Spring 2013

Fingerprint fuzzy vault: Security analysis and a new scheme

Patrick J.B. Perry
James Madison University

Fingerprint Fuzzy Vault: Security Analysis and a New Scheme

Patrick JB Perry

A thesis submitted to the Graduate Faculty of

JAMES MADISON UNIVERSITY

In

Partial Fulfillment of the Requirements

for the degree of

Master of Science

Department of Computer Science

May 2013

Dedication

This work is dedicated to my children Noah, Aden and Allegra. Without your constant questions and interruptions, this would have been a far easier but less enjoyable process.

I love you all. Never stop asking questions.

Dad

Acknowledgments

First, I would like to thank my advisor, Dr. Xunhua Wang, for his expertise, patience and friendship this year. Knowing I would have the opportunity to work with a person that is as dedicated to his students as much as his craft made the pursuit of this thesis a worthwhile and enjoyable endeavor. I would also like to thank Dr. Mohammed Heydari and Dr. Brett Tjaden for serving on my thesis committee. I have been fortunate to have a class with each of them over the last two years and am delighted they were so willing to offer me their help.

Finally, I need to thank my wife, Melissa. Without her love, support, brilliance and amazing trigonometric insights this would not have been possible.

Table of Contents

List of Figures

Abstract

A fingerprint fuzzy vault uses a fingerprint A to lock a strong secret k and only a close fingerprint from the same finger can be used to unlock k. An attacker who has stolen the vault will *not* be able to get useful information about A or k.

In this research, we shall study the security of a major fingerprint fuzzy vault developed by Nandakumar et al. through investigating the security implication of helper data, which are stored in the fuzzy vault for fingerprint alignment. We will show that helper data leak information about fingerprints and thus compromise the security claim on the fingerprint fuzzy vault scheme. Next, we will propose a new fingerprint fuzzy vault scheme, which is based on traditional representation of fingerprints in minutia points and does not need helper data for alignment.

Keywords: fuzzy vault, helper data, fingerprint authentication, biometric authentication

Chapter 1

Introduction

Overview

Entity Authentication

The notion of authentication is central to secure computing. In recent years there has been a marked increase in all forms of computer crime and digital malfeasance. A common approach to attempt to combat this problem is through the use of passwords to restrict access to systems and/or data. This is the most common form of entity authentication. That is verifying an individual is whom they claim to be based on something that they know. This form of authentication has many significant drawbacks. For instance, users tend to employ passwords that are as simple and as easy to remember as possible. Additionally, passwords are often reused for multiple systems. These facts are all the more troubling in light of security researcher, Moxie Marlinspike, releasing Chapcrack. Chapcrack reduces the overall effectiveness of MS-CHAPv2 into a single DES operation and proceeds to break this encryption within twenty-four hours [13]. It is understandable why those in the security industry are leery of passwords.

Authentication based on something the user knows is but one form of entity authentication. There are also issues that arise when using authentication based on something a user has. This introduces the problem of a user not always having his/her token with him/her for authenticating or the possibility of the token being stolen. This paradigm shifts the authentication problem from a user's mind to that of a physical token. Because of the fundamental flaws with each of these approaches we have seen a push in more recent time to biometric authentication – authentication based on something a user is. This biometric authentication is commonly regarded as stronger than more traditional mechanisms, such as password based authentication. This mechanism is more reliable as it is composed of things that are not easily forgotten or misplaced. It has the added advantage of being difficult to forge another user's traits. There are several possible biometric traits which can be used for authentication. These include types such as signature dynamics, typing patterns, retinal scans, voice recognition, facial recognition, palm geometry and fingerprint recognition. It is this last type of biometric trait that we will concern ourselves with here.

General Fuzzy Vault

While many times it is useful to require a cryptographic algorithm to depend on an exact match, there are times that an exact match will not work. When requiring a password, it is easy to obtain an exact match. If you allowed variation in the acceptance of a password you would greatly reduce its security. However, there are situations when an exact match is neither possible nor required for adequate security. For example, when using biometrics you rely on something from a human, this might be a scan of a fingerprint, a typing pattern or voice recognition. When relying on humans there will always be some sort of variation and therefore exact matching is impractical. A *general fuzzy vault* is a cryptographic system to address this issue of needing a close-enough match [6, 7, 3, 4]. A general fuzzy vault takes a non-ordered set of "target" integers and stores them along with a non-ordered set of random integers in a vault. When trying to unlock the secret, it is considered a match as long as the supplied target set is close enough to the stored target set.

Fingerprint Fuzzy Vault

The general fuzzy vault provides a foundation for a cryptographic system that deals with fingerprints but it needs to be adapted in order to be effective.

In one such fingerprint fuzzy vault [14], called *NJP07* hereafter, the following adaptations are used in the fingerprint fuzzy vault. When processing a fingerprint, minutia points are used. These points are made up of three components, a horizontal location (x), a vertical location (y) and an angle (θ). To use a fuzzy vault these three numerical values are concatenated. However, it is still important to understand that for fingerprints to match, it is not necessary for the three components to match exactly. Given two minutia (x_1, y_1, θ_1) and (x_2, y_2, θ_2) the spatial distance between (x_1, y_1) and (x_2, y_2) must be small enough (not larger than a distance threshold) and the directional distance between (θ_1) and (θ_2) must also be small enough (not larger than a directional threshold). As long as those values are close enough it is considered a match and the exact matching of integers is not required. Additionally, the genuine minutia points used for the vault are selected such that they are adequately spaced out.

Helper Data

In the fingerprint fuzzy vault scheme developed in *NJP07* [14], another adaptation used in the fingerprint fuzzy vault is the use of helper data. In order to compare the minutia points of two fingerprints it is necessary to align them. Helper data is used to do this. The fingerprint is processed to find a set of high curvature points that is used to align the two images. This helper data is publicly available so it is important that knowledge of this data does not decrease the security of the vault. The argument will be made here that this helper data does in fact decrease the security of the vault. Therefore, it will be proposed that a fuzzy vault, which does not use helper data, is a better choice.

Problem Statement

This thesis research aims to answer the following two questions:

1. In *NJP07*, if an attacker has stolen a copy of the fuzzy vault, how much useful information can be retrieved from the helper data about the genuine minutia points? In other words, do the helper data leak useful information about the fingerprint?

2. If helper data do leak much information, how to develop a secure solution to fix this flaw?

Contributions

The results of this thesis research are two-fold.

1. We show that helper data contains useful information, such as the orientation, that can be used by an attacker to filter chaff points from the fuzzy vault. This is contrary to what has been claimed in *NJP07* [14] and as a result, *NJP07* does not achieve the security level that it claims.

2. We develop a new fuzzy vault scheme that does not require helper data for alignment. Our scheme is based on the traditional representation of genuine minutia points that have been well tested by the community.

Organization

The remainder of this thesis is organized as follows. Chapter 2 gives background information and related work. In Chapter 3, we analyze the security of helper data. Chapter 4, we present a new fingerprint fuzzy vault scheme that does not use helper data. Finally, concluding remarks are given in Chapter 5.

Chapter 2

Background Information and Related Work

In this section, we give a more detailed description of the information on fingerprint authentication, the general fuzzy vault *JS02*, the fingerprint fuzzy vault *NJP07*, and an alternate fingerprint matcher that we propose for implementation in a new fingerprint fuzzy vault scheme.

Fingerprint Authentication

Minutia points are unique to a finger just as a fingerprint is. This is because these points are composed of unique areas of a fingerprint. In order to understand why these unique areas exist we need to consider what makes up a fingerprint. A human fingerprint has friction *ridges* with the space between ridges known as *valleys*. Together, these ridges and valleys form patterns, which also include special areas [12]. It is these special areas that are the fingerprint's minutia points. A minutia point may be the *ending* of a ridge (ridge ending) or where a ridge splits into two. This splitting of the ridge is known as a ridge *bifurcation*. For fingerprint authentication a fingerprint can be modeled as the collection of its minutia points. We represent these points with their (x, y) coordinate pair, the directional angle θ and the quality of the point.

In order for a user to be authenticated, the user first needs to enroll a fingerprint with an authentication server. Upon registration, this server obtains the user's fingerprint. This fingerprint image is then used to extract its minutia points and in creating a *reference template* which is stored. This is necessary to authenticate the user at a later time. When this user swipes his/her fingerprint the minutia points can be extracted and compared to those stored in the authentication server's reference templates [12, 8].

Minutia point extraction from a fingerprint image begins with a sequence of preprocessing steps [12]. These steps are *normalization*, *orientation image estimation*, *frequency image estimation*, *region mask generation*, and *filtering to remove noise*. After preprocessing, the image is then binarized and thinned. Binarization is the process by which each pixel has a value of 0 or 1. Thinning is the process where it is ensured that ridges are only one pixel wide. It is at this point that minutia points may be detected. These points still require a certain amount of post-processing. This post-processing is necessary to remove incorrect minutia points as well as those too close to the edge of the image or too close to others.

