Equation (4.4) an attacker can expect to break such a random vault within $\mathcal{I} \approx 1.76 \cdot 10^9$ iterations.

4.1.2 Experiments

Another important size affecting the efficiency of the brute-force attack is the number of iterations per time unit an attacker is capable to perform. Therefore, we have experimentally determined how many iterations in the brute-force attack are possible to perform on an `AMD Phenom(tm) II X4 955` processor with 3.2 `Ghz` using an own `C++` implementation.

Once the vault parameters n, t, and k were given we constructed 10 different vault instances at random. For each vault instance we, first, randomly selected a polynomial f of degree $< k$ with coefficients in the finite field $\mathbb{F}_{2^{16}}$. Second, we have randomly chosen t different abscissa values from the field $\mathbb{F}_{2^{16}}$. Third, the abscissa values x were evaluated on f to obtain the genuine vault points $(x, f(x))$. Finally, $n-t$ chaff points (x_c, y_c) were randomly generated with $f(x_c) \neq y_c$ such that all x_c are distinct and also no abscissa values of genuine points. Furthermore, the set of points were randomly shuffled. To estimate the number of iterations an attacker is able to perform each second, we measured the time consumed by the first $100,000$ iterations of the brute-force attack. The final estimation has been deduced by averaging all of these 10 measurements for each vault instance.

As in Algorithm 4.1.1, within each iteration of the brute-force attack we 1) selected k random vault points, 2) determined its interpolation polynomial f^*, and

3) tested how many vault points are interpolated by f^* by evaluating f^* on $n-k$ additional vault points.

For example, against a vault instance of parameters as in Uludag and Jain (2006), we were able to iterate through 37,160.9 polynomials per second. Thus, an attacker can expect to succeed after approximately $1.76 \cdot 10^9 / 30,459.9 \approx 47,432$ seconds, which corresponds to ≈ 13 hours. If all four processor cores were run in parallel the genuine polynomial — and thus the template too — would show up after an expected time of just approximately $3-4$ hours.

Table 4.1: Brute-force attack performance on an `AMD Phenom(tm) II X4 955` with four processor cores of 3.2 `Ghz`. Within each iteration the correctness of the candidate polynomial is checked by testing how many vault points it interpolates.

parameters as in	comment	alignment	GAR (FAR)	security in bits	polynomials per second per core	**expected time before successful iteration**
Uludag et al. (2005)		manually	79% (0%)	≈ 36	36,643.5	$\approx$ 3 days
Uludag and Jain (2006)		automatic	72.6% (0%)	≈ 31	37,160.9	3–4 hours
Nandakumar et al. (2007a)		automatic	91% (0.13%)	≈ 27	43,047.8	$\approx$ 13 minutes
"		automatic	91% (0.01%)	≈ 31	37,160.9	3–4 hours
"		automatic	86% (0%)	≈ 39	21,588.9	$\approx$ 52 days
Clancy et al. (2003)		manually	- (-)	≈ 50	17,850.8	293–294 years
Li et al. (2010)	alignment-free		94% (0.04%)	≈ 48	13,900.5	$\approx$ 102 years
"	"		92% (0%)	≈ 52	13,269.6	$\approx$ 1,693 years
Merkle et al. (2010b)	2 finger	automatic	- (-)	≈ 72	8,859.75	$\approx 4.1 \cdot 10^9$ years
Merkle et al. (2010b)	3 finger	automatic	- (-)	≈ 95	3,348.63	$\approx 8.51 \cdot 10^{16}$ years
Merkle et al. (2010b)	3 finger	automatic	- (-)	≈ 95	4,924.17	$\approx 5.15 \cdot 10^{16}$ years

In Table 4.1 for different implementations of the fuzzy fingerprint vault, the expected times of a successful attack are reported.[1]

While the correct polynomial can be identified by counting the number of vault points it interpolates, an attacker can often identify the correct polynomial more efficiently.

Identification of the Correct Polynomial by its Hash Value

In the case a hash value $h(f)$ of the secret polynomial is stored along with the vault, within each iteration of the brute-force attack the step of evaluating f^* at a large number of vault points can be circumvented: It is sufficient to compute the hash value $h(f^*)$ and to compare it with $h(f)$.

We modified the above experiments to simulate this situation. For each vault instance we additionally computed the correct polynomial's SHA-1 hash value. Within each iteration, after interpolation, we did not evaluate the candidate polynomial f^* on vault points but computed its SHA-1 hash value instead and compared it with the correct polynomial's SHA-1 hash value.

For example, against vault instances of parameters as in Uludag and Jain (2006), we were able to test 124,533 candidate polynomials per second on a single

---

[1] In Table 4.1 the vault configuration referred by Uludag et al. (2005) corresponds to $(n,t,k) = (218,18,9)$, Uludag and Jain (2006) to $(n,t,k) = (224,24,9)$, Nandakumar et al. (2007a) corresponds to $(n,t) = (224,24)$ where $k = 8$, 9, and 10 for FAR = 0.13%, 0.01%, and 0%, respectively. Furthermore, the configuration referred with Clancy et al. (2003) corresponds to their "optimal" configuration $(n,t,k) = (313,38,15)$ (see page 41). The configuration of Li et al. (2010) corresponds to $(n,t,k) = (440,40,12)$, and $(440,40,13)$. The configurations of the multi-finger implementation of Merkle et al. (2010b) corresponds to $(n,t,k) = (240,52,28)$, $(351,90,42)$, and $(360,70,35)$.

processor core. If all four processor cores were used, this would correspond to only approximately 59 minutes of computing time in where an attacker can expect to break the vault.

Table 4.2: Brute-force attack performance on an `AMD Phenom(tm) II X4 955` with four processor cores of 3.2 `Ghz` against our implementation of Section 3.4 with configuration as in Section 3.4.6 and different polynomial degrees $< k$

length of secret polynomial	security in bits	polynomials per second per core	**expected time before successful iteration**
$k = 7$	≈ 24	$184,162$	13–14 seconds
$k = 8$	≈ 27	$154,560$	3–4 minutes
$k = 9$	≈ 31	$124,533$	$\approx$ 59 minutes
$k = 10$	≈ 35	$111,359$	15–16 hours
$k = 11$	≈ 39	$97,656.2$	11–12 days
$k = 12$	≈ 43	$83,822.3$	$\approx$ 7 months

Our implementation of Section 3.4 also identifies the correct polynomial by its SHA-1 hash value. For the vault parameters as in Table 3.1 the corresponding attack complexities can be found in Table 4.2.

Table 4.3: Brute-force attack performance on an `AMD Phenom(tm) II X4 955` with four processor cores of 3.2 `Ghz`. Within each iteration the correctness of the candidate polynomial is checked by comparing its SHA-1 hash value.

parameters as in	security in bits	polynomials per second per core	**expected time before successful iteration**
Uludag et al. (2005)	≈ 36	$129,032$	19–20 hours
Uludag and Jain (2006)	≈ 31	$124,533$	$\approx$ 59 minutes
Nandakumar et al. (2007a)	≈ 27	$154,560$	3–4 minutes
"	≈ 31	$124,533$	$\approx$ 59 minutes
"	≈ 39	$97,656.2$	11–12 days
Clancy et al. (2003)	≈ 50	$60,350.0$	$\approx$ 87 years
Li et al. (2010)	≈ 48	$72,150.1$	19–20 years
"	≈ 52	$66,357.0$	338–339 years
Merkle et al. (2010b)	≈ 72	$20,136.9$	$1.80 \cdot 10^9$ years
Merkle et al. (2010b)	≈ 95	$9,523.81$	$2.99 \cdot 10^{16}$ years
Merkle et al. (2010b)	≈ 95	$12,651.8$	$2.01 \cdot 10^{16}$ years

Table 4.3 lists the results of Table 4.1 correspondingly. The theoretical estimation for „optimal“ parameters of Clancy et al. (2003) results in a rather good security

against the brute-force attack. However, a similar high security is not provided by any fuzzy vault implementation protecting single-finger minutiae templates in the table. A reason for this curiosity might be that Clancy et al. (2003) determined the parameters for minutiae extracted from fingerprints of sufficient good quality. Furthermore, they assumed a good alignment of the query minutiae templates to the vault.

Therefore, it is interesting to observe that the implementation of Li et al. (2010) outperforms the implementations of Uludag et al. (2005), Uludag and Jain (2006), and Nandakumar et al. (2007a) with respect to authentication performance seemingly just because it does not require an alignment (see Section 3.3). Furthermore, the implementation's security against the brute-force is rather good. On the other hand, to claim that attacking and instance of the implementation of Li et al. (2010) where GAR $= 94\%$ would last 19–20 years would be somewhat contradicting due to a false acceptance rate of 0.04%. In fact, the false-accept attacks seems to be the best choice when one attempts to attack an intercepted vault (see Section 4.3). Furthermore, the resistance of the implementation by Li et al. (2010) against attacks via record multiplicity is not addressed and is (most likely) vulnerable against a correlation attack (see Section 4.4).

4.1.3 Fuzzy Vault with Minutiae Descriptors

Another single-finger fuzzy vault implementation is the one proposed by Nagar et al. (2008, 2010). This implementation aims for improving vault security by protecting each vault point's ordinate value using a fuzzy commitment of minutiae descriptors (see Section 2.2.8 and Section 3.2.4). Therefore, in a brute-force attack, if k candidates for genuine vault points are chosen, additionally the choices for k corresponding minutiae descriptors must be correct. This increases the security.

The minutiae descriptors are encoded as m-bit vectors v. The 16-bit ordinate values $y \in \mathbb{F}_{2^{16}}$ of vault points are encoded by an appropriate word $c(y)$ of an m-length code with error-correction capability e. Instead of only publishing a vault point as (x, y) the pair $(x, c(y)+v)$ is published. If the pair $(x, c(y)+v)$ is considered, for an attacker to obtain the unprotected ordinate value y he has to guess a minutiae descriptor v' of sufficient similarity to v: Only if $\operatorname{dist}(v, v')$ is sufficiently small, the attacker is able to obtain the codeword $c(y)$ and thus y in clear. According to Nagar et al. (2008, 2010) the difficulty in guessing a single minutiae descriptor is $R \approx 4.27$. Therefore, for vaults of size n, t genuine points, and a polynomial of degree $< k$ an additional security of $\approx \log_2(R) \cdot k \approx 2 \cdot k$ is added against an attack where the adversary in addition guesses k minutiae descriptors within each iteration of the brute-force attack above (Algorithm 4.1.1). For example, if $n = 224$, $k = 24$, and $k = 9$ instead of protecting fingerprints at a security level of 31 bits the resistance is improved to $31 + 2 \cdot 9 = 49$ bits against this attack.

4.1.4 Information Leakage from Imperfect Codes

However, the above security analysis does only hold in case a perfect code (see Section 2.3.3) is used. Therefore, we next give a heuristic on how much advantage an attacker can achieve from an error-correcting code used in the fuzzy commitment scheme that is of low sphere packing density.

Let $\mathbf{C}$ be the underlying binary error-correcting code of error-correction capability $\nu^* = \lfloor (d^* - 1)/2 \rfloor$ and of length m. Let $(x, c(y) + v)$ be a vault point with ordinate value y protected by the minutiae descriptor v encoded as an m-bit vector. Furthermore, let R denote the difficulty in guessing a random minutiae descriptor of sufficient similarity (this depends on the error-correction capability). Then for a random minutiae descriptor w such that $\mathrm{dist}(v, w) > \nu^*$ the probability p' that $c(y) + v - w$ can be corrected to another codeword $c(z) \neq c(y)$ estimates as

$$p' = \frac{\mathcal{S}(\mathbf{C}) - \mathcal{B}(\mathbf{C})}{2^m - \mathcal{B}(\mathbf{C})} \tag{4.5}$$

where $\mathcal{B}(\mathbf{C})$ and $\mathcal{S}(\mathbf{C})$ are the sizes given by the equations (2.15) and (2.16), respectively (see page 30). Thus, an attacker has found the correct ordinate value y given $c(y) + v - w$ can be decoded with probability

$$p = 1 - p' = 1 - \frac{\mathcal{S}(\mathbf{C}) - \mathcal{B}(\mathbf{C})}{2^m - \mathcal{B}(\mathbf{C})}. \tag{4.6}$$

Note that the above probability can be overwhelming depending on the parameters of the underlying error-correcting code.

4.1.5 Decoupling Attack

For each vault point $(x, c(y) + v)$ (genuine an chaff) an attacker may guess a minutiae descriptor w such that $c(y) + v - w$ can be decoded. In this way, the attacker obtains a vault where the ordinate values are unprotected. With probability p^t (4.6) the reduced vault will contain all genuine points (i.e. constituted with their correct ordinate value). In other words, with probability estimated as p^t an attacker is able to decouple the brute-force attack from the problem of guessing minutiae descriptors. Therefore, we refer to this attack as *decoupling attack*.

Efficiency of the Decoupling Attack

We assume that an attacker has established a collection of well-separated minutiae descriptors (encoded as m-bit vectors). We estimate the size of this set to be of order R, which corresponds to the difficulty in guessing a random minutiae descriptor. For each vault point $(x, c(y) + v)$ (among n) and all pre-established minutiae descriptors w the attacker attempts to round $c(y) + v - w$ to its nearest codeword. On successful decoding to a codeword $c(z)$ the attacker keeps track of the value z as a candidate for the correct ordinate value associated with the abscissa value x. In this way, the attacker might obtain more than one candidate for the ordinate value for each point. The efficiency of this reduction step is estimated as

$$\mathbf{rd}(\mathbf{V}) = n \cdot R \tag{4.7}$$

decoding attempts in $\mathbf{C}$. For each vault point, the attacker selects one of the tracked ordinate values to obtain a reduced vault instance with clear ordinate values. As a next step, the attacker tries to break the vault using a generic attack (e.g., brute-force) requiring at most, say, Θ operations. The attacker's chance that a random selection of the tracked ordinate values will enable him to break the vault is estimated as p^t where p is as in Equation (4.6). Thus, the overall cost for conducting the decoupling attack can be estimated as

$$\mathbf{dc}(\mathbf{V}) = \mathbf{rd}(\mathbf{V}) \cdot \alpha + p^{-t} \cdot \Theta \tag{4.8}$$

where α denotes the computation time for a decoding attempt in $\mathbf{C}$. If β denotes the computing time of a single step in the brute-force attack, the brute-force attack against an ordinary vault instance requires at most $\Theta = \mathbf{bf}(n,t,k) \cdot \beta$ operations. Finally, we can estimate the cost for the decoupling attack (with brute-force) as

$$\mathbf{dc}(\mathbf{V}) = \mathbf{rd}(\mathbf{V}) \cdot \alpha + p^{-t} \cdot \mathbf{bf}(n,t,k) \cdot \beta. \tag{4.9}$$

Roughly speaking, for overwhelming p^t, which is possibly caused by a low sphere packing density of the code, the partial securities for guessing minutiae descriptors essentially add up but do not multiply. An important impact from this is that the overall security of the vault implementation remains similar to the base vault implementation without minutiae descriptors.

Example

Nagar et al. (2008) implemented the fuzzy fingerprint vault with minutiae descriptors using the $(511, 19, 239)$-BCH code $\mathbf{C}$ for protecting ordinate values on the basis of the implementation of Nandakumar et al. (2007a) where the vaults are of size $n = 224$ containing $t = 24$ genuine points.

Using Equation (2.15) and Equation (2.16) on page 30 we get that $\mathcal{B}(\mathbf{C}) \approx 1.66 \cdot 10^{119}$ and $\mathcal{S}(\mathbf{C}) \approx 8.72 \cdot 10^{124}$. Therefore, the probability for a genuine vault point with protected ordinate value to reveal its correct one is $p = 1 - p'$ where $p' \approx 2^{-96}$. Thus, the probability of a successful reduction is $p^{24} \approx 1 - 2^{-91}$.

We have empirically determined that a single decoding attempt of a $(511, 19)$-BCH codeword can be performed within $\alpha \approx 0.024$ seconds.[2] For $k = 9$ the time consumed by a single iteration of the brute-force attack has been determined as $\beta \approx 3.28 \cdot 10^{-5}$ seconds. Nagar et al. (2008, 2010) estimate the difficulty in guessing a random minutiae descriptor as $R \approx 4.27$. Since

$$\mathbf{dc}(\mathbf{V}) = \left(\mathbf{rd}(\mathbf{V}) \cdot \frac{\alpha}{\beta} + p^{-t} \cdot \mathbf{bf}(n,t,k)\right) \cdot \beta \approx \underbrace{2.54 \cdot 10^9}_{\approx \mathbf{bf}(224,24,9)} \cdot \beta \tag{4.10}$$

the time consumed by the decoupling attack is estimated as to take approximately the time of an ordinary brute-force attack, which correspond to 31 bits (and not 49 bits). The security of the fuzzy fingerprint vault implementation with protected ordinate values is therefore similar to the fuzzy vault implementation without protected ordinate values.

Nagar et al. (2010) in fact recognized that their analysis resulting in a security of 49 is actually not correct in case the imperfect $(511, 19)$-BCH code is used. To fix this, they additionally analyzed the use of the $(31, 6, 15)$- and $(15, 5, 7)$-BCH

---

[2] The average time for a single decoding attempt of a $(511, 19)$-BCH codeword has been determined using an own non-optimized `C++` implementation based on `NTL` linked with `libgf2x` and `GMP` (see `http://www.shoup.net/ntl/`, `http://gforge.inria.fr/projects/gf2x/`, and `http://gmplib.org/`). The test was performed on a single core of an `AMD Phenom(tm) II X4 955`.

code.[3] We have empirically determined that the time α for decoding a $(31, 6)$- and $(15, 5)$-BCH codeword is 10^{-7} and $2 \cdot 10^{-8}$ seconds, respectively. Analogous to the estimation for the security of the $(511, 19)$-BCH code, the overall security against the decoupling attack is estimated as 35 and 59, respectively. An overview for the results of our analysis is given by Table 4.4.

Table 4.4: Securities of the implementations by Nagar et al. (2008, 2010) for $k = 9$ against the decoupling attack

code parameters	probability of correct recovery of single ordinate (p)	successful reduction probability (p^t)	**security in bits**
$(511, 19)$	$\approx 1 - 2^{-96}$	$\approx 1 - 2^{-91}$	≈ 31
$(31, 6)$	$\approx 89.5\%$	$\approx 6.98\%$	≈ 35
$(15, 5)$	$\approx 44.53\%$	$\approx 3.7 \cdot 10^{-9}$	≈ 59

The bit securities of Table 4.4 are reports with respect to the decoupling attack as it has been described above. For the $(15, 5)$-BCH code, the difficulty in running the generic decoupling attack is 59 bits. We will next describe a *decoupling brute-force attack* being more efficient than the decoupling attack with brute-force. In particular, for the $(15, 5)$-BCH code the difficulty in running the decoupling brute-force attack turns out to be 45.

4.1.6 Decoupling Brute-Force Attack

As usual, let $\mathbf{V}$ denote a vault of cardinality n, containing t genuine points that are interpolated by a polynomial of degree $< k$. For the decoupling attack we assumed that the identification of the genuine polynomial is given by the number of vault points it interpolates. However, most fuzzy fingerprint vault implementations associate additional information with the vault or secret polynomial to allow safe recovery, e.g., by storing a strong hash value. In other words, once the correct polynomial is seen, with very high reliability its correctness can be verified. For the decoupling attack we assumed correct recovery of protected ordinate values for all genuine vault points. However, a slight modification of the brute-force attack performs decoupling within each iteration more dynamically.

Algorithm 4.1.2 (Decoupling Brute-Force Attack).

Input: A vault $\mathbf{V}$ hiding a polynomial of degree $< k$ interpolating t vault points where each vault point's ordinate value is protected by a minutiae descriptor

---

[3] Using codes of smaller length requires dimension reduction of the representation of the minutiae descriptors. Nagar et al. (2008, 2010) proposed *principal component analysis* to extract the most reliable bits from initial encodings of minutiae descriptors, which are 684-bit vectors.

```
encoded as a vector $v$ of length $m$; a criterion for identifying the correct polynomial.
Output: A polynomial $f$ satisfying the criterion.

1: for each vault point, pre-compute candidates of ordinate values as in the decoupling attack;
2: select $k$ different vault abscissas at random;
3: for each selected abscissa, choose one of the precomputed ordinate values at random;
4: find the interpolation polynomial $f^*$ of the $k$ chosen points;
5: if $f^*$ can be verified to be correct, output $f^*$ and terminate; otherwise continue with Step 2.
```

A major difference in the *decoupling brute-force attack* is that it contains an additional step for choosing a candidate ordinate value for each vault point.

Efficiency of the Decoupling Brute-Force Attack

By R we use to denote the difficulty in guessing a random minutiae descriptor. For each vault point, we assume the set of candidate ordinate values to contain at least one element. Furthermore, for genuine vault points we assume that the correct ordinate value is an element of the candidate set. Therefore we estimate the number of candidate ordinate values associated with each vault point to be approximately

$$\mathbf{cov}(\mathbf{V}) = 1 + (R-1) \cdot p', \tag{4.11}$$

where p' is as in Equation (4.5) on page 62. Finally, the difficulty in choosing k genuine vault points and their respective correct ordinate values estimates as

$$\mathbf{dbf}(\mathbf{V}) = \mathbf{cov}(\mathbf{V})^k \ \cdot \ \mathbf{bf}(n,t,k). \tag{4.12}$$

Example

For the implementation of Nagar et al. (2010) ($n = 224$, $t = 24$) where the $(15,5)$-BCH code $\mathbf{C}$ is used to protect ordinate values the probability that a minutiae descriptor of sufficient dissimilarity will succeed in decoding the protected ordinate value is estimated as $p' \approx 55.47\%$ using Formula (4.5) where $\mathcal{S}(\mathbf{C}) = 18,432$, $\mathcal{B}(\mathbf{C}) = 576$, and $m = 15$. By Equation (4.11) the set from where an attacker chooses candidates for a vault point's ordinate values is of size estimated as $\mathbf{cov}(\mathbf{V}) \approx 2.81$. Thus, for $k = 9$

$$\mathbf{dbf}(\mathbf{V}) \approx 2.79 \cdot 10^{13} \approx 2^{45}. \tag{4.13}$$

Therefore, the decoupling brute-force attack supersedes the generic decoupling attack (with brute-force) for the selected parameters. Since the security of a cryptographic system is lower bounded by its resistance against the most efficient attack, the bit security of the corresponding implementation can be at most 45 bit (and not 59 as suggested by Table 4.4).

In Table 4.5 the bit securities of the implementations by Nagar et al. (2008, 2010) against the decoupling brute-force attack are listed. Not surprisingly, the more information is leaked by the imperfectness of the used error-correcting code (i.e. the larger p') the more the security against the decoupling brute-force attack drops.

Table 4.5: Securities of the implementations by Nagar et al. (2008, 2010) for $k = 9$ against the decoupling brute-force attack

code parameters	**security in bits**
$(511, 19)$	≈ 31
$(31, 6)$	≈ 35
$(15, 5)$	≈ 45

We finish this section by noting that our analysis supports the analysis given by Nagar et al. (2010) for the $(31, 6)$ and $(15, 5)$-BCH code. For example, if $k = 7$ then Nagar et al. (2010) report a security gain of approximately 10 bits for the $(15, 5)$-BCH code. In fact, the formula in Equation (4.12) evaluates to

$$\mathbf{dbf}(\mathbf{V}) \approx 2.05 \cdot 10^{10} \approx 2^{34} \tag{4.14}$$

while

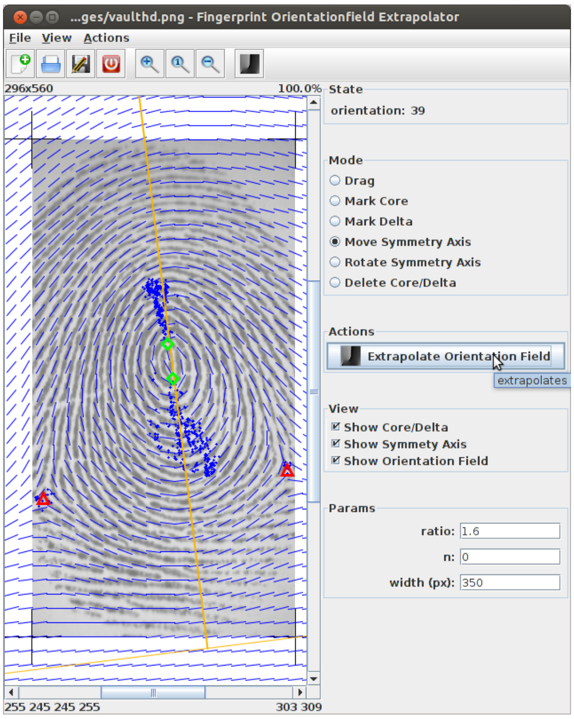
$$\mathbf{bf}(224, 24, 7) \approx 1.48 \cdot 10^{7} \approx 2^{24} \tag{4.15}$$

in this case.

4.2 Auxiliary Alignment Data May Help to Improve Attacks

The attacks above are brute-force attacks, which definitely can be improved. For example, it is possible to make use of the statistics of fingerprint features or to run a false-accept attack (see Section 4.3). Assuming hardness for the instances of the polynomial reconstruction problem given by the vault (see Bleichenbacher and Nguyen (2000)), the overall difficulty in breaking a fingerprint vault with similar complexity of a brute-force attack can only be guaranteed if, first, vault minutiae are independent and identically distributed and, second, no additional information about the finger is stored along with the vault. For many implementations of the fingerprint fuzzy vault there is, however, additional alignment helper data that is stored along with the vault. This data leaks information about the finger and may be used by an attacker to accelerate attacks. In the following we discuss approaches for the attacker to gain advantage of auxiliary alignment data that we briefly reviewed in Section 3.3.

Yang and Verbaudwhede (2005) proposed to store reliable reference minutiae publicly along with the vault. Even if the reference minutiae are excluded from encoding genuine vault points, they allow an adversary to cross-match vault records across different application's databases with unprotected minutiae. As already indicated at the beginning of this chapter (see page 57), cross-matching belongs to the most serious risks the fuzzy fingerprint vault is concerned with. Even worse, once genuine


...ges/vaulthd.png - Fingerprint Orientationfield Extrapolator
File
View
Actions
296x560
100.0%
Extrapolate Orientation Field
extrapolates
255 245 245 255
303 309

State
orientation: 39

Mode
○ Drag
○ Mark Core
○ Mark Delta
● Move Symmetry Axis
○ Rotate Symmetry Axis
○ Delete Core/Delta

View
☑ Show Core/Delta
☑ Show Symmety Axis
☑ Show Orientation Field

Params
ratio: 1.6
n: 0
width (px): 350

Figure 4.1: Tool used to extrapolate a fingerprint's orientation field from singular points using the model of Sherlock and Monro (1993) (i.e. $n = 0$). The singular points, cores (green diamonds) and deltas (red triangles), have been marked by an expert depending on the points of high ridge curvature. The symmetry axis (yellow) was chosen such that its ordinate axis passes through both the core points while its abscissa barely lies outside the image's region. The stretching factor 1.6 has been chosen such that it good applies to most fingerprints (private communication with Thomas Hotz (Hotz (2007))). Note, the parameter *width* does not affect the orientation field in the model of Sherlock and Monro (1993).

vault correspondences have been filtered out, they are highly vulnerable against the *correlation attack* (Kholmatov and Yanikoglu (2008)), which is an attack via record multiplicity (Scheirer and Boult (2007)): An attacker uses the correlation attack to break vault records given multiple instances of the vault captured from the same user are given to him. In Section 4.4 we will describe a cross-matching algorithm as well as a correlation attack and conduct them against the implementation presented in Section 3.4.

Uludag and Jain (2006) proposed to store points of high ridge curvature (see Figure 3.4 on page 46) publicly along with the vault. This approach has been refined later by Nandakumar et al. (2007a). Again, we feel that using points of high ridge curvature might assist in cross-matching but there is another interesting point to consider.

Clusters of high curvature points tend to appear near the singular points. From those clusters, in many cases, it is possible to derive the type of the singular point as well as its rough location. Using a model for a fingerprint orientation field (Hotz (2007); Huckemann et al. (2008); Sherlock and Monro (1993)) it is possible to estimate the orientation field to some extent. If an attacker can assume to have a good estimation of the fingerprint protected by the vault then every vault minutiae of direction inconsistent with the orientation field can be excluded from being a genuine vault point (assuming vault features also use minutiae directions). This reduces the effort for performing a subsequent brute-force attack. Furthermore, if minutiae descriptors are used to protect the vault's ordinate values by a fuzzy commitment scheme (Nagar et al. (2008, 2010)) then parts of its corresponding orientation descriptor are leaked. This reduces the difficulty in guessing a single minutiae descriptor causing a reduction in the overall difficulty in successfully running the decoupling brute-force attack (see Section 4.1.6).

For example. in Figure 4.1 the process of deriving an orientation field from points of high ridge curvature is visualized. The points of high ridge curvature have been calculated according to Nandakumar et al. (2007a). The estimation fits very well and leaks virtually the whole orientation field (depending on the level of quantization).

For the three structures proposed by Jeffers and Arakala (2007) we make the following remarks.

First, note that each structure the authors propose to enable vault alignment leaks a certain number of template minutiae. For each *five nearest neighbor structure* there are six minutiae leaked (reference minutia plus it five nearest neighbors). Similarly, minutiae are leaked if *Voronoi neighbor structures* are used (reference minutia plus its Voronoi neighbors). For the *triangle based structure* each vertex corresponds to a location of a template minutiae. Furthermore, each vertex is constituted with a ridge orientation such that the position as well as the undirected orientation of each vertex minutia can be computed from a given triangle based structure.

In this way, an attacker is able to build a set of directed/undirected minutiae leaked from the public minutiae structures. Assuming the set of leaked minutiae significantly overlaps with the set of genuine minutiae, the attacker evidently can gain great advantage to accelerate brute-force attacks. Moreover, again, we feel that cross-matching on the basis of the proposed minutiae structures might help to quickly determine genuine vault correspondences across different application's databases.

The *topo-structure* proposed by Li et al. (2008) leaks an upper bound k_0 on the number of minutiae laying within the *sampling area*. Even if the additional data may not leak minutiae locations. In a brute-force attack (Step 1 of Algorithm 4.1.1), the attacker can exclude any choice for k genuine vault minutiae where more than k_0 lay within the sampling area. This will reduce the effort for an attacker to conduct a brute-force attack against the corresponding minutiae fuzzy vault, which can already be highly vulnerable.

Again, the topo-structure might weaken a fingerprint fuzzy vault against cross-matching. Amongst other information, the alignment helper data encodes a set where each element has been derived from a minutia's location within the sampling area of the corresponding fingerprint. From that we assume that matching two topo-structures can be performed with similar accuracy as matching two fingerprints where only the minutia's locations within the sampling circle are taken into account. Thus, again, cross-matching seems to be possible with sufficient accuracy to reliably find most genuine vault correspondences across different application's databases.

All the proposals for auxiliary alignment data above to deal with the challenging problem of aligning query fingerprints to the vault are causing problems. Either, they leak information about the fingerprint, which furthermore decreases vault security, or they might help an adversary in cross-matching. In our opinion, publicly storing individual fingerprint features to enable vault alignment (such as multiple reference minutiae, high curvature points, minutiae structures, or topo-structure) is problematic from a security perspective. The amount of information leaked from alignment helper data about the template as well as their effects to support cross-matching still have to be analyzed.

Merkle et al. (2010b) base their vault work on pre-alignment. Furthermore, on authentication, they propose to refine the alignment by maximizing the number of matching query minutiae with vault minutiae. However, the way the authors rely on refinement in the pre-alignment, requires random generation of chaff points.

This makes allows the vault records to be cross-matched via correlation (see Section 4.4). Furthermore, merely choosing vault features that are invariant to rotation and translation (Li et al. (2010)), will not solve this problem.

A secure alignment scheme were enabled if fingerprint minutiae could be pre-aligned to an *intrinsic coordinate systems* (Bazen and Gerez (2001)). Then no public alignment data is needed. Unfortunately, current methods that extract intrinsic coordinate systems lack of robustness to allow fingerprint pre-alignment of sufficient accuracy. Although challenging, it seems to be worth to seek for more robust methods to extract intrinsic coordinate systems.

The discussion of this section emphasize that the alignment in a minutiae based fuzzy vault is a not yet solved. When one chooses a particular solution to deal with vault alignment, two main question arise. First, how much does vault security suffer, and second, how does the alignment procedure affect the vault's authentication performance? To get rid of the open problem of vault alignment (as already argued in Section 3.4.4), we investigate vault security and authentication performance assuming no alignment data is stored and the query fingerprints are well pre-aligned to the vault. Thus, throughout this thesis wherever vault alignment is required we achieve it by aligning the corresponding minutiae templates in clear as described in Section 3.4.4.

4.3 False-Accept Attack

The attacks of Section 4.1 are brute-force attacks, the worst case for the attacker, and thus the best case for the user. The analyses of the brute-force attacks against single-finger minutiae fuzzy vaults are indicators of the security limitation for the fuzzy vault scheme applied to the biometric trait *single-finger*. Brute-force attacks can definitely be improved. For example, it is possible to use some statistics of fingerprint minutiae: Vault minutiae that lay within the images center may have a higher chance to be genuine minutiae than vault minutiae that lay near the boundary. Another example: If minutiae angles are used in the vault, it is unlikely that spatially close minutiae have inconsistent direction. Thus, with some probability, combinations of vault minutiae can be excluded from being simultaneously genuine. Evidence advocating the existence of attacks that are much more efficient than brute-force attacks is for example given by a non-negligible false acceptance rate even though the complexity for a brute-force attack would require much more effort. For example, in Li et al. (2010) if $n = 440$, $t = 40$, and $k = 12$ a successful brute-force attack would require approximately 2^{48} iterations whilst the false-accept requires approximately 2^{11} iterations (for a false acceptance rate of 0.04%). Similar to other attacks, an upper bound for the security of a fuzzy fingerprint vault implementation is given by the difficulty in running a successful *false-accept attack*.

An adversary might try to break an intercepted vault $\mathbf{V}$ by taking advantage of the false acceptance rate ϵ of the corresponding vault implementation. In such a scenario, we assume the adversary to have access to a large database $\mathbf{DB}$ containing fingerprint templates from real fingers. The attacker is able to iteratively use the templates from $\mathbf{DB}$ and simulate authentication attempts until the vault $\mathbf{V}$ can be opened.

4.3.1 Algorithmic Description

We will next describe our version of the false-accept attack in algorithmic terms.

Algorithm 4.3.1 (False-Accept Attack).

Input: An intercepted vault instance $\mathbf{V}$ hiding a polynomial of degree k interpolating t genuine points;
a pre-established database $\mathbf{DB} = \{T_1, \ldots, T_N\}$ containing fingerprint templates from different fingers;
a criterion for verfifying the correct polynomial;
Output: Failure or a polynomial interpolating t vault points;

```
1: for $i = 1, \ldots, N$
2:     use the template $T_i$ to extract the unlocking set $\mathbf{U}_i \subset \mathbf{V}$;
3:     try to decode a polynomial $f^*$ from the unlocking set $\mathbf{U}_i$;
4:     if $f^*$ can be verified to be correct output $f^*$ and terminate;
5: endfor
6: output Failure and terminate the algorithm.
```

Note, if the attack is run against our implementation of Section 3.4 then Step 3 consists of the iteration through all polynomials interpolating k points from $\mathbf{U}_i$ and checking whether the interpolation polynomial's SHA-1 hash value is equals to the hash value of the genuine polynomial.

The probability that Algorithm 4.3.1 outputs the correct polynomial is estimated as

$$1 - (1 - \epsilon)^N. \tag{4.16}$$

A vault's vulnerability against the false-accept attack can be easily calculated as the false acceptance rate ϵ is known. However, a reliable estimation for a vault implementation's false acceptance rate is difficult to achieve but nonetheless important for reliably analyzing a vault's resistance against the false-accept attack. Anyway, it is possible to compute confidence intervals for the false acceptance rate. For a closer view on this topic see Section 2.2.6 on page 21.

4.3.2 Performance of the False-Accept Attack

If the average impostor decoding time τ and the false acceptance rate ϵ of a vault implementation are known, the efficiency of the false-accept attack can be easily calculated.

Assume that the database input to the false-accept attack is sufficiently large. An attacker who is going to run the attack against an intercepted vault $\mathbf{V}$ will need to spend

$$\frac{\log(0.5)}{\log(1-\epsilon)} \cdot \tau \tag{4.17}$$

of computation time in average to successfully open the vault $\mathbf{V}$.

Examples

For example, if $k = 9$, our implementation (see Section 3.4) requires an attacker to wait approximately 247 ms (on a single processor core of an `AMD Phenom(tm) II X4 955`) for a template $T_i \in \mathbf{DB}$ until he can proceed with the next template T_{i+1}. Using Table 3.1 on page 53 we see that 16 among 4856 (i.e. a false acceptance rate $\approx 0.33\%$) non-genuine templates successfuly opened the vault. Using the Clopper-Pearson confidence interval (see Equation (2.13)) at a confidence level of 95% we expect the false acceptance rate to lay in the range $[0.19\%, 0.53\%]$. Consequently, with a confidence of 95% an attacker can expect to successfully break the vault within 32–91 seconds (see Equation (4.17)). If all four processor cores were used in parallel, the attacker can expect to be successful after just 8–23 seconds. In comparison to the time that is expected to be consumed by the brute-force attack (i.e. 59 minutes according to Table 4.2) the false-accept attack significantly outperforms the brute-force attack.

Table 4.6: Range of false-accept attack complexities for a confidence level of 95% on all four cores of an `AMD Phenom(tm) II X4 955`

length of secret polynomial	expected time for a successful brute-force attack	**expected time range for a successful false-accept attack**
$k = 7$	13 – 14 seconds	348–466 milliseconds
$k = 8$	3 – 4 minutes	1.46–2.28 seconds
$k = 9$	59 minutes	8–23 seconds
$k = 10$	15 – 16 hours	13–51 seconds
$k = 11$	11 – 12 days	35 seconds – 18 minutes
$k = 12$	$\approx$ 7 months	$>$ 1 minute

In Table 4.6 the estimated complexities for a successful false-accept attack against our implementation for different k are listed. It was not possible to estimate an upper bound for the complexity when $k = 12$. This is due to the fact that the false acceptance rate can be arbitrarily close to zero. However, under security aspects with a confidence of 95% we can only expect that a false-accept attack will be successful within 1 minute given the adversary is in possession of a sufficiently large database containing fingerprint templates.

4.3.3 False-Accept Attack against Fuzzy Vault with Minutiae Descriptors

In this section, we show how to attack the implementation of Nagar et al. (2008, 2010) using a false-accept attack (note, the vault ordinate values are additionally protected using the fuzzy commitment scheme where the remaining is as in Nandakumar et al. (2007a)). For example, if the $(511, 19, 239)$-BCH code $\mathbf{C}$ is used, then for $k = 7$ the false acceptance rate is reported as to be 0.01% while it has been reported to be 0.7% for the implementation without protected ordinate values. This suggests a tremendous improvement of the implementation's resistance against the false-accept attack as well. However, as in the brute-force attack, an attacker has the ability to decouple the false-accept attack from the problem of guessing minutiae descriptors to break a vault $\mathbf{V}$.

Combining the Decoupling Attack with the False-Accept Attack

We start with repeating some notions from Section 4.1.5.
Let $\mathbf{V}$ be a vault of size n and t genuine points whose ordinate values $c(y) + v$ are protected using minutiae descriptors v via the fuzzy commitment scheme. Let the fuzzy commitment scheme be based on the error-correcting code $\mathbf{C}$. Assume p to be the probability that $c(y) = c(z)$ under the condition that w is a random minutiae descriptor such that $c(y)+v-w$ can be decoded to $c(z)$ (see Equation (4.6) on page 63).

Next, if R is the difficulty for guessing a random minutia descriptor, an attacker can expect to obtain a candidate set containing ordinate values of size $\mathbf{rd}(\mathbf{V}) = n \cdot R$. If α is the time for a decoding attempt in $\mathbf{C}$, this pre-computation step is estimated as to last $\mathbf{rd}(\mathbf{V}) \cdot \alpha$. The probability, that a selection of n ordinate values from their corresponding candidate sets will contain all t genuine points correct ordinate value is estimated as p^t.

Assume that the false-accept attack is run to break the reduced vault (without protected ordinate values). Let the time for a successful false-accept attack be Θ. Analogous to Equation (4.8) the decoupling attack is estimated as to take

$$\mathbf{dc}(\mathbf{V}) = \mathbf{rd}(\mathbf{V}) \cdot \alpha + p^{-t} \cdot \Theta \tag{4.18}$$

of computing time.

Example

For the implementation of Nagar et al. (2008), if the $(511, 19)$-BCH code $\mathbf{C}$ is used, we have experimentally determined that a single decoding attempt in $\mathbf{C}$ can be done within $\alpha = 0.024$ seconds (see the example of Section 4.1.5). Thus, with an effort of approximately $n \cdot R \cdot \alpha \approx 23$ seconds (i.e. ≈ 5 seconds if four processor cores are used) and overwhelming probability of $p^t \approx 1 - 2^{-91}$ the vault implementation with protected ordinate value is as vulnerable to the false-accept attack as if the ordinate values were not protected.

Intermediate Discussion

Using the implementation of Nagar et al. (2008, 2010) the improvement of the false acceptance rate from 0.7% to 0.01% at a genuine acceptance rate of 95% was used as a teaser to advocate for the benefits of the implementation. The validity of the reduction in the false acceptance rate is certainly true if using their authentication procedure. However, the report of the false acceptance rate is not valid under security aspects. The reason is that the false-accept attack also applies in combination with the decoupling attack.

It is furthermore claimed in Nagar et al. (2010) that although the use of imperfect error-correcting codes will not cause a gain in security from an information theoretic point of view there would be a certain improvement for there is a significant cost to the adversary to find decoding minutiae descriptors. According to our tests, finding matching minutiae descriptors can be performed within approximately 5 seconds in a single pre-computation step using four processor cores of an `AMD Phenom(tm) II X4 955`. This does not correspond to a significant cost an adversary has to pay.

The work of Nagar et al. (2010) contains good analysis of how minutiae descriptors can be used to improve fingerprint vault security. Valid countermeasures are given to deal with the imperfectness of error-correcting codes. In fact, the problems we have emphasized in terms of the decoupling attack and the decoupling brute-force attack were also recognized by Nagar et al. (2010)—even though missing in the abstract or in the final discussion. This might lead to confusion of the reader and misunderstanding of the article's benefit.

4.4 Cross-Matching and the Correlation Attack

The low amount of time needed to run a successful false-accept attack against single-finger fuzzy vaults is highly alarming. Even brute-force attack can be feasible for reasonable genuine acceptance rates. This suggests that single-finger is not sufficient to provide a cryptographically secure biometric authentication scheme. Instead analyses of multi-finger fuzzy vaults should have the focus for future research. Unfortunately, there remain problems of the fuzzy fingerprint vault that can not be solved merely using multiple finger.

One of the most serious risks the fuzzy fingerprint vault is concerned with is its high vulnerability against a special kind of attack via record multiplicity (Scheirer and Boult (2007)).

Two scenarios can be distinguished concerning attacks via record multiplicity. The first scenario is where the attacker has intercepted two vault records $\mathbf{V}$ and $\mathbf{W}$ for which he has to decide whether both protect templates from the same finger, i.e. *cross-matching*. The second scenario is concerned with where the attacker already knows that $\mathbf{V}$ and $\mathbf{W}$ do protect the same finger and he is attempting to get into the possession of the template and/or key (i.e. secret polynomial f). The attacker's possibility to recover the templates and/or key is not necessarily at stake given the attacker is able to cross-match vault records: Cross-matching, for instances, might be enabled just by matching alignment helper data (for example as in (Li et al. (2008); Nandakumar et al. (2007a); Uludag and Jain (2006))). The converse, on the other hand, i.e. efficient cross-matching, is assumed by the possibility to efficiently reconstruct vault templates via record multiplicity: Given the reconstruction of the templates and/or key is successful the vaults are considered to match.

Cross-matching is always possible by, for instance, the brute-force attack: The vault $\mathbf{V}$ is attacked until its template reveals; this template is then used to open another vault $\mathbf{W}$; if successful, $\mathbf{V}$ and $\mathbf{W}$ are considered to match. While such an approach is always possible there exists a more efficient method to separate genuine points from chaff points if two vaults protecting the same finger are given: By correlating the vaults, barring alignment, genuine minutiae have a bias to be in agreement in both vaults while chaff minutiae are likely to be separate since they have been generated randomly. An example illustrating this approach is given by Figure 4.2.

While correlation has the inadvertent effect that vault records can be cross-matched, it is even possible for the attacker to efficiently break two vault's templates and keys given the vaults protect templates from the same finger. As a consequence, separating genuine points from chaff points via correlation has the potential to be much more efficient than merely attacking one of the vault separately.

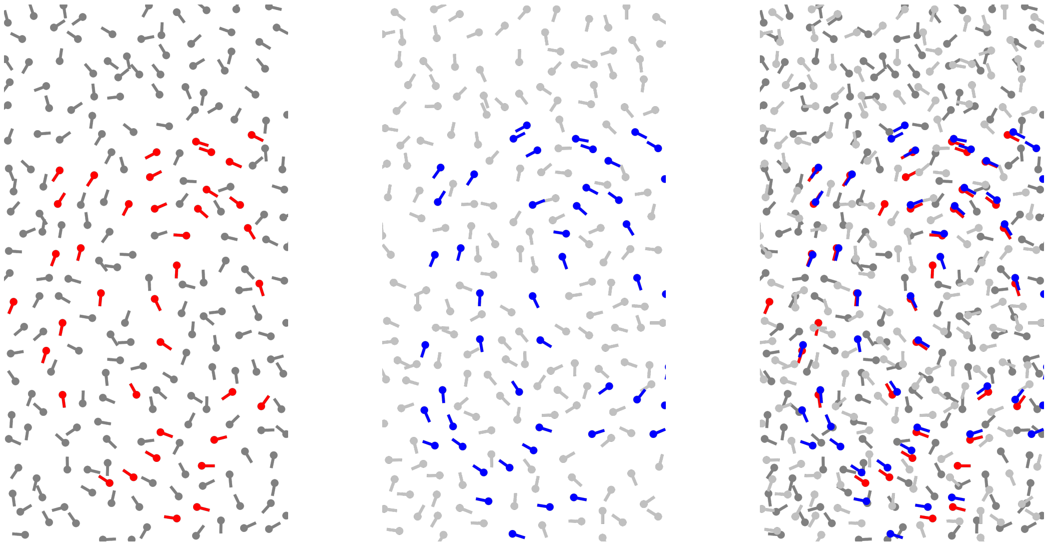


Figure 4.2: Two aligned vaults with chaff minutiae (gray and light-gray) and genuine minutiae (red and blue). The genuine minutiae have a bias to be in agreement

4.4.1 The Correlation Attack

We describe the attack more generally, not only restricted to minutiae features.

Let $\mathbf{V}$ and $\mathbf{W}$ be two vaults. Each vault point $v \in \mathbf{V}$ and $w \in \mathbf{W}$ encodes a fingerprint feature (on the abscissa) $\mathfrak{m}$ and $\mathfrak{m}'$, respectively. Assume there is a function $d(\mathfrak{m}, \mathfrak{m}')$ that measures the distance (or similarity) between $\mathfrak{m}$ and $\mathfrak{m}'$.

For instance, if $\mathfrak{m}$ and $\mathfrak{m}'$ are minutiae, then $d(\mathfrak{m}, \mathfrak{m}')$ might correspond to the minutiae distance map as in Equation (2.5) on page 19.

Since each vault pair $(v, w) \in \mathbf{V} \times \mathbf{W}$ corresponds to a feature pair $(\mathfrak{m}, \mathfrak{m}')$ we define the corresponding distance map on $\mathbf{V} \times \mathbf{W}$ by setting

$$d : \mathbf{V} \times \mathbf{W} \to \mathbb{R}_{\geq 0}, \ (v, w) \mapsto d(\mathfrak{m}, \mathfrak{m}'). \tag{4.19}$$

For a vault point $v \in \mathbf{V}$ consider

$$d_{\mathbf{W}}(v) := \min_{w \in \mathbf{W}} d(v, w) \tag{4.20}$$

the minimal distance of v to one of its nearest points in $\mathbf{W}$.

On the basis of the map $d_{\mathbf{W}}(\cdot)$ the *correlation attack* can be described as follows.

Algorithm 4.4.1 (Correlation Attack)**.**

Input: Two vaults $\mathbf{V}$ and $\mathbf{W}$; a real number $x \geq 0$;

Output: The secret polynomial f hidden by $\mathbf{V}$ or FAILURE;

```
1: $\mathbf{U} \leftarrow \{ v \in \mathbf{V} \mid d_{\mathbf{W}}(v) \leq x \}$;
2: try to decode the unlocking set $\mathbf{U}$;
3: on success, output the decoded polynomial $f$;
4: otherwise, output FAILURE.
```

Note, that there is no particular reason to only consider the unlocking set $\mathbf{U}$ to consist of elements in $\mathbf{V}$. It is also possible to let $\mathbf{U}$ consist of the elements in $w \in \mathbf{W}$ with $d_{\mathbf{V}}(w) \leq x$. For matter of notational convenience we restrict our considerations on $\mathbf{U}$ as a subset of $\mathbf{V}$.

Let $\mathbf{H}$ be the subset of the genuine vault points $\mathbf{G} \subset \mathbf{V}$ that have a ground-truth match in $\mathbf{W}$: If a genuine vault point $v \in \mathbf{V}$ encodes a scan of a minutia $\mathfrak{m}$ and there is a genuine vault point $w \in \mathbf{W}$ encoding a scan $\mathfrak{m}'$ of the same ground-truth minutia then $\mathbf{H}$ is defined to contain v.

If the values $d_{\mathbf{W}}(\mathbf{H})$ have a bias to be significantly smaller than $d_{\mathbf{W}}(\mathbf{V})$, an attacker has good reason to expect the correlation attack to successfully output the secret polynomial f — for a reasonable $x \geq 0$.

4.4.2 Criterion for Cross-Matching

In this section, we derive a criterion for an attacker to cross-match two vaults. Just in case the vaults can be positively cross-matched using this criterion the attacker runs the time-consuming step of decoding the unlocking set, i.e. Step 2 of Algorithm 4.4.1. Otherwise, if both the vaults do not cross-match, the attacker continues with another pair of vaults.

As usual, let $\mathbf{V}$ be a fingerprint vault of size n hiding a polynomial of degree $< k$ which interpolates t vault points. As a function in x, let $\gamma(x)$ be the distribution function for a random $v \in \mathbf{G}$ also to be an element in $\mathbf{U}$, i.e.

$$\gamma(x) = \mathbb{P}(\ d_{\mathbf{W}}(v) \leq x \mid v \in \mathbf{G} \ ). \tag{4.21}$$

Analogously, let $\beta(x)$ be the distribution function for a random $v \in \mathbf{V} \setminus \mathbf{G}$ also to be contained in $\mathbf{U}$, i.e.

$$\beta(x) = \mathbb{P}(\ d_{\mathbf{W}}(v) \leq x \mid v \in \mathbf{V} \setminus \mathbf{G} \ ). \tag{4.22}$$

Now, if $v \in \mathbf{G}$ then v either has a ground-truth match in $\mathbf{W}$ (in this case we write $v \in \mathbf{H}$) or v is a genuine point that does not have a ground-truth match in $\mathbf{W}$. By

$$\alpha(x) = \mathbb{P}(\ d_{\mathbf{W}}(v) \leq x \mid v \in \mathbf{H} \ ) \tag{4.23}$$

denote the distribution function of genuine vault points v with a ground-truth match in $\mathbf{W}$ being an element of the unlocking set $\mathbf{U}$. Similarly, write

$$\tilde{\alpha}(x) = \mathbb{P}(\ d_{\mathbf{W}}(v) \leq x \mid v \in \mathbf{G} \setminus \mathbf{H} \ ) \tag{4.24}$$

for the distribution function of a genuine vault point v with no ground-truth match in $\mathbf{W}$. Now, if $t_{\mathbf{H}} = |\mathbf{H}|$ then

$$\gamma(x) = \frac{t_{\mathbf{H}} \cdot \alpha(x) + (t - t_{\mathbf{H}}) \cdot \tilde{\alpha}(x)}{t}. \tag{4.25}$$

Note that the elements of $\mathbf{G} \setminus \mathbf{H}$ are closest to the elements in $\mathbf{W}$ that are not confirmed by a ground-truth match in $\mathbf{W}$. Therefore, we assume that the elements in $\mathbf{G} \setminus \mathbf{H}$ appear random to the attacker. As a consequence, an attacker can not distinguish the elements in $\mathbf{G} \setminus \mathbf{H}$ from the elements in $\mathbf{V} \setminus \mathbf{H}$. In other words, we assume that

$$\tilde{\alpha}(x) = \beta(x). \tag{4.26}$$

So the probability $\gamma(x)$ for a genuine point to be an element of the unlocking set fulfills

$$\gamma(x) = \frac{t_{\mathbf{H}} \cdot \alpha(x) + (t - t_{\mathbf{H}}) \cdot \beta(x)}{t}. \tag{4.27}$$

For random vaults $\mathbf{V}$ and $\mathbf{W}$ (but fixed x) in Step 1 of the correlation attack we expect the size of the unlocking set $u = |\mathbf{U}|$ to have the following distribution given $t_{\mathbf{H}}$

$$\begin{aligned} u &= (n - t) \cdot \beta(x) + t \cdot \gamma(x) \\ &= (n - t) \cdot \beta(x) + t_{\mathbf{H}} \cdot \alpha(x) + (t - t_{\mathbf{H}}) \cdot \beta(x) \\ &= n \cdot \beta(x) + (\alpha(x) - \beta(x)) \cdot t_{\mathbf{H}}. \end{aligned} \tag{4.28}$$

In correlating two vaults, we assume that the elements in $d_{\mathbf{W}}(\mathbf{H})$ have a bias to be significantly smaller than the elements of $d_{\mathbf{W}}(\mathbf{V} \setminus \mathbf{G})$. More precisely,

$$\alpha(x) \gg \beta(x). \tag{4.29}$$

In particular, an implication of this assumption is $\alpha(x) \neq \beta(x)$. Hence

$$t_{\mathbf{H}} = \frac{u - n \cdot \beta(x)}{\alpha(x) - \beta(x)}. \tag{4.30}$$

We build the conditional expectation given the size of the unlocking set u, i.e.

$$\begin{aligned} \mathbb{E}(\ t_{\mathbf{H}} \mid u\ ) &= \frac{\mathbb{E}(\ u \mid u\ ) - n \cdot \beta(x)}{\alpha(x) - \beta(x)} \\ &= \frac{u - n \cdot \beta(x)}{\alpha(x) - \beta(x)}. \end{aligned} \tag{4.31}$$

Consequently, if in Step 1 the size of the unlocking set $u = |\mathbf{U}|$ is already known, we expect the value for $t_{\mathbf{H}}$ to be

$$t_{\mathbf{H}}(u) = \mathbb{E}(\ t_{\mathbf{H}} \mid u\ ) = \frac{u - n \cdot \beta(x)}{\alpha(x) - \beta(x)}. \tag{4.32}$$

Finally, in an unlocking set of size u we expect to find

$$\begin{aligned} t(u) &= t_{\mathbf{H}}(u) \cdot \alpha(x) + (t - t_{\mathbf{H}}(u)) \cdot \beta(x) \\ &= (\alpha(x) - \beta(x)) \cdot t_{\mathbf{H}}(u) + t \cdot \beta(x) \\ &= u + (t - n) \cdot \beta(x) \end{aligned} \tag{4.33}$$

genuine points.

In case

$$t(u) \geq k \tag{4.34}$$

the attacker has good reason to believe that decoding the unlocking set **U** (i.e. Step 2 in Algorithm 4.4.1) will be successful. On the other hand, for an adversary to feasibly decode **U** a limiting factor is the size u of the unlocking set. Anyway, if an attacker chooses a distance threshold x such that $\alpha(x) - \beta(x)$ is maximal he can expect that the unlocking set **U** dominantly contains genuine vault points (see Section 4.4.5 below).

The distribution $\alpha(x)$ is affected by the distribution of ground-truth matching fingerprint features that are used for the vault work. In the following we describe what the relation is.

4.4.3 Relation of $\alpha(x)$ to the Distribution of Ground-Truth Matching Features of Distance $\leq x$

Let Ω consist of all pairs $(\mathfrak{m},\mathfrak{m}')$ such that $\mathfrak{m}$ and $\mathfrak{m}'$ are ground-truth matching measurements of fingerprint features. By

$$\delta(x) = \mathbb{P}(\ d(\mathfrak{m},\mathfrak{m}') \leq x \mid (\mathfrak{m},\mathfrak{m}') \in \Omega\ ) \tag{4.35}$$

we denote the distribution function of the random variable

$$\Omega \to \mathbb{R}_{\geq 0},\ \ (\mathfrak{m},\mathfrak{m}') \mapsto d(\mathfrak{m},\mathfrak{m}'), \tag{4.36}$$

which measures the distribution of ground-truth matching fingerprint features that are bounded by the distance x.

If $v \in \mathbf{V}$ and $w \in \mathbf{W}$ encode the features $\mathfrak{m}$ and $\mathfrak{m}'$, respectively, then $d(v,w) = d(\mathfrak{m},\mathfrak{m}')$. Since $d_{\mathbf{W}}(v) \leq d(v,w)$ it follows by a coupling argument

$$\alpha(x) \geq \delta(x). \tag{4.37}$$

By (4.37) the size of **H** given u is bounded by

$$t_{\mathbf{H}}(u) \leq \frac{u - n \cdot \beta(x)}{\delta(x) - \beta(x)}. \tag{4.38}$$

For an attacker it would be more useful to have a sharp upper bound on $\alpha(x)$ rather than a lower bound. As a consequence, the value $t_{\mathbf{H}}(u)$ could be lower bounded. For simplicity, however, we will assume that for reasonable x the distributions $\alpha(x)$ and $\delta(x)$ are close, i.e.

$$\alpha(x) \approx \delta(x), \tag{4.39}$$

in which case

$$t_{\mathbf{H}}(u) \approx \frac{u - n \cdot \beta(x)}{\delta(x) - \beta(x)}. \tag{4.40}$$

4.4.4 Cross-Matching prior the Correlation Attack

Assume an intruder has intercepted two databases $\mathbf{DB}_1$ and $\mathbf{DB}_2$ from different applications that share common users. Furthermore, assume that the databases $\mathbf{DB}_1$ and $\mathbf{DB}_2$ contain fuzzy vault records. The intruder might run the following attack to gain user access to one (or both) of the applications.

For all $(\mathbf{V},\mathbf{W}) \in \mathbf{DB}_1 \times \mathbf{DB}_2$ run Algorithm 4.4.1 with input **V**, **W**, and a fixed $x \geq 0$. If the polynomial f protected by **V** has been discovered successfully, then the intruder is capable in discovering the template protected by $\mathbf{V}$ and has thus broken the corresponding user account.

Running the correlation attack for all pairs $(\mathbf{V},\mathbf{W})$ can become very time-consuming. As a countermeasure the attacker could make use of the criterion (see (4.34)) that we heuristically derived during Section 4.4.2. Depending on the size of the unlocking set built up in Step 1 of the correlation attack, the attacker has a criterion to cross-match. In algorithmic description the *cross-matcher* can be written as follows.

Algorithm 4.4.2 (Cross-Matching Vault Records)**.**

Input: Two vaults $\mathbf{V}$ and $\mathbf{W}$; a real number $x \geq 0$; the distribution $\beta(x)$ as in Section 4.4.2 of minimal distances of non-confirmed vault points.
Output: Either MATCH or NON-MATCH.

```
1: /* Initialization */
   $t \leftarrow$ "number of genuine vault points in $\mathbf{V}$";
   $k \leftarrow$ "length of secret polynomial hidden by $\mathbf{V}$";

2: /* Correlation */
   $\mathbf{U} \leftarrow \{\, v \in \mathbf{V} \mid d_{\mathbf{W}}(v) \leq x \,\}$;

3: /* Apply criterion (4.34) */
   $u \leftarrow |\mathbf{U}|$;
   $t(u) \leftarrow u + (t-n) \cdot \beta(x)$;
   if $t(u) \geq k$
      return MATCH;
   else
      return NON-MATCH;
   endif
```

For a pair $(\mathbf{V},\mathbf{W})$ an attacker may decide to run the correlation attack (see Algorithm 4.4.1) only if Algorithm 4.4.2 outputs MATCH. The purpose for cross-matching prior possibly running the correlation attack is to eliminate virtually all pairs $(\mathbf{V},\mathbf{W})$ where $\mathbf{V}$ and $\mathbf{W}$ protect templates acquired from different users whilst for a preponderant amount of genuine pairs $(\mathbf{V},\mathbf{W})$ the correlation attack is run by the attacker.

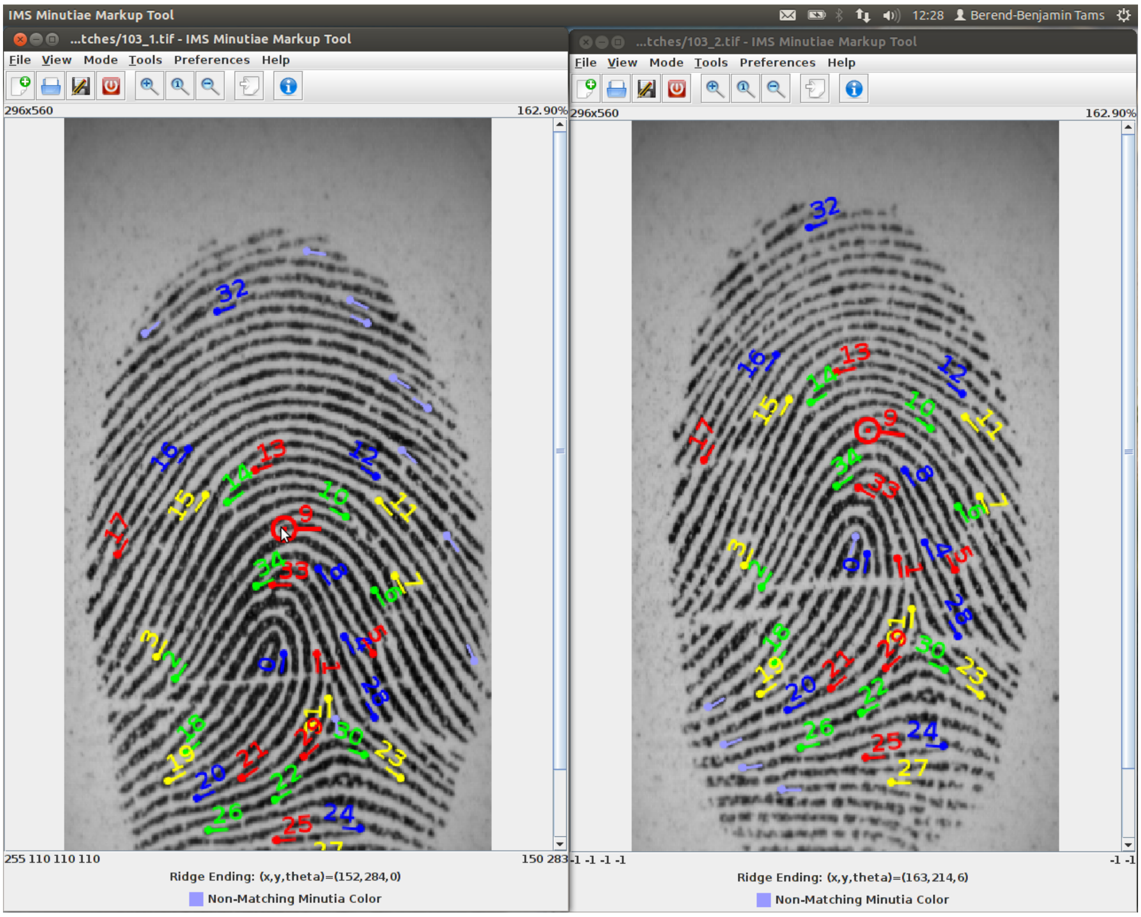
4.4.5 Finding a Reasonable Distance Threshold x

Before an attacker actually runs the decoding step in the correlation attack, he might want to apply criterion (4.34) to decide whether decoding the unlocking set $\mathbf{U}$ is worth it. For a fixed x, to apply the criterion, he must know the distance distribution function $\beta(x)$, which corresponds to the distance of vault points without a ground-truth matching minutia. However, the attacker still needs to choose a good distance threshold x. As already mentioned in Section 4.4.2, the attacker has a good reason to expect a large amount of genuine points in the unlocking set $\mathbf{U}$ if x is chosen such that $\alpha(x) - \beta(x)$ is maximal. Under the assumption $\alpha(x) \approx \delta(x)$ (see (4.39)) a reasonable choice for x is

$$x \approx \arg\max \delta(x) - \beta(x). \tag{4.41}$$

In the following we set our focus to the minutiae fuzzy vault implementation described in Section 3.4.