Fingerprint matching algorithms based on minutia points are different than normal password authentication. This is because the matching of fingerprints is an inexact science. In matching fingerprints a threshold must be met for a match to be considered as occurring. The security of this changes with the value of the threshold. That is to say the higher the threshold for minutia points, the more secure it will be. However, as the threshold value increases so does the number of false negatives, making it more difficult to match the fingerprints.

General Fuzzy Vault JS02

Juels and Sudan [6, 7] gave the first general fuzzy vault scheme, *JS02*, for the set difference metric, under which two sets A and B are considered a match if their set difference is smaller than a given value d. The order of the sets does *not* matter in determining if they match.

A *JS02* vault consists of a set of points and the number of points, r, is determined by the security level to be achieved. Let n be the number of elements in set $A = \{a_1, a_2, \ldots, a_n\}$, from which the vault will be constructed, and d be the maximum number of errors tolerated. (Both n and d are system-wide parameters; d is also the maximum set difference between A and any close set $B = \{b_1, b_2, \ldots, b_n\}$.)

Define $t = (n - 2 \times d)$. This scheme requires a finite field with q elements, where $r \leq q$. This finite field is denoted as F_q and it can be either a prime field (where q is a prime number and $F_q = \{0, 1, 2, \ldots, q-1\}$) or a Galois field (where $q = 2^m$ for some integer m) [13,4].

This general fuzzy vault assumes that all set elements (i.e., a_i of A and b_i of B) are integers and it works as follows:

- Vault encoding: Let k be the secret to be protected by the vault; the fuzzy vault is constructed from $A = \{a_1, a_2, \ldots, a_n\}$ where a_i are integers, as follows.

    1. Generate valid points:

(a) Split k into t pieces of equal size, $k_0, k_1, \ldots, k_{t-1}$, where $k_i, 1 \leq i \leq t-1$, is an element of F_q.

(b) Construct a polynomial degree $(t-1), p(x) = k_{t-1}x^{t-1} + k_{t-2}x^{t-2} + \ldots + k_1 t + k_0 \bmod q$.

(c) Calculate $\beta_i = p(a_i), 1 \leq i \leq n$. These points (a_i, β_i) are valid points and they form *locking set* S.

2. Generate chaff points: randomly select $(r-n)$ points $(\gamma_j, \zeta_j), 1 \leq j \leq (r-n)$, where γ_j and ζ_j are randomly selected from F_q with two conditions. First, $\gamma_j \neq a_i$. Second, $\zeta_j \neq p(\gamma_j)$; that is, (γ_j, ζ_j) are *not* on polynomial. All points (γ_j, ζ_j) form *chaff set* C.

3. The union of sets S and $C, P = S \cup C$, forms the points stored in the vault.

- Vault decoding: Let $B = \{b_1, b_2, \ldots, b_n\}$ be a fresh set and it can be used to unlock vault P if B is close to A.

1. Use each $b_i, 1 \leq i \leq n$, as the x-coordinate to search, in an exact manner, P for a point. Let V be the set of points found.

2. Apply the Reed-Solomon decoding algorithm to points in V to reconstruct a polynomial.

In a Reed-Solomon code with n-symbol codewords, up to d errors can be corrected and thus, when B is close to A (with set difference not larger than d), $p(x)$ and k can be reconstructed to decrypt the vault data.

Selection of r. Juels and Sudan [6, 7] also observed in their Lemma 4 [7] that, when chaff set C is randomly chosen, given any $\mu, 0 < \mu < 1$, with probability $(1-\mu)$, there are at least $\tau = \mu\binom{r}{n}q^{1-r}(q-1)^{r-n}$ polynomials similar to $p(x)$ (that is, each such polynomial has a degree of less than t and there are exactly n vault points on the polynomial). As a result, an attacker who has seized P will *not* be able to find out which of the polynomials is $p(x)$. This security is information-theoretic, as it does not depend on the attacker's computational power.

Dodis et al. [3, 4] further improved JS02 and also proposed a general fuzzy scheme based on the edit-distance metric.

Fingerprint Fuzzy Vault NJP07

As described earlier, the general fuzzy vault scheme *JS02* is not directly applicable to fingerprint applications for two reasons. First, a fingerprint is not a set of integers, but a set of minutia points represented by coordinates and angles. Second, unlike exact integer comparison, the comparison of minutia points is close, not exact.

Let a fingerprint reference template be $\bar{A} = \{\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_n\}$, where minutia point $\bar{a}_i = \{\bar{x}, \bar{y}\}$, are the coordinates of the minutia point and $\bar{\theta}$ is its angle/orientation. (The quality attribute of a minutia point is not used, as it is less reliable.)

Nandakumar et al. [14] developed the following adaptations to construct a fingerprint fuzzy vault *NJP07*.

1. Each minutia point is converted to an integer by concatenating its three values (coordinates and angle) together. In other words, $\bar{a}_i = \bar{x}_i||\bar{y}_i||\bar{\theta}_i$, where $||$ stands for string concatenation.

2. Using fuzzy vault *JS02* requires the two fingerprints being compared to be aligned. Nandakumar et al. [14] introduces a *helper data set* into fuzzy vault for this purpose. The helper data set of a fingerprint is composed of the high curvature points (coordinates and angle) of the image. This data is stored publicly in the fuzzy vault. The helper data set from each image is used to align the images properly. Nandakumar et al. [14] claims that storing helper data in the fuzzy vault does not compromise the security of the vault.

3. Third, Step 2 of the vault encoding in *JS02*is revised as follows: to generate a chaff point, a fake minutia point is randomly generated and is checked to make sure that its distance to all existing points in the vault is larger than $\delta_1 = 25$; this fake minutia point is then encoded into a chaff point and is added into the vault.

4. Fourth, fingerprint matching is often based on set intersection, not set difference. Two fingerprint samples $\bar{A}$ and B are considered matched when the number of their close points is larger than a similarity threshold $\bar{t}$. This shifting from set difference to set intersection is accomplished by testing each $\bar{t}$-subset of V to see whether $p(x)$ can be reconstructed. If one such $\bar{t}$-subset exists, then the set intersection condition is met and B is considered matched to $\bar{A}$.

Accordingly, the vault decoding step was completely revamped as follows. First, the abscissa values of all vault points are parsed into $\{\bar{x}_i, \bar{y}, \bar{\theta}_i\}$ and their distances to minutia points of B are calculated. Those vault points with distance larger than $\delta_2 = 30$ are deemed as chaff points and are ignored. For the set of remaining vault points, next, all $\bar{t}$-subset of this set are brute-forced to check whether $p(x)$ and k can be recovered. To test each $\bar{t}$-subset, the Reed-Solomon decoding is not used and instead, the Lagrange interpolation over finite field is used. This change allows a shift from the set difference metric to the set intersection metric, which is more appropriate for and widely used by fingerprint matching. In *NJP07*, through the use of δ_2, minutia points of $\bar{A}$ are compared with minutia points of B in a close, not exact manner. This change is necessary because of distortion in fingerprint images, which is one nature of fingerprint applications.

Security claim. Nandakumar et al. [14] performed an ad hoc security analysis on *NJP07*. In [14], $\delta_1 = 25, \delta_2 = 30, \bar{t} = 9$, (i.e, two fingerprints are considered matched if they share 9 or more close minutia points); the number of genuine minutia points n is chosen as 24 and the number of chaff points in a vault is 200. Under these parameters, Nandakumar et al. [14] claims that an attacker who has stolen a fingerprint fuzzy vault will need to perform 2.5×10^9 decoding attempts to unlock the vault.

BOZORTH3 Fingerprint Matcher

BOZORTH3 is an algorithm used to compute a score in comparing minutia points from any two fingerprints [21], assuming that these minutia points have already been extracted by other algorithms. A significantly high score indicates the two prints are a match. NIST modified this algorithm based on the work of Allan S. Bozorth while he was at the FBI. The NIST enhancements to Bozorth's algorithm are primarily technical fixes to correct memory leaks and increase the speed of the program.

BOZORTH3 uses the (x, y) location and orientation (θ) of the minutia points in determining if fingerprints match. It does this by generating a unique table for each print being compared that stores the distance between minutia points as well as the orientation between them. In determining whether or not one fingerprint matches another, that prints table must be compared to its counterpart's corresponding table. A match score is computed based on the amount of apparently congruent minutia point clusters.

This algorithm is of particular interest to our research as it is rotation and translation invariant. Our interest in this algorithm is clear. If helper data does in fact leak data, reducing the security of a fingerprint vault, then this algorithm offers a potential solution. Helper data are needed to align fingerprints before comparing their minutia points. An algorithm that is rotation and translation invariant would not rely on helper data and would allow for that vulnerability to be removed in the creation of a new fingerprint fuzzy vault scheme.