4.4.6 Distance Threshold for a Minutiae Fuzzy Vault


IMS Minutiae Markup Tool
12:28 Berend-Benjamin Tams
...tches/103_1.tif - IMS Minutiae Markup Tool
...tches/103_2.tif - IMS Minutiae Markup Tool
File View Mode Tools Preferences Help
296x560
162.90%
Non-Matching Minutia Color

255 110 110 110   150 283

Ridge Ending: (x,y,theta)=(152,284,0)

-1 -1 -1 -1   -1 -1

Ridge Ending: (x,y,theta)=(163,214,6)

Figure 4.3: Tool used for marking ground-truth minutiae correspondences where the minutiae have been extracted by a commercial extractor

In order to choose a good distance threshold as in (4.41) an attacker should have knowledge about the distribution functions $\delta(x)$ and $\beta(x)$. To measure both distribution functions, we consulted the FVC 2002 DB2-B dataset (Maio et al. (2002)), which consists of 8 scans each of 10 different fingers.

To measure $\delta(x)$, we only considered the first and the second respective impressions of the 10 fingers contained in the FVC 2002 DB2-B database. For each finger, the minutiae templates of the first and second impressions have been extracted using a commercial extractor.[4] Afterwards, an expert marked ground-truth minutiae correspondences between the first and the second impression for each of the ten fingers (see Figure 4.3). After having determined all minutiae correspondence between two matching templates, we determined an isometry that minimized the sum of squared distances between point correspondence given by ground-truth matching minutiae (see (Eggert et al. (1997)); see Section 2.2.3). Using the isometry, the second impressions are aligned to the first impressions of the corresponding finger. In this way, a

---

[4] Verifinger SDK 5.0 (Neurotechnology Ltd (2006))

set of ground-truth matching minutiae pairs $\mathbf{D} = \{(\mathfrak{m}, \mathfrak{m}')\}$ of cardinality $|\mathbf{D}| = 377$ was obtained. The distribution function $\delta(x)$ then was measured as

$$\delta(x) \approx \frac{\#\{(\mathfrak{m}, \mathfrak{m}') \in \mathbf{D} \mid d(\mathfrak{m}, \mathfrak{m}') \leq x\}}{\#\mathbf{D}} \tag{4.42}$$

where $d(\cdot, \cdot)$ is as in (2.5) with $w = 11.46$.

To measure $\beta(x)$ for each finger $i = 1, \ldots, 9$ in the FVC 2002 DB2-B database and for each finger index $j = i + 1, \ldots, 10$ two vaults $\mathbf{V}$ and $\mathbf{W}$ were built using the implementation described in Section 3.4. For the ith finger let T_{vault} denote the set of vault minutiae where T'_{vault} denotes the vault built from the first impression of the jth finger. For each $\mathfrak{m} \in T_{\text{vault}}$ let $\mathfrak{m}' \in T'_{\text{vault}}$ denote a minutia closest (with respect to $d(\cdot, \cdot)$) to T_{vault}. Since both vaults protect templates from different fingers it follows that (with the notation of Section 4.4.2) if $v \in \mathbf{V}$ is encoded by the minutia $\mathfrak{m}$ then $d_{\mathbf{W}}(v) = d(\mathfrak{m}, \mathfrak{m}')$. All pairs $(\mathfrak{m}, \mathfrak{m}')$ occurring this way were collected up into a set of non-matching close vault minutiae correspondences $\mathbf{B} = \{(\mathfrak{m}, \mathfrak{m}')\}$. Finally, $\#\mathbf{B} = 10\ 080$. The distribution function of vault points without a ground-truth match was measured as

$$\beta(x) = \frac{\#\{(\mathfrak{m}, \mathfrak{m}') \in \mathbf{B} \mid d(\mathfrak{m}, \mathfrak{m}') \leq x\}}{\#\mathbf{B}}. \tag{4.43}$$

In this a way we were able to estimate the distribution functions $\delta(x)$ and $\beta(x)$ for a minutiae fuzzy vault with respect to the distance measure as in Equation (2.5) on page 19 with $w = 11.46$. A plot of the estimated $\delta(x)$ and $\beta(x)$ can be found in Figure 4.4.

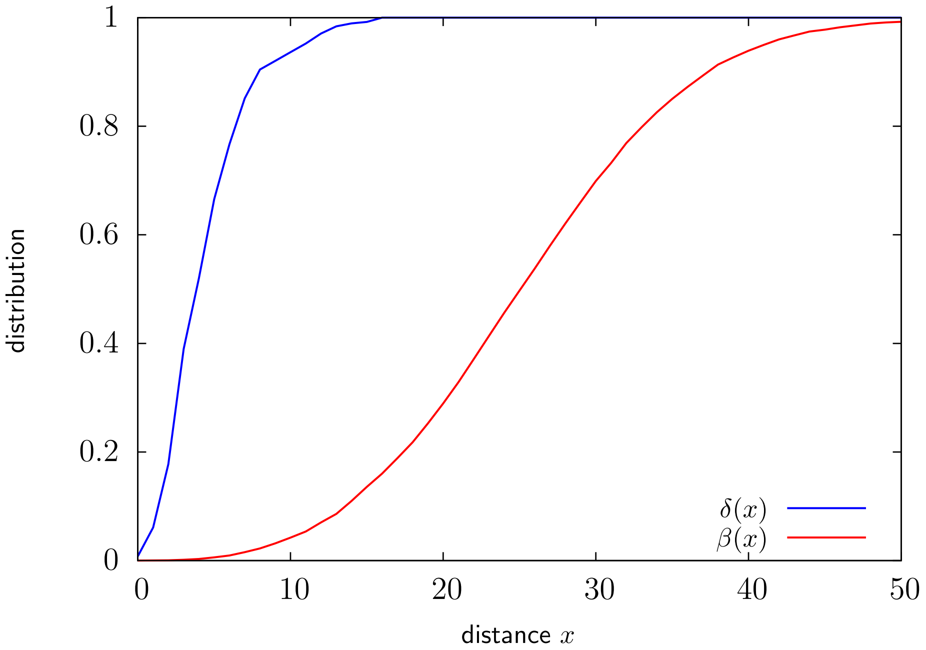


Figure 4.4: distance distribution of ground-truth matching minutiae $\delta(x)$ and for vault minutiae without a ground-truth mate $\beta(x)$

Finally, an estimation for the optimal x, i.e. which maximizes $\delta(x) - \beta(x)$, was determined by a global iterative search through all

$$x_i = 0.0001 \cdot i \text{ where } 0 \leq x_i \leq \min(\max \mathbf{B}, \max \mathbf{D}). \tag{4.44}$$

The estimation $\hat{x}$ for the optimal x has been determined as

$$\hat{x} = \arg\max_{x_i} \delta(x_i) - \beta(x_i), \tag{4.45}$$

which turned out to be

$$\hat{x} = 11.0869. \tag{4.46}$$

4.4.7 Correlating Minutiae Fuzzy Vaults

In the subsequent we demonstrate the effectiveness of cross-matching and the correlation attack using the example of minutiae fuzzy vaults.

Experimental Database

The test we performed used the implementation of Section 3.4. The database containing minutiae vault records have been built out of minutiae templates extracted from the images in the FVC 2002 DB2-A database (Maio et al. (2002)).[5] Analogously to Section 3.4.2 a minutiae fuzzy vault was built out of every minutiae template unless a *failure to capture* was encountered. In this way, we established a database containing minutiae fuzzy vaults.

Rates and Timings Measured by our Experiment

For all *genuine cross-matching attempts* (corresponding to genuine authentication attempts in the FVC protocol (see Section 3.4.7)) the second vault was aligned to the first vault.[6] Whenever Algorithm 4.4.2 output MATCH, with the aligned vaults and $\hat{x} = 11.0869$ as input, this was accounted as a *genuine cross-match*. In case of a genuine cross-match the aligned vaults and $\hat{x}$ were input to Algorithm 4.4.1 and the correlation attack was run. On success, the event was accounted as a *successful genuine correlation attack*. The proportion of all *genuine cross-matches* within all genuine cross-matching attempts is referred to as *genuine cross-match rate* and denoted by GCMR. The proportion of all successful genuine correlation attacks within all genuine cross-matches is referred as the *genuine correlation attack success rate* and denoted by GCASR.

Analogously, the impostor authentication attempts in the FVC protocol correspond to *false cross-matching attempts*. For each false cross-matching attempt whenever Algorithm 4.4.2 output MATCH this was accounted as a *false cross-match*. For a false positive cross-match the vaults were input to Algorithm 4.4.1. Whenever the correlation attack was successful this was accounted as a *successful false correlation attack*. The proportion of all *false cross-matches* within all false cross-matching attempts we refer to as *false cross-match rate* and denote it by FCMR. Consequently, the amount of successful false correlation attacks within false cross-matches is referred to as *false correlation attack success rate* and denoted by FCASR. Note that both the genuine correlation attack success rate and the false correlation attack success rate are conditional probabilities: A correlation attack was conducted only on a positive cross-match.

---

[5] As usual, we used minutiae features extracted with a commercial extractor (Verifinger SDK 5.0 (Neurotechnology Ltd (2006))).

[6] Analogously to Section 3.4.4, we decoupled the alignment from the vault: The vault minutiae of the second vault were mapped via the isometry aligning the second protected minutiae template to the first protected minutiae template in clear.

In addition to mere rates, we also report cross-match/correlation attack timings determined from our experiment. An attacker attempting to cross-match different application's database encounters mostly false cross-matching attempts before he cross-matches genuine vault correspondence. Within all false cross-matching attempts some wrongly cross-match resulting in a subsequent correlation attack. Hence it seems reasonable to measure the average amount of time an attacker has to spend for running the cross-matcher (Algorithm 4.4.2) and possibly the correlation attack (Algorithm 4.4.1). This corresponds to an iteration procedure. Consequently, we refer to this time as *impostor iteration time* and abbreviate it by IIT.

As a genuine cross-match attempt occurs (a priori unknown by the attacker) and the cross-matcher positively matches them, the attacker needs to spend some time in performing the subsequent correlation attack. The average time by performing a genuine correlation attack (given a positive cross-match) we refer to as *genuine correlation attack time* and abbreviate it by GCAT. Note that the genuine correlation attack time is not affected by whether the attack successfully breaks the vault or not.

Results of our Experiment

Table 4.7: Performance of cross-matching/correlation attack against the implementation described in Section 3.4 evaluated on the FVC 2002 DB2-A database. Rates in brackets correspond to results we measured on the sub-database of good quality as described in Section 3.4.8. Due to failure to captures the genuine cross-match rate corresponds to 2632 observed genuine authentication attempts (99 on the sub-database of good quality) while the false cross-match rate is based on a total of 4851 false cross-match attempts.

length of secret polynomial	genuine cross-match rate GCMR (sub-GCMR)	false cross-match rate FCMR	genuine correlation attack success rate GCASR (sub-GCASR)	false correlation attack success rate FCASR
$k = 7$	$\approx 39.02\%$ ($\approx 74\%$)	$\approx 3.15\%$	$\approx 99.03\%$ ($= 100\%$)	$\approx 0.65\%$
$k = 8$	$\approx 33.24\%$ ($\approx 68\%$)	$\approx 1.69\%$	$\approx 98.51\%$ ($= 100\%$)	$= 0\%$
$k = 9$	$\approx 27.05\%$ ($\approx 67\%$)	$\approx 1.11\%$	$\approx 97.89\%$ ($= 100\%$)	$= 0\%$
$k = 10$	$\approx 22.6\%$ ($\approx 64\%$)	$\approx 0.39\%$	$\approx 96.8\%$ ($\approx 97\%$)	$= 0\%$
$k = 11$	$\approx 18.92\%$ ($\approx 54\%$)	$\approx 0.21\%$	$\approx 96.59\%$ ($= 100\%$)	$= 0\%$
$k = 12$	$\approx 14.74\%$ ($\approx 46\%$)	$\approx 0.08\%$	$\approx 97.68\%$ ($= 100\%$)	$= 0\%$

For example, if $k = 9$ an amount of

$$\mathrm{GCMR} = 27.05\% \tag{4.47}$$

matching vault records have been successfully determined by Algorithm 4.4.2 while

$$\mathrm{FCMR} = 1.11\%. \tag{4.48}$$

The genuine correlation attack success rate (given a genuine cross-match) has been measured as

$$\mathrm{GCASR} = 97.89\% \tag{4.49}$$

Table 4.8: Computational performance of cross-matching/correlation attack against the minutiae fuzzy vaults as in Table 4.7. The timings have been measured on a single core of an `AMD Phenom(tm) II X4 955`.

length of secret polynomial	average genuine correlation attack time GCAT(sub-GCAT)	average impostor indexing time IIT
$k = 7$	$\approx 0.81s$ ($\approx 1.15s$)	$\approx 0.02s$
$k = 8$	$\approx 3.5s$ (≈ 6.14)	$\approx 0.03s$
$k = 9$	$\approx 12.94s$ ($\approx 19.5s$)	$\approx 0.05s$
$k = 10$	$\approx 42.53s$ ($\approx 54.51s$)	$\approx 0.04s$
$k = 11$	$\approx 88.35s$ ($\approx 157.35s$)	$\approx 0.05s$
$k = 12$	$\approx 172.2s$ ($\approx 221.84s$)	$\approx 0.03s$

at a false correlation attack success rate (given a false cross-match) of

$$\text{FCASR} = 0\%. \tag{4.50}$$

The genuine cross-match rate corresponding to the sub-database of good quality has been determined as

$$\text{sub-GCMR} = 67\% \tag{4.51}$$

where the corresponding genuine correlation attack success rate turned out to be

$$\text{sub-GCASR} = 100\%. \tag{4.52}$$

The false indexing time has been measured to be

$$\text{IIT} = 0.05s \tag{4.53}$$

in average while the correlation attack of a genuine vault correspondence lasted

$$\text{GCAT} \approx 12.94s \tag{4.54}$$

in average. The genuine correlation attack time on the database of good quality has been measured as

$$\text{sub-GCAT} \approx 19.5s. \tag{4.55}$$

For different $k = 7, \ldots, 12$ corresponding rates can be found in Table 4.7 while the timings are listed in Table 4.8.

Intermediate Discussion, Conclusion, and Outlook

According to our experiments, due to an impostor iteration time of $\text{IIT} \leq 0.05s$ the attacker is able to quickly eliminate false vault correspondences using the combination of cross-matching and correlation attack. Furthermore, the genuine cross-match rate $\text{GCMR} \geq 14.74\%$ ($\text{sub-GCMR} \geq 46\%$) are of an order that is sufficient if the attacker aims for gaining user-access for just one or a few users. Note, that an attacker can be quite certain to break at least one vault as the database share at least a certain amount of common users.

A conclusion from our experiments is that, given a well performing alignment mechanism, minutiae fuzzy vault implementations (as in Section 3.4) are highly vulnerable against cross-matching via correlation. Even worse, the probability of a successful correlation attack given a genuine vault correspondence is much too high to claim that minutiae fuzzy vault implementations are resistant against attacks via record multiplicity (see (Scheirer and Boult (2007))). For instance, for $k = 9$ on minutiae templates with good quality we were able to successfully break 67% of genuine vault correspondences using the combination of the cross-matching algorithm and possibly the correlation attack. With respect to successfully breaking genuine vault correspondences, we can support the result of Kholmatov and Yanikoglu (2008), who performed an experiment where the correlation attack successfully broke 59% of genuine vault correspondences but with vaults built using the implementation of Kholmatov et al. (2006).

Kholmatov and Yanikoglu (2008) additionally incorporate an exhaustive search for vault alignment in their implementation of the correlation attack. In our experiment, however, we assumed a well-solved alignment framework. If we would also perform alignment based on vault minutiae, our cross-matching rates would probably drop. Moreover, the impostor iteration time would increase. On the other hand, for most minutiae fuzzy vault implementations an automatic mechanism based on auxiliary alignment data is proposed (see Section 3.4.4). The alignment, thus, could be achieved using the public alignment helper data making exhaustive search obsolete. Moreover, auxiliary alignment helper data may help to filter out false vault correspondences, i.e. it may assist cross-matching (see Section 4.2). For these reasons, we decided to perform our experiments assuming the vaults are already aligned well.

Either way, our experiments clearly confirm that the fuzzy vault implementation of Section 3.4 is vulnerable to attacks via record multiplicity—in particular, if genuine vault correspondences are known in advance.

Furthermore, merely substituting minutiae features by other features (such as alignment-free features as proposed by Li et al. (2010)) is not sufficient to secure a fuzzy fingerprint vault against cross-matching or the correlation attack: As long as similarity of ground-truth matching vault features is expected to be significantly smaller than the similarity of non-matching vault features, correlating two vault records is possible to some extent.

In Chapter 5 we propose a minutiae fuzzy vault implementation refusing vault records to be correlated in the domain of vault minutiae. In this respect, via correlation, the implementation is resistant against cross-matching and the attack via record multiplicity.

5. Cross-Matching Resistant Minutiae Fuzzy Vault

In the last chapter, we have analyzed security of the fuzzy fingerprint vault. In particular, we simulated different attack scenarios by performing attacks against an own reference implementation incorporating ideas found in the literature. Even for naive brute-force attacks the fuzzy vault for fingerprints turns out to be highly vulnerable and can be broken within a reasonable amount of time. To furthermore increase the efficiency of an attack, the attacker can run statistical attacks or the false-accept attack. The success rates achievable by the attacker are highly alarming: Even in the best case for the system user we can only expect (with a confidence of 95%) the attacker to run a successful false-accept attack within approximately 1 minute (where the brute-force attack is expected to require approximately 7 months) on an ordinary personal desktop computer. This necessarily requires to conclude that single-finger are (with current methods) not sufficient to be incorporated in a secure authentication scheme with usable authentication performance. Rather, with respect to both security and usability, the capacity of multi-finger fuzzy vaults should be investigated in more detail (Merkle et al. (2010b)). But the fuzzy fingerprint vault has difficulties that can not be solved only by using multi-finger. If an intruder has intercepted two application's databases, he can expect to find genuine vault correspondences (given both databases share common users) by cross-matching via correlation. Even worse, with high probability he can even break a user's account given a genuine vault correspondence.

The problem of cross-matching and attacks via record multiplicity is not just a problem for fuzzy vault but also concerns other authentication schemes. In a password-based authentication scheme that are based on cryptographic hash functions there exists a simple countermeasure against cross-matching: Each user password is concatenated with an additional public string uniquely linked with the service provider, called *salt*; e.g., its web-address. Assume an adversary has to decide whether two salted hash values (with different salts) are originating from the same password. Both hash values, no matter if they came from the same or from different passwords, are (with overwhelming probability) different. The only conclusion the adversary can make, is that the salted passwords are different, which yields no additional insight to the adversary since the salts are indeed different.

In a biometric authentication scheme, based on the fuzzy commitment scheme, without preventions, an adversary can cross-match too. Assume two commitments $c+v$ and $c'+v'$ are given where v and v' are encodings of biometric templates and c and c' are codewords of a linear error-correcting code with error-correction capability ν. Now, the difference $c+v-(c'+v')=(c-c')+(v-v')$ can be corrected to the codeword $c-c'$ if $v-v'$ is of hamming weight $\leq \nu$. Otherwise, if $v-v'$ is of hamming weight $>\nu$ with overwhelming probability (depending on the parameters of the error-correcting code; see Section 2.3.3), the difference can not be corrected to any codeword. Thus, only from whether the intruder is able to decode he has a criterion to cross-match. Furthermore, the more v and v' agree, the easier it is for the attacker to even break both the commitments using a *decodability attack*, which is an attack via record multiplicity (Kelkboom et al. (2011)). Fortunately, for each application, an individual permutation of the positions in the vector v can effectively prevent an attacker to cross-match across different application's databases or to perform the decodability attack. The individual permutations in the fuzzy commitment scheme can be considered as to play the role of password salts.

Most implementations of the minutiae fuzzy vault (see Section 3.2.4) are vulnerable to cross-matching and the correlation attack (see Section 4.4). In order to achieve resistance against cross-matching we can incorporate a user password additionally to the vault as proposed by Nandakumar et al. (2007b). Using a user password, however, causes inconveniences we used to have with a password based authentication scheme, e.g., efforts for the user to keep the passwords secret (see the introducing discussion of Section 2.2).

In this chapter we develop an implementation of a minutiae fuzzy vault that is resistant against cross-matching without additional passwords. Resistance against cross-matching is achieved by rounding genuine minutiae to a rigid grid within the image region. Instead of generating a few chaff features, i.e. selecting a few grid points, at random, the vault features comprise the whole grid. In such a way, barring alignment, the feature set of two different vault instances are equal yielding no criterion to an intruder to match vault features via correlation. As a consequence, if an attacker has run Step 1 of Algorithm 4.4.1 the unlocking would consist of the entire vault points. This makes the correlation attack at least as hard as attacking one of the vault instances individually. Irrefutably, such an implementation is resistant against the correlation attack, and with the same argument also against cross-matching via correlation.

For our implementation we have deduced parameter configuration in a training stage that would result in good authentication performance. Unfortunately, on authentication the decoding timings would be much too costly when performed analogously to the decoding mechanism used in most implementation from the literature (Li et al. (2010); Nagar et al. (2008, 2010); Nandakumar et al. (2007a,b); Uludag and Jain (2006); Uludag et al. (2005)). On the other hand, if one of the efficient polynomial time algorithm for decoding Reed-Solomon codes were used (Guruswami and Sudan (1998); Massey (1969)), the genuine acceptance rate would drop drastically. As a trade-off we propose the use of a randomized decoder in where in each of a preliminarily fixed number of iterations the candidates for interpolation points are chosen randomly from the unlocking set rather to choose all combinations of all interpolation points systematically. For our proposed vault construction the randomized decoder turns out to have the potential of providing good genuine acceptance rate at low false acceptance rate as we observed in an experiment conducted on a public-domain database. Well conceived, the randomized decoder might be incorporated into a multi-finger fuzzy vault implementation.

Another advantage of using exactly reproducible vault features (such as coarsely quantized minutiae) is that the modified fuzzy vault construction proposed by Dodis et al. (2008), which avoids the generation of chaff points, can be applied without affecting authentication performance or security against the brute-force attack. As a consequence, the storage size of each vault instance reduces significantly. On the other hand, although we see no efficient way to perform an attack via record multiplicity, without prevention multiple instances of the modified vault construction can be cross-matched to some extent. Fortunately, by choosing different encodings from vault features to finite field elements for each application, cross-matching can be effectively prevented (similar to as a permutation can prevent cross-matching in a fuzzy commitment scheme (Kelkboom et al. (2011))).

This chapter is outlined as follows. In Section 5.1 we describe how minutiae from an input minutiae template are selected and quantized. Furthermore, we describe how they can be protected using a fuzzy vault scheme without weaken it against cross-matching via correlation. Afterwards, in Section 5.2 the vault work of the modified fuzzy vault scheme proposed by Dodis et al. (2008) is reviewed and we describe how it can be used to protect the quantized minutiae templates we propose in Section 5.1. Afterwards, in Section 5.3 we specify how we selected vault parameter in a training conducted on the FVC 2002 DB2-B dataset (Maio et al. (2002)). In Section 5.4 we define the randomized decoder and describe how it can be used on authentication. After a description of the details of our proposed implementation in Section 5.5, we report the result of a performance evaluation we conducted on the FVC 2002 DB2-A database in Section 5.6. Subsequently, analyses of the three different attack scenarios that we discussed in Chapter 4 are given in Section 5.7. This comprises a discussion on to what extent cross-matching is possible and how it can be prevented when the modified vault construction of Dodis et al. (2008) is used for building the vault records. Finally, in Section 5.8 we compare our implementation's authentication performance and security with those of other implementations of the fuzzy fingerprint vault proposed in the literature.

5.1 Basic Vault Construction

We start with the description on how we encode a minutia as an element of a finite field.

5.1.1 Vault Feature from a Single Minutia

Let $R_0, \ldots, R_{r-1}$ be those coordinates of a hexagonal grid of distance λ that lay in the regions of a fingerprint's image. We fix a parameter α controlling how coarsely minutiae directions are quantized: Using α, we divide the range $[0, 2\pi)$ into equally sized connected quanta; more precisely, for $j = 0, \ldots, 2^\alpha - 1$ define the interval

$$D_j = [\ j \cdot 2\pi/2^\alpha\ ,\ (j+1) \cdot 2\pi/2^\alpha\ ).$$

Now, for each $i = 0, \ldots, r-1$ and each $j = 0, \ldots, 2^\alpha - 1$ we fix a unique element $x_{i,j}$ of an appropriate finite field $\mathbf{F}$.[1] More precisely, $i \neq i'$ or $j \neq j'$ implies $x_{i,j} \neq x_{i',j'}$. A minutia $\mathfrak{m} = (a, b, \theta)$ at the location (a, b) and of direction θ is then encoded as follows. First, we determine which grid point R_i is closest to (a, b).[2] Second, let j be the unique index j such that $\theta \in D_j$. In such a way, we encode the minutia $\mathfrak{m}$ via the map

$$q : \mathfrak{m} \mapsto x_{i,j}. \tag{5.1}$$

Note, different minutiae can have an equal quantization, e.g., if they are very close and have similar direction. In this way, minutiae quantizations can occur in the universe

$$\mathbf{E} = \{ \ q(\ R_i \ , \ j \cdot 2\pi/2^\alpha \ ) \ \mid \ i = 0, \ldots, r-1, \ j = 0, \ldots 2^\alpha - 1 \ \} \tag{5.2}$$

of size $r \cdot 2^\alpha$.

In the following we describe how we encode a full minutiae template as the set of genuine features.

5.1.2 Vault Feature Set from Minutiae Template

Let t be a bound on the template size and T be an input minutiae template. In order to only quantize the most reliable minutiae from T we assume that T is sorted in dependence of minutiae quality (Chen et al. (2005)). More precisely, write

$$T = \{\mathfrak{m}_1, \mathfrak{m}_2, \ldots\} \tag{5.3}$$

such that if $\mathfrak{m}_i$ is of better quality than $\mathfrak{m}_j$ then $i < j$ is assumed. We quantize the minutiae from T successively collecting the quantization in a feature set $\mathbf{A}$ until it reaches as size at most t.

More precisely, the feature set after the ith minutia has been quantized contains

$$\mathbf{A}_i = \{ \ q(\mathfrak{m}_1), \ldots, q(\mathfrak{m}_j) \ \mid \ j = \min(i, \#T) \ \}. \tag{5.4}$$

Note, that

$$\mathbf{A}_1 \subseteq \mathbf{A}_2 \subseteq \ldots \subseteq \mathbf{A}_t. \tag{5.5}$$

If $q(\mathfrak{m})_i \in \mathbf{A}_{i-1}$ then $\mathbf{A}_{i-1} = \mathbf{A}_i$. Thus, $|\mathbf{A}_i| \leq i$.

We define the feature set to contain at most t elements. Hence, we stop to quantize minutiae from T if $|\mathbf{A}_i| = t$ or if all minutiae from T have already been quantized, i.e. if $i = t$. More precisely, the final feature set is defined to be

$$\mathbf{A} = \underset{\mathbf{A}_i}{\arg\max}\{ \ \#\mathbf{A}_i \mid \#\mathbf{A}_i \leq t \ \}. \tag{5.6}$$

In Algorithm 5.1.2 (below) the construction of the set $\mathbf{A}$ is described in algorithmic description.

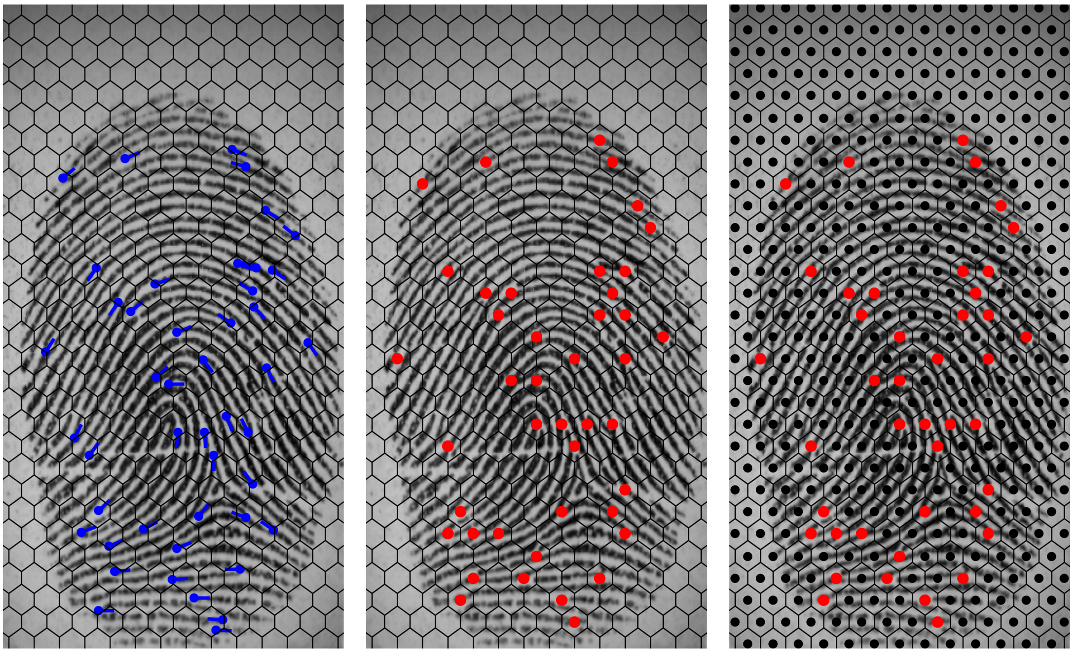

(a) minutiae template (b) genuine vault features (c) entire vault features

Figure 5.1: If $\alpha = 0$, depending on where minutiae locations occur (a) points of a hexagonal grid are selected (b) to result in the template of coarsely quantized minutiae. Every possible grid point (c) encodes a vault point such that the feature set between different vaults are equal. As a consequence, correlation of multiple vault records (see Section 4.4) does not help to filter out chaff points.

5.1.3 Enrollment: Vault Construction

A simple way to protect $\mathbf{A}$ using a vault construction that is resistant against cross-matching via correlation is to encode the genuine points by the elements of $\mathbf{A}$ while the chaff points are encoded by the remaining elements in $\mathbf{E}$, i.e. $\mathbf{E}\setminus\mathbf{A}$. In such a way, an attacker is not able to gain any benefit from correlating two matching vault records—neither to cross-match nor to recover the templates via record multiplicity. The reason is, that (barring alignment) the vault features are equal. Hence, the unlocking set in Step 1 of Algorithm 4.4.1 consists of the entire vault, no matter what $x \geq 0$ has been chosen. As a consequence, cross-matching two vault records via correlation is at least as hard as breaking one of the vaults individually.

The remaining details to construct the vault can be implemented following the formal description of Section 3.1: Given a secret, encoded as a polynomial $f \in \mathbf{F}[X]$ of degree $< k$, for every $x \in \mathbf{A}$ the genuine set

$$\mathbf{G} = \{ \ (x, f(x)) \mid x \in \mathbf{A} \ \} \tag{5.7}$$

consists of those vault points binding the feature set $\mathbf{A}$ to the polynomial f. The genuine points are hidden by the chaff points, which are encoded by the elements of $\mathbf{E} \setminus \mathbf{A}$, where the ordinate values are chosen randomly such that the corresponding points do not lay on the polynomial's graph. More formally,

$$\mathbf{C} = \{(x, y) \mid x \in \mathbf{E} \setminus \mathbf{A}, \ y \in_R \mathbf{F} \setminus \{f(x)\} \ \}. \tag{5.8}$$

---

[1] Note, for the finite field $\mathbf{F}$ it is required that $\#\mathbf{F} \geq r \cdot 2^{\alpha}$.

[2] Note, that there does not need to exist a unique nearest grid point. The grid point R_i has to be determined by a rounding-convention.

Finally, the public vault data is

$$(\mathbf{V}, h(f)) \tag{5.9}$$

where $\mathbf{V} = \mathbf{G} \cup \mathbf{C}$ and $h(f)$ denotes a cryptographic hash value of the polynomial f.

5.1.4 Authentication: Ordinary Iterative Decoder

Assume a minutiae template aligned to the vault $(\mathbf{V}, h(f))$ is given on authentication. Query vault features $\mathbf{B} \subset \mathbf{F}$ are extracted in the same manner as genuine vault features have been extracted on enrollment, i.e. $\mathbf{B}$ contains at most t elements. Next, the unlocking set is constructed by letting

$$\mathbf{U} = \{\ (x, y) \in \mathbf{V} \mid x \in \mathbf{B}\ \}. \tag{5.10}$$

Now, following the ordinary iterative decoding procedure that we already used for our implementation of Section 3.4 (see Section 3.4.3), and which has been used in most implementations of the fuzzy fingerprint vault from the literature (Li et al. (2010); Nagar et al. (2008, 2010); Nandakumar et al. (2007a,b); Uludag and Jain (2006); Uludag et al. (2005)), we iterate through all combinations of k different unlocking points from $\mathbf{U}$ and determine its interpolation polynomial f^*. If $h(f^*) = h(f)$ then the secret f^* is output and the authentication was successful. Otherwise, we continue with the next combination of k different unlocking points until one of its interpolation polynomial can be successfully re-identified with the help of its hash value. If no combination yields a polynomial with hash value $h(f)$ the authentication is considered to be unsuccessful.

The ordinary iterative decoder requires the unlocking set $\mathbf{U}$ to have quite a limited size, which contains as many elements as $\mathbf{B}$, while the size of the secret k must be well-balanced. More precisely, to feasibly decode the unlocking set $\mathbf{U}$ the number of iterations

$$\binom{t}{k} = \frac{t!}{k! \cdot (t-k)!} \tag{5.11}$$

must be possible to perform within a reasonable amount of time.

We will see below, that the above expression becomes too large for parameters that we determined in a training stage. As an effective countermeasure, we will propose a randomized decoder that slightly differs from the systematic decoding procedure above.

In other implementations of the fuzzy vault to fingerprints, on authentication, there is an extraction step. For example, in the implementation described in Section 3.4 the query minutiae template is used to extract vault minutiae, and thus their corresponding vault points. If among the extracted vault minutiae there are at least k genuine vault minutiae the decoding procedure will be successful in recovering the protected polynomial. For our vault construction, there is no extraction step on authentication. Rather the success of authentication depends on how many genuine vault features have been successfully reproduced and are contained in the query feature set. In other words, if $\mathbf{A} \cap \mathbf{B}$ contains at least k elements, the iterative decoder will successfully recover the protected polynomial.

5.1.5 Implementation Details

It remains to specify some of the details how we build the feature sets from a minutiae template.

Hexagonal Grid

Minutiae locations, for example, are rounded to the points of a hexagonal grid laying in the region of the corresponding fingerprint image of dimension $M \times N$. Furthermore, this grid depends on the minimal distance $\lambda > 0$ of grid points. Given $M \times N$ and λ the hexagonal grid we use for quantizing minutiae can be determined by the following algorithm.

Algorithm 5.1.1 (Hexagonal Grid Points)**.**

Input: The dimension in where minutiae locations can occur $M \times N$; the minimal distance $\lambda > 0$ of the hexagonal grid.

Output: A list $R_0, \ldots, R_{r-1}$ of hexagonal grid points laying within the fingerprint image's dimension $M \times N$.

```
1: /* initialize */
   $dx \leftarrow \lambda \cdot \tan(60^\circ)/2$;
   $r \leftarrow 0$;

2: /* shifts for centering grid points */
   $x_0 \leftarrow (M - \lfloor (M-1)/dx \rfloor \cdot dx)/2$;
   $y_0 \leftarrow (N - \lfloor (N-1)/\lambda \rfloor \cdot \lambda)/2$;

3: /* loop on $x$-coordinate */
   $x \leftarrow x_0$;
   while $x < M$

4:     /* determine alternating offset of $y$-coordinate */
       if $i$ is even
           $y \leftarrow y_0 + \lambda/2$;
       else
           $y \leftarrow y_0$;
       endif

5:     /* loop on $y$-coordinate */
       while $y < N$

6:         /* add to list */
           $R_r \leftarrow (x, y)$;

7:         /* increment $y$-coordinate */
           $y \leftarrow y + \lambda$;
           $r \leftarrow r + 1$;
       end

8:     /* increment $x$-coordinate */
       $x \leftarrow x + dx$;
   endfor

9: /* output */
   return $R_0, \ldots, R_{r-1}$.
```

The values x_0 and y_0 are used to center the grid points such that the grid points do well fit into the fingerprint's image region.

When the algorithm is run with input $M = 296$, $N = 560$, and $\lambda = 22$ we obtain a list of $r = 405$ grid coordinates centered in an image of dimension 296×560, which is the dimension of the images contained in the FVC 2002 DB2 database as visualized in Figure 5.1.

Finite Field

The quantization of minutiae are represented as elements of a finite field $\mathbf{F}$. Except its size there is no particular restriction for the finite field. More precisely, the finite field must have cardinality

$$|\mathbf{F}| \geq 2^{\alpha} \cdot r, \tag{5.12}$$

where α controls how significantly minutiae angles are taken into account and r denotes the size of the hexagonal grid used for quantization. If the above inequality is satisfied, the finite field is of sufficient size such that it can encode each minutia's quantization (index of grid point and α-bit quantization for the direction) following the description of Section 5.1.1.

If α and r are kept within reasonable limits (e.g, $\alpha \leq 4$ and $r \leq 2^{12} = 4096$) then each minutia's quantization can be encoded using 16 bits. Thus, it suffices to use a finite field of cardinality 2^{16}, which is widely used for many purposes. Furthermore, the size of this finite field is not too large. This makes the pre-computation of logarithm and anti-logarithm multiplication tables possible, which significantly accelerates subsequent finite field arithmetic.

For these reasons, throughout we use the finite field

$$\mathbf{F} = \mathbb{F}_{2^{16}} \tag{5.13}$$

for computations with the vault, even if we do not mention this explicitly.[3]

Hash Value of Secret Polynomial

To allow safe recovery of the secret polynomial f we additionally store a cryptographic hash value $h(f)$ along with the vault. Again, there is no particular restriction which cryptographic hash function we use for this purpose except security and performance.

For simplicity, we use the *secure hash algorithm* (National Institute of Standards and Technology (1995)). Even though there are more secure hash algorithms (National Institute of Standards and Technology (2002)), we claim that the SHA-1 is sufficient for our evaluation purposes.

Given the polynomial f, by $h(f)$ we denote its SHA-1 hash value. The value $h(f)$ depends on how the data of f is represented. Write

$$f(X) = f_0 + f_1 \cdot X + f_2 \cdot X^2 + \ldots + f_d \cdot X^d. \tag{5.14}$$

where $d \leq k$ and $f_d \neq 0$. Then $f_0, \ldots, f_d \in \mathbf{F}$ are 2-byte (16-bit) values. The values f_i are concatenated $(f_0 \| \cdots \| f_d)$ such that an array of $(d+1) \cdot 2$ bytes is obtained. This array is passed as input to the *secure hash algorithm* as described in (National Institute of Standards and Technology (1995)) to obtain the corresponding hash value as a 20-byte (160-bit) array.

---

[3] More specifically, we have chosen the isomorphy class $\mathbf{F} \simeq \mathbb{F}_2[X]/(1 + X + X^3 + X^5 + X^{16})$.

Quantization of Minutiae

When the hexagonal grid, α, and the finite field are given, we can perform the quantization of a single minutia. This quantization procedure corresponds to the map $q : \{\mathfrak{m}\} \to \mathbf{F}$ as we have already generically used it in Section 5.1.1. We next outline how we use to quantize minutiae, i.e. how we implement the map q—even though there are other valid ways to realize this.

Let $\mathfrak{m} = (a, b, \theta)$ be a minutia at location (a, b) and of direction $\theta \in [0, 2\pi)$. The way the minutia $\mathfrak{m}$ is quantized depends on both the minutia's position (a, b) and its direction θ. More precisely, in numerical representation the minutia $\mathfrak{m}$ quantizes as

$$q(\mathfrak{m}) = j + r \cdot \left\lfloor (2^\alpha - 1) \cdot \frac{\theta}{2\pi} \right\rceil \tag{5.15}$$

where

$$\begin{aligned} j &= \min\{i \ : \ \|R_i - (a,b)\|_2 = d\} \\ d &= \min\{\|R_i - (a,b)\|_2\}. \end{aligned} \tag{5.16}$$

Note, that there might be multiple i such that R_i is closest to (a, b), i.e. $\|R_i - (a,b)\|_2 = d$, where d is the minimal distance of the position to the hexagonal grid points $R_0, \ldots, R_{r-1}$. Since we desire that only a single hexagonal grid point should be associated with (a, b), we extract the first hexagonal grid point occurring in the list $R_0, \ldots, R_{r-1}$ that is of minimal distance to (a, b). In other words, the way we defined the index j corresponds to a rounding convention.

To encode the α-bit quantization of the direction θ of the minutia $\mathfrak{m}$ independently from the grid point index j the quantization of the direction

$$\left\lfloor (2^\alpha - 1) \cdot \frac{\theta}{2\pi} \right\rceil \tag{5.17}$$

must be shifted by a multiple of r. The sum gives the quantization in integer representation.

The expression (5.15) interpreted bit-wisely as an element of the finite field $\mathbf{F}$ finally encodes a quantization of the minutia $\mathfrak{m}$.

For the matter of completeness, we give an algorithmic description that quantizes a full minutiae template according to (5.6). The description on how a minutiae template is quantized is better understood by an informal description, so we give such a description prior to an algorithmic description.

Roughly speaking, the procedure requires a list of minutiae (e.g., sorted by their respective qualities achieved by some convention). The minutiae in the list are quantized iteratively according to (5.15) and interpreted as an element of a finite field. These elements are inserted into the set containing the minutiae quantization. The iteration stops if there are no more minutiae contained in the list or if the set of quantized minutiae has reached a size of a pre-determined bound t. More formally,

Algorithm 5.1.2 (Quantization of Minutiae Template)**.**

Input: A minutiae template given as a list $\mathfrak{m}_1, \ldots, \mathfrak{m}_s$; a bound on the size t of the set containing collecting quantized minutiae.

Output: The set of $\mathbf{A} \subset \mathbf{F}$ as in (5.6).

```
1: /* initialize */
   $\mathbf{A} \leftarrow \emptyset$;
   $i \leftarrow 1$;

2: /* iterate */
   while $i \leq s$ and $|\mathbf{A}| < t$

      /* if $q(\mathfrak{m}_i)$ is not contained in A the size of A will increase */
      $\mathbf{A} \leftarrow \mathbf{A} \cup \{q(\mathfrak{m}_i)\}$;

      $i \leftarrow i + 1$;
   end

3: /* output */
   return $\mathbf{A}$;
```

5.2 Modified Vault Construction without Chaff Points

An advantage of our construction is that it can be easily modified such that the *modified vault construction* proposed by Dodis et al. (2008). Using the construction, the generation of chaff points is avoided which results in significantly smaller record sizes.

The modified vault construction is easily described. Let $\mathbf{A} \subset \mathbf{F}$ be the set of genuine quantized minutiae and $f \in \mathbf{F}$ be the secret polynomial of degree $< k$. The modified vault construction essentially is the polynomial

$$V(X) = f(X) + \prod_{x \in \mathbf{A}} (X - x) \tag{5.18}$$

instead of points in $\mathbf{F} \times \mathbf{F}$.

Note that if $x \in \mathbf{A}$ then

$$V(x) = f(x) \tag{5.19}$$

and thus $(x, V(x))$ is a genuine point. Conversely, if $x \notin \mathbf{A}$ then

$$V(x) \neq f(x) \tag{5.20}$$

and thus $(x, V(x))$ is a chaff point. Consequently, the genuine points for the modified vault constructions are

$$\mathbf{G} = \{\ (x, V(x)) \mid x \in \mathbf{A}\ \} \tag{5.21}$$

and the chaff points are

$$\mathbf{C} = \{\ (x, V(x)) \mid x \in \mathbf{E} \setminus \mathbf{A}\ \} \tag{5.22}$$

where $\mathbf{E} \subset \mathbf{F}$ is the universe in where vault features can occur (also see Equation (5.2)). Using a vault polynomial that is of degree $t = |\mathbf{A}|$ instead of vault points, the generation and storage of chaff points can be effectively avoided.

On authentication, an (alleged genuine) user provides a second minutiae template aligned to the vault from where quantized minutiae $\mathbf{B} \subset \mathbf{F}$ are extracted. The unlocking set is obtained by letting

$$\mathbf{U} = \{ \ (x, V(x)) \mid x \in \mathbf{B} \ \}. \tag{5.23}$$

The success of decoding the unlocking set $\mathbf{U}$, as for the basic vault construction, depends on how many elements the genuine feature set $\mathbf{A}$ and the query feature set $\mathbf{B}$ share.

The big advantage of the modified vault construction, is that the record sizes are significantly smaller than those for the original vault construction.

For a discussion of the information theoretic security of the above construction we refer to (Dodis et al. (2008)). We make the remark that an instance of the original vault construction can be derived from the modified vault construction by letting

$$\mathbf{V} = \{ \ (x, V(x)) \mid x \in \mathbf{E} \ \} = \mathbf{G} \cup \mathbf{C}. \tag{5.24}$$

In this way, the difficulty in breaking a modified vault construction is equivalent to breaking an instance of the original vault construction. Yet the chaff points are not generated randomly anymore. However, an adversary has a minor chance to cross-match multiple instances of the modified vault construction even if this does not necessarily enable an attack via record multiplicity. This point as well as a countermeasure will be discussed during Section 5.7.

Anyway, on authentication the success of decoding the unlocking sets does not depend on what vault construction is used and thus authentication performance is not affected. But the modified vault construction is a simple and elegant solution to avoid the storage of chaff points, which can become very large in number.

5.3 Training

For the parameters introduced in Section 5.1 we performed systematical tests to find a good vault configuration.

5.3.1 Configuration

According to our description, vault configuration depends on

- λ, which controls the number of grid points r,
- α determining how coarsely a minutia's direction is to be taken into account,
- t is the upper bound of the size of the genuine set, and
- k bounds the degree the secret polynomial can have.

5.3.2 Training Database

We used FVC 2002 DB2-B (Maio et al. (2002)), which is intended for training purposes, to find a good parameter configuration. This set consists of scans of 10 different fingers (indexed $101, \ldots, 110$). For each finger, 8 scans are contained in the dataset yielding a total of 80 fingerprint images.

To obtain the corresponding minutiae templates (sorted with respect to their estimated qualities), a commercial extractor (Neurotechnology Ltd (2006)) has been used.

5.3.3 Genuine Acceptance Rate

For a given configuration, we determined the genuine acceptance rate and the false acceptance rate according to the FVC protocol (Maio et al. (2002)).

More precisely, the genuine acceptance rate for a fixed configuration, was determined as follows. For each finger index $i = 101, \ldots 110$ and each scan $j = 1, \ldots, 7$ the corresponding minutiae template T was quantized as $\mathbf{A}$ following the procedure of Section 5.1.2. Then for each $j' = j + 1, \ldots, 8$ the j'th scan of the ith minutiae template aligned to T was quantized as $\mathbf{B}$.

We counted how many elements the sets $\mathbf{A}$ and $\mathbf{B}$ share. If $|\mathbf{A} \cap \mathbf{B}| \geq k$ then the authentication attempt was counted as a success. Otherwise, if $|\mathbf{A} \cap \mathbf{B}| < k$ the authentication attempt was unsuccessful.

The proportion of successful genuine authentication attempts within the total number of simulated genuine authentication attempts (i.e. $10 \cdot (7 \cdot 8)/2 = 280$) yielded the genuine acceptance rate.

Note, it is not necessary to build the vaults (basic or modified construction) or to perform the exhaustive decoding work for determining the authentication rates. A successful decoding using the ordinary iterative decoding procedure occurs if and only if $|\mathbf{A} \cap \mathbf{B}| \geq k$.

5.3.4 Alignment

In the protocol used to obtain the genuine acceptance rate, we assumed that the query fingerprints were aligned to the vaults. To simulate authentication performances in a well-solved alignment framework, we decoupled the alignment work from the vault. Consequently, the alignment was obtained by aligning both minutiae templates in clear.

For our training and experiments, we used the same alignment procedure as we already used for the performance evaluation of our standard minutiae vault implementation (see Section 3.4.4). However, even if we performed our investigations by assuming a well-solved alignment framework, fingerprint alignment under security aspects still asks for a valid solution.

5.3.5 False Acceptance Rate

Analogously to the FVC protocol, the *false acceptance rates* for a given configuration were determined as follows: For each finger $i = 101, \ldots, 109$ and each finger $i' = i + 1, \ldots, 110$ the templates T and S corresponding to their respective first impressions were acquired. T was quantized as $\mathbf{A}$ and S as $\mathbf{B}$ analogously to the description of Section 5.1.2. If $|\mathbf{A} \cap \mathbf{B}| \geq k$, the impostor authentication attempt was counted as a successful match and as a non-match otherwise. The number of successful impostor authentication attempts within the number of totally observed impostor authentication attempts ($(10 \cdot 9)/2 = 45$) yielded the *false acceptance rate*.

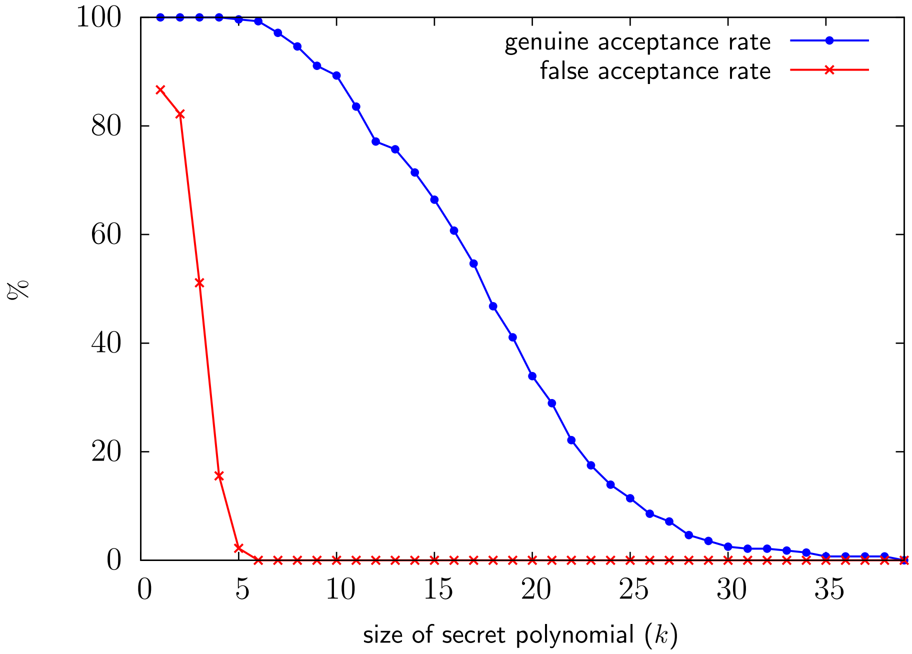


Figure 5.2: Genuine- and false acceptance rates on FVC 2002 DB2-B for $\lambda = 22$, $\alpha = 2$, $t = 46$, and varying k

Table 5.1: Training result: Genuine acceptance rates and false acceptance rates for varying k on the FVC 2002 DB2-B

size of secret polynomial k	genuine acceptance rate in % GAR	false acceptance rate in % FAR	security against brute-force attack $\mathbf{bf}(2^{\alpha} \cdot n, t, k)$
4	$= 100\%$	$\approx 15.56\%$	$\approx 2^{21}$
5	$\approx 99.64\%$	$\approx 2.22\%$	$\approx 2^{26}$
6	$\approx 99.29\%$	$= 0$	$\approx 2^{31}$
7	$\approx 97.14\%$	$= 0$	$\approx 2^{36}$
8	$\approx 94.64\%$	$= 0$	$\approx 2^{42}$
9	$\approx 91.07\%$	$= 0$	$\approx 2^{47}$
10	$\approx 89.29\%$	$= 0$	$\approx 2^{53}$
11	$\approx 83.57\%$	$= 0$	$\approx 2^{58}$
12	$\approx 77.14\%$	$= 0$	$\approx 2^{64}$

5.3.6 Training Results

We applied the protocols to determine the *genuine acceptance rate* and *false acceptance rate* for all configurations of

$$\begin{aligned} \lambda &= 8,\ldots,32 \\ \alpha &= 0,\ldots,3 \\ t &= 10,\ldots,60 \text{ , and} \\ k &= 1,\ldots,t. \end{aligned} \tag{5.25}$$

The best configuration, with respect to the highest *genuine acceptance rate* (GAR) at the zero *false acceptance rate* (FAR), was obtained as

$$\begin{aligned} \lambda &= 22 \\ \alpha &= 2 \\ t &= 46 \\ k &= 6 \end{aligned} \tag{5.26}$$

with corresponding authentication rates

$$\begin{aligned} \text{GAR} &\approx 99.29\% \quad \text{and} \\ \text{FAR} &= 0\%. \end{aligned} \tag{5.27}$$

Note, the *false acceptance rate* is not identical to zero but there occurred no false accept within 45 impostor authentication attempts.

For the configuration determined in the above training, the security against the brute-force attack of Section 4.1 can be determined as follows. The images of the FVC 2002 DB2 are of dimension 296 × 560 pixels. As a consequence, the number of hexagonal grid points that fit in the images region is

$$r = 405. \tag{5.28}$$

As a result, vault security against the brute-force attack of Section 4.1 estimates as

$$\mathbf{bf}(2^{\alpha}\cdot r,t,k)=\mathbf{bf}(4\cdot 405,46,6)=\binom{1620}{6}\cdot\binom{46}{6}^{-1}\approx 2^{31}. \tag{5.29}$$

Higher resistances against the brute-force attack are achievable for higher k while the genuine acceptance rate drops correspondingly as sketched by Figure 5.2.

Consequently, from our training the configuration we used for our experiments is

$$\begin{aligned} \lambda &= 22 \\ \alpha &= 2 \\ t &= 46 \end{aligned} \tag{5.30}$$

and k is varying.

5.4 Randomized Decoder

Even if the authentication performances as well as the brute-force securities (for higher k) of our minutiae fuzzy vault implementation look promising (see Table 5.1), there remains a problem concerning the decoding work.

For example, if 2^{40} security is sought, a secret size at least $k = 8$ should be used. The ordinary iterative decoder (see Section 5.1.4) would require a computational cost up to

$$\binom{t}{k} = \binom{46}{8} \approx 2^{28} \tag{5.31}$$

before a user gets authenticated or rejected. This, however, is far away from being acceptable for a user-friendly authentication scheme. In the following, we propose a mechanism to deal with this problem.

5.4.1 Bound the Number of Decoding Iterations

On decoding an unlocking set $\mathbf{U}$ of size t we fix a bound of iterations $\mathcal{D}$. Instead of iterating through all

$$\binom{t}{k} = \frac{t!}{k! \cdot (t-k)!} \tag{5.32}$$

combinations of k distinct unlocking points systematically, we randomize this procedure: For at most $\mathcal{D}$ iterations, we randomly select k elements from the unlocking set $\mathbf{U}$, determine its interpolation polynomial f^*, and test whether $h(f^*) = h(f)$. If $h(f^*) = h(f)$ then f^* is output as the correct polynomial; otherwise, we repeat with the next iteration.

The iteration bound $\mathcal{D}$ must be chosen such that decoding will be likely successful if $\mathbf{U}$ contains a reasonable amount of genuine points.

5.4.2 Change in the Genuine Acceptance Rate

When using the randomized decoder, we first should investigate how genuine acceptance rate and false acceptance rate change.

Within each iteration in the randomized decoder there is a non-zero chance that the secret polynomial can be reconstructed if the unlocking set contains at least k genuine points. More precisely, for a single iteration the chance that the randomized decoder will output the secret polynomial if the unlocking set of size t contains exactly $\omega \geq k$ vault points is

$$\mathbf{bf}(t, \omega, k) = \binom{t}{k} \cdot \binom{\omega}{k}^{-1}. \tag{5.33}$$

Consequently, if $\mathcal{D}$ iterations are performed to decode, the probability for a success is

$$\epsilon(\omega, t, k, \mathcal{D}) = 1 - (1 - \mathbf{bf}(t, \omega, k)^{-1})^{\mathcal{D}}. \tag{5.34}$$

Let

$$\mathrm{GAR}(\omega) = \mathbb{P}(\ \#(\mathbf{A} \cap \mathbf{B}) \geq \omega \mid \mathbf{A} \text{ and } \mathbf{B} \text{ are true matches }) \tag{5.35}$$

denote the chance for a genuine query template to overlap with the reference template in at least ω elements (note, these rates coincide with the rates for if the ordinary iterative decoder were used). Then the genuine acceptance rate using the randomized decoder extrapolates as

$$\mathrm{GAR} \geq \sum_{\omega \geq k} \epsilon(\omega, t, k, \mathcal{D}) \cdot (\mathrm{GAR}(\omega) - \mathrm{GAR}(\omega + 1)). \tag{5.36}$$

For a sharp estimate it would be necessary to make use of the distribution function on the size of unlocking sets. Even more inconvenient, conditional probabilities for an unlocking set to contain at least ω genuine points given the size of the unlocking sets would be required. Measurements for these conditional probabilities can be done on very small test sets for the FVC 2002 DB2 database. As a result, they are unreliable. For these reasons, we will use the bound (5.36) to relate the change of genuine acceptance rate of our implementation using the randomized decoder to the genuine acceptance rates that would be achievable for the ordinary iterative decoder. Even if the genuine acceptance rate would be larger than (5.36) it is safer to give a lower bound for the genuine acceptance rate rather an upper bound. Thus, we will use (5.36) to estimate the genuine acceptance rate.

5.4.3 Change in the False Acceptance Rate

The false acceptance rate changes analogously.

For varying ω let

$$\mathrm{FAR}(\omega) = \mathbb{P}(\ \#(\mathbf{A} \cap \mathbf{B}) \geq \omega \mid \mathbf{A} \text{ and } \mathbf{B} \text{ are false matches }) \tag{5.37}$$

denote the chance for a non-genuine query template to overlap with the reference template in at least ω elements. Then the false acceptance rate fulfills

$$\mathrm{FAR} \geq \sum_{\omega \geq k} \epsilon(\omega, t, k, \mathcal{D}) \cdot (\mathrm{FAR}(\omega) - \mathrm{FAR}(\omega + 1)). \tag{5.38}$$

However, for the false acceptance rate it is safer to have an estimation as an upper bound.

Easily, an upper bound of the false acceptance rate can be deduced by observing that a necessary condition for a false accept is that the unlocking set contains at least k genuine points. Thus,

$$\mathrm{FAR} \leq \mathrm{FAR}(k). \tag{5.39}$$

However, this estimation is very coarse and it will allow us to report only quite high false acceptance rates.

To give a sharper upper bound, we first establish notation. Let $u(s)$ be the distribution for unlocking sets of size smaller than s. The size of an unlocking set is equals to the size of the corresponding query template $\mathbf{B}$ in our implementation. Thus for random query templates $\mathbf{B}$

$$u(s) = \mathbb{P}(\ \#\mathbf{B} \leq s\ ). \tag{5.40}$$

Furthermore, by $\mathrm{FAR}_s(\omega)$ denote the chance for two non-matching templates to contain at least ω elements given the unlocking set contains exactly $s \leq t$ elements, i.e.

$$\mathrm{FAR}_s(\omega) = \mathbb{P}(\ \#(\mathbf{A} \cap \mathbf{B}) \geq \omega \mid \mathbf{A} \text{ and } \mathbf{B} \text{ are false matches},\ \#\mathbf{B} = s\ ). \tag{5.41}$$

With this notation the false acceptance rate can be written as

$$\begin{aligned}\mathrm{FAR} &= \sum_{\omega\geq k}\sum_{s\geq\omega}\epsilon(\omega,s,k,\mathcal{D})\cdot(u(s)-u(s-1))\cdot(\mathrm{FAR}_s(\omega)-\mathrm{FAR}_s(\omega+1))\\ &\leq \sum_{\omega\geq k}\sum_{s\geq\omega}\epsilon(\omega,s,k,\mathcal{D})\cdot(u(s)-u(s-1))\cdot\mathrm{FAR}_s(\omega).\end{aligned} \tag{5.42}$$

We assume that

$$\mathrm{FAR}_s(\omega)\leq\mathrm{FAR}_t(\omega) \tag{5.43}$$

as $s<t$. In other words, the assumption states that the more elements an unlocking set contains the more genuine points it contains. Using this assumption it follows that

$$\begin{aligned}\mathrm{FAR} &\leq \sum_{\omega\geq k}\sum_{s\geq\omega}\epsilon(\omega,s,k,\mathcal{D})\cdot(u(s)-u(s-1))\cdot\mathrm{FAR}_t(\omega)\\ &= \sum_{\omega\geq k}\mathrm{FAR}_t(\omega)\cdot\sum_{s\geq\omega}\epsilon(\omega,s,k,\mathcal{D})\cdot(u(s)-u(s-1)).\end{aligned} \tag{5.44}$$

The above bound might exceed 1. But (5.39) gives an upper bound for the false acceptance rate in $[0,1]$. Thus, the false acceptance rate fulfills

$$\begin{aligned}\mathrm{FAR}\leq\min\{ &\sum_{\omega\geq k}\mathrm{FAR}_t(\omega)\cdot\sum_{s\geq\omega}\epsilon(\omega,s,k,\mathcal{D})\cdot(u(s)-u(s-1))\ ,\\ &\mathrm{FAR}(k)\ \}.\end{aligned} \tag{5.45}$$

Above, we derived a lower bound for the genuine acceptance rate and an upper bound for the false acceptance rate achievable using our proposed randomized decoder. We related these bounds to the genuine and false acceptance rates that would be achievable when the ordinary iterative decoder were used—although the ordinary iterative decoder would consume an unrealistic large amount of computer time for being implemented in practice. Therefore, we refer to these rates $\mathrm{GAR}(\omega)$ and $\mathrm{FAR}(\omega)$ as *hypothetical genuine acceptance rate* and *hypothetical false acceptance rate* as both reflect authentication rates. Below, for the performance evaluation using the randomized decoder we additionally measured the hypothetical authentication rates and used them to derive the bounds for the genuine and false acceptance rate.

Since the bound given in (5.45) can be used to estimate an upper bound for the false acceptance rate, it is useful to find a reasonable vault configuration that is based on the randomized decoder. In particular, it is useful to find a reasonable choice for the number of iterations in the randomized decoder, as we discuss in the following.

5.4.4 Configuration for the Randomized Decoder

As above, the vault's configuration consists of the parameters

- λ, which controls the number of grid points r,
- α determining how coarsely a minutia's direction is to be taken into account,

- t is the upper bound of the size of the genuine set, and
- k bounds the degree the secret polynomial can have.

Since we base our vault on the randomized decoder, there is a further parameter to be taken into account for vault configuration, namely

- $\mathcal{D}$, which is the number of decoding iterations.

To some extent, we are free in choosing $\mathcal{D}$, for it does not influence the complexity of the brute-force attack: The magnitude $\mathbf{bf}(n, t, k)$ is not affected by $\mathcal{D}$.

The genuine acceptance rate and the false acceptance rate, however, are influenced by $\mathcal{D}$: With the notation of Section 5.4.2 and Section 5.4.3 for the genuine acceptance rate GAR and the false acceptance rate FAR we have that

$$\begin{aligned} \mathrm{GAR} &\nearrow \mathrm{GAR}(k), \text{ and} \\ \mathrm{FAR} &\nearrow \mathrm{FAR}(k), \text{ respectively, as } \mathcal{D} \to \infty. \end{aligned} \tag{5.46}$$

In other words, the larger $\mathcal{D}$ the more GAR and FAR approach the hypothetical genuine acceptance rate GAR(k) and the hypothetical false acceptance rate FAR(k), respectively. This might have some impact for an attacker who is attempting to perform a false-accept attack: The attacker also is free to choose a bound of iterations such that he minimizes the time needed for a successful false-accept attack. On the other hand, the time for a single iteration in the false-accept attack increases as $\mathcal{D}$ increases. Thus, the attacker may choose the $\mathcal{D}$ that minimizes the overall time for a successful false-accept attack. Anyway, in Section 5.7.2 we investigate how the attacker may choose a $\mathcal{D}$ that yields the fastest false-accept attack.