Other Related Fingerprint Fuzzy Vault Schemes

Chang and Li [1] studied how to generate chaff points to minimize entropy leaking. Kotlarchyk et al. [9] performed a *simulation* to determine acceptable parameters and thresholds for a fingerprint fuzzy vault based on *JS02*. They showed that accurate fingerprint alignment is crucial for such a fingerprint fuzzy vault.

Wang et al. [19] discussed how to speed up the decoding of a fingerprint fuzzy vault by taking advantage of the connection of Shamir secret sharing [17] and error-correcting code [11].

Through showing the insecurity of a fingerprint-protected USB drive, Rodes and Wang [16] showed the importance of fingerprint fuzzy vaults.

Li et al. [10] argued that high curvature points-based helper data do leak, especially in those fingerprint subareas close to the high curvature curves, but they did *not* give detailed security analysis.

Li et al. [10] proposed an alignment-free fingerprint fuzzy vault scheme based on known minutia description [18] and minutia local structure [5]. Unlike this scheme, the fingerprint fuzzy vault scheme proposed in Chapter 4 is based on the well-tested and well-accepted minutia points.

Chapter 3

Security Analysis on Helper Data

In this chapter, we describe our security analysis on helper data of the *NJP07* fingerprint fuzzy vault scheme.

Attack Model

In our security analysis, the adversary is assumed to have stolen a copy of the *NJP07* fuzzy vault and thus has both the helper data and all vault points, which is the union of valid points and chaff points. The adversary tries to use the helper data to differentiate valid points from chaff points.

Data Set

The prototype implementation of *NJP07* was performed on fingerprint database 2 of the 2002 fingerprint verification competition (FVC2002). This database consists of 100 fingers (numbered from 1 to 100) and 8 fingerprints (called *impressions*, numbered from 1 to 8) per finger. The image size of each fingerprint is 296×560 pixels. Nandakumar et al. [14] note that for each finger of FVC2002 database, fingerprints $1, 2, 7, 8$ were obtained with the cooperation of the finger owners and are in good quality. (In contrast, artificial displacement and rotation were used in obtaining fingerprints $3, 4, 5, 6$.) This good-quality subset of the FVC 2002 database 2 is referred to as *FVC02-db2-good* hereafter and was used by *NJP07*. The same fingerprint subset was used in our following experiments.

How Are Helper Data Generated

The first step in generating helper data is to find the orientation field of the fingerprint image. The orientation field provides the direction of the ridge flow at any given point. Once the orientation field is found, we must generate the orientation field flow curve (OFFC). In order to do this first you must generate a set of equidistant starting points (s_0 to s_n) across the fingerprint. Typically this is done in the middle (either horizontally or vertically) of the fingerprint. Each starting point is used to begin tracing a ridge line in opposite directions, ultimately generating multiple OFFCs that cover the fingerprint. For each OFFC we need to find the point with the greatest curvature. These high curvature points are then filtered and the remaining points become the helper data.

Our View

It is our view that a correlation can be drawn between the set of helper data and either minutia points and/or chaff points. That is to say that the high curvature points represented as helper data potentially leak much information thus reducing the overall security of the fuzzy vault.

Implementation

In order to begin exploring the correlation between helper data, minutia and chaff points, we first had to generate the helper data. We were able to use Matlab code that was already written to generate the orientation field. Once that was done we implemented our own code to generate the helper data. The following describes the method we used in that generation.

The first step was to generate the OFFCs. This was done by first finding the starting points. These points are spread out evenly along either the horizontal or vertical midline of the fingerprint. In order to begin the process, we found the starting points along the horizontal midline. This was done in the following manner: $s_{0i} = r_{start}k + w, c_{start} + lw$, where $k = 1, 2, \ldots, \frac{r_{end} - r_{start}}{w}, l = \frac{c_{end} - c_{start}}{2w}$, and $r_{start}, r_{end}, c_{start}, c_{end}$ are the top, bottom, left and right boundaries of the fingerprint and w is the sampling width which was 5.

Each of these starting points was then used to trace an OFFC. This was done by using s_0 as the starting point in the following equation: $s_j = s_{j-1} + d_j \times l_j \times o_{s_{j-1}}$ for $j = 1, 2, \ldots n$, d_j is -1 and 1 to trace the curves in opposite directions, l_j is the length between points which was set to 5, and $o_{s_{j-1}}$ is the orientation vector at s_{j-1} . When implementing this equation, boundaries were set so that the curve did not go beyond the limitations of $r_{start}, r_{end}, c_{start}, c_{end}$. Additionally, there was a maximum n (limiting how large j could be) set to control the number of points in the curve. This maximum n can fluctuate depending on the image. Different fingerprints will have varying OFFC lengths and if n is too large for an image it can create unnecessary noise. For our purposes, a value of 75 was used as it seemed to generate sufficient OFFCs. Further research on this topic may want to focus on developing a more robust algorithm which can adapt to a given ridge length. This would ensure that the entirety of a ridge was traced while eliminating noise that is generated from tracing beyond the edge of a ridge line due to a maximum that is too large.

We notice that this does not always generate a complete set of flow curves, so the process was repeated along the vertical midline of the fingerprint. The reason this did not always generate a complete set of flow curves is simple. When bisecting the image on its horizontal equator only those flow curves that flow through this region are traced. If a flow curve is located entirely above or below the equator without traversing it no starting point is engaged, thus the line does not get traced. The process was the same as the horizontal with changes made to k and l noted here: $k = \frac{r_{end}-r_{start}}{2w}, l = 1, 2, \ldots, \frac{c_{end}-c_{start}}{w}$. The same problem is encountered when the image is bisected with a vertical midline. Those flow curves existing entirely to the left or right of the midline without traversing it never have a starting point engaged and thus are never traced. Our solution to this problem was to use the OFFCs generated from both the horizontal and vertical starting points and then combine their results to create the final OFFCs. This allowed us to consistently generate OFFC's that covered large portions of the fingerprint images.

In order to find the points of the OFFC you must use orientation vectors. This is seen in the equation $s_j = s_{j-1} + d_j \times l_j \times o_{s_{j-1}}$, where $o_{s_{j-1}}$ is the orientation vector at point s_{j-1}. $o_{s_{j-1}} = (\cos\theta_{s_{j-1}}, \sin\theta_{s_{j-1}})$, where θ is the value taken from the orientation field at point s_{j-1}. There are areas of the OFFC where the points are more closely clustered than others. These areas occur near horizontal and vertical portions of the curve due to the nature of the sine and cosine functions along with the angles used.

θ represents the directional angle relative to the horizontal. Since opposite directions are equivalent, the only way to represent unique angles is to limit the angle domain to a range of π radians. Given this, the options are either angles from 0 to π or angles from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$. When using the angles 0 to π, you get stuck along the horizontal portions of the flow curves. The slope of the tangent line to a curve at a point close to a vertex of a curve will be close to horizontal. On either side of that vertex those close to horizontal slopes will have opposite values (one will be negative and the other positive). Given the angle restrictions this means that on one side of the vertex the directional angle will be very close to π while on the other side of the vertex the angle will be very close to zero. Using these angles with the equation given above you can see that once you get close to the vertex you will simply oscillate around the vertex rather than continue along the curve.

Explicitly, this is what occurs. We use the equation $s_j = s_{j-1} + d_j \times l_j \times o_{s_{j-1}}$ to get from one point in the curve to the next. Here d_j is either 1 or -1 depending on the direction from the starting point, for our purposes, let's consider it to be 1. The length between the points is l_j and that is 5. This leaves the following equation, $s_j = s_{j-1} + 5 \times o_{s_{j-1}}$. Since $o_{s_{j-1}} = (\cos\theta_{s_{j-1}}, \sin\theta_{s_{j-1}})$ the angle can now be used to demonstrate what occurs around the vertex. On one side the angle will be close to zero which will leave you with the following results: $s_j = s_{j-1} + 5 \times (1, 0)$, indicating the new point would be 5 units to the right of the point that came before it. If you are close to the vertex this lateral movement would bring you to the other side of the vertex and therefore the new angle would be close to π and would produce the following result: $s_j = s_{j-1} + 5 \times (-1, 0)$. This indicates the new point would be 5 units to the left of the previous point and therefore back to the original point we started with. This process continues until you reach the limit on the number of points allowed in the curve. Not only does this prevent you from moving beyond this section of the curve, it also generates many more points than should be in that area. This oscillation that occurs can be remedied by using angles from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$.

However, this causes the same sort of situation to occur along the vertical areas of the curve. When using the angles $-\frac{\pi}{2}$ to $\frac{\pi}{2}$, you get stuck along the vertical portions of the flow curves. The slope of the tangent line to a curve at a point close to a horizontal vertex of a curve will be close to vertical. On either side of that vertex those close to vertical slopes will have opposite values (one will be negative and the other positive). Given the angle restrictions this means that on one side of the vertex the directional angle will be very close to $-\frac{\pi}{2}$ while on the other side of the vertex the angle will be very close to $\frac{\pi}{2}$. Using these angles with the equation given above you can see that once you get close to the vertex you will simply oscillate around the vertex rather than continue along the curve.