5.5 Implementation

We implemented our proposed cross-matching resistant fuzzy vault for performing evaluations. The implementation was realized in the `C++` programming language (Stroustrup (2000)) and is part of an own software library that we call `thimble`. The main functionalities of `thimble` are described in the appendix in Section A on page 127. In the following, we give algorithmic descriptions on how we implemented vault construction and authentication.

5.5.1 Enrollment

On enrollment, a list of minutiae $\mathfrak{m}_1, \ldots, \mathfrak{m}_s$ and a secret encoded as a polynomial $f \in \mathbf{F}[X]$ of degree $< k$ are given. The minutiae are quantized using Algorithm 5.1.2 resulting in the reference template $\mathbf{A} \subset \mathbf{F}$. Using the modified fuzzy vault scheme (see Section 5.2) a monic polynomial $V \in \mathbf{F}[X]$ is constructed by binding f to the reference template $\mathbf{A}$. If $t' = |\mathbf{A}|$, then V has degree exactly t'. Moreover, the SHA-1 hash value $h(f)$ is stored publicly along with the vault to allow safe recovery.

In algorithmic description, the above can be described as follows.

Algorithm 5.5.1 (Enrollment)**.**

Input: A configuration as in Section 5.4.4; a minutiae template given as a list $\mathfrak{m}_1, \ldots, \mathfrak{m}_s$ sorted by their respective qualities; a secret encoded as a polynomial $f \in \mathbf{F}[X]$ of degree $< k$.

Output: A monic polynomial $V(X)$ of degree $\leq t$ and the hash value $h(f)$ or Failure.

```
1: /* extract template */
   use Algorithm 5.1.2 with input $\mathfrak{m}_1, \ldots, \mathfrak{m}_s$ and $t$ to obtain the discretized set $\mathbf{A} \subset \mathbf{F}$ containing at most $t$ quantized minutiae;

2: /* test if the template can encode the secret */
   $t' \leftarrow |\mathbf{A}|$;
   if $t' \leq k$
      return Failure;
   end

3: /* polynomial interpolation */
   determine the vault polynomial as described in Section 5.2, i.e.
      $V(X) \leftarrow f(X) + \prod_{x \in \mathbf{A}} (X - x);$      (5.47)

4: /* output */
   return $V$ and $h(f)$.
```

5.5.2 Authentication

On authentication, the query minutiae template is quantized as in (5.6). The vault polynomial V is evaluated on the quantized query minutiae to obtain the unlocking set $\mathbf{U} \subset \mathbf{F} \times \mathbf{F}$.

As the unlocking set $\mathbf{U}$ is extracted using the query template, we propose to apply the randomized decoder (see Section 5.4). It works by choosing *randomly* k different vault points from $\mathbf{U}$ that are used to determine its corresponding interpolation polynomial f^*. If $h(f^*) = h(f)$ then f^* is output and the query is accepted. Otherwise, we continue with a new iteration by randomly choosing a new subset containing k different points from $\mathbf{U}$. At most $\mathcal{D} = 2^{16}$ iterations are performed this way. If no polynomial f^* with $h(f^*) = h(f)$ is seen in an iteration, the query is rejected.

In algorithmic description, the randomized decoder can be formulated as follows.

Algorithm 5.5.2 (Randomized Decoder)**.**

Input: An unlocking set $\mathbf{U} \subset \mathbf{F} \times \mathbf{F}$; a hash-value $h(f)$; a bound on decoding iterations $\mathcal{D}$.

Output: A polynomial f^* with $h(f^*) = h(f)$ or Failure.

```
1: /* test */
   if $|\mathbf{U}| < k$
      return Failure;
   end
```

```
2: /* iterate */
   for $j = 1, \ldots, \mathcal{D}$

      choose $k$ points $(x_1, y_1), \ldots, (x_k, y_k) \in \mathbf{U}$ uniformly at random;

      determine a polynomial $f^*$ of degree $< k$ such that
      $f^*(x_1) = y_1, \ldots, f^*(x_k) = y_k$;

      if $h(f^*) = h(f)$
         return $f^*$;
      end

   end

3: /* the decoding was not successful */
   return FAILURE;
```

Consequently, the authentication procedure can be described as follows.

Algorithm 5.5.3 (Authentication)**.**

Input: A vault $(V(X), h(f))$ as output by Algorithm 5.5.1 and its configuration as in Section 5.4.4; a query minutiae template sorted increasingly by their qualities $\mathfrak{m}'_1, \ldots, \mathfrak{m}'_{s'}$, respectively, aligned to the vault $(V(X), h(f))$;
Output: Either a polynomial f^* with $h(f^*) = h(f)$ or FAILURE.

```
1: /* quantize minutiae */
   call Algorithm 5.1.2 with input $\mathfrak{m}'_1, \ldots, \mathfrak{m}'_{s'}$ and $t$ to obtain a set $\mathbf{B} \subset \mathbf{F}$ of size
   at most $t$ containing quantized minutiae encoded as finite field elements;

2: /* unlocking set */
   $\mathbf{U} \leftarrow \emptyset$;
   for all $x \in \mathbf{U}$
      $\mathbf{U} \leftarrow \mathbf{U} \cup \{(x, V(x))\}$;
   end

3: /* randomized decoding procedure */
   return the result of Algorithm 5.5.2 with input $\mathbf{U}$, $h(f)$, and $\mathcal{D}$.
```

5.6 Performance Evaluation

To evaluate the performance of our vault implementation, we conducted tests on the FVC 2002 DB2-A dataset (Maio et al. (2002)).

5.6.1 Evaluation Configuration

For our experiments we have chosen the configuration according to the result of our training (5.30) and for varying polynomial length k. Furthermore, we have chosen $\mathcal{D} = 2^{16}$, which corresponds to a number of decoding operations that is feasible to perform on current hardware. Therefore, the vault we used for evaluation was configured by the following parameters:

$$\begin{aligned} \lambda &= 22 \\ \alpha &= 2 \\ t &= 46 \\ \mathcal{D} &= 2^{16} \end{aligned} \tag{5.48}$$

where the length of the secret polynomial k is varying.

5.6.2 Evaluation Database

The FVC 2002 DB2-A database consists of a total of 800 images from 100 different fingers (index $1, \ldots, 100$). As in FVC 2002 DB2-B, 8 scans of each finger are contained in the database. Again, as for the training stage, a commercial extractor (Verifinger SDK 5.0 (Neurotechnology Ltd (2006))) has been utilized to obtain the corresponding minutiae templates sorted by their qualities, which are the very same templates that have been used for the performance evaluation of our reference implementation described in Section 3.4. Furthermore, we assume that for a genuine authentication attempt the query images are aligned to the vault. The way we align query templates to the vault is described in Section 3.4.4 on page 50. Moreover, the section also contains a discussion for the reasons why we investigate vault performance in a well-solved alignment framework.

5.6.3 Evaluation Protocol

For a given vault configuration, the way we conducted performance evaluation of our implementation follows exactly the description of Section 3.4.7, which corresponds to the FVC protocol (Maio et al. (2002)). Moreover, to compare the performance of our implementation with those of other implementations in the literature we also report rates on the sub-database whose templates have been extracted from fingerprint images of good quality. This corresponds to the description of Section 3.4.8. While the false acceptance rate (sample size is 4950) for both the entire database and the sub-database is the same, the genuine acceptance rates differ. For the entire database the genuine acceptance rate is estimated from 2800 samples while for the sub-database there are 100 samples. Although the genuine acceptance rate for the sub-database of good quality is statistical less significant, genuine acceptance rates have been reported for this set for virtually all implementations of the fuzzy fingerprint vault found in the literature (see Nagar et al. (2008, 2010); Nandakumar et al. (2007a,b); Uludag and Jain (2006)), while for only a few implementations there have been reports for the entire database (see Li et al. (2008)).

5.6.4 Prediction of Authentication Rates

For our implementation with the randomized decoder, using (5.36) and (5.45), we can predict lower bounds of the genuine acceptance rate and an upper bound of the false acceptance rate. This can be done without performing actual vault construction or vault decoding but only by performing a simulation in where we determine hypothetical authentication rates and the distribution of the size of the unlocking sets. But before applying the aforementioned bounds, we have to measure to hypothetical authentication rates.

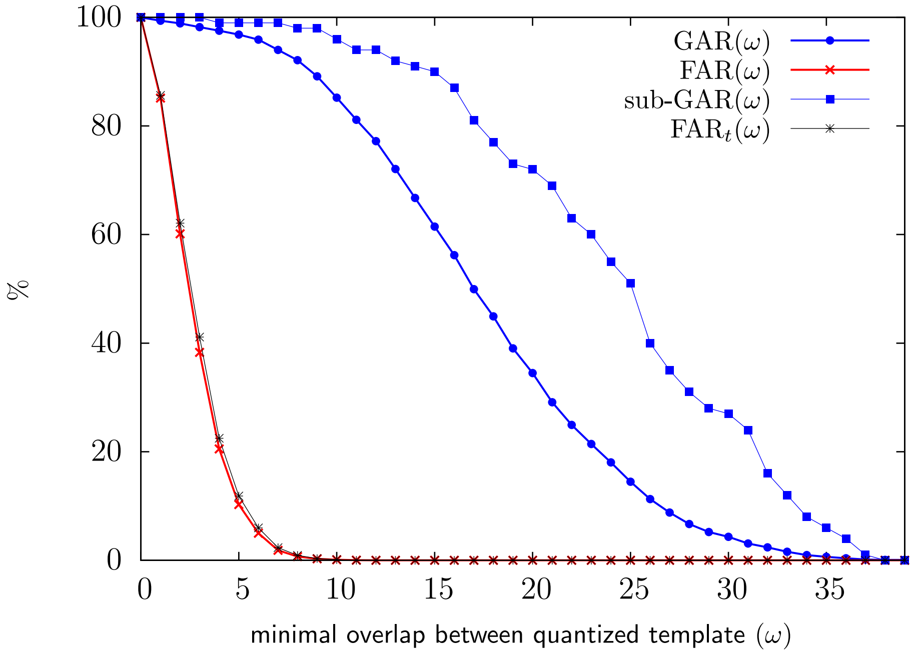


Figure 5.3: Hypothetical authentication rates $\mathrm{GAR}(\omega)$ and $\mathrm{FAR}(\omega)$ as in (5.35) and (5.37), respectively. These are probabilities for two templates to share at least ω elements. Note, these probabilities are no authentication rates that we achieved in our implementation due to a missing efficient decoder. Furthermore, sub-$\mathrm{GAR}(\omega)$ denotes the hypothetical genuine acceptance rate measured on the reduced FVC 2002 DB2 consisting of the first two impressions per finger, only. $\mathrm{FAR}_t(\omega)$ indicates the upper bound for the hypothetical false acceptance rate where $t = 46$.

Hypothetical Authentication Rates

Hypothetically, if the implementation with the ordinary iterative decoder were used, the genuine acceptance rate $\mathrm{GAR}(\omega)$ and false acceptance rate $\mathrm{FAR}(\omega)$ can be determined for varying ω by counting how many elements corresponding minutiae quantization sets share. As already pointed out in Section 5.3.3, it is not necessary to perform actual vault construction or decoding to determine these rates. These rates can be efficiently be deduced by counting common elements of quantized templates on an authentication attempt.

For varying $\omega = 0, \ldots, 46$, we have determined the hypothetical genuine acceptance rate $\mathrm{GAR}(\omega)$, the hypothetical false acceptance rate $\mathrm{FAR}(\omega)$, the hypothetical genuine acceptance rate measured on the reduced database sub-$\mathrm{GAR}(\omega)$ (see Section 3.4.8), and the upper bound for the hypothetical false acceptance rate $\mathrm{FAR}_t(\omega)$ as it is used to predict an upper bound for the false acceptance rate of our implementation (Section 5.5).

According to the FVC protocol, $\mathrm{GAR}(\omega)$ has been determined from 2800 samples while sub-$\mathrm{GAR}(\omega)$ is based on only 100 observations. $\mathrm{FAR}(\omega)$ are results from 4950 tests. The upper bound $\mathrm{FAR}_t(\omega)$ (where $t = 46$) is the false acceptance rate under the condition that the unlocking sets are of size exactly $t = 46$. Among 4950 unlocking sets 3049 unlocking sets were of size $t = 46$. The determined rates are plotted in Figure 5.3. In addition, their values are listed in Table B.2 on page 148 in the appendix.

Although authentication rates look promising, a systematic decoder, however, would require an amount of computational cost which would be unacceptable (see Section 5.4).

Distribution of the Size of Unlocking Sets

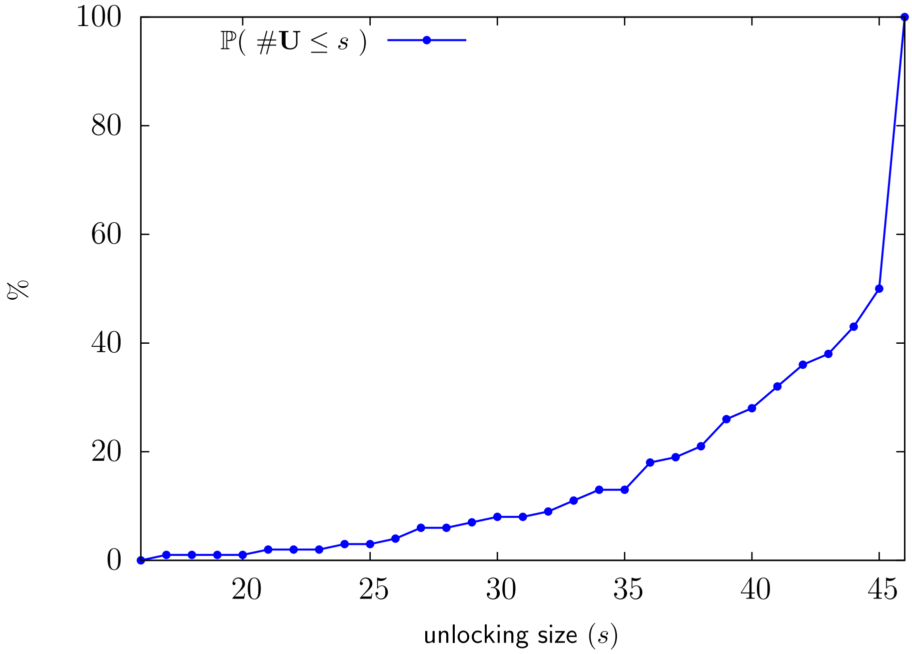


Figure 5.4: Distribution function on the size of unlocking sets, i.e. $u(s)$ as in (5.40)

To predict upper bounds for the false acceptance rate of our implementation using (5.45) we need to determine the distribution function of sizes of unlocking sets, i.e.

$$u(s) = \mathbb{P}(\ \#\mathbf{U} \leq s\ ). \tag{5.49}$$

To determine the distribution, we only used the first impression of each finger $i = 1, \ldots, 100$ of the FVC 2002 DB2-A database. Each corresponding minutiae template was input to Algorithm 5.1.2 to obtain a sample of quantized minutiae. Since the size of these templates, if they are used as a query, agree with the corresponding size of the unlocking set, we inserted the size of the query templates into an increasingly sorted vector. After this has been done for all 100 fingers, the final vector was used to determine $u(s)$ for each $s = 0, \ldots, 46$.

For a particular s, we counted how many elements of the sorted vector were smaller than or equals to s. This number divided by 100 yielded our estimation for $u(s)$.

The results for $u(s)$ are plotted in Figure 5.4. Moreover, the estimated distributions $u(s)$ for $s = 0, \ldots, 46$ are listed in Table B.1 on page 147 in the appendix.

Predicted Bounds

To predict bounds of our implementation's authentication rates, we may use the hypothetical authentication rates as well as the distribution of the unlocking set sizes from above. This enables us to use (5.36) and (5.45) to estimate a lower bound of the genuine acceptance rate as well as to estimate an upper bound for the false acceptance rate.

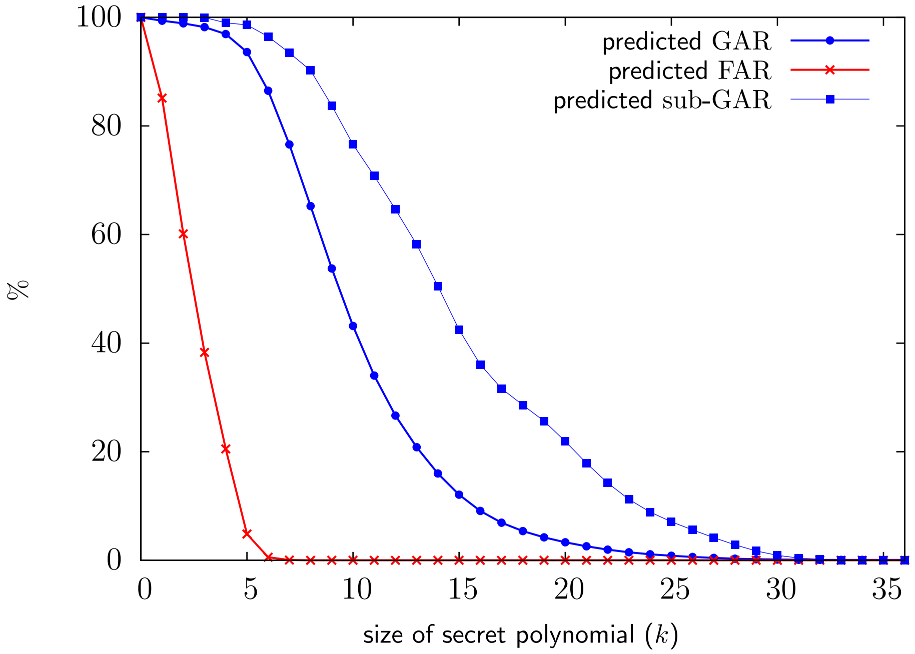


Figure 5.5: Predicted genuine acceptance rate GAR and false acceptance rate FAR using (5.36) and (5.45), respectively; sub-GAR denotes the prediction for the genuine acceptance rate on the reduced FVC 2002 DB2 consisting of the first two impressions per finger, only.

For example, with the configuration as in Section 5.6.1 where $k = 8$ using (5.36) and (5.45) we can expect that

$$\begin{aligned} &\text{GAR} > 65.24\% \text{ and} \\ &\text{FAR} < 2.45 \cdot 10^{-5}. \end{aligned} \tag{5.50}$$

If the reduced FVC 2002 DB2-A database consisting of the first two impressions for each finger is considered, we plug the estimated sub-GAR(ω) instead of GAR(ω) into (5.36) to predict the lower bound of the genuine acceptance rate. Then

$$\text{sub-GAR} > 90.22\%. \tag{5.51}$$

For varying k, corresponding plots for predicted bounds on the genuine acceptance rates and on the false acceptance rate are given in Figure 5.5. Moreover, the predicted bounds are listed in Table B.3 on page 149 in the appendix.

5.6.5 Authentication Performance

In addition to predicted authentication rates and hypothetical authentication rates, we have evaluated the performance of our proposed fingerprint vault implementation by performing actual vault construction (see Section 5.5.1) and actual vault decoding (see Section 5.5.2). By actually performing these steps, we determined genuine acceptance rate as well as false acceptance rate according to the FVC protocol (see Section 5.6.3). Furthermore, decoding timings have been determined.

For example, for the evaluation configuration (see Section 5.6.1) and secret polynomial lengths up to $k = 8$ the genuine acceptance rate and the false acceptance rate were determined as

$$\begin{aligned} &\text{GAR} \approx 77.14\% \ (> 65.25\%) \text{ and} \\ &\text{FAR} = 0\% \ (< 2.45 \cdot 10^{-5}), \end{aligned} \tag{5.52}$$

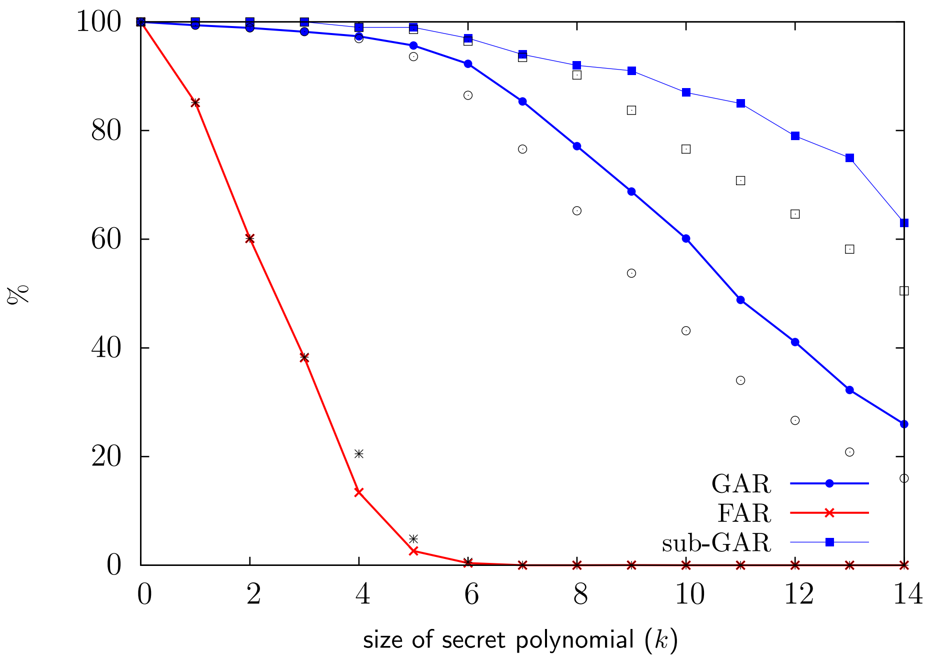


Figure 5.6: Genuine acceptance rate GAR and false acceptance rate FAR of our vault implementation using the configuration of Section 5.6.1; sub-GAR denotes the genuine acceptance rate measured on the reduced FVC 2002 DB2 consisting of the first two impressions per finger, only; the rates that have been predicted only from the hypothetical authentication rates are indicated, correspondingly.

Table 5.2: Authentication performances for varying k of our cross-matching resistant minutiae fuzzy vault implementation (see Section 5.5) evaluated on the FVC 2002 DB2-A database; the rates in brackets correspond to the bounds predicted in Section 5.6.4.

size of secret polynomial k	genuine acceptance rate GAR	genuine acceptance rate on reduced dataset of scans of good quality sub-GAR	false acceptance rate FAR
0	$= 100\%$ $(= 100\%)$	$= 100\%$ $(= 100\%)$	$= 100\%$ $(= 100\%)$
1	$\approx 99.39\%$ $(> 99.39\%)$	$= 100\%$ $(= 100\%)$	$\approx 85.13\%$ $(< 85.14\%)$
2	$\approx 98.89\%$ $(> 98.89\%)$	$= 100\%$ $(= 100\%)$	$\approx 60.12\%$ $(< 60.13\%)$
3	$\approx 98.17\%$ $(> 98.2\%)$	$= 100\%$ $(> 99.98\%)$	$\approx 38.18\%$ $(< 38.31\%)$
4	$\approx 97.34\%$ $(> 96.92\%)$	$= 99\%$ $(> 98.99\%)$	$\approx 13.39\%$ $(< 20.51\%)$
5	$\approx 95.64\%$ $(> 93.6\%)$	$= 99\%$ $(> 98.26\%)$	$\approx 2.62\%$ $(< 4.84\%)$
6	$\approx 92.29\%$ $(> 86.49\%)$	$= 97\%$ $(> 96.47\%)$	$\approx 0.38\%$ $(< 0.57\%)$
7	$\approx 85.36\%$ $(> 76.59\%)$	$= 94\%$ $(> 93.46\%)$	$= 0\%$ $(< 0.05\%)$
8	$\approx 77.14\%$ $(> 65.25\%)$	$= 92\%$ $(> 90.22\%)$	$= 0\%$ $(< 2.45 \cdot 10^{-5})$
9	$= 68.75\%$ $(> 54.75\%)$	$= 91\%$ $(> 83.74\%)$	$\approx 0.02\%$ $(< 9.76 \cdot 10^{-7})$
10	$\approx 60.14\%$ $(> 43.15\%)$	$= 87\%$ $(> 76.58\%)$	$= 0\%$ $(< 2.11 \cdot 10^{-8})$
11	$\approx 48.79\%$ $(> 34.01\%)$	$= 85\%$ $(> 70.79\%)$	$= 0\%$ $(= 0\%)$
12	$\approx 41.07\%$ $(> 26.65\%)$	$= 79\%$ $(> 64.61\%)$	$= 0\%$ $(= 0\%)$
13	$\approx 32.29\%$ $(> 20.83\%)$	$= 75\%$ $(> 58.16\%)$	$= 0\%$ $(= 0\%)$
14	$\approx 25.96\%$ $(> 15.99\%)$	$= 63\%$ $(> 50.50\%)$	$= 0\%$ $(= 0\%)$

Table 5.3: Average decoding timings for varying k of our cross-matching resistant minutiae fuzzy vault implementation (see Section 5.5) evaluated on the FVC 2002 DB2-A database; the timings have been measured on a single core of an `AMD Phenom(tm) II X4 955`.

size of secret polynomial k	average genuine decoding time GDT	average genuine decoding time on reduced dataset of scans of good quality GDT	average impostor decoding time IDT
0	$\approx 0.001s$	$\approx 0.0007s$	$\approx 0.001s$
1	$\approx 0.001s$	$\approx 0.0015s$	$\approx 0.01s$
2	≈ 0.002	$\approx 0.0013s$	$\approx 0.04s$
3	≈ 0.003	$\approx 0.0018s$	$\approx 0.09s$
4	$\approx 0.007s$	$\approx 0.0045s$	$\approx 0.17s$
5	$\approx 0.015s$	$\approx 0.0061s$	$\approx 0.24s$
6	$\approx 0.032s$	$\approx 0.012s$	$\approx 0.29s$
7	$\approx 0.067s$	$\approx 0.026s$	$\approx 0.36s$
8	$\approx 0.14s$	$\approx 0.053s$	$\approx 0.50s$
9	$\approx 0.24s$	$\approx 0.084s$	$\approx 0.63s$
10	$\approx 0.31s$	$\approx 0.11s$	$\approx 0.65s$
11	$\approx 0.47s$	$\approx 0.17s$	$\approx 0.82s$
12	$\approx 0.58s$	$\approx 0.25s$	$\approx 0.89s$
13	$\approx 0.70s$	$\approx 0.32s$	$\approx 0.95s$
14	$\approx 0.81s$	$\approx 0.48s$	$\approx 1.04s$

respectively, on the entire FVC 2002 DB2-A database. The values in brackets correspond to the respective predicted rate. On the reduced database consisting of the first two impressions per finger only the genuine acceptance rate was determined as

$$\text{sub-GAR} = 92\% \ (> 90.22\%). \tag{5.53}$$

The average decoding time, on a single core of an `AMD Phenom(tm) II X4 955`, of a genuine user before he is either accepted or rejected has been measured as

$$\text{GDT} \approx 140 \ ms \tag{5.54}$$

while an impostor has to wait

$$\text{IDT} \approx 500 \ ms \tag{5.55}$$

in average. Moreover, on the reduced database the average time a genuine user must wait before being authenticated or rejected was determined as

$$\text{GDT} \approx 53 \ ms \tag{5.56}$$

We determined corresponding rates for each secret size $k = 0, \ldots, 14$. The authentication rates are plotted in Figure 5.6 and can additionally be found in Table 5.2. The decoding timings are listed in Table 5.3.

5.7 Security Analysis and Evaluation

In this section, we evaluate the resistance of our implementation against the attacks of Chapter 4.

5.7.1 Brute-Force Attack

Table 5.4: Brute-force attack performance against our cross-matching resistant minutiae fuzzy vault implementation with an `AMD Phenom(tm) II X4 955` using all four processor cores in parallel

size of secret polynomial k	security in bits	brute-force iterations per second per core	**expected time before successful iteration**
3	≈ 15	$343,643$	$\approx$ 23 milliseconds
4	≈ 20	$271,739$	$\approx$ 1 second
5	≈ 25	$215,983$	$\approx$ 54 seconds
6	≈ 31	$180,180$	$\approx$ 43 minutes
7	≈ 36	$147,059$	$\approx$ 35 hours
8	≈ 41	$127,389$	$\approx$ 2 months
9	≈ 47	$105,820$	$\approx$ 10 years
10	≈ 52	$95,328.9$	$\approx$ 472 years
11	≈ 58	$83,194.7$	$\approx 24,177$ years
12	≈ 63	$74,571.2$	$\approx 1.24 \cdot 10^6$ years
13	≈ 69	$65,919.6$	$\approx 6.63 \cdot 10^7$ years
14	≈ 75	$58,445.4$	$\approx 3.64 \cdot 10^9$ years

In this section, we report average attack timings against our cross-matching resistant minutiae fuzzy vault implementation analogous to Section 4.1.[4]

For example, if a vault of configuration given in Section 5.6.1 with $k = 8$ was intercepted, the difficulty in guessing $k = 8$ genuine vault points is

$$\mathbf{bf}(1620, 46, 8) \approx 4.43 \cdot 10^{12} \approx 2^{42}, \tag{5.57}$$

Thus, an attacker can expect to successfully break a vault hiding $t = 46$ genuine points within

$$\mathcal{I} \approx 3.07 \cdot 10^{12} \approx 2^{41} \tag{5.58}$$

iterations. Analogous to the experiments in Section 4.1.2, on a single core of an `AMD Phenom(tm) II X4 955`, we empirically determined that within one second it is possible to make

$$127,389 \tag{5.59}$$

guesses for $k = 8$ genuine vault points, determine its interpolation polynomial, compute its SHA-1 hash-value, and to compare it with the stored hash of the correct polynomial. Thus, if all four cores were used, an attacker can expect to successfully break the vault after approximately 2 months ($\approx$ 70 days). For varying $k = 0, \ldots, 14$ corresponding expected attack timings are listed in Table 5.4.

---

[4] For a given configuration we randomly generated 10 vaults of corresponding parameters. For each vault, the time needed for the first $100,000$ iterations of the brute-force was measured.

We see that our implementation provides good security against naive brute-force attacks. However, we have seen during Section 4.3 that in comparison to the false-accept attack, which could be conducted much more efficiently, the security against the brute-force attack is yet of minor relevance.

5.7.2 False-Accept Attack at a First Glance

To evaluate the security of our implementation against the false-accept attack we could follow the way as in Section 4.3: For a given configuration, let FAR be its false acceptance rate and let IDT denote its average impostor decoding time. Then the expected time needed for a successful false-accept attack estimates as

$$\frac{\log(0.5)}{\log(1-\mathrm{FAR})}\cdot \mathrm{IDT}. \tag{5.60}$$

For example, for the configuration we used throughout for examples (see Section 5.6.1), where $k = 8$ then FAR is between 0% and 0.06% with a confidence of 95% by the rule of three (see Section 2.2.6). Furthermore, $\mathrm{IDT} = 0.5s$. The time needed for a successful false-accept attack thus is at least 571.67 s. If all four processor cores were used in parallel, the false-accept attack would consume at least $142.92s$, which corresponds to 2–3 minutes. In comparison to the brute-force attack, which would require approximately 2 months on the same processor, an attacker may prefer to run the false-accept attack. For varying k, corresponding times can be found in Table 5.5.
Against the false-accept attack our implementation does not provide significant improvement in comparison to other implementations. This highly advocates that a single finger is not sufficient for cryptographic security. Rather fuzzy vaults protecting multiple finger should be investigated to achieve high resistance against the false-accept attack.

In the above evaluation we assumed that the attacker who is performing a false-accept attack uses the same decoder as used by the system for authorizing users. But in fact, the attacker can use any decoder that works best for his purposes. In particular, for each impostor authentication attempt that the attacker simulates he may choose a number of decoding iterations $\mathcal{D}$ for that the expected number of time consumed by the false-accept is minimized while in the evaluation above we assumed that the attacker uses $\mathcal{D} = 2^{16}$.