Explicitly, this is what occurs. We use the equation $s_j = s_{j-1} + d_j \times l_j \times o_{s_{j-1}}$ to get from one point in the curve to the next. Here d_j is either 1 or -1 depending on the direction from the starting point, for our purposes, let's consider it to be 1. The length between the points is l_j and that is 5. This leaves the following equation, $s_j = s_{j-1} + 5 \times o_{s_{j-1}}$. Since $o_{s_{j-1}} = (\cos\theta_{s_{j-1}}, \sin\theta_{s_{j-1}})$ the angle can now be used to demonstrate what occurs around the vertex. On one side the angle will be close to $-\frac{\pi}{2}$ which will leave you with the following results: $s_j = s_{j-1} + 5 \times (0, -1)$, indicating the new point would be 5 units below the point that came before it. If you are close to the vertex this vertical movement would bring you to the other side of the vertex and therefore the new angle would be close to $\frac{\pi}{2}$ and would produce the following result: $s_j = s_{j-1} + 5 \times (0, 1)$. This indicates the new point would be 5 units above the previous point and therefore back to the original point we started with. This process continues until you reach the limit on the number of points allowed in the curve. Not only does this prevent you from moving beyond this section of the curve, it also generates many more points than should be in that area.

As you can see, depending on the angles used there will be noise along the horizontal or vertical areas of the curve. To overcome the problem of stopping curve generation in these sections and to minimize the noise, the range of angles to use is carefully chosen for each point when generating the OFFC. This was done by using angles from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$ as the default. At each point the directional angle was evaluated. If it was greater than 1.4 radians or less than -1.4 radians then the corresponding angle from the range of 0 to π was used. While it does minimize the noise, it does not totally eliminate it.

Once the OFFCs were generated, we need to find the point of highest curvature for each flow curve. The curvature value (ω) for each point l_j on the curve is calculated using the following: $\omega_{l_j} = 1 - \cos \alpha_j$, where α_j = the orientation value of l_{j-5} minus the orientation value of l_{j+5}. According to Dass and Jain [2] all points that had a curvature value of less than 0.3 were eliminated. Theoretically, the point on the curve with the highest curvature value is added to the helper data set.

However, it was necessary to do some filtering before adding points to the helper data set. When examining the curvature values, it was evident that the noise that occurred in the OFFC generation was resulting in artificially high curvature values. It is known that the high curvature values should occur when the curve is changing direction, this occurs at local maximums and local minimums. So in order to eliminate these false high curvature values, the point with the highest curvature value in each individual flow curve was checked to see if it was a local maximum or local minimum of the curve. If it was not, that point was eliminated and the next highest curvature value was examined. When a local maximum or minimum was found that also had the highest curvature value, that point (x, y, ω) was stored as a high curvature point in the helper data set. This helper data set is then manually filtered using the high curvature points drawn on the fingerprint image. For example in figure 3.7 we see a fingerprint image with high curvature points before manual filtration. The areas highlighted in green are those areas which require manual removal before analysis could begin.

Figure 3.1: An example fingerprint to be marked

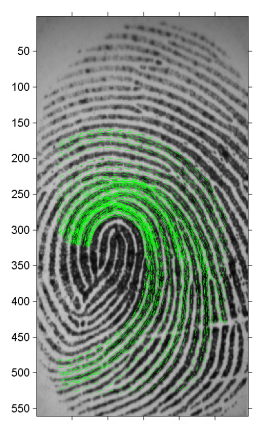


Figure 3.2: OFFC through horizontal

Exploring Chaff Point Elimination

There are two major components of the fuzzy vault, the minutia points and the chaff points. It is possible that the high curvature points leak data about both of these. Since

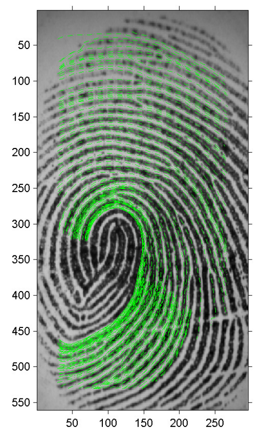


Figure 3.3: OFFC through vertical

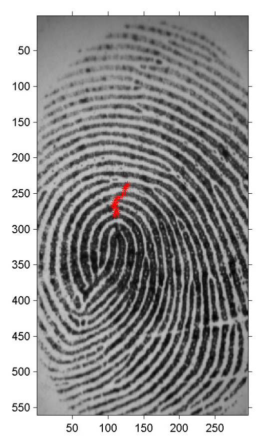


Figure 3.4: Example helper data through horizontal

high curvature points occur in areas where the flow curves are changing direction, the points around them tend to have a directional angle that is somewhat predictable. Given a set of high curvature points, it is likely that any minutia points that are relatively close to the high curvature points and on the left of them will have directional angle between 0 and $\frac{\pi}{2}$ when adjusted for the tilt of the high curvature points. While points on the right that are relatively close will have a directional angle that is between 0 and $-\frac{\pi}{2}$, which is equivalent to the range between $\frac{\pi}{2}$ and π when adjusted for the tilt of the high curvature points.

In order to explore this concept you need to obtain high curvature points, minutia points

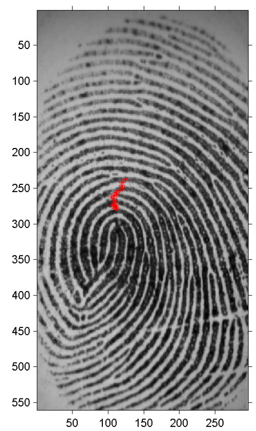


Figure 3.5: Example helper data through vertical

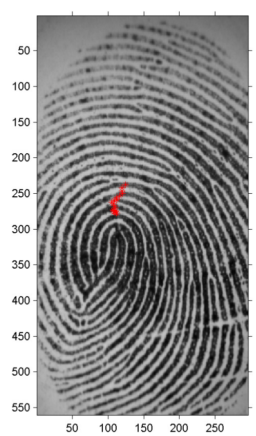


Figure 3.6: Example helper data combined

and chaff points. We use .xyt files provided by NIST to obtain the minutia points for our set of fingerprints. The high curvature points are obtained using the code we wrote in Appendix A. Unfortunately, we did not have a convenient way to obtain chaff points and time prevented us from developing the proper code to do this.

This hypothesis initially involved examining the minutia points that surrounded the high curvature points. In order to do this we first use the high curvature point with the greatest y value and that with the lowest y value to define a high curvature line segment. We used the slope and length of this line segment to create a square centered over the high curvature

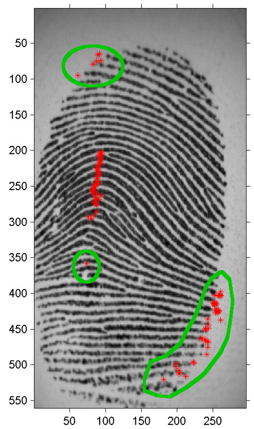
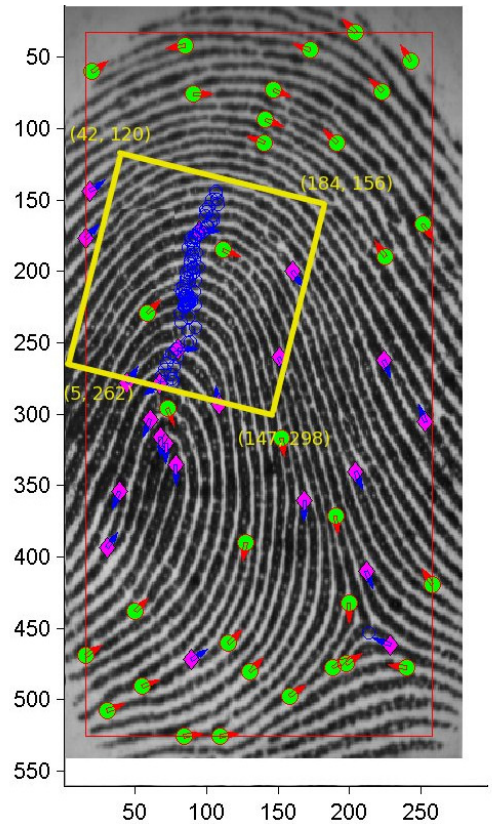


Figure 3.7: HCP to be manually filtered

points. The top and bottom sides of the square had slopes that were perpendicular to the high curvature line segment and pass through the high curvature point with the greatest y value and lowest y value points respectively. The length of each of these sides is equivalent to the length of the high curvature line segment with the midpoint of each of the segments being at the intersection of the high curvature points. The left and right sides have slopes parallel to the high curvature line segment, are of equal length and connected with the end points of the top and bottom lines.

We then determine which minutia points are present within those boundaries and examine the directional angle of those points. We complete the analysis in this manner on three fingerprints. The three fingerprints had a combined total of 10 minutia points in the described zones. Two of those points fell on the high curvature points and so were not considered as part of the analysis. This left a total of 8 minutia points, of these 7 had expected angle measures. Obviously further analysis of more fingerprints is needed to determine if there is an actual correlation between directional angle and proximity to the high curvature points. However, the numerical data we examine as well as informal visual observations from other fingerprints look like a correlation might be promising. Additionally, we note through visual observation that it appears as though if we extend the bottom line of the square across the entire image that a large majority of minutia points above that line would follow the expected directional angles.

While time constraints prevent us from doing an in-depth study of this hypothesis, we



Minutia Points in Box

x	y	theta	side of HCP line	Expected theta value
62	229	22	left	0-90
85	256	0	right	90-180
101	168	0	left	0-90
118	182	168	right	90-180
170	198	152	right	90-180

Figure 3.8: Fingerprint with HCP, minutia points and square drawn for illustration.

are able to perform an initial exploration using a pictorial example.

We use the photo-editing software, Gimp, to superimpose the high curvature and minutia points from the middle image over the last picture.

The image above has the superimposed image. While we are unable to get an exact match (it seems the middle image was not the same scale as the image on the right), it was close enough for an initial investigation.

We create a straight line through the high curvature points and then draw a perpendicular line at the bottom. We eliminate all the chaff points on the left of the high curvature line that had a negative slope and all the chaff points on the right of the high curvature line that had a positive slope. The slope is predicated on the high curvature line as the y-axis and the perpendicular line as the x-axis.

In the below picture, 36 chaff points are removed based on directional angle relative to the high curvature points. There are a total of 223 points on the original (including the

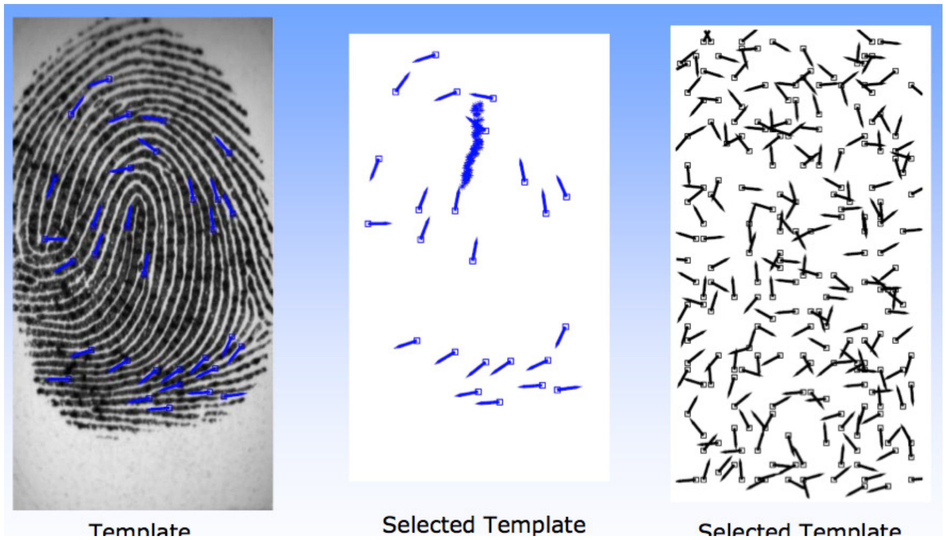


Figure 3.9: An example of a fingerprint with minutia points (left), minutia points and helper data (middle) and minutia points with chaff points (right).

actual minutia points). By using the described procedure, we are able to remove 36 points (16%) that are determined to be chaff points. We believe it might be possible to remove even more points with further analysis. It is possible that the angle range could be further limited depending on how close the point is to the high curvature points.

This was not a precise process for this example. It was an initial exploration to see if it might be worthy of an in depth exploration. For that purpose, this estimation was sufficient. While this process seems promising time constraints are too restrictive to continue with further analysis.

Correlation of Helper Data and Minutia Points

In addition to the potential to eliminate chaff points it is also a possibility that high curvature points leak data about some minutia points. The hypothesis being there is some area around the high curvature points that has a greater concentration of minutia points than would be expected in an uniform distribution. If this were true an attacker may be able to focus efforts on the area of greater concentration, thus decreasing the time needed for a brute force attack.

An initial survey of fingerprint images which included minutia and high curvature points suggest there may be a greater concentration of minutia points within a relatively close

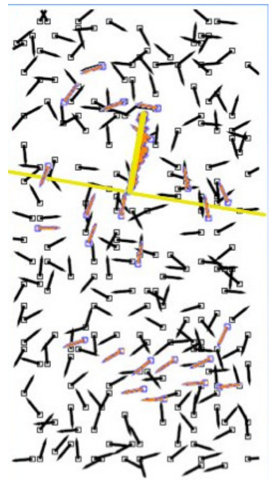


Figure 3.10: High curvature points superimposed over minutia.

proximity to the high curvature points. In order to do a mathematical analysis of this theory a rectangular HCP zone is created around the high curvature points. First, a least squares regression is calculated on the high curvature points. The length of the high curvature points is determined by finding the distance between the maximum and minimum high curvature points.

The rectangle that binds the HCP zone is created in the following manner:

1. The top line is perpendicular to the best fit line of the high curvature points. The length of the line is a percentage (10% or 25%) of the high curvature length. The midpoint of the line is at the high curvature points with the lowest y-value. See figure 3.13.

2. The left and right lines are parallel to the best fit line and begin at the endpoints of the top line. The length of these sides is a percentage (125%) of the length of the high curvature points.

3. The bottom line is perpendicular to the best fit line and connects lower endpoints of the left and right sides.

We determine what percent of the total image area is contained in the HCP zone. The number of minutia points present in the HCP zone is then found. One would logically expect that if the area contained inside the HCP zone was 10% of the total area that this same

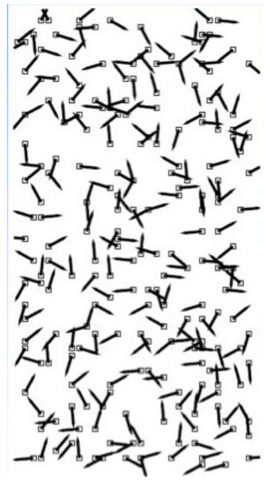


Figure 3.11: Original picture with all chaff and minutia.

area would also contain approximately 10% of all available minutia. A statistical analysis is conducted on the data to determine if there is uniform distribution.

A chi-square goodness of fit test is performed on the resulting data using the following: $\chi^2 = \sum \frac{(o-e)^2}{e}$

where o is the observed number of minutia points in the HCP zone and e is the expected number of minutia points in the HCP zone. The value for e is calculated by multiplying the total number of minutia points by the percent area of the HCP zone.

Most fingerprints have a greater number of minutia points than expected and some have fewer minutia points than expected. In order to better understand how the data was distributed the chi-square goodness of fit test was done on all 200 fingerprints. Then again on the subset of fingerprints with a greater number of minutia than expected and finally on the subset of all fingerprints with fewer than expected minutia points.

The results for the 10% by 125% rectangle are given in Figure 3.14:

The results for the 25% by 125% rectangle are given in Figure 3.15:

The chi-square goodness of fit test indicates that in both rectangular regions, the minutia points do not exhibit a uniform distribution. It also shows that when the observed minutia is greater than the expected minutia the distribution is still not uniform. Furthermore, it shows that when the observed minutia is less than the expected minutia there is a uniform distribution. These combined results suggest the concentration of minutia is greater inside the HCP zone than it would be with a uniform distribution. Therefore, the high curvature

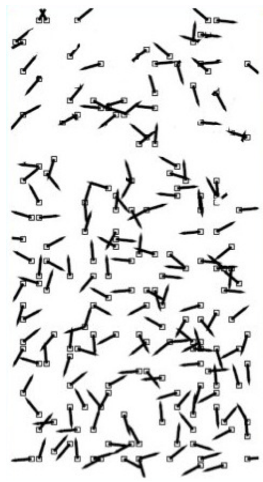


Figure 3.12: Edited picture with some chaff deleted.

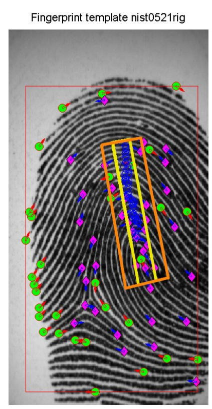


Figure 3.13: HCP Zones.

points that are publicly stored do leak information about the minutia points. This means that it would be easier to find minutia points in this area. Given the small area of the HCP zone, generally less than 10% of the total image area, it would take significantly less time to attack while also increasing the probability of finding minutia.

According to [20], with fewer than nine minutia points a fingerprint fuzzy vault is vulnerable to a partial fingerprint attack. Further work should be done determining an optimal size of the HCP zone to find an area that contains an average of nine (currently the 25% by 125% HCP zone rectangle has an average of 6 minutia points) minutia points. This combined with the partial fingerprint attack make the public high curvature points a vul-

	All fingerprints	Fingerprints with greater minutia points than expected	Fingerprints with fewer minutia points than expected
Chi-square	720.41	697.89	22.52
Degrees of freedom	199	148	50
p-value	< 0.001	< 0.001	0.9997

Figure 3.14: 10% by 125% rectangle results.