5.7.3 False-Accept Attack at a Second Glance

In the above evaluation, we assumed that the attacker also uses $\mathcal{D} = 2^{16}$ for simulating an impostor authentication attempt. But actually, he is free in choosing whichever $\mathcal{D}$ he wants. For simplicity, we assume that the time needed for the attacker to build the unlocking set on a simulated impostor authentication attempt is negligible. Moreover, we assume that the attacker is in the possession of an overwhelming large database containing real minutiae templates. This corresponds to an ideal situation for the attacker. Therefore, we assume that the time for a successful false-accept attack using the randomized decoder is

$$\frac{\log(0.5)}{\log(1-\mathrm{FAR})}\cdot \mathcal{D}\cdot \ell \tag{5.61}$$

Table 5.5: False-accept attack performance with $\mathcal{D} = 2^{16}$ against our cross-matching resistant minutiae fuzzy vault implementation with an `AMD Phenom(tm) II X4 955` using all four processor cores in parallel

size of secret polynomial k	number of false accepts within 4950 samples	interval for the false acceptance rate	average impostor decoding time on a single core IDT	**estimated time for a successful false-accept attack**
1	4214	84.11%–86.11%	$\approx 0.01s$	3.5–3.8 milliseconds
2	2976	58.74%–61.49%	$\approx 0.04s$	29–31 milliseconds
3	1890	36.83%–39.55%	$\approx 0.09s$	124–136 milliseconds
4	663	12.46%–14.37%	$\approx 0.17s$	759–886 milliseconds
5	130	2.2%–3.11%	$\approx 0.24s$	5.26–7.48 seconds
6	19	0.23%–0.6%	$\approx 0.29s$	33.5–86.8 seconds
7	0	0%–0.06%	$\approx 0.36s$	$>$ 102 seconds
8	0	0%–0.06%	$\approx 0.5s$	$>$ 142 seconds
9	1	0.0005%–0.11%	$\approx 0.63s$	$\approx$ 97 seconds – 6 hours
10	0	0%–0.06%	$\approx 0.65s$	$>$ 3 minutes

where ℓ corresponds to the time for 1) selecting k unlocking points, 2) determine their interpolation polynomial, 3) compute the polynomial's hash value, and 4) compare the hash value with the hash value of the correct polynomial. We assume that the factor ℓ is constant. Thus, it can be omitted if for different number of decoding iterations the expected time for a successful false-accept attack are compared.

We have the following conjecture.

Hypothesis 5.7.1. *For a realistic implementation using our vault construction, the expected time for a successful false-accept is minimized if the attacker chooses* $\mathcal{D} = 1$.

To see that the hypothesis is a realistic assumption, we derive a property for the false-acceptance rate as a function in $\mathcal{D}$ such that the expected time for a successful false-accept (also a function in $\mathcal{D}$) is minimized at $\mathcal{D}$. Therefore, we rename the variable $\mathcal{D}$ as x and write $v(x) = \mathrm{FAR}$ as the false-acceptance rate as a function x. Note that v is an increasing function that varies in $(0, v^*)$ where by $v^* \ll 1$ we denote the hypothetical false-acceptance rate $\mathrm{FAR}(k)$, i.e. $v(x) \to \mathrm{FAR}(k)$ as $x \to \infty$ (see Equation (5.46) on page 104). Furthermore, let

$$L(x) = \frac{\log(0.5)}{\log(1 - v(x))} \cdot x \tag{5.62}$$

be the expected time (i.e. number of iterations each consuming constantly ℓ operations) for a successful false-accept attack.

Proposition 5.7.2. *If for all $x \geq 1$ the bound*

$$v(x) \leq 1 - (1 - v(1))^x \tag{5.63}$$

is fulfilled then L is minimal at $x = 1$.

Proof. First note that L is minimal if and only if

$$\tilde{L}(x) = \frac{x}{\log(1 - v(x))}, \tag{5.64}$$

which is negative for $x \geq 1$, is maximal. Assume by contradiction, there is an $x > 1$ such that

$$\tilde{L}(1) < \tilde{L}(x). \tag{5.65}$$

Then (note that the logs are negative)

$$\begin{aligned}
& \frac{1}{\log(1 - v(1))} < \frac{x}{\log(1 - v(x))} \\
\Rightarrow\quad & \frac{\log(1 - v(x))}{\log(1 - v(1))} > x \\
\Rightarrow\quad & \log(1 - v(x)) < x \cdot \log(1 - v(1)) \\
\Rightarrow\quad & \exp(\log(1 - v(x))) < \exp(x \cdot \log(1 - v(1))) \\
\Rightarrow\quad & 1 - v(x) < (1 - v(1))^x \\
\Rightarrow\quad & -v(x) < (1 - v(1))^x - 1 \\
\Rightarrow\quad & v(x) > 1 - (1 - v(1))^x.
\end{aligned} \tag{5.66}$$

This, however, would contradict our assumption (5.63). Thus the statement of the proposition is true. □

It is my belief that for a realistic implementation the false-acceptance rate as a function of the decoding iteration is bounded as required by the above proposition. Unfortunately, I was not yet able to derive a simple criterion that is 1) easily fulfilled, 2) one can easily verify it for an implementation, and 3) that implies (5.63).

I tried to derive such a criterion based on the following observations. Using the arguments of Section 5.4.3, we see that false acceptance rate can be written as the finite sum

$$\begin{aligned}
v(x) = & \sum_{\omega = k,\ldots,t} \\
& \sum_{s = \omega,\ldots,t} \epsilon(\omega, s, k, x) \cdot (u(s) - u(s-1)) \cdot (\mathrm{FAR}_s(\omega) - \mathrm{FAR}_s(\omega + 1))
\end{aligned} \tag{5.67}$$

where $u(s)$ is the distribution of unlocking sets of size $\leq s$, $\mathrm{FAR}_s(\omega)$ denotes the hypothetical false-acceptance rate given the size of the unlocking set is of size s, and

$$\epsilon(\omega, s, k, x) = 1 - (1 - \mathbf{bf}(s, \omega, k)^{-1})^x \tag{5.68}$$

where

$$\mathbf{bf}(s, \omega, k) = \binom{s}{k} \cdot \binom{\omega}{k}^{-1}. \tag{5.69}$$

To simplify notation, we write

$$\begin{aligned} b_{s,\omega} &= 1 - \mathbf{bf}(s,\omega,k) \qquad \text{and} \\ c_{s,\omega} &= (u(s) - u(s-1)) \cdot (\mathrm{FAR}_s(\omega) - \mathrm{FAR}_s(\omega+1)). \end{aligned} \tag{5.70}$$

Then

$$v(x) = \sum_{s,\omega} c_{s,\omega} \cdot (1 - b_{s,\omega}^x). \tag{5.71}$$

At this point I got stuck. Although the above terms look similar to what is required by Proposition 5.7.2, I was not able to derive a simple bound for $v(1)$ or the coefficients $c_{s,\omega}$ that are easily fulfilled and that can be easily verified such that $v(x) \leq 1-(1-v(1))^x$. However, it is my belief that $v(x) \leq 1-(1-v(1))^x$. In fact, as $x \to \infty$ the false acceptance rate approaches a value, which is $\mathrm{FAR}(k)$, that is much smaller than 1 for realistic vault parameters as confirmed within our performance evaluation. On the other hand the right-handed expression of the assumption 5.63 approaches 1 as $x \to \infty$. For $\mathrm{FAR}(k) \ll 1$ it is my belief that in fact Hypothesis 5.7.1 holds. Moreover, the following evaluation supports this conjecture.

Evaluation of the False-Accept Attack using a Single Decoding Iteration

Within our performance evaluation reported in Section 5.6 we also experimentally observed for each impostor authentication attempt, how many genuine vault points the corresponding unlocking set contained.

Assume the first impression of the ith finger of the FVC 2002 DB2 database (see Section 3.4.7) was used for vault construction and the first impression of the jth finger was used as a query. Assume the corresponding unlocking set $\mathbf{U}$ was of size s and it contained ω genuine points. For a fixed k let

$$p_{i,j} = \begin{cases} \mathbf{bf}(s,\omega,k)^{-1}, & \omega \geq k \\ 0, & \omega < k \end{cases} \tag{5.72}$$

denote the chance to select k genuine unlocking points. We model the false acceptance rate as

$$v(1) = \mathrm{FAR} = \frac{1}{4,950} \cdot \sum_{\substack{i=1,\ldots,99 \\ j=i+1,\ldots,100}} p_{i,j}. \tag{5.73}$$

We measured the time for a million 1) choices of k unlocking points, 2) determination of their interpolation polynomial f^*, 3) computation of its SHA-1 hash value $h(f^*)$, and 4) comparison of $h(f^*)$ with a preliminary randomly chosen 160-bit hash value h^*. Thereby, we preliminary generated a random set of $\mathbf{U} \subset \mathbf{F} \times \mathbf{F}$. Essentially, the time that we determined this way is $10^6 \cdot \ell$ (see Equation (5.61)).

For example, if $k = 6$ using (5.73) we estimated the false acceptance rate as

$$v(1) \approx 7.84 \cdot 10^{-8}. \tag{5.74}$$

Furthermore, a million decoding iterations have been measured as

$$10^6 \cdot \ell = 4.47s \tag{5.75}$$

on a single core of an `AMD Phenom(tm) II X4 955`. Correspondingly, the expected time for a successful false-accept attack is

$$\frac{\log(0.5)}{\log(1-v(1))}\cdot\ell\approx 39.52s. \tag{5.76}$$

If all four processor cores were used in parallel[5] the time for a successful false accept becomes approximately $9.88s$. In comparison to the false-accept attack with $\mathcal{D}=2^{16}$ decoding iterations (in where the time for a successful was estimated to last between 33.5–86.8s) our analysis for $\mathcal{D}=1$ yields a more efficient false-accept attack.

Of course, if the attacker is only in the possession of a small attack database, he may run more than one decoding iterations to decrease the chance that he will be unsuccessful with only simulating a few impostor authentication attempts. But analyses of the false-accept should be performed by assuming ideal conditions for the attacker in where he has access to arbitrarily many pre-processed minutiae templates.

For varying k, expected timings for a successful false-accept attack using a single decoding iteration are listed in Table 5.6. Note that, some of the estimated timings in Table 5.5 are smaller than in 5.6 which may be an effect caused by our randomized experiments. In any case, we may not conclude that Hypothesis 5.7.1 is not true.

For our evaluation of the false-accept using $\mathcal{D}=2^{16}$, we estimated a range of the false acceptance rate for a confidence level of 95% using the Clopper-Pearson confidence interval (see Section 2.2.6 on page 21). In the evaluation of the false-accept attack for $\mathcal{D}=1$, we estimated the false acceptance rate in another way using (5.73). This better involves the structure of the false acceptance rate and thus we can estimate the false acceptance rate even if for a performance evaluation using $\mathcal{D}=1$ no false-accept would occur.

For example, in Table 5.6 where $k=11$ the false acceptance rate was yielded as follows. For only one impostor authentication attempt it was observed that an unlocking set of size 45 contained exactly $\omega=11$ genuine vault points. Using (5.73) this yields

$$v(1)=\frac{1}{4,950}\cdot\binom{11}{11}\cdot\binom{45}{11}^{-1}\approx 1.99\cdot 10^{-14}. \tag{5.77}$$

However, the problem of non-confident estimations also concerns our evaluation. Therefore, an expected time for a successful false-accept attack of 20 days or 3 years for $k=10$ or $k=11$, respectively, should be taken with caution and we may not conclude that the need for multiple finger fuzzy vault does not exist anymore.

5.7.4 Correlation Attack and Cross-Matching

It is not necessary to evaluate our implementation's resistance against the correlation attack adopting the evaluations of Section 4.4.7 on page 80. An intruder who has intercepted multiple vault records protecting minutiae templates extracted from the same finger will apparently not be able to filter out chaff points by correlating the vaults. The reason is that chaff points and genuine points are encoded by a

---

[5] Note that a false-accept attack can be parallelized by running multiple false-accept attacks each using a different query templates.

Table 5.6: False-accept attack performance with $\mathcal{D} = 1$ against our cross-matching resistant minutiae fuzzy vault implementation with an `AMD Phenom(tm) II X4 955` using all four processor cores in parallel

length of secret polynomial k	false acceptance rate for a single decoding iteration $\mathrm{FAR} = v(1)$	time for a million impostor decoding attempts $10^6 \cdot \ell$	**estimated time for a successful false-accept attack**
1	$\approx 5.11\%$	$\approx 1.1s$	$\approx 3.63 \cdot 10^{-3}$ milliseconds
2	$\approx 0.3\%$	$\approx 1.5s$	$\approx 8.56 \cdot 10^{-2}$ milliseconds
3	$\approx 0.02\%$	$\approx 2.06s$	≈ 1.78 milliseconds
4	$\approx 1.42 \cdot 10^{-5}$	$\approx 2.78s$	≈ 33.81 milliseconds
5	$\approx 1.06 \cdot 10^{-6}$	$\approx 3.57s$	≈ 586 milliseconds
6	$\approx 7.84 \cdot 10^{-8}$	$\approx 4.47s$	≈ 9.88 seconds
7	$\approx 5.68 \cdot 10^{-9}$	$\approx 5.47s$	≈ 167 seconds
8	$\approx 3.85 \cdot 10^{-10}$	$\approx 6.62s$	≈ 50 minutes
9	$\approx 2.18 \cdot 10^{-11}$	$\approx 7.8s$	≈ 17 hours
10	$\approx 8.95 \cdot 10^{-13}$	$\approx 9.13s$	≈ 20 days
11	$\approx 1.99 \cdot 10^{-14}$	$\approx 10.78s$	≈ 3 years

feature set which is equal for different vault instances (see Figure 5.1 on page 91). As a consequence, via correlation, an attacker will neither be successful to cross-match nor he will be able to filter out any chaff points to ease brute-force attacks.

Cross-Matching of Modified Vault Records

For the basic vault construction in where the ordinate values of the rigid chaff features are generated randomly, we do not see an efficient way to even cross-match. But if the modified vault construction by Dodis et al. (2008) (see Section 5.2) is used, there might be a chance for an intruder to determine whether to vault records protect templates from the same biometric source.

To see this, assume that a user is enrolled using the same template $\mathbf{A} \subset \mathbf{E} \subset \mathbf{F}$ of size t for two applications.[6] Let

$$\chi_{\mathbf{A}}(X) = \prod_{x \in \mathbf{A}} (X - x) = \chi_0 + \chi_1 \cdot X + \ldots + \chi_t \cdot X^t \tag{5.78}$$

be the *characteristic polynomial of* $\mathbf{A}$. For the first application a secret polynomial

$$f(X) = f_0 + f_1 \cdot X + \ldots + f_{k-1} \cdot X^{k-1} \tag{5.79}$$

---

[6] Note, $\mathbf{E}$ is the subset of the finite field $\mathbf{F}$ in where the template features are encoded (see Equation (5.2) on page 90).

is generated randomly and the vault is published as the polynomial

$$\begin{aligned} V(X) =& f(X) + \chi_{\mathbf{A}}(X) \\ =& \chi_t \cdot X^t + \ldots + \chi_k \cdot X^k + \sum_{i=0}^{k-1} (f_i + \chi_i) \cdot X^i. \end{aligned} \tag{5.80}$$

For the second application another secret polynomial

$$g(X) = g_0 + g_1 \cdot X + \ldots + g_{k-1} \cdot X^{k-1} \tag{5.81}$$

is generated and the vault polynomial is

$$\begin{aligned} W(X) =& g(X) + \chi_{\mathbf{A}}(X) \\ =& \chi_t \cdot X^t + \ldots + \chi_k \cdot X^k + \sum_{i=0}^{k-1} (g_i + \chi_i) \cdot X^i. \end{aligned} \tag{5.82}$$

Now, an intruder who has intercepted the polynomials V and W may observe that the upper $t-k$ coefficients of V and W are equal if both protect the same template $\mathbf{A}$. Note, the lower k coefficients appear random as the polynomials f and g have been generated randomly. We see that the upper $t-k$ only depend on the template $\mathbf{A}$. Therefore, an intruder may cross-match using the upper $t-k$ coefficients of the polynomials V and W — although we do not see an efficient approach for the attacker to attack the vaults via record multiplicity. The risk of cross-matching, however, must be prevented.

Hardening Cross-Matching of Modified Vault Records

The mechanism we propose to harden cross-matching of modified vault records is adopted from the idea of Kelkboom et al. (2011) who proposed to incorporate a random bit-permutation process to prevent the fuzzy commitment scheme to become vulnerable against cross-matching via the decodability attack.

Assume an individual public permutation is associated with each application. Therefore, assume $P : \mathbf{E} \to \mathbf{E}$ to be an application's individual permutation. Instead of protecting $\mathbf{A}$ with the modified vault construction directly, its permutation is protected. Therefore, let

$$P(\mathbf{A}) = \{ \ P(x) \mid x \in \mathbf{A} \ \}. \tag{5.83}$$

Let

$$\chi_{P(\mathbf{A})}(X) = \prod_{x \in \mathbf{A}} (X - P(x)). \tag{5.84}$$

If f is the secret polynomial of degree $< k$ then the vault consists of the polynomial

$$V(X) = f(X) + \chi_{P(\mathbf{A})}(X). \tag{5.85}$$

Using a public permutation this way (similar to a password salt), the possibility for the attacker to cross-match vaults may be effectively hardened.

It would be interesting to mathematically analyze how much knowledge about the templates an intruder can gain given multiple modified vault records protecting templates generated from the same biometric source. In any case, if the risk for cross-matching or attack via record multiplicity remains too high we can still use the ordinary basic vault construction — although this requires more storage size for the vault records. In this thesis we just draw attention to these open topics but do not tackle them.

5.8 Comparison with other Implementations

In this section, we compare our implementation that we developed in this chapter with other implementations of the fuzzy fingerprint vault found in the literature. For comparison we use the rates that have been evaluated empirically in the tests of Section 5.6.5. Moreover, the common dataset we use for comparison is the reduced FVC 2002 DB2-A database consisting of the first two impressions per finger only (see Section 3.4.8) as this is a dataset publicly available for that authentication rates have been reported for most fuzzy fingerprint vault implementations found in literature.

A major advantage of our finger fuzzy vault implementation is given by its inherent property of being resistant against the correlation attack while most other fingerprint fuzzy vault implementations proposed in the literature do not address resistance against the correlation attack. To the best of our knowledge, the only finger fuzzy vault implementation in the literature that proposes a solution causing resistance against the correlation attack is the implementation of Nandakumar et al. (2007b): They proposed to additionally incorporate a user password into the implementation of Nandakumar et al. (2007a) to improve security. The big advantage of our implementation is that it does not need a password for being cross-matching resistant which would require mechanism for keeping the additional password secret.

For our implementation we did not propose a solution to cope with the alignment of query finger to the vault. Although alignment would be enabled by storing public alignment helper data (such as points of high ridge curvature (Nagar et al. (2008, 2010); Nandakumar et al. (2007a); Uludag and Jain (2006))) this may easily weaken vault security (see Section 4.2 for a discussion). To get rid of such issues caused by additional alignment data one can use an approach in where only fingerprint features that are invariant to translation and rotation are used in the vault. Such an implementation is provided by Li et al. (2010) who use minutiae descriptors and minutiae local structures. However, the implementation may still be vulnerable to the correlation attack (see Section 4.4).

Regarding authentication performance, we observe that our implementation performs well when compared to other implementations found in the literature (see Table 5.7), even though other implementations are not resistant against the correlation attack. For instance, if at least 39 bits of security is sought, then the implementation of Nandakumar et al. (2007a) report a genuine acceptance rate of 86% at no experienced false accept. Our implementation, even with a security of 41 bits against the brute-force attack, performs with a genuine acceptance rate of 92% and zero false accepts. For even a higher security of 47 bits we experienced an amount of 91% of genuine accepts but even a single false accept.[7] The implementation of Nagar et al. (2010) at a security level of 45 bits (see Table 4.5 on page 66) performs with a genuine acceptance rate of 91% and no false accept.[8]

One may argue that our implementation is hardly comparable with other implementations since the rates that we reported only hold under a well-solved alignment framework. In my opinion, a valid counterargument is that the security of the

---

[7] The single false accept was due to the randomness of our decoder. Note that the false acceptance rate is never identically to zero.

[8] We obtained these rates from Figure 7 in (Nagar et al. (2010))

Table 5.7: Comparison with other implementations from the literature; all authentication rates have been measured on the reduced FVC 2002 DB2-A database which contains the first two impression for each finger only (see Section 3.4.8).

implementation	**alignment**	**cross-matching resistant**	**security in bits**	GAR (FAR)
Uludag et al. (2005)	manually	no	36	79% (0%)
Uludag and Jain (2006)	automatically	no	32	72.6% (0%)
Nandakumar et al. (2007a)	automatically	no	24 31 39	91% (0.13%) 91% (0.01%) 86% (0%)
Nandakumar et al. (2007b)	automatically	yes	24+password 31+password 39+password	90% (0%) 88% (0%) 81% (0%)
Nagar et al. (2010)	automatically	no	45	$\approx$ 91% ($\approx$ 0%)
Li et al. (2010)	alignment-free	no	52	92% (0%)
our implementation	manually	yes	36 41 47 52 58 63	94% (0%) 92% (0%) 91% ($\approx$ 0.02%) 87% (0%) 85% (0%) 79% (0%)

other implementations may be lower than reported due to the information leaked from additional alignment helper data (see Section 4.2). Furthermore, if alignment invariant features are used, it is not yet clear whether they can be encoded as exactly reproducible features (in the manner as quantized minutiae) such that a fuzzy vault construction that is resistant against cross-matching via correlation can be applied for template protection. Moreover, the genuine acceptance rates that we compare in Table 5.7 only correspond to a sample size of 100 (barring failure to captures), which is of low statistical significance.

If all impressions of the FVC 2002 DB2-A database are considered, the only fuzzy fingerprint vault implementation for that authentication rates have been reported, to the best of our knowledge, is the implementation of Li et al. (2010). They achieve a genuine acceptance rate of 72% at zero false accepts. Our implementation performs even with a genuine acceptance rate of 85.36% at which no false accepts have been observed in our experiment (see Table 5.2).

6. Discussion

6.1 Conclusion

We investigated the security of current implementations of the fuzzy fingerprint vault. We found that brute-force attacks are not only feasible but rather easy to perform (see Table 4.3 on page 61). We also found that, even if the brute-force attack is impractical against some implementations, this does not hold for the false-accept attack. This attack is feasible for every authentication scheme in which the false acceptance rate is non-negligible and thus it is for current implementations of the fuzzy fingerprint vault. In addition, according to our observations, the false-accept attack can be performed much more efficiently than the brute-force attack. Apparently, this problem can only be solved when multiple fingers or even multiple biometric modalities are combined. Therefore, multiple finger fuzzy vaults should be investigated as a potential method wherever high security is important. And yet a significant risk remains: In our view, the correlation attack cannot be solved merely by using multiple fingers.

Therefore we endeavored to solve the problem of the correlation attack. In this thesis we have demonstrated that it is possible to implement a minutiae fuzzy vault that is resistant against the correlation attack without loss of authentication performance. Our implementation primarily relies on the simple innovation of rounding minutiae to a rigid grid while using the entire grid as vault features, thereby preventing attackers from distinguishing genuine from chaff features via correlation. Furthermore, to make vault authentication practical we proposed to use a randomized decoder rather than systematically iterating through all candidate polynomials. Since the randomized decoder only affects vault authentication and not vault construction, the randomized decoder does not adversely affect vault security. Well conceived, the randomized decoder may be incorporated into a wide variety of fuzzy vault implementations, not only fuzzy vaults with the express purpose of protecting minutiae templates of a single finger. Furthermore, our single-finger fuzzy vault construction that is resistant against the correlation attack may be generalized to a construction that protects multiple fingers. However, an important step that we did not explore, but which we suggest should be the focus of future research, is the problem of finger alignment in the vaults.

6.2 Outlook

We did not propose a mechanism for dealing with alignment for our vault construction. Although it would have been possible to adopt the ideas available in the literature that propose to store additional alignment-helper data publicly with the vault (Jeffers and Arakala (2007); Li et al. (2008); Nandakumar et al. (2007a); Uludag and Jain (2006); Yang and Verbaudwhede (2005)) it is not yet clear how this would affect vault security. Moreover, some of the proposals find accurate alignment via multiple candidate alignments: During authentication, for each candidate alignment an authorization attempt is performed until the correct secret is seen. Translating this method to multiple fingers is problematic because the number of candidate alignments grows exponentially with the number of fingers. For example, if for a single-finger fuzzy vault there were an average of four such candidate alignments, the amount of candidate alignments for a three-finger fuzzy vault would be approximately 56. Consequently, fingerprint alignment techniques for multi-finger fuzzy vaults should be reconsidered.

Ideally, fingerprints could be pre-aligned. This would make iterations through several candidate alignments obsolete. Moreover, fuzzy vaults protecting accurately pre-aligned fingers do not need to store additional alignment-helper data which can cause unwanted information leakage regarding the corresponding finger. Pre-alignment of fingerprints is strongly related to the concept of intrinsic coordinate systems (Bazen and Gerez (2001)). Unfortunately, current methods that extract intrinsic coordinate systems are not robust enough to produce fingerprint pre-alignment of sufficient accuracy. Although challenging, it may be worthwhile to seek more robust methods to extract intrinsic coordinate systems.

On the other hand, the vault performance achievable with pre-aligned fingers may be less than that achievable using a well-solved alignment framework. So an alternative would be to design fuzzy vaults which protect feature sets that are invariant to fingerprint alignment. In contrast to the construction of Li et al. (2010) the vault features of different records must be equal to achieve resistance against the correlation attack. It would be interesting to investigate these various approaches and find out which provides the best performance. We plan to tackle these topics in the future.

6.3 Open Problems

Even from a mathematical perspective, security of the fuzzy fingerprint vault is not yet proven. While Juels and Sudan (2002) were able to show information-theoretic security of the fuzzy vault scheme for appropriate vault parameters, the fuzzy *fingerprint* vault has vault parameters for which no information-theoretic security can be asserted (see Lemma 3.1.1 on page 38). Most likely, the lack of information-theoretic security cannot be solved by using multiple fingers. The length of the secret polynomial k will likely remain small compared to the number of genuine vault points t; consequently, Lemma 3.1.1 can not be used to assert vault security (see Merkle et al. (2010b)).

It may be possible that the security of the fuzzy fingerprint vault can be assumed because of the problem of the hardness of polynomial reconstruction, which the difficulty of breaking a vault instance can be reduced to. In fact, Guruswami and Vardy (2005) proved NP-hardness of the polynomial reconstruction problem for finite fields of very large cardinality. Moreover, it is known that random instances of the polynomial reconstruction problem for random parameters are as hard as the worst case (see Kiayias and Yung (2008)). But this does not prove security for the fuzzy fingerprint vault barring additional contributions that clearly establish the hardness of polynomial reconstruction in all relevant cases.

Although commonly believed to be true, it remains to be proven that the polynomial reconstruction problem remains hard even for arbitrary finite fields. Furthermore, a solution to the famous *P versus NP problem* has yet to be found.[1]

6.4 Final Remarks

Even if the above questions were resolved, there remains an important additional one regarding the eventual obsolescence of all biometric protection schemes. The number of fingers a single human can provide is obviously limited, and as a consequence, the security capacity of the fuzzy fingerprint vault is limited.[2] Computational resources, however, become exponentially more powerful with time. Therefore, it remains to be seen whether we can design any biometric template protection scheme that will not eventually lose its effectiveness.

---

[1] Security of the polynomial reconstruction problem assumes that $P \neq NP$.

[2] Note, this observation can be generalized even for multiple biometric modalities.

A. Software Library

In this section we give an overview to the implementation we used for the experiments in this thesis. This implementation has been build up into a `C++` library we refer to as the `libthimble` or `thimble` library in the following.

Not surprisingly, implementations of the fuzzy vault scheme and the fuzzy commitment scheme require a comprehensive amount of sub-implementations for doing number theory. A powerful library providing them is the `Number Theory Library` (`NTL`) by Shoup (2009).

`NTL` provides a well-rounded programming interface that has the capability of serving as the basis for many purposes. However, in case finite field arithmetic is sought in very small binary finite fields, the computation can be heavily accelerated by making use of logarithm and antilogarithm tables. The running times concerning cryptanalysis of the fuzzy vault, for instance, strongly benefit from fast finite field computations. For this reason, we decided to implement arithmetic of small finite field by ourselves. Concerning basic number theory, `libthimble` does not provide the functionalities as `NTL` does. A certain functionality has been included into `libthimble` only if it was needed for a certain purpose. On the other hand, concerning error-correcting codes, `libthimble` provides functionalities that are not provided by `NTL`. These are for instance different en- and decoders for error-correcting codes such as Golay codes, BCH codes, and Reed-Solomon codes. Moreover, it is worth to mention that our library even provides an interface for a Guruswami-Sudan decoder (Guruswami and Sudan (1998)) which can correct a received Reed-Solomon codeword even beyond the error-correction bound.

In the subsequent, we will give an excerpt of `libthimble`'s programming interface providing only the most important functionalities.

A.1 Programming Interface

In an early version, our library was based upon `NTL` but has now reached a state that gets along without any additional libraries, except the `C++` standard library. For this reason, the interface for the number theoretical part of our library looks similar to that of `NTL`.

The programming interface providing number theoretical functionalities, for instances, are collected up in the header file `nttools.h` and can be included via the directive

```
#include <thimble/nttools.h>
```

`libthimble`'s programming interface is wrapped into the namespace `thimble`. Consequently, to avoid frequent use of `thimble::` we can insert the command

```
using namespace thimble;
```

A minimal example of code implementing an executable using functionalities provided by `libthimble` could thus have the following shape

```
#include <iostream>

#include <thimble/nttools.h>

using namespace std;
using namespace thimble;

int main( int argc , char *args[] ) {

        // Commands using thimble's functionalities

        return 0;
}
```

We start with describing the interface of our library that provides simple finite field arithmetic.

A.2 Number Theory

Number theoretical functions are provided by the `nttools.h` header.

A.2.1 Prime Fields

Roughly speaking, if $p = 2, 3, 5, 7, \ldots$ is a prime number, finite field arithmetic is yielded by canonical addition/multiplication of integers in $\{0, 1, \ldots, p-1\}$ whose respective results are reduced modulo p afterwards, i.e. remainder of division by p. Thus, the way we represent elements of finite prime fields is given by a representative in $\{0, \ldots, p-1\}$ wrapped in a structure called `gfp_t`. Before prime field computation can be performed, the modulus of the prime p must be set. This can achieved by invoking the setter function

```
void gfp_init( uint32_t p );
```

to enable subsequent computations in the finite field of cardinality p. To access the cardinality of the prime field we may consult the functions

```
uint32_t gfp_modulus();
uint32_t gfp_cardinality();
```

The structure `gfp_t` has the variable `rep` that is a `uint32_t` intended to hold the representative of the corresponding finite field element in the range $\{0, \ldots, p-1\}$.

A declared `gfp_t` must be initialized before used, e.g., via

```
void SetZero( gfp_t & x ); // x = 0
void SetOne ( gfp_t & x ); // x = 1
```

or by setting `x.rep` manually.

Arithmetic

Addition, subtraction, multiplication, and division can be performed using the following functions.

```
// c = a + b
void add( gfp_t & c , gfp_t a , gfp_t b );
gfp_t add( gfp_t a , gfp_t b );

// c = a - b
void sub( gfp_t & c , gfp_t a , gfp_t b );
gfp_t sub( gfp_t a , gfp_t b );

// c = a * b
void mul( gfp_t & c , gfp_t a , gfp_t b );
gfp_t mul( gfp_t a , gfp_t b );

// c = a^e, if e<0 then a must be non-zero
void power( gfp_t & c , gfp_t a , long e );

// c = a / b
void div( gfp_t & c , gfp_t a , gfp_t b );
gfp_t div( gfp_t a , gfp_t b );
```

Furthermore, the negative of an `gfp_t` can be obtained via

```
void negate( gfp_t & c , gfp_t a ); // c =-a
gfp_t negate( gfp_t a );
```

while the multiplicative inverse can be computed by

```
void inv( gfp_t & c , gfp_t a ); // c = a^(-1)
gfp_t inv( gfp_t a );
```

Random Elements

Sometimes we may want to generate a `gfp_t` at random. For this purpose, the function

```
void random( gfp_t & c , bool tryRandom = false );
```

is provided. If `tryRandom` is `true` then the function tries to use a random generator with more entropy, e.g., `/dev/urandom` on a Unix systems; otherwise, if `false`, the standard pseudo random generator is used.

Miscellaneous

A `gfp_t` can be printed in a `std::ostream` via the `<<` operator, i.e.

```
std::ostream & operator<<( std::ostream & out , gfp_t f );
```

A.2.2 Binary Polynomials of Small Degree

In a similar manner as for prime fields, arithmetic for binary fields is provided, in particular for extension fields over $\mathbb{F}_2$. A convenient way to define binary fields is with the help of polynomials over $\mathbb{F}_2$.