	All fingerprints	Fingerprints with greater minutia points than expected	Fingerprints with fewer minutia points than expected
Chi-square	506.97	492.84	14.13
Degrees of freedom	199	157	41
p-value	< 0.001	< 0.001	0.9999

Figure 3.15: 25% by 125% rectangle results.

nerability.

Chapter 4

A New Fingerprint Fuzzy Vault Scheme

In this chapter, we propose an alignment-free fingerprint fuzzy vault scheme. Our scheme is based on an observation used in NIST Biometric Image Software (NBIS) [8]. NBIS is a big package that contains several components, including image enhancing software, image conversion software, minutiae detection MINDTCT, and a fingerprint matching algorithm BOZORTH3 [21]. The BOZORTH3 algorithm was considered in the category of export control and was not included in the public release of NBIS. This decision has been reversed in the latest NBIS release and document [21] was then made available.

In BOZORTH3, it has been observed that a set of minutia points marked on a fingerprint image can be treated as vertices of a graph whose edges are the lines connecting two vertices. (The lines are called *intervening lines* in [21].) For two fingerprints from the same finger, the distance between two same minutia points should remain the same (or at least, very close). So are the relative angles of their orientations relative to the interveneing lines. For example, Figure 4.1 and Figure 4.2 are part of two fingerprints from the same finger and they have two minutia points, called points i and j in Figure 4.1 and points u and v in Figure 4.2 respectively. Due to various environmental factors, these two minutia points have different coordinates and orientations in these two fingerprints. However, their relative distances, labeled d_{ij} in Figure 4.1 and d_{uv} in Figure 4.2, are very close, if not the same. Their orientation angles relative to the interveneing line in 4.1, θ_i and θ_j, are very close to their counterparts in 4.2, θ_u and θ_v respectively.

Under this view, the matching of a reference template and a fresh query fingerprint can be accomplished through comparing the corresponding two graphs. Unlike other fingerprint matching algorithms that require fingerprint alignment, BOZORTH3 is alignment-free, as the graphs are rotation and translation-variant. This characteristic naturally lends itself to fingerprint fuzzy vault.

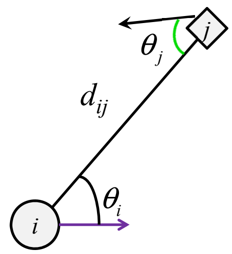


Figure 4.1: Minutia pair in the reference template

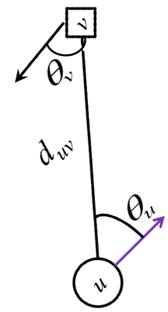


Figure 4.2: Minutia pair in the fresh query template

Invariants

As shown in Figure 4.1, each minutia pair is identified by their Euclidean distance d_{ij} and two relative angles, θ_i and θ_j. This triplet $\{d_{ij}, \theta_i, \theta_j\}$ should remain largely unchanged on fingerprints from the same finger. Given a fingerprint A, we will use its minutia triplets to construct/decode a fingerprint fuzzy vault.

The scheme

Let d_{max} be a distance difference threshold between two minutia points and θ_s be an angle difference threshold. Let t be a threshold value that if two fingerprints have t or more common minutia points, then they are considered a match.

Vault construction

Let k be the secret to be protected by the vault.

Given a fingerprint reference $A = \{a_1, a_2, \ldots, a_n\}$, where a_i is a minutia point, the steps to construct our alignment-free fingerprint fuzzy vault scheme as follows:

1. Sort the minutia points in the descending order in terms of their quality. Let the sorted set be $\bar{A} = \{\bar{a_1}, \bar{a_2}, \ldots, \bar{a}_n\}$.

2. Calculates the triplet invariant on $\bar{A}$, as described earlier in Section 4. For each triplet, concatenate them into an integer. Let the resulting integer set be $\{\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_m\}$.

3. Generate valid points:

(a) Split k into t pieces of equal size, $k_0, k_1, \ldots, k_{t-1}$, where $k_i, 0 \leq i \leq (t-1)$, is an element in F_q.

(b) Construct a polynomial of degree $(t-1)$, $p(x) = k_{t-1}x^{t-1} + k_{t-2}x^{t-2} + \ldots + k_1 t + k_0 \bmod q$

(c) Calculate $\beta_i = p(\tilde{a}_i), 1 \leq i \leq n$. These points (a_i, β_i) are valid points and they form *locking set* S.

4. Generate chaff points: randomly select $(r-n)$ points $(\gamma_j, \zeta_j), 1 \leq j \leq (r-n)$, where γ_j and ζ_j are randomly selected from F_q with two conditions. First, $\gamma_j \neq \tilde{a}_i$. Second, $\zeta_j \neq p(\gamma_j)$; that is, (γ_j, ζ_j) are not on polynomial $p(x)$.

   All points (γ_j, ζ_j) form *chaff set* C.

5. The union of sets S and C, $\mathcal{P} = S \cup C$, forms the points stored in the vault.

Vault decoding

Let $B = \{b_1, b_2, \ldots, b_n\}$ be a fresh set and it can be used to unlock vault $\mathcal{P}$ if B is close to A.

1. Sort the minutia points in B in the descending order in terms of their quality. Let the sorted set be $\bar{B} = \{\bar{b_1}, \bar{b_2}, \ldots, \bar{b}_l\}$.

2. Calculates the triplet invariant on $\bar{B}$, as described earlier in Section 4. For each triplet, concatenate them into an integer. Let the resulting integer set be $\{\tilde{b}_1, \tilde{b}_2, \ldots, \tilde{b}_m\}$.

3. Use each $\tilde{b}_i, 1 \leq i \leq n$, as the x-coordinate to search, in a close manner with consideration of d_{max} and θ_s, $\mathcal{P}$ for a point. Let $\mathcal{V}$ be the set of points found.

4. Apply the Reed-Solomon decoding algorithm to points in $\mathcal{V}$ to reconstruct a polynomial $p'(x)$ [11, 15].

Chapter 5

Conclusion

Fingerprint fuzzy vault schemes have great potentials to address the security and privacy concerns of fingerprint applications and may see real-world deployment very soon. In this thesis, we first analyze the security of an existing fingerprint fuzzy vault scheme called *NJP07* [14].

In [14] Nandakumar et al. [14] conclude that helper data which is stored as public information does *not* affect the security of the fuzzy vault. In our work we take steps to show that the helper data required to align fingerprint images in the fingerprint fuzzy vault does leak information thereby compromising the security of the system. A rectangular region surrounding the helper data is shown to have a higher percentage of minutia points than would be expected in a uniform distribution. This area is substantially smaller than the total fingerprint area. This small area combined with the greater concentration of minutia points within it gives an attacker an advantage when attempting to compromise the system. Furthermore, obtaining some minutia points allows an attacker to employ a partial fingerprint attack. This sort of attack is a natural complement to the data that is leaked by the publicly available high curvature points as shown in this paper.

Next, we suggest a new fuzzy vault scheme that is based on an observation in the Bozorth fingerprint matching algorithm [21]. This scheme allows for the implementation of a fuzzy vault without any helper data. This eliminates the significant security risk identified in this paper. Additionally, the scheme eliminates the common problem of fingerprint alignment as it is rotation and translation invariant.

Areas of Further Research

As discussed in Chapter 3, it would be beneficial to further analyze and research different sizes of rectangles to be used in binding the HCP zone. More testing of this could reveal an optimal size for maintaining a small area while exposing the greatest number of minutia. Additionally, it may be useful to explore different shapes for binding the region around the HCP zone. For instance, it appears that an elliptical shape may reduce the area while still containing the same number of minutia points. It would also be interesting to learn if there is any relationship between the shapes and sizes used to bind an HCP zone and the different types of fingerprints.

We have shown the helper data may leak additional information. We outlined a method to begin an investigation of using helper data along with the expected directional angles of minutia points to eliminate chaff points within close proximity to the helper data. If this procedure were successfully implemented then it is possible a significant number of chaff points could be eliminated from the system. The reduction of this chaff should reduce the overall security of the vault. By decreasing the chaff while simultaneously having an increased probability of finding minutia within a small area the fuzzy vault quickly becomes more vulnerable.

Finally, security analysis is required of the proposed fuzzy vault scheme.

Appendix A

Code to Generate curvature points

The Matlab code to generate curvature points consists of three scripts, *highcurvaturepoints_horiz.m*, *highcurvaturepoints_vert.m*, and *highcurvaturepoints_total.m*

highcurvaturepoints_horiz.m

```
% HIGHCURVATUREPOINTS
%
% Function to d
%
% Usage:
%
% Argument:
%
% Returns:

% Patrick Perry
%
% James Madison University
%
%
%
% October 2012

fvc02_files = dir(fullfile('./results/', '*-fvc02_orient.txt'));

for a = 1:size(fvc02_files)
    currentFile = fvc02_files(a).name;
[pathstr, nameWOext, ext] = fileparts(currentFile);
disp (nameWOext);
    short_name = strrep(nameWOext, '-fvc02_orient', '');
    short_nameWext = [short_name '.tif'];
    M = dlmread(['./results/' short_name '-fvc02_orient.txt']);
%[0..pi]==> sin table

    %PJP-10-29 Patch to convert range from [0..pi] to [-pi/2..pi/2]
    [rows,cols] = size(M);
    M_transform = M;
    for i_patch = 1:cols
        for j_patch = 1:rows
```

```
            if (M_transform(j_patch,i_patch) > pi/2)
                M_transform(j_patch,i_patch) = M(j_patch,i_patch) - pi;
            end
        end
    end
% dlmwrite(['./results/' short_name '-fvc02_cos_orient.txt'],
% M_transform,'\t');
%[-pi/2..pi/2]==> cos table
    %PJP-10-29 End Patch

    rect_info = dlmread(['./results/' short_name 'myfv2002datab.txt'],
'\t', 0, 1);
    display(rect_info);
    r_start = rect_info(1);
    r_end = rect_info(3);
    c_start = rect_info(2);
    c_end = rect_info(4);
    n_0 = 75; %DassJain04 Section 3.1 refers to this prespecified constant.
      %I am choosing 25 for something to test.
% I am not sure what value to use here.
    w = 5;
    l = ((c_end - c_start) / (2 * w));

    final_k = ((r_end - r_start) / w);
    max_k = ceil(final_k);
% We use this for initializing our array and as an
% upper bound on the for loop

    s_0_x = zeros(1, max_k);
    s_0_y = c_start + l * w;
    for i = 1:max_k
        k = i;
    if i == max_k
    k = final_k;
    end
    s_0_x(i) = r_start + k * w;
    end
    % At this point we have an array of s_0 x values and the corresponding
% unchanging y value for those points.

    Curves_M_transform = zeros(2 * n_0 + 1, max_k * 2);
% For rows we * 2 to account for + & -.  Then add 1 for s_0.
% Columns are * 2 to store x & y.
    % Calculate s_j
    for i = 1:max_k % We will deal with the top half of the rectangle first
    s_x_jminus1 = round(s_0_x(i));
    s_y_jminus1 = round(s_0_y);
    d_j = 1;
```

```
    l_j = 5;
    Curves_M_transform(n_0 + 1, 2 * i - 1) = s_x_jminus1;
% We have to begin each curve by putting the starting point in place.
    Curves_M_transform(n_0 + 1, 2 * i) = s_y_jminus1;

    for j = 1:n_0
            theta_s_jminus1 = M_transform(round(s_y_jminus1),
round(s_x_jminus1));
            %Patch to correct oscillation at vertical areas
            if ( (theta_s_jminus1 > 1.4) | (theta_s_jminus1 < -1.4) )
                theta_s_jminus1 = M(round(s_y_jminus1),round(s_x_jminus1));
                d_j = 1;
            end
            s_x_j = s_x_jminus1 + d_j * l_j * cos(theta_s_jminus1);
    s_y_j = s_y_jminus1 + d_j * l_j * sin(theta_s_jminus1);
    % First, test if we are in the red rectangle.
% If not break out of the loop.
    if (s_x_j < r_start) | (s_x_j > r_end)
    break
    end
    if (s_y_j < c_start) | (s_y_j > c_end)
    break
    end
            Curves_M_transform(n_0 + 1 + j, 2 * i - 1) = s_x_j;
    Curves_M_transform(n_0 + 1 + j, 2 * i) = s_y_j;
    % Reset the variables before the next iteration.
            if ( Curves_M_transform(n_0 + 1 + j - 1, 2 * i - 1) >
Curves_M_transform(n_0 + 1 + j, 2 * i - 1) )
                d_j = -1;
            end
    s_x_jminus1 = s_x_j;
    s_y_jminus1 = s_y_j;
    end
    end

    for i = 1:max_k
% We will deal with the bottom half of the rectangle second
    s_x_jminus1 = round(s_0_x(i));
    s_y_jminus1 = round(s_0_y);
    d_j = -1;
    l_j = 5;
    Curves_M_transform(n_0 + 1, 2 * i - 1) = s_x_jminus1;
% We have to begin each curve by putting the starting
% point in place.
    Curves_M_transform(n_0 + 1, 2 * i) = s_y_jminus1;

    for j = 1:n_0
            theta_s_jminus1 = M_transform(round(s_y_jminus1),
```

```
round(s_x_jminus1));
            %Patch to correct oscillation at vertical areas
            if ( (theta_s_jminus1 > 1.4) | (theta_s_jminus1 < -1.4) )
                theta_s_jminus1 = M(round(s_y_jminus1),round(s_x_jminus1));
                d_j = -1;
            end
            s_x_j = s_x_jminus1 + d_j * l_j * cos(theta_s_jminus1);
    s_y_j = s_y_jminus1 + d_j * l_j * sin(theta_s_jminus1);
    % First, test if we are in the red rectangle.
% If not break out of the loop.
    if (s_x_j < r_start) | (s_x_j > r_end)
    break
    end
    if (s_y_j < c_start) | (s_y_j > c_end)
    break
    end
            Curves_M_transform(n_0 + 1 - j, 2 * i - 1) = s_x_j;
    Curves_M_transform(n_0 + 1 - j, 2 * i) = s_y_j;
    % Reset the variables before the next iteration.
            if ( Curves_M_transform(n_0 + 1 - j + 1, 2 * i - 1) <
Curves_M_transform(n_0 + 1 - j, 2 * i - 1) )
                d_j = 1;
            end
    s_x_jminus1 = s_x_j;
    s_y_jminus1 = s_y_j;
    end
    end

%++++++++++
    [mm1, nn1] = size (M);
    disp (mm1);
    disp (nn1);
    for j = 1:max_k %
mynewfolder = ['./results/' short_name];
isexistent = exist(mynewfolder);
if (isexistent ~= 7) mkdir(mynewfolder);
end
        thisfilename = ['./results/' short_name
'/oneline-offc-horiz-' num2str(j) '.txt'];
        disp (thisfilename);
        fid = fopen(thisfilename, 'w');
        formatSpec = '%f\t %f\t %f\n';
        for i=1:2*n_0+1
            if (Curves_M_transform(i, 2*j-1) > 0) &
(Curves_M_transform(i, 2*j) > 0)
                if (Curves_M_transform(i, 2*j-1) > nn1) |
(Curves_M_transform(i, 2*j) > mm1)
                    disp ('ERROR');
```

```
                    disp (Curves_M_transform(i, 2*j-1));
                    disp (Curves_M_transform(i, 2*j));
                else
                    fprintf(fid, formatSpec, Curves_M_transform(i, 2*j-1),
Curves_M_transform(i, 2*j),
M_transform(round(Curves_M_transform(i, 2*j)),
round(Curves_M_transform(i, 2*j-1))));
                end %% end of inner if
            end %% end of outer if
        end %% end of for i
        fclose (fid);
    end %% end of for j
%++++++++++

%    Curves_with_noise = ['./results/' short_name '_curve_w_noise.txt'];
%    dlmwrite(Curves_with_noise, Curves_M_transform);

    thisIm = imread (['./Sample_Fingerprints/' short_name '.tif']);
    f=figure('visible', 'off'), imshow (thisIm);
    hold on;
    axis on;
    [m n] = size (Curves_M_transform);
    Curvature_Values = zeros(m, n/2);
    for i=1:n/2
        x = round(nonzeros(Curves_M_transform(:,2*i-1)));
        y = round(nonzeros(Curves_M_transform(:,2*i)));
        plot (x, y, '-.g');%, '-.og');
        hold on;
        %This is where we create a matrix with curvature values.
        for j=6:size (x) - 5
% We subtract 5 from the size of x so we do not
% exceed the number of points when we use j+5
            theta_left = M_transform(y(j-5), x(j-5));
            theta_right = M_transform(y(j+5), x(j+5));
            alpha = theta_left - theta_right;
            Curvature_Values(j,i) = 1 - cos(alpha);
        end
        %We have the OFFC curvature values.
    end

mynewfolder2 = ['./results/' short_name];
% disp (mynewfolder2);
isexistent2 = exist(mynewfolder2);
% disp (isexistent2);
if (isexistent2 ~= 7) mkdir(mynewfolder2);
end
    Curve_Vals = ['./results/' short_name '/Curvature_Vals_Horiz.txt'];
% disp (Curve_Vals);
```

```
    dlmwrite(Curve_Vals, Curvature_Values);
%%%%
    %We need to remove some false high curvature values
    Filtered_Curvature_Values = Curvature_Values;
    for i=1:n/2
        while (1)
            [Filtered_peaks_max, Filtered_index] =
max(Filtered_Curvature_Values(:,i));
            filtered_y = nonzeros(Curves_M_transform(:,2*i));
            Filtered_index = round(Filtered_index);
            if (Filtered_peaks_max == 0)
                break
            end
            if ( (filtered_y(Filtered_index) >
filtered_y(Filtered_index - 1)) &
(filtered_y(Filtered_index) > filtered_y(Filtered_index + 1)) )
                break
            elseif ( (filtered_y(Filtered_index) <
filtered_y(Filtered_index - 1)) &
(filtered_y(Filtered_index) < filtered_y(Filtered_index + 1)) )
                break
            else
                Filtered_Curvature_Values(Filtered_index, i) = 0;
            end
        end
    end
    Curve_Vals = ['./results/' short_name
'/FILTERED_Curvature_Vals_Horiz.txt'];
    dlmwrite(Curve_Vals, Filtered_Curvature_Values);
    %False values have been removed now.

    %Initialize our High Curvature Points matrix
    High_Curvature_Points = zeros(1, 3); %Initialize array to store 1 HCP
    for i=1:n/2
        [HCP_peaks, HCP_index] = max(Filtered_Curvature_Values(:,i));
        HCP_index = round(HCP_index);
        x = nonzeros(Curves_M_transform(:,2*i-1));
        y = nonzeros(Curves_M_transform(:,2*i));
        for j=1:length(HCP_index)
            HCP_x = x(HCP_index(j));
            HCP_y = y(HCP_index(j));
            HCP_omega = HCP_peaks(j);
            %Based on DJ04 p 750 we are not going to include
%HCP less than .3
            if (HCP_omega >= .3)
                next_row = [HCP_x HCP_y HCP_omega];
                High_Curvature_Points = [High_Curvature_Points ; next_row];
            end
```

```
        end
    end
    [r c] = size(High_Curvature_Points);
%    for i=2:r
%        plot (High_Curvature_Points(i,1),
%  High_Curvature_Points(i,2), '*r');
%        hold on;
%    end
    HCP_Vals = ['./results/' short_name '/HCP_horiz.txt'];
    dlmwrite(HCP_Vals, High_Curvature_Points);
%%%%
    newFilename1 = ['./results/' short_name '/OFFC_horiz.tif'];
    print (f, '-dtiff', newFilename1);

end
disp ('Done');
```