Binary polynomials of degree ≤ 63 are wrapped in the class `gf2x`:

```
class gf2x {

        uint64_t rep;

        //zero polynomial
        gf2x();

        //polynomial encoded by 'rep'
        gf2x( uint64_t rep );

        //copy constructor
        gf2x( const gf2ex & x );


        // a = b; copies b into a
        gf2x operator=( const gf2x & x );
};
```

The meaning of the variable `rep`, which defines a polynomial f as a `gf2x`, is as follows. Write for a polynomial

$$f(X) = f_0 \cdot X^0 + f_1 \cdot X^1 + f_2 \cdot X^2 + \ldots + f_{63} \cdot X^{63},$$

with $f_i \in \{0, 1\}$. Then

$$\texttt{rep} = f_0 \cdot 2^0 + f_1 \cdot 2^1 + f_2 \cdot 2^2 + \ldots + f_{63} \cdot 2^{63}.$$

To access the ith coefficient f_i of a f, we can use the follow relation

$$f_i = \begin{cases} 1, & \texttt{(rep}\gg\texttt{i)\&0x1 != 0} \\ 0, & \texttt{(rep}\gg\texttt{i)\&0x1 == 0} \end{cases}.$$

This relation is implemented for our convenience in the function

```
bool coeff( const gf2x & f , long i );
```

which is `true` if $f_i = 1$ and `false` if $f_i = 0$. Similarly, if the coefficient of f_i should be set to either 1 (`true`) or 0 (`false`) we may use

Listing A.1: Setting a coefficient

```
void SetCoeff( gf2x & f , long i , bool fi );
void SetCoeff( gf2x & f , long i ); // fi = 1
void ClearCoeff( gf2ex & f , long i ); // fi = 0
```

We may often need to know the degree of a certain polynomial. The function

```
long deg( const gf2x & f );
```

does the job. Sometimes a specific `gf2x` has to be set to zero, one, or the monomial polynomial X. For such purposes, the following convenience functions are provided

```
void SetZero( gf2x & f ); // f = 0
void SetOne( gf2x & f );  // f = 1
void SetX( gf2x & f );    // f = X
```

Conversely, testing a polynomial on whether it is zero, one, or the monomial polynomial X can be achieved by

```
bool IsZero( gf2x & f ); // true iff f = 0
bool IsOne( gf2x & f );  // true iff f = 1
bool IsX( gf2x & f );    // true iff f = X
```

Arithmetic

```
// h = f+g, this corresponds to an xor-operation
void add( gf2x & h , const gf2x & f , const gf2x & g );

// h = f-h, which yields the same result as addition in
// the binary case
void sub( gf2x & h , const gf2x & f , const gf2x & g );

// h = f*g, if deg(f),deg(g)<=31
void mul( gf2x & h , const gf2x & f , const gf2x & g );
```

Using `mul`, the correct product of two polynomials f and g is only guaranteed if $\deg f,\ \deg g \leq 31$ due to the limited size a `gf2x` can store. Furthermore, there are functions that provide polynomial division.

```
// q*b+r=a with deg(r)<deg(b)
void DivRem( gf2x & q , gf2x & r , const gf2x & a ,
             const gf2x & b );
void div( gf2x & q , const gf2x & a , const gf2x & b );
void rem( gf2x & r , const gf2x & a , const gf2x & b );
```

Extended Euclidean Algorithm

```
//Computes g, s, and t such that GCD(a,b)=g=s*a+t*b
void XGCD( gf2x & g , gf2x & s , gf2x & t ,
           const gf2x & a , const gf2x & b );
```

Modular Arithmetic

```
// h = (f*g) rem m
void MulMod( gf2x & h , const gf2x & f , const gf2x & g ,
             const gf2x & m );

// g = f^(-1) rem m, i.e. g with deg(g)<deg(m) such
// that (f*g) rem m = 1; this assumes that
// GCD(f,m)=1, deg(m) >= 1
void InvMod( gf2x & g , const gf2x & f , const gf2x & m );

// h = f^e rem m, if e<0 then m and f have to be co-prime
void PowerMod( gf2x & h , const gf2x & f , long e ,
               const gf2x & m );

// h = f(g) rem m
void CompMod( gf2x & h , const gf2x & f , const gf2x & g ,
              const gf2x & m );
```

Miscellaneous

A representation of a `gf2x` can be printed to a `std::ostream` using the `<<` command, i.e.

```
std::ostream & operator<<( std::ostream & out , const gf2x & f );
```

A.2.3 Small Binary Field Extensions

The following is provided by the `nttools.h` header file.

Binary field extension are defined with the help of polynomials with coefficients in a ground field that are irreducible. Given an irreducible polynomial f, the corresponding extension of the field with two elements, i.e. $\mathbf{F} = \mathbb{F}_2[X]/(f(X) \cdot \mathbb{F}_2[X])$, can be initialized via

```
void gf2e_init( const gf2x & f );
```

This function globally establishes logarithmic and anti-logarithmic multiplication tables to accelerate subsequent arithmetic in the corresponding finite field. It assumes that all elements of the entire finite field $\mathbf{F}$ fit in memory. Barring computer memory, the finite field can have cardinality at most 2^{30}. For convenience, irreducible polynomials to define finite fields are already provided by `libthimble`. These can be obtained using

```
gf2x gf2_1();
gf2x gf2_2();
...
gf2x gf2_16();
...
gf2x gf2_29();
gf2x gf2_30();
```

For example, a finite field of cardinality 2^{16} can be easily initialized globally via

```
gf2e_init(gf2_16());
```

If the irreducible polynomial defining the finite field is required during computation, it can be obtained by

```
const gf2x & gf2e_modulus();
```

Its degree and and the cardinality of the finite field are available by

```
long gf2e_degree();
uint32_t gf2e_cardinality();
```

If a list of all elements in the finite field is wanted,

```
gf2e_elements( std::vector<gf2e_t> & F );
```

may be useful.

An element of a small binary field is wrapped into the structure `gf2e_t`, which contains a member variable `rep` encoded as an `uint32_t`. The variable `rep` defines a polynomial `gf2x` which is the represantive of the corresponding finite field element. Therefore, the zero element is given by `rep=0` while the unity element is guaranteed to be given by `rep=1`. As for a prime field element, an `gf2e_t` must be initialized before it is used, e.g., via

```
void SetZero( gf2e_t & x ); // x = 0
void SetOne( gf2e_t & x );  // x = 1
```

or by setting the member variable `rep` manually. Conversely, the following functions allow for testing a `gf2e_t` of being zero or one.

```
bool IsZero( gf2e_t x ); // true if x = 1; false otherwise
bool IsOne( gf2e_t x );  // true if x = 0; false otherwise
```

Arithmetic

```
// c = a+b, this corresponds to an xor-operation
void add( gf2e_t & c, gf2e_t a, gf2e_t b );
gf2e_t add( gf2e_t a, gf2e_t b );

// c = a-b, this is equivalent to a+b
void sub( gf2e_t & c, gf2e_t a, gf2e_t b );
gf2e_t sub( gf2e_t a, gf2e_t b );

// c = a*b
void mul( gf2e_t & c, gf2e_t a, gf2e_t b );
gf2e_t mul( gf2e_t a, gf2e_t b );

// c = a^e, if e<0, a must be non-zero
void power( gf2e_t & c, gf2e_t a, long e );

// c = a/b, b non-zero
void div( gf2e_t & c, gf2e_t a, gf2e_t b );
gf2e_t div( gf2e_t a, gf2e_t b );
```

The inverse of a non-zero `gf2e_t` can be computed by

```
void inv( gf2e_t & b , gf2e_t a );
gf2e_t inv( gf2e_t a );
```

Random Elements

```
void random( gf2e_t & x , bool tryRandom = false );
```

If `tryRandom` is `true` then the function tries to use a random generator with more entropy, e.g., `/dev/urandom` on a Unix systems; otherwise, if `false`, the standard pseudo random generator is used.

Miscellaneous

A representation of `gf2e_t` can be printed to a `std::ostream` via the `<<` command, i.e.

```
std::ostream & operator<<( std::ostream & out , gf2e_t f );
```

A.2.4 Polynomials

The following is provided by the `nttools.h` header file.

Polynomials with coefficients in a finite field are essential for fuzzy vault. In particular, they are important for many purposes related with coding theory and error-correcting codes.

`libthimble` provides three classes of polynomials over finite fields. These are polynomials with `gfp_t` and `gf2e_t` as coefficients as well as polynomials with coefficients in $\mathbb{F}_2$. All of these polynomials can have arbitrary degree that is only limited by computer memory. The polynomials are wrapped in the classes

```
class gfpx;  //polynomial over gfp_t
class gf2ex; //polynomial over gf2e_t
class GF2X;  //polynomial over the binary field
```

`GF2X` provides much faster arithmetic compared to the arithmetic of `gf2ex` using the finite field initialized with `gf2e_init(gf2_1())`.

For many functions provided for on the class there exists an analogous one for the other two polynomial classes. Thus, if we write `ELX`, we mean one of the polynomials `gfpx`, `gf2ex`, or `GF2X`. Furthermore, by `EL` we refer to the type over that the polynomial of type `ELX` is defined, i.e. the type of its coefficient. Thus `EL` is either `gfp_t`, `gf2e_t`, or `bool` as `ELX` is either `gfpx`, `gf2ex`, or `GF2X`, respectively.

With this notation, each polynomial over finite fields that `libthimble` provides has the following constructors.

```
ELX(); //zero
ELX( const ELX & f ); //copy constructor
ELX( EL b ); //constant polynomial equals b
```

Furthermore, `GF2X` has the constructor

```
GF2X( gf2x & f ); // conversion of f
```

Each `ELX` implement the assignment operator such that polynomials can be copied, i.e.

```
ELX & ELX::operator=( const ELX & f );
```

Comparison of polynomials can be realized via the `==` operator, i.e.

```
// true if and only if f=g
bool operator==( const ELX & f , const ELX & g );
```

The degree of an `ELX` can be accessed via

```
// if f=0 then returns -1, otherwise the position
// of the highest non-zero term
long deg( const ELX & f );
```

Polynomials can be tested on being zero or one using

```
bool IsZero( const ELX & f );
bool IsOne( const ELX & f );
```

Furthermore, a particular polynomial instance can be set to zero/one via

```
bool SetZero( ELX & f );
bool SetOne( ELX & f );
```

Setter and getter for the coefficients of polynomials are

```
//returns coefficient of the X^i monimial
EL coeff( const ELX & f , long i );

// Sets coefficient of X^i to fi , i >= 0
void SetCoeff( ELX & f , long i , EL fi );
```

Arithmetic

```
// c = a+b
void add( ELX & c , const ELX & a , const ELX & b );

// c = a-b
void sub( ELX & c , const ELX & a , const ELX & b );

// c = a*b
void mul( ELX & c , const ELX & a , const ELX & b );

// c = a^e
void power( ELX & c , const ELX & a , uint64_t & e );

// q*b+r=a with deg(r)<deg(b); assumes b is non-zero
void DivRem( ELX & q , ELX & r ,
             const ELX & a , const ELX & b );
void div( ELX & q , const ELX & a , const ELX & b );
void rem( ELX & r , const ELX & a , const ELX & b );
```

Greatest Common Divisor via the Euclidean Algorithm

```
// g = gcd(a,b)
void GCD( ELX & g , const ELX & a , const ELX & b );

//Computes g, s, and t such that GCD(a,b)=g=s*a+t*b
void XGCD( ELX & g , ELX & s , ELX & t ,
           const ELX & a , const ELX & b );
```

Modular Arithmetic

```
// h = f*g rem m
void MulMod( ELX & h , const ELX & f , const ELX & g ,
             const ELX & m );

// h = f^(-1) rem m
void InvMod( ELX & h , const ELX & f , const ELX & m );

// h = f^e rem m, if e<0 then m and f must be co-prime
void PowerMod( ELX & h , const ELX & f , long e ,
               const ELX & m );

// h = f(g) rem m
void EvalMod( ELX & h , const ELX & f , const ELX & g ,
              const ELX & m );
```

Random Polynomials

```
void random( ELX & f , long n , bool tryRandom = false );
```

Generates a random polynomial of degree $< n$. If `tryRandom` is `true` then the function tries to use a random generator with more entropy, e.g., `/dev/urandom` on a Unix systems; otherwise, if `false`, the standard pseudo random generator is used.

Miscellaneous

A representation of `ELX` can be printed to a `std::ostream` via the `<<` command, i.e.

```
std::ostream & operator<<( std::ostream & out , const ELX & f );
```

A.2.5 Polynomial Evaluation, Interpolation, and Factorization

The functions described here are provided by the `nttools.h` header file.

Functions for performing multi-point polynomial evaluation and polynomial interpolation are implemented for the classes `gfpx` and `gf2ex`.

Multipoint Evaluation

```
// y[i] = f(x[i])
void eval( std::vector<gfp_t> & y , const gfpx & f ,
           const std::vector<gfp_t> & x );
void eval( std::vector<gf2e_t> & y , const gf2ex & f ,
           const std::vector<gf2e_t> & x );
```

Interpolation

```
// computes f of minimal degree such that f(x[i])=y[i]
// the elements of x must be pairwise distinct;
// x and y should have the same length;
void interpolate( gfpx & f ,
                  const std::vector<gfp_t> & x ,
                  const std::vector<gfp_t> & y );
void interpolate( gf2ex & f ,
                  const std::vector<gf2e_t> & x ,
                  const std::vector<gf2e_t> & y );
```

Factorization

Functions for root finding are provided for `gf2ex` and `gfpx`.

```
//Find all roots[i] such that f(roots[i])=0
void FindRoots( std::vector<gfp_t> & roots , const gfpx & f );
void FindRoots( std::vector<gf2e_t> & roots , const gf2ex & f );
```

For the classes `gfpx`, `gf2ex`, and `GF2X` methods for finding factors are provided.

```
// Finds an irreducible non-constant factor of f
void IrreducibleFactor( ELX & h , const ELX & f );
```

Irreducibility Testing

The following function tests whether a polynomial is irreducible.

```
bool IsIrreducible( const ELX & f );
```

A.3 Cryptographic Tools

`thimble` provides some implementations related to cryptography. These functions are provided by the `cryptools.h` header file.

A.3.1 SHA-1

The following method implements the secure hash algorithm (National Institute of Standards and Technology (1995)) of data `message` that holds n bytes. After invocation `hash` holds the SHA-1 hash value of `message`.

```
void SHA1( uint32_t hash[5] , uint8_t *message , uint64_t n );
```

For convenience, `thimble` provides wrapper functions that compute the SHA-1 hash value of the data of polynomials, which is frequently needed for authentication in `thimble`'s fuzzy fingerprint vault implementations documented below.

```
void SHA1( uint32_t hash[5] , const gfpx & f );
void SHA1( uint32_t hash[5] , const gf2ex & f );
```

A.4 Error-Correcting Codes

Classes, types, and functions related to error-correcting codes are provided by the `ecctools.h` header file.

A.4.1 Binary BCH Codes

Prior encoding and decoding in a certain binary BCH code it must be generated. Therefore the class `BCHCode` is provided which has the following constructor.

```
BCHCode( long n , long nu );
```

BCH codewords are represented as binary polynomials, i.e. as `GF2X`. Given a polynomial `r` of degree smaller than `n` we can compute its nearest BCH codeword (as a polynomial) via the member function

```
// computes nearest codeword c to r
// and returns true if possible;
// otherwise returns false
bool round( GF2X & c , const GF2X & r ) const;
```

of the class `BCHCode`. Furthermore, the class `BCHCode` provides the following member function which may be useful for encoding and decoding.

```
//generator polynomial of the BCH code
const GF2X & get_g() const;
```

A.4.2 Golay Codes

The following functions implement encoding and decoding of Golay codes.

```
//add redundancy to a 12-bit word
//and returns its corresponding
//23-bit Golay codeword
uint32_t GolayEnc( uint16_t msg );

//rounds a 23-bit word to
//its nearest 23-bit Golay codeword
//and returns it
uint32_t GolayRound( uint32_t r );

//essentially performs
//GolayRound(r) but returns only
//the first 12 bits
uint16_t GolayDec( uint32_t r );
```

The function `GolayRound` implements the low-complexity scheme proposed by Ching-Lung et al. (2006) which is faster than exhaustive search.

A.4.3 Reed-Solomon Codes

Given different code locators $x[0], \ldots, x[n-1]$ of a finite field $\mathbf{F}$ and an integer $k < n$ they define the Reed-Solomon code

$$\mathbf{C} = \{(f(x[0]), \ldots, f(x[n-1])) \mid f \in \mathbf{F}[X],\ \deg f < k\}. \tag{A.1}$$

If we received $y = (y[0], \ldots, y[n-1]) \in \mathbf{F}^n$, we may ask for the polynomial f that defines the nearest codeword $c = (f(x[0]), \ldots, f(x[n-1]))$ to y. The following functions can be used to compute such a polynomial with coefficients in `gfp_t` and `gf2e_t` in the case where $\operatorname{dist}(c, y) \leq (n-k+1)/2$.

```
// finds f if there exist one which already interpolates
// at least (n+k)/2 pairs f(x[i])=y[i] and returns true;
// otherwise returns false
bool gaodecode( gf2ex & f , const std::vector<gf2e_t> & x ,
                const std::vector<gf2e_t> & y , long d );
bool gaodecode( gfpx & f , const std::vector<gfp_t> & x ,
                const std::vector<gfp_t> & y , long d );
```

The above functions implement the algorithm of Gao (2002).

A.4.4 Guruswami-Sudan Decoder

`thimble` also provides a *Guruswami-Sudan list decoder* (Guruswami and Sudan (1998)) for both `gfp_t` and `gf2e_t`. The interface is the following

```
GSDecTime gsdecode
( std::vector<gf2ex> & P ,
  const std::vector<gf2e_t> & x ,
  const std::vector<gf2e_t> & y ,
  long k , long m );

GSDecTime gsdecode
( std::vector<gfpx> & P ,
  const std::vector<gfp_t> & x ,
  const std::vector<gfp_t> & y ,
  long k , long m );
```

m is a parameter controlling how many errors the decoder can tolerate, i.e. the multiplicity in the interpolation step of the Guruswami-Sudan decoder. In particular, if m has been chosen sufficiently large, the list P contains all polynomials f such that $f(x[i]) = y[i]$ for more than $\sqrt{n \cdot k}$ elements.

`thimble`'s implementation of the Guruswami-Sudan decoder follows the description of Trifonov (2010) for the *interpolation step* and the description of Roth and Ruckenstein (2000) for the *factorization step*.

The class `GSDecTime` contains timings a decoding attempt lasted. In particular, the time in seconds needed for the entire decoding attempt is achievable via the member function `totalSecs()`, for the interpolation step via `interpolationSecs()`, and for the factorization step via `rootSecs()`.

A.5 Fingerprints

The main purpose of the `thimble` library is to implement fuzzy fingerprint vaults and functionalities to support security analysis. In this respect `thimble` provides an interface for minutiae templates.

A.5.1 Minutiae

The header `Minutia.h` provides the class `Minutia` with the following constructors.

```
//Minutia at (0,0) with angle 0 and unknown type
Minutia();

//Minutia at (X,Y) with angle 'theta' and unknown type
Minutia( long X , long Y , long theta )

//Minutia at (X,Y) with angle 'theta' and type 'typ'
Minutia( long X , long Y , long theta , MINUTIATYPE_T typ );

//copy constructor
Minutia( const Minutia & minutia );
```

A `MINUTIATYPE_T` can have the values `UNKNOWN_MT`, `ENDING_MT`, and `BIFURCATION_MT`. Angles are encoded as integers between 0 and 359. Furthermore, the attributes of a `Minutia` can be assigned using its member functions

```
void setX( long X );
void setY( long X );
void setAngle( long angle );
void setType( MINUTIATYPE_T typ );
```

Moreover, a `Minutia` can store a quality measure between 0 and 255 which can be set via

```
void setQuality( uint8_t q );
```

Correspondingly, the attributes can be obtained using the member functions

```
long getX() const;
long getY() const;
long getAngle() const;
uint8_t getQuality() const;
```

Minutiae Templates

A minutiae template in `thimble` is a `std::vector<Minutia>`. We do not provide a documented interface for reading or writing minutiae templates. When using our library, minutiae templates have to be converted manually.

Alignment

Given two minutiae templates

```
std::vector<Minutia> A , B;
```

we can compute an alignment of `B` to `A` via

```
void align
( std::vector<Minutia> & C ,
  const std::vector<Minutia> & A ,
  const std::vector<Minutia> & B );
```

Note that for our experiments in Section 3.4.4 on page 50, the alignment was achieved differently to the alignment given by the above function.

A.6 Fuzzy Vault

In the class `FuzzyVault` provided by the header `FuzzyVault.h` there are some static functions that we implemented to perform tests in this thesis.

A.6.1 Brute-Force Attack

The following functions perform N iterations of the brute-force attack where the “correct“ polynomial is identified by the number of points it interpolates. The attack attempts to find a polynomial f of degree d that interpolates at least D points $f(X[i]) = Y[i]$. If such a polynomial can be successfully found in one of the performed iteration, the functions return `true` and f contains the data of the found polynomial; otherwise `false` is returned.

```
static bool bfattack
( gf2ex & f , long d , long D ,
  const std::vector<gf2e_t> & X ,
  const std::vector<gf2e_t> & Y ,
  long N );

static bool bfattack
( gfpx & f , long d , long D ,
  const std::vector<gfp_t> & X ,
  const std::vector<gfp_t> & Y ,
  long N );
```

The following functions also perform N iterations of the brute-force attack. But this time the correct polynomial is identified with its SHA-1 hash value. If a polynomial with SHA-1 hash value equals to `hash` is found then the functions return `true` and f contains the found polynomial; otherwise `false` is returned.

```
static bool bfattack
( gf2ex & f , long d ,
  const std::vector<gf2e_t> & X ,
  const std::vector<gf2e_t> & Y ,
  long N , const uint32_t hash[5] );
```

```
static bool bfattack
( gfpx & f , long d ,
  const std::vector<gfp_t> & X ,
  const std::vector<gfp_t> & Y ,
  long N , const uint32_t hash[5] );
```

We used these functions to predict the expected timings for a successful brute-force attack of given vault parameters.

A.6.2 Ordinary Iterative Decoder

The following functions perform decoding by systematically iterating through all candidate polynomials. Essentially this corresponds to a brute-force attack that is run systematically.

Each of the following functions searches a polynomial f with SHA-1 hash value `hash` of degree $\leq d$ interpolating at least $d+1$ points $(x[i], y[i])$ and return `true` if it is found; otherwise `false` is returned.

```
static bool bfdecode
( gf2ex & f , long d ,
  const std::vector<gf2e_t> & x ,
  const std::vector<gf2e_t> & y ,
  const uint32_t hash[5] );

static bool bfdecode
( gfpx & f , long d ,
  const std::vector<gfp_t> & x ,
  const std::vector<gfp_t> & y ,
  const uint32_t hash[5] );
```

A.6.3 Randomized Decoder

Essentially, the randomized decoder of Section 5.4 agrees with the implementation of the brute-force attack in where the SHA-1 hash value is used for identifying the correct polynomial. Nonetheless, to allow clear distinction of the meaning `thimble` provides an interface for running the randomized decoder.

The following functions attempt to determine a polynomial f with SHA-1 hash value `hash` that is of degree $\leq d$ and that interpolates at least $d+1$ pairs $(x[i], y[i])$. They perform at most I iterations. If such a polynomial is found, the returned value will `true` and the decoded polynomial will be stored in f; otherwise `false` will be returned.

```
static bool RandDecode
( gf2ex & f , long d ,
  const std::vector<gf2e_t> & x ,
  const std::vector<gf2e_t> & y ,
  const uint32_t hash[5] , long I );

static bool RandDecode
```

```
( gfpx & f , long d ,
  const std::vector<gfp_t> & x ,
  const std::vector<gfp_t> & y ,
  const uint32_t hash[5] , long I );
```

A.7 Minutiae Fuzzy Vault

The header file `MinutiaeFuzzyVault.h` provides the interface of our reference implementation mainly following the description of Nandakumar et al. (2007a) (see Section 3.4 on page 47).

An instance of a minutiae fuzzy vault is constructed from the class `MinutiaeFuzzyVault`.

A.7.1 Initialization

Before minutiae can be protected via a `MinutiaeFuzzyVault` we should specify the region in where minutiae locations can occur. For this purposes the method

```
void setFixedBounds( long x0 , long y0 , long x1, long y1 );
```

is provided. If invoked, the fuzzy vault is deemed to protect a minutiae template whose minutiae are at the locations $(x, y) \in [x0, x1] \times [y0, y1]$. For minutiae templates from the FVC 2002 DB2 database a `MinutiaeFuzzyVault` thus has to be initialized with

```
MinutiaeFuzzyVault V;
V.setFixedBounds(0,0,295,559);
```

The length of the secret polynomial (default is $k = 9$) can be specified via

```
void set_k( long k );
```

A.7.2 Enrollment

Enrollment at a vault can be realized via

```
bool enrol
( const std::vector<Minutia> & T , const std::string & s );
```

using a minutiae template T and a secret string s. The minutiae template T is assumed to be ordered with respect to the quality of its minutiae. If no sufficient many minutiae could be selected from T, the function returns `false`; otherwise, after successful enrollment `true` is returned.

A.7.3 Authentication

An authentication attempt using a query template (aligned to the vault) can be achieved via

```
bool open
( std::string & s , const vector<Minutia> & Q ) const;
```

If Q is of sufficient agreement with the enrolled template T, the function returns `true` and s contains secret specified on enrollment; otherwise `false` is returned.

A.7.4 Correlation Attack

The correlation attack (i.e. Algorithm 4.4.1 on page 75) can be run via the function

```
CORRATTACK_T CorrelationAttack
( std::string & s ,
  const MinutiaeFuzzyVault & V ,
  const std::vector<Minutia> & W ,
  double x );
```

The result can either be `CORRATTACK_SUCCESS` in case the vault V could be successfully broken, or `CORRATTACK_FAILED` if unsuccessful. On success, s contains the secret string that was used on enrollment.

A.8 Cross-Matching Resistant Minutiae Fuzzy Vault

In `CMRMinutiaeFuzzyVault.h` the interface of our implementation of the cross-matching resistant minutiae fuzzy vault is provided (see Chapter 5 on page 87).

The vault implementation is wrapped in the class `CMRFminutiaeFuzzyVault`. The following variables are members of the class.

```
long alpha , t , k , I;
std::vector<std::pair<long double,long double> > grid;
```

`alpha` controls how significant minutiae angles are taken into account (default is 2); t bounds the number of genuine vault points (default is 46); k controls the length of the secret polynomial (default is 8); I corresponds to the number of iterations that are performed with the randomized decoder (default is $65,536$) on authentication; `grid` contains the rigid points to where minutiae locations on quantization (default is the hexagonal grid visible in Figure 5.1 on page 91 of distance $\lambda = 22$ centered in the region $[0,295] \times [0,559]$).

A.8.1 Quantization of Minutiae

The interface of our implementation that quantizes a minutia is given by the member function

```
uint32_t quantize( const Minutia & minutia ) const;
```

where the result (as a `uin32_t`) encodes a quantization of the specified minutia as an element of the vault's finite field. An entire template can be quantized using

```
void quantize
( std::set<uint32_t> & Q ,
  const std::vector<Minutia> & T )
const;
```

or equivalently

```
void quantize
( std::vector<uint32_t> & Q ,
  const std::vector<Minutia> & T )
const;
```

A.8.2 Enrollment

Binding a minutiae template T to a secret key s on enrollment can be performed with

```
bool enrol
( const std::vector<Minutia> & T ,
  const std::string & s );
```

which will return `false` if it was not able to obtain a quantization set of sufficient size; otherwise the result will be `true`.

A.8.3 Authentication

Similarly, authentication using a query template qT (aligned to the vault) is provided as

```
bool open
( std::string & s , const std::vector<Minutia> & qT );
```

which returns `true` if s contains the correct secret and `false` otherwise.

A.9 Miscellaneous

In `mtools.h` there is the function

```
void HexaGrid
( std::vector< std::pair<long double,long double> > & grid ,
  long double x0 , long double y0 ,
  long double x1 , long double y1 ,
  long double g );
```

which implements Algorithm 5.1.1 on page 93. That is, it computes points of a hexagonal grid `grid` that are centered in the region $[x0, y0] \times [x1, y1]$.

B. Tables

Table B.1: Distribution of unlocking set sizes $u(s)$ as described in Section 5.6.4

$u(0) = 0\%$	$u(10) = 0\%$	$u(20) = 1\%$	$u(30) = 8\%$	$u(40) = 28\%$
$u(1) = 0\%$	$u(11) = 0\%$	$u(21) = 2\%$	$u(31) = 8\%$	$u(41) = 32\%$
$u(2) = 0\%$	$u(12) = 0\%$	$u(22) = 2\%$	$u(32) = 9\%$	$u(42) = 36\%$
$u(3) = 0\%$	$u(13) = 0\%$	$u(23) = 2\%$	$u(33) = 11\%$	$u(43) = 38\%$
$u(4) = 0\%$	$u(14) = 0\%$	$u(24) = 3\%$	$u(34) = 13\%$	$u(44) = 43\%$
$u(5) = 0\%$	$u(15) = 0\%$	$u(25) = 3\%$	$u(35) = 13\%$	$u(45) = 50\%$
$u(6) = 0\%$	$u(16) = 0\%$	$u(26) = 4\%$	$u(36) = 18\%$	$u(46) = 100\%$
$u(7) = 0\%$	$u(17) = 1\%$	$u(27) = 6\%$	$u(37) = 19\%$	
$u(8) = 0\%$	$u(18) = 1\%$	$u(28) = 6\%$	$u(38) = 21\%$	
$u(9) = 0\%$	$u(19) = 1\%$	$u(29) = 7\%$	$u(39) = 26\%$	

Table B.2: Hypothetical Authentication Rates of the Cross-Matching Resistant Fuzzy Vault Implementation of Chapter 5. The rates reported have been determined on the FVC 2002 DB-A database (Maio et al. (2002))

minimal overlap of templates ω	hypothetical genuine acceptance rate $\mathrm{GAR}(\omega)$	hypothetical genuine acceptance rate on reduced dataset consisting of the first two impressions per finger only sub-$\mathrm{GAR}(\omega)$	hypothetical false acceptance rate $\mathrm{FAR}(\omega)$	upper bound of hypothetical false acceptance rate $\mathrm{FAR}_t(\omega)$
0	$= 100\%$	$= 100\%$	$= 100\%$	$= 100\%$
1	$\approx 99.39\%$	$= 100\%$	$\approx 85.14\%$	$\approx 85.64\%$
2	$\approx 98.89\%$	$= 100\%$	$\approx 60.13\%$	$\approx 62.09\%$
3	$\approx 98.21\%$	$= 100\%$	$\approx 38.31\%$	$\approx 41.13\%$
4	$\approx 97.53\%$	$= 99\%$	$\approx 20.51\%$	$\approx 22.47\%$
5	$\approx 96.82\%$	$= 99\%$	$\approx 10.27\%$	$\approx 11.84\%$
6	$\approx 95.89\%$	$= 99\%$	$\approx 4.99\%$	$\approx 5.97\%$
7	$= 94\%$	$= 99\%$	$\approx 1.84\%$	$\approx 2.33\%$
8	$\approx 92.1\%$	$= 98\%$	$\approx 0.69\%$	$\approx 0.92\%$
9	$\approx 89.14\%$	$= 98\%$	$\approx 0.27\%$	$\approx 0.33\%$
10	$\approx 85.21\%$	$= 96\%$	$\approx 0.11\%$	$\approx 0.14\%$
11	$\approx 81.14\%$	$= 94\%$	$\approx 0.03\%$	$= 0$
12	$\approx 77.21\%$	$= 94\%$	$= 0$	$= 0$
13	$\approx 72.07\%$	$= 92\%$	$= 0$	$= 0$
14	$= 66.75\%$	$= 91\%$	$= 0$	$= 0$
15	$\approx 61.46\%$	$= 90\%$	$= 0$	$= 0$
16	$\approx 56.21\%$	$= 87\%$	$= 0$	$= 0$
17	$\approx 49.92\%$	$= 81\%$	$= 0$	$= 0$
18	$\approx 44.92\%$	$= 77\%$	$= 0$	$= 0$
19	$\approx 39.03\%$	$= 73\%$	$= 0$	$= 0$
20	$= 34.5\%$	$= 72\%$	$= 0$	$= 0$
21	$\approx 29.1\%$	$= 69\%$	$= 0$	$= 0$
22	$\approx 24.92\%$	$= 63\%$	$= 0$	$= 0$
23	$\approx 21.42\%$	$= 60\%$	$= 0$	$= 0$
24	$\approx 18.03\%$	$= 55\%$	$= 0$	$= 0$
25	$\approx 14.46\%$	$= 51\%$	$= 0$	$= 0$
26	$\approx 11.28\%$	$= 40\%$	$= 0$	$= 0$
27	$\approx 8.78\%$	$= 35\%$	$= 0$	$= 0$
28	$\approx 6.67\%$	$= 31\%$	$= 0$	$= 0$
29	$\approx 5.21\%$	$= 28\%$	$= 0$	$= 0$
30	$\approx 4.32\%$	$= 27\%$	$= 0$	$= 0$
31	$\approx 3.1\%$	$= 24\%$	$= 0$	$= 0$
32	$\approx 2.39\%$	$= 16\%$	$= 0$	$= 0$
33	$\approx 1.57\%$	$= 12\%$	$= 0$	$= 0$
34	$\approx 0.96\%$	$= 8\%$	$= 0$	$= 0$
35	$\approx 0.64\%$	$= 6\%$	$= 0$	$= 0$
36	$\approx 0.35\%$	$= 4\%$	$= 0$	$= 0$
37	$\approx 0.14\%$	$= 1\%$	$= 0$	$= 0$
38	$\approx 0.07\%$	$= 0\%$	$= 0$	$= 0$
39	$\approx 0.07\%$	$= 0\%$	$= 0$	$= 0$
40	$\approx 0.07\%$	$= 0\%$	$= 0$	$= 0$
41	$\approx 0.03\%$	$= 0\%$	$= 0$	$= 0$
42	$\approx 0.03\%$	$= 0\%$	$= 0$	$= 0$
43	$\approx 0.03\%$	$= 0\%$	$= 0$	$= 0$
44	$= 0\%$	$= 0\%$	$= 0$	$= 0$
45	$= 0\%$	$= 0\%$	$= 0$	$= 0$
46	$= 0\%$	$= 0\%$	$= 0$	$= 0$

Table B.3: Predicted Authentication Rates of the Crossmatching Resistant Fuzzy Vault Implementation of Chapter 5. The rates reported have been determined on the FVC 2002 DB database (Maio et al. (2002)). The predictions make use of the values in Table B.2

size of secret polynomial k	predicted genuine acceptance rate GAR	predicted genuine acceptance rate on reduced dataset sub-GAR	predicted false acceptance rate FAR
0	$= 100\%$	$\geq 100\%$	$\leq 100\%$
1	$> 99.39\%$	$\geq 100\%$	$< 85.14\%$
2	$> 98.89\%$	$\geq 100\%$	$< 60.13\%$
3	$> 98.2\%$	$> 99.98\%$	$< 38.31\%$
4	$> 96.92\%$	$> 98.99\%$	$< 20.51\%$
5	$> 93.6\%$	$> 98.62\%$	$< 4.84\%$
6	$> 86.49\%$	$> 96.47\%$	$< 0.57\%$
7	$> 76.59\%$	$> 93.46\%$	$< 0.05\%$
8	$> 65.24\%$	$> 90.22\%$	$< 2.45 \cdot 10^{-5}$
9	$> 53.75\%$	$> 83.74\%$	$< 9.76 \cdot 10^{-7}$
10	$> 43.15\%$	$> 76.58\%$	$< 2.11 \cdot 10^{-8}$
11	$> 34.01\%$	$> 70.79\%$	$= 0\%$
12	$> 26.65\%$	$> 64.61\%$	$= 0\%$
13	$> 20.83\%$	$> 58.16\%$	$= 0\%$
14	$> 15.99\%$	$> 50.5\%$	$= 0\%$
15	$> 12.07\%$	$> 42.48\%$	$= 0\%$
16	$> 9.08\%$	$> 36.06\%$	$= 0\%$
17	$> 6.91\%$	$> 31.62\%$	$= 0\%$
18	$> 5.38\%$	$> 28.6\%$	$= 0\%$
19	$> 4.23\%$	$> 25.65\%$	$= 0\%$
20	$> 3.32\%$	$> 21.9\%$	$= 0\%$
21	$> 2.58\%$	$> 17.87\%$	$= 0\%$
22	$> 1.97\%$	$> 14.23\%$	$= 0\%$
23	$> 1.47\%$	$> 11.21\%$	$= 0\%$
24	$> 1.1\%$	$> 8.88\%$	$= 0\%$
25	$> 0.82\%$	$> 7.09\%$	$= 0\%$
26	$> 0.6\%$	$> 5.6\%$	$= 0\%$
27	$> 0.43\%$	$> 4.18\%$	$= 0\%$
28	$> 0.3\%$	$> 2.83\%$	$= 0\%$
29	$> 0.2\%$	$> 1.71\%$	$= 0\%$
30	$> 0.13\%$	$> 0.9\%$	$= 0\%$
31	$> 0.1\%$	$> 0.41\%$	$= 0\%$
32	$> 0.08\%$	$> 0.16\%$	$= 0\%$
33	$> 0.07\%$	$> 0.05\%$	$= 0\%$
34	$> 0.07\%$	$> 0.01\%$	$= 0\%$
35	$> 0.07\%$	$> 0\%$	$= 0\%$
36	$> 0.06\%$	$> 0\%$	$= 0\%$
37	$> 0.05\%$	$> 0\%$	$= 0\%$
38	$> 0.04\%$	$\geq 0\%$	$= 0\%$
39	$> 0.03\%$	$\geq 0\%$	$= 0\%$
40	$> 0.03\%$	$\geq 0\%$	$= 0\%$
41	$> 0.03\%$	$\geq 0\%$	$= 0\%$
42	$> 0.03\%$	$\geq 0\%$	$= 0\%$
43	$> 0.03\%$	$\geq 0\%$	$= 0\%$
44	$\geq 0\%$	$\geq 0\%$	$= 0\%$
45	$\geq 0\%$	$\geq 0\%$	$= 0\%$
46	$\geq 0\%$	$\geq 0\%$	$= 0\%$

Bibliography

Alekhnovich, M. (2002). Linear diophantine equations over polynomials and soft decoding of reed-solomon codes. In *Proc. of the 43rd Symp. on Foundations of Computer Science*, FOCS '02, pages 439–448, Washington, DC, USA. IEEE Computer Society.

Arakala, A., Jeffers, J., and Horadam, K. J. (2007). Fuzzy extractors for minutiae-based fingerprint authentication. In *Proc. Int'l Conf. on Biometrics*, LNCS 4642, pages 760–769.

Bazen, A. M. and Gerez, S. H. (2001). An intrinsic coordinate system for fingerprint matching. In *Proc. Int'l Conf. on Audio- and Video-based Biometric Person Authentication*, pages 198–204.

Bazen, A. M. and Gerez, S. H. (2001). Segmentation of fingerprint images. In *PRORISC 2001 WORKSHOP ON CIRCUITS, SYSTEMS AND SIGNAL PROCESSING*, pages 276–280.

Berlekamp, E. (1966). *Nonbinary BCH decoding.* Institute of Statistics mimeo series. University of North Carolina. Dept. of Statistics.

Berlekamp, E. R. (1984). *Algebraic coding theory.* Aegean Park Press, Laguna Hills, CA, USA.

Berlekamp, E. R., McEliece, R. J., and Van Tilborg, H. C. A. (1978). On the inherent intractability of certain coding problems. *IEEE Trans. Inf. Theory*, 24.

Besl, P. J. and McKay, N. D. (1992). A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256.

Blahut, R. (2003). *Algebraic Codes for Data Transmission.* Cambridge University Press.

Bleichenbacher, D. and Nguyen, P. Q. (2000). Noisy polynomial interpolation and noisy chinese remaindering. In *Proc. of the 19th Int'l Conf. on Theory and application of cryptographic techniques*, EUROCRYPT'00, pages 53–69, Berlin, Heidelberg.

Bose, R. C. and Ray-Chaudhuri, D. K. (1960). On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79.

Bose, R. C. and Shrikhande, S. S. (1959). A note on a result in the theory of code construction. *Information and Control*, 2(2):183–194.

Brassard, G., Chaum, D., and Crépeau, C. (1988). Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189.

Brent, R., Gaudry, P., Thome, E., and Zimmermann, P. (2010). GF2X: a library for multiplying polynomials over the binary field, version `1.0`. available from `http://gforge.inria.fr/projects/gf2x/`.

Bringer, J., Chabanne, H., Izabachène, M., Pointcheval, D., Tang, Q., and Zimmer, S. (2007a). An Application of the Goldwasser-Micali Cryptosystem to Biometric Authentication. In Pieprzyk, J., Ghodosi, H., and Dawson, E., editors, *The 12th Australasian Conf. on Information Security and Privacy (ACISP '07)*, volume 4586 of *Lecture Notes in Computer Science*, pages 96–106, Townsville, Queensland, Australia. Springer.

Bringer, J., Chabanne, H., Pointcheval, D., and Tang, Q. (2007b). Extended private information retrieval and its application in biometrics authentications. In *Proc. of the Int'l Conf. on Cryptology and Network Security (6th)*, CANS'07, pages 175–193, Berlin, Heidelberg. Springer-Verlag.

Buchmann, J. (2003). *Einführung in die Kryptographie.* Springer-Verlag, Berlin, third edition.

Cavoukian, A. and Stoianov, A. (2009). *Biometrics: theory, methods, and applications*, chapter 26 - Biometric Encryption: The New Breed of Untraceable Biometrics. John Wiley & Sons, Inc., Hoboken, NJ, USA.

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

Chen, Y., Dass, S., and Jain, A. (2005). Fingerprint quality indices for predicting authentication performance. In *Proc. Int'l Conf. Audio- and Video-Based Biometric Person Authentication (5th)*, pages 160–170.

Chetverikov, D., Svirko, D., Stepanov, D., and Krsek, P. (2002). The trimmed iterative closest point algorithm. In *Proc. Int'l Conf. on Pattern Recognition*, pages 545–548.

Chien, R. (1964). Cyclic decoding procedures for bose- chaudhuri-hocquenghem codes. *IEEE Trans. Inf. Theory*, 10:357–363.

Ching-Lung, C., Szu-Lin, S., and Shao-Wei, W. (2006). Decoding the (23, 12, 7) golay code using a low-complexity scheme. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 89-A(8):2235–2238.

Clancy, T. C., Kiyavash, N., and Lin, D. J. (2003). Secure smartcardbased fingerprint authentication. In *Proc.s of the 2003 ACM SIGMM workshop on Biometrics methods and applications*, WBMA '03, pages 45–52, New York, NY, USA. ACM.

Clopper, C. J. and Pearson, E. S. (1934). The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):pp. 404–413.

Cohn, H. and Heninger, N. (2011). Ideal forms of coppersmith's theorem and guruswami-sudan list decoding. In *Innovations in Computer Science - ICS 2010*, pages 298–308.

Cormen, T., Leierson, C., Rivest, R., and Stein, C. (2001). *Introduction To Algorithms*. MIT Press, Cambridge, Massachusetts, 2 edition.

Daemen, J. and Rijmen, V. (2002). *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer.

Dass, S. C. and Jain, A. K. (2004). Fingerprint classification using orientation field flow curves. In *Proc. Indian Conf. on Computer Vision, Graphics and Image Processing*, pages 650–655.

Daugman, J. (2004). How iris recognition works. *IEEE Trans. Circuits Syst. Video Technol.*, 14(1):21–30.

Davida, G. I., Frankel, Y., and Matt, B. J. (1998). On enabling secure applications through off-line biometric identification. In *IEEE Symp. on Security and Privacy*, pages 148–157. IEEE Computer Society.

Dodis, Y., Ostrovsky, R., Reyzin, L., and Smith, A. (2008). Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139.

Eggert, D. W., Lorusso, A., and Fisher, R. B. (1997). Estimating 3-d rigid body transformations: a comparison of four major algorithms. *Mach. Vision Appl.*, 9:272–290.

El Gamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proc. of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA. Springer-Verlag New York, Inc.

Feistel, H. (1973). Cryptography and computer privacy. *Scientific American*, 228(5):15–23.

Feng, J. (2008). Combining minutiae descriptors for fingerprint matching. *Pattern Recognition*, 41(1):342–352.

Gao, S. (2002). A new algorithm for decoding reed-solomon codes. In *Communications, Information and Network Security, V.Bhargava, H.V.Poor, V.Tarokh, and S.Yoon*, pages 55–68. Kluwer.

Gao, S., Shokrollahi, A., and Joyner, D. (1999). Computing roots of polynomials over function fields of curves. In *Proc. of the Annapolis Conference on Number Theory, Coding Theory, and Cryptography*, pages 214–228. Springer-Verlag.

Gilat, A. (2011). *MATLAB: An Introduction with Applications*. John Wiley & Sons Ltd, 4 edition.

Golay, M. J. E. (1949). Notes on digital coding. *Proc. IRE*, 37:657.

Goldwasser, S. and Micali, S. (1982). Probabilistic Encryption and How To Play Mental Poker Keeping Secret All Partial Information. In *Proc. 14th ACM Symp. on Theory of Computing*, pages 270–299. ACM.

Gorenstein, D., Peterson, W. W., and Zierler, N. (1960). Two-error correcting bose-chaudhuri codes are quasi-perfect. *Information and Control*, 3(3):291–294.

Gottschlich, C. (2011). *Fingerprint Growth Prediction, Image Preprocessing and Multi-level Judgment Aggregation.* PhD thesis, University of Göttingen, Germany.

Gottschlich, C., Mihailescu, P., and Munk, A. (2009). Robust orientation field estimation and extrapolation using semilocal line sensors. *IEEE Trans. Inf. Forensics Security*, 4:802–811.

Gray, F. (1953). Pulse code communication. US Patent 2,632,058.

Guruswami, V. and Sudan, M. (1998). Improved decoding of reed-solomon and algebraic-geometric codes. *IEEE Trans. Intell. Transp. Syst.*, 45:1757–1767.

Guruswami, V. and Vardy, A. (2005). Maximum-likelihood decoding of reed-solomon codes is np-hard. In *Proc. of the ACM-SIAM Symp. on Discrete algorithms (16th)*, SODA '05, pages 470–478, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Hanley, J. and Lippman-Hand, A. (1983). If nothing goes wrong, is everything allright? interpreting zero numerators. *Journal of the American Medical Association*, 249(13):1743–1745.

Hao, F., Anderson, R., and Daugman, J. (2006). Combining crypto with biometrics effectively. *IEEE Trans. Comput.*, 55(9):1081–1088.

Hasse, H. (1936). Theorie der höheren Differentiale in einem algebraischen Funktionenkörper mit vollkommenem Konstantenkörper bei beliebiger Charakteristik. *J. Reine Angew. Math.*, 175:50–54.

Hocquenghem, A. (1959). Codes correcteurs d'erreurs. *Chiffres (paris)*, 2:147–156.

Hong, L., Wan, Y., and Jain, A. K. (1998). Fingerprint image enhancement: Algorithm and performance evaluation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(8):777–789.

Hotz, T. (2007). *Modelling and Analysing Orientation Fields of Fingerprints.* PhD thesis, University of Göttingen, Germany.

Huckemann, S., Hotz, T., and Munk, A. (2008). Global models for the orientation field of fingerprints: An approach based on quadratic differentials. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(9):1507–1519.

Jain, A., Hong, L., and Bolle, R. (1997). On-line fingerprint verification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19:302–314.

Jain, A. K., Flynn, P., and Ross, A. A. (2007). *Handbook of Biometrics.* Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Jeffers, J. and Arakala, A. (2007). Fingerprint Alignment for a Minutiae-Based Fuzzy Vault. In *Proc. Biometrics Symp.*, pages 1–6.

Jiang, X. and Yau, W.-y. (2000). Fingerprint minutiae matching based on the local and global structures. *Proc. of the 15th Int'l Conf. on Pattern Recognition ICPR*, 2:1038–1041.

Jovanovic, B. D. and Levy, P. S. (1997). A look at the rule of three. *The American Statistician*, 51(2):137–139.

Juels and Sudan (2002). A fuzzy vault scheme. In Lapidoth, A. and Teletar, E., editors, *Proc. IEEE Int'l Symp. Inf. Theory*, page 408.

Juels and Wattenberg (1999). A fuzzy commitment scheme. In *CCS '99: Proc. of the 6th ACM Conf. on Computer and Communications Security*, pages 28–36, New York, NY, USA. ACM.

Kamei, T. and Mizoguchi, M. (1995). Image filter design for fingerprint enhancement. In *Proc. Int'l Symp. on Computer Vision*, pages 109–114.

Kass, M. and Witkin, A. P. (1987). Analyzing oriented patterns. *Computer Vision, Graphics, and Image Processing*, 37(3):362–385.

Kawagoe, M. and Tojo, A. (1984). Fingerprint pattern classification. *Pattern Recognition*, 7:296–303.

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

Kholmatov, A. and Yanikoglu, B. (2008). Realization of correlation attack against the fuzzy vault scheme. In *Proc. SPIE*.

Kholmatov, A., Yanikoglu, B. A., Savas, E., and A., L. (2006). Secret sharing using biometric traits. In *Proc. of SPIE, Biometric Technology For Human Identification III, 6202, 18 April.*

Kiayias, A. and Yung, M. (2008). Cryptographic hardness based on the decoding of reed-solomon codes. *IEEE Trans. Inf. Theory*, 54(6):2752–2769.

Knuth, D. E. (1981). *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition.* Addison-Wesley.

Koetter, R. and Vardy, A. (2003a). Algebraic soft-decision decoding of reed-solomon codes. *IEEE Trans. Inf. Theory*, 49(11):2809–2825.

Koetter, R. and Vardy, A. (2003b). A complexity reducing transformation in algebraic list decoding of reed-solomon codes. In *Proc. of IEEE Information Theory Workshop*, pages 200–203.

Li, J., Yang, X., Tian, J., Shi, P., and Li, P. (2008). Topological structure-based alignment for fingerprint Fuzzy Vault. In *Proc. Int'l Conf. on Pattern Recognition*, pages 1–4.

Li, P., Yang, X., Cao, K., Shi, P., and Tian, J. (2009). Security-enhanced fuzzy fingerprint vault based on minutiae's local ridge information. In *Proc. Int'l Conf. on Advances in Biometrics*, ICB '09, pages 930–939, Berlin, Heidelberg. Springer-Verlag.

Li, P., Yang, X., Cao, K., Tao, X., Wang, R., and Tian, J. (2010). An alignment-free fingerprint cryptosystem based on fuzzy vault scheme. *J. Netw. Comput. Appl.*, 33:207–220.

Maio, D., Maltoni, D., R., C., J.L., W., and A.K., J. (2002). FVC2002: Second Fingerprint Verification Competition. In *Proc. Int'l Conf. on Pattern Recognition (16th)*, pages 811–814.

Maltoni, D., Maio, D., Jain, A. K., and Prabhakar, S. (2009). *Handbook of Fingerprint Recognition.* Springer Publishing Company, Incorporated, 2nd edition.

Massey, J. L. (1969). Shift-register synthesis and bch decoding. *IEEE Trans. Inf. Theory*, 15(1):122–127.

McEliece, R. (2003). The guruswami-sudan decoding algorithm for reed-solomon codes.

Merkle, J. et al. (2010a). Provable security for the fuzzy fingerprint vault. In *Proc. Int'l Conf. on Internet Monitoring and Protection*, ICIMP '10, pages 65–73, Washington, DC, USA. IEEE Computer Society.

Merkle, J., Ihmor, H., Korte, U., Niesing, M., and Schwaiger, M. (2010b). Performance of the fuzzy vault for multiple fingerprints (extended version). *CoRR*, abs/1008.0807.

Mieloch, K. (2008). *Hierarchically linked extended features for fingerprint processing.* PhD thesis, University of Goettingen, Germany.

Mieloch, K., Munk, A., and Mihailescu, P. (2008). Hierarchically linked extended features for fingerprint processing. In *Proc. SPIE.*

Mihăilescu, P. (2007). The fuzzy vault for fingerprints is vulnerable to brute force attack. *CoRR*, abs/0708.2974.

Mihăilescu, P., Munk, A., and Tams, B. (2009). The fuzzy vault for fingerprints is vulnerable to brute force attack. In *BIOSIG*, pages 43–54.

Nagar, A., Nandakumar, K., and Jain, A. K. (2008). Securing fingerprint template: Fuzzy vault with minutiae descriptors. In *Proc. Int'l Conf. on Pattern Recognition.*

Nagar, A., Nandakumar, K., and Jain, A. K. (2010). A hybrid biometric cryptosystem for securing fingerprint minutiae templates. *Pattern Recogn. Lett.*, 31:733–741.

Nandakumar, K., Jain, A. K., and Pankanti, S. (2007a). Fingerprint-based fuzzy vault: Implementation and performance. *IEEE Trans. Inf. Forensics Security*, 2(4):744–757.

Nandakumar, K., Nagar, A., and Jain, A. K. (2007b). Hardening fingerprint fuzzy vault using password. In *Proc. Int'l Conf. on Biometrics*, LNCS 4642, pages 927–937.

Nandakumar, K. (2008). *Multibiometric systems: Fusion strategies and template security*. PhD thesis, Department of Computer Science and Engineering, Michigan State University.

National Institute of Standards and Technology (1995). Announcing the Secure Hash Standard. available online `http://csrc.nist.gov/`.

National Institute of Standards and Technology (1999). *FIPS PUB 46-3: Data Encryption Standard (DES)*. National Institute of Standards and Technology. supersedes FIPS 46-2.

National Institute of Standards and Technology (2001). *FIPS PUB 197: Announcing the Advanced Encryption Standard (AES)*. National Institute for Standards and Technology.

National Institute of Standards and Technology (2002). *FIPS 180-2: Secure Hash Standard*. National Institute for Standards and Technology.

Neurotechnology Ltd (2006). Verifinger SDK 5.0. `http://www.neurotechnology.com`.

Ortega-Garcia, J., Fierrez-Aguilar, J., Simon, D., Gonzalez, J., Faundez-Zanuy, M., Espinosa, V., Satue, A., Hernaez, I., Igarza, J. J., Vivaracho, C., Escudero, D., and Moro, Q. I. (2003). MCYT baseline corpus: a bimodal biometric database. *IEE Proc. on Vision, Image and Signal Processing*, 150(6):395–401.

Papadimitriou, C. H. (1994). *Computational complexity*. Addison-Wesley.

Ratha, N. K., Chen, S., and Jain, A. K. (1995). Adaptive flow orientation-based feature extraction in fingerprint images. *Pattern Recognition*, 28(11):1657–1672.

Ratha, N. K., Pandit, V. D., Bolle, R. M., and Vaish, V. (2000). Robust fingerprint authentication using local structural similarity. In *Proc. Workshop on applications of Computer Vision*, pages 29–34.

Reed, I. S. and Solomon, G. (1960). Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304.

Rerkrai, K. and Areekul, V. (2000). A new reference point for fingerprint recognition. In *Proc. Int'l Conf. on Image Processing*.

Rivest, R. L. (1991). The md4 message digest algorithm. In *Proc. of the 10th Annual Int'l Cryptology Conference on Advances in Cryptology*, CRYPTO '90, pages 303–311, London, UK, UK. Springer-Verlag.

Rivest, R. L., Shamir, A., and Adleman, L. (1983). A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26:96–99.

Roth, R. (2006). *Introduction to Coding Theory*. Cambridge University Press.

Roth, R. M. and Ruckenstein, G. (2000). Efficient decoding of reed-solomon codes beyond half the minimum distance. *IEEE Trans. Inf. Theory*, 46:246–257.

Sahai, A. and Waters, B. (2005). Fuzzy Identity-Based Encryption. *Advances in Cryptology – EUROCRYPT 2005*, pages 457–473.

Sarier, N. (2011). *Biometric Cryptosystems: Authentication, Encryption and Signature for Biometric Identities.* PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, Germany.

Scheirer, W. J. and Boult, T. E. (2007). Cracking fuzzy vaults and biometric encryption. In *Proc. of Biometrics Symp.*, pages 1–6.

Sharp, G. C., Lee, S. W., and Wehe, D. K. (2002). Icp registration using invariant features. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(1):90–102.

Sherlock, B. G. and Monro, D. M. (1993). A model for interpreting fingerprint topology. *Pattern Recognition*, 26(7):1047–1055.

Shoup, V. (2005). *A Computational Introduction to Number Theory and Algebra.* Cambridge University Press.

Shoup, V. (2009). NTL: A library for doing number theory, version `5.5.2`. available from `http://www.shoup.net/ntl/`.

Soutar, C., Roberge, D., Stoianov, A., Gilroy, R., and Kumar, B. (1998a). Biometric encryption: Enrollment and verification procedures. In *Proc. of SPIE*, volume 3386, pages 24–35.

Soutar, C., Roberge, D., Stoianov, A., Gilroy, R., and Kumar, B. (1998b). Biometric encryption using image processing. In *Proc. of SPIE*, volume 3314, pages 178–188.

Stroustrup, B. (2000). *The C++ Programming Language.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition.

Sudan, M. (1997). Decoding of reed solomon codes beyond the error-correction bound. *Journal of Complexity*, 13:180–193.

Joachim von zur Gathen, Jürgen Gerhard (2003). *Modern Computer Algebra.* Cambridge University Press, Cambridge (UK), second edition.

The GMP Team (2011). GMP*: the GNU Multiple Precision Arithmetic Library, version* `5.0.2`. avaiable from `http://gmplib.org/`.

Tico, M. and Kuosmanen, P. (2003). Fingerprint matching using an orientation-based minutia descriptor. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25:1009–1014.

Tomko, G., Soutar, C., and Schmidt, G. (1994). Fingerprint controlled public key cryptographic system. US Patent 5,541,994.

Trifonov, P. (2010). Efficient interpolation in the guruswami-sudan algorithm. *IEEE Trans. Inf. Theory*, 56(9):4341–4349.

Uludag, U. and Jain, A. K. (2006). Securing fingerprint template: fuzzy vault with helper data. In *Proc. IEEE Workshop on Privacy Research In Vision*, pages 163–169.

Uludag, U., Pankanti, S., and Jain, A. (2005). Fuzzy vault for fingerprints. In *Proc. Int'l Conf. on Audio- and Video-Based Biometric Person Authentication*, pages 310–319.

Welch, L. R. and Berlekamp, E. R. (1983). Error correction for algebraic block codes. US Patent 4,633,470.

Yang, S. and Verbaudwhede, I. (2005). Automatic secure fingerprint verification system based an fuzzy vault scheme. In *Proc. Int'l Conf. on Acoustics, Speech and Signal Processing*, pages 609–612.

Zhang, T. Y. and Suen, C. Y. (1984). A fast parallel algorithm for thinning digital patterns. *Commun. ACM*, 27:236–239.

Index

A

advanced encryption standard 15
AES . . . . *see* advanced encryption standard
ambient space 28
ambient vector 28
angle
  ∼ of a minutia 18
arch 26
attack via record multiplicity . . 57, 73

B

BCH code . . . . *see* Bose-Chaudhuri-Hocquenghem code
bifurcation
  ride ∼ 18
binary Golay code 30
binomial distribution
  ∼ function 22
  probability mass function of the ∼ 22
block cipher
  asymmetric ∼ 14
  symmetric ∼ 14
Bose-Chaudhuri-Hocquenghem code 29
brute-force attack
  ∼ against the fuzzy vault 39
  decoupling ∼ against the fuzzy vault 65

C

check-sum 13
Clopper-Pearson interval 22
code locators
  ∼ of a Reed-Solomon code 31
collision 15
  strong ∼ resistance 15, 33
  weak ∼ resistance 15
confidence interval 22
confidence level 22
control matrix
  ∼ of a linear code 28
core point 25
correlation attack 75
cross-matching 57, 73
cryptographic hash function 15
cryptographic system 14
cryptography 13
cryptosystem 14

D

data encryption standard 14
decoupling attack 63
decoupling brute-force attack
  ∼ against the fuzzy vault 65
decryption function 14
delta point 25
DES . . . . *see* data encryption standard
digital signature 13
direction
  ∼ of a minutia 18
double loop 26

E

encryption function 14
encryption key 14
ending
  ride ∼ 18
error-correcting code 28
  linear ∼ 28
error-correction bound 29

F

false acceptance rate 21
false match rate . . *see* false acceptance rate
false non-match rate 21
false-accept attack 7, 56
false-acceptance rate 7
feature-level fusion 7
Feistel cipher 14
five nearest neighbor minutiae structure 45

Fourier transform
  two-dimensional ∼ 5
frequency field 25
fuzzy commitment scheme 3, 32
fuzzy vault scheme 4

G
Gabor filter 25
generator matrix 28
genuine acceptance rate 21
genuine authentication attempt 21
genuine match rate . . . . *see* genuine acceptance rate
genuine set 37
global matching 21
Golay code
  binary ∼ 30
  ternary ∼ 30
guaranteed error-correction bound . 29

H
hamming code 30
hamming distance 28
hamming weight 28
hash function 15
  cryptographic ∼ 15
hypothetical false acceptance rate 103
hypothetical genuine acceptance rate 103

I
impostor authentication attempt . . . 21
inter-ridge distance 25
intrinsic coordinate system 69
iris code 33

L
left loop 26
libthimble . . . . *see* thimble library
linear code 28
liveness detection 6
local matching 21
local structure
  minutiae ∼ 47
locators
  ∼ of a Reed-Solomon code 31

M
maximum likelihood decoding 29
minimal distance 28
minutia 18
  ∼ angle . . . . . *see* minutia direction
  ∼ direction 18
  triangle based ∼ structure 46
  undirected ∼ 46
minutiae
  ∼ local structure 47
  ∼ template 18
  five nearest neighbor ∼ structure 45
  Voronoi ∼ neighbor structure . . 46
minutiae descriptor 7, 26
minutiae local structure 21
MLD . . . . *see* maximum likelihood decoding
modified vault construction 96

N
non-invertible transformation 2
NTL . . . . . *see* Number Theory Library
Number Theory Library 127

O
one-way function 15
orientation field 25

P
parity check matrix 28
password
  ∼ salt 16, 87
perfect code 7
personal identification number 1
PIN *see* personal identification number
Poincaré index 25
point estimation 22
polynomial reconstruction problem 38
private template 5
public-key cryptography 13

Q
quality index 18

R
rainbow-table 16
rate
  ∼ of a linear code 28
redundancy
  ∼ of a linear code 28
Reed-Solomon code 31
repetition code 30

ridge
  ∼ bifurcation 18
  ∼ ending 18
ridge curvature 25
ridge frequency 25
right loop 26
Rijndael cipher 15
Rivest-Shamir-Adleman cryptosystem 14
RSA . . . . . *see* Rivest-Shamir-Adleman cryptosystem
rule of three 23

S
salt
  password ∼ 16, 87
secure hash algorithm 16
secure hash standard 16
SHA . . . . *see* secure hash algorithm
SHS . . . . *see* secure hash standard
singleton bound 31
singular point 25
sphere packing bound 30
sphere packing density 30
sphere-packing density 7
strong collision resistance 15, 33
strongly binding 33
Sudan algorithm 32
syndrome 29
system users 2

T
ternary Golay code 30
thimble library 127
topo-structure 46
triangle based minutiae structure . . 46

U
undirected minutia 46
unlocking set 38
user 2

V
Voronoi neighbor minutiae structure 46

W
weak collision resistance 15
whorl 26

Curriculum Vitae

Berend-Benjamin Tams
born 9$^{\text{th}}$ July 1981 in Göttingen, Germany

September 1988 – June 2001
Schooling
General Qualification for University Entrance
Berufsbildende Schulen Osterholz-Scharmbeck (Fachgymnasium Technik)

September 2001 – June 2002
Civil Service
Kreiskrankenhaus OHZ in *Osterholz-Scharmbeck*

October 2002 – December 2008
Study of Mathematics and Computer Science (minor)
Faculty of Mathematics, University of Göttingen
diploma thesis: *Diskreter Logarithmus mittels Zahlkörpersieb (in Erweiterungskörpern)*
supervised by *Prof. Dr. Preda Mihăilescu*

April 2005 – March 2008
Student Assistant
Faculty of Mathematics, University of Göttingen

since January 2009
Ph.D. Studies in Mathematics
Faculty of Mathematics, University of Göttingen
supervised by *Prof. Dr. Preda Mihăilescu*

January 2009 – December 2011
member of the *DFG Graduate Program 1023*
"Identification in Mathematical Models"

since January 2012
Research Assistant (*wissenschaftliche Hilfskraft*)
Institute for Mathematical Stochastics, University of Göttingen