highcurvaturepoints_vert.m

```
% HIGHCURVATUREPOINTS
%
% Function to d
%
% Usage:
%
% Argument:
%
% Returns:

% Patrick Perry
%
% James Madison University
%
%
%
% October 2012

fvc02_files = dir(fullfile('./results/', '*-fvc02_orient.txt'));

for a = 1:size(fvc02_files)
    currentFile = fvc02_files(a).name;
[pathstr, nameWOext, ext] = fileparts(currentFile);
disp (nameWOext);
    short_name = strrep(nameWOext, '-fvc02_orient', '');
    short_nameWext = [short_name '.tif'];
    M = dlmread(['./results/' short_name '-fvc02_orient.txt']);
%[0..pi]==> sin table
```

```
    %PJP-10-29 Patch to convert range from [0..pi] to [-pi/2..pi/2]
    [rows,cols] = size(M);
    M_transform = M;
    for i_patch = 1:cols
        for j_patch = 1:rows
            if (M_transform(j_patch,i_patch) > pi/2)
                M_transform(j_patch,i_patch) = M(j_patch,i_patch) - pi;
            end
        end
    end
% dlmwrite(['./results/' short_name '-fvc02_cos_orient.txt'],
%  M_transform, '\t');
  %[-pi/2..pi/2]==> cos table
    %PJP-10-29 End Patch

    rect_info = dlmread(['./results/' short_name
'myfv2002datab.txt'], '\t', 0, 1);
    display(rect_info);
    r_start = rect_info(1);
    r_end = rect_info(3);
    c_start = rect_info(2);
    c_end = rect_info(4);
    n_0 = 50; % DassJain04 Section 3.1 refers to
% this prespecified constant.
% I am choosing 25 for something to test.
% I am not sure what value to use here.
    w = 5;
    k = ((r_end - r_start) / (2 * w));

    final_l = ((c_end - c_start) / w);
    max_l = ceil(final_l)
% We use this for initializing our array and as an upper bound
% on the for loop

    s_0_x = r_start + k * w;
    s_0_y = zeros(1, max_l);
    for i = 1:max_l
        l = i;
    if i == max_l
    l = final_l;
    end
    s_0_y(i) = c_start + l * w;
    end
    % At this point we have an array of s_0 x values and the
% corresponding unchanging y value for those points.

    Curves_M_transform = zeros(2 * n_0 + 1, max_l * 2);
```

```
% For rows we * 2 to account for + & -.
% Then add 1 for s_0. Columns are * 2 to store x & y.
    % Calculate s_j
    for i = 1:max_l %We will deal with the top half of the rectangle first
    s_x_jminus1 = round(s_0_x);
    s_y_jminus1 = round(s_0_y(i));
    d_j = 1;
    l_j = 5;
    Curves_M_transform(n_0 + 1, 2 * i - 1) = s_x_jminus1;
% We have to begin each curve by putting the
% starting point in place.
    Curves_M_transform(n_0 + 1, 2 * i) = s_y_jminus1;

    for j = 1:n_0
            theta_s_jminus1 = M_transform(round(s_y_jminus1),
round(s_x_jminus1));
            %Patch to correct oscillation at vertical areas
            if ( (theta_s_jminus1 > 1.4) | (theta_s_jminus1 < -1.4) )
                theta_s_jminus1 = M(round(s_y_jminus1),
round(s_x_jminus1));
                d_j = 1;
            end
            s_x_j = s_x_jminus1 + d_j * l_j * cos(theta_s_jminus1);
    s_y_j = s_y_jminus1 + d_j * l_j * sin(theta_s_jminus1);
    % First, test if we are in the red rectangle.
% If not break out of the loop.
    if (s_x_j < r_start) | (s_x_j > r_end)
    break
    end
    if (s_y_j < c_start) | (s_y_j > c_end)
    break
    end
            Curves_M_transform(n_0 + 1 + j, 2 * i - 1) = s_x_j;
    Curves_M_transform(n_0 + 1 + j, 2 * i) = s_y_j;
    % Reset the variables before the next iteration.
            if ( Curves_M_transform(n_0 + 1 + j - 1, 2 * i - 1) >
Curves_M_transform(n_0 + 1 + j, 2 * i - 1) )
                d_j = -1;
            end
    s_x_jminus1 = s_x_j;
    s_y_jminus1 = s_y_j;
    end
    end

    for i = 1:max_l
% We will deal with the bottom half of the rectangle second
    s_x_jminus1 = round(s_0_x);
    s_y_jminus1 = round(s_0_y(i));
```

```
    d_j = -1;
    l_j = 5;
    Curves_M_transform(n_0 + 1, 2 * i - 1) = s_x_jminus1;
% We have to begin each curve by putting the starting
% point in place.
    Curves_M_transform(n_0 + 1, 2 * i) = s_y_jminus1;

    for j = 1:n_0
            theta_s_jminus1 = M_transform(round(s_y_jminus1),
round(s_x_jminus1));
            %Patch to correct oscillation at vertical areas
            if ( (theta_s_jminus1 > 1.4) | (theta_s_jminus1 < -1.4) )
                theta_s_jminus1 = M(round(s_y_jminus1),
round(s_x_jminus1));
                d_j = -1;
            end
            s_x_j = s_x_jminus1 + d_j * l_j * cos(theta_s_jminus1);
    s_y_j = s_y_jminus1 + d_j * l_j * sin(theta_s_jminus1);
    % First, test if we are in the red rectangle.
% If not break out of the loop.
    if (s_x_j < r_start) | (s_x_j > r_end)
    break
    end
    if (s_y_j < c_start) | (s_y_j > c_end)
    break
    end
            Curves_M_transform(n_0 + 1 - j, 2 * i - 1) = s_x_j;
    Curves_M_transform(n_0 + 1 - j, 2 * i) = s_y_j;
    % Reset the variables before the next iteration.
            if ( Curves_M_transform(n_0 + 1 - j + 1, 2 * i - 1) <
Curves_M_transform(n_0 + 1 - j, 2 * i - 1) )
                d_j = 1;
            end
    s_x_jminus1 = s_x_j;
    s_y_jminus1 = s_y_j;
    end
    end

%++++++++++
    [mm1, nn1] = size (M);
    disp (mm1);
    disp (nn1);
    for j = 1:max_l %
        thisfilename = ['./results/' short_name '/oneline-offc-vert-'
num2str(j) '.txt'];
        disp (thisfilename);
        fid = fopen(thisfilename, 'w');
        formatSpec = '%f\t %f\t %f\n';
```

```
        for i=1:2*n_0+1
            if (Curves_M_transform(i, 2*j-1) > 0) &
(Curves_M_transform(i, 2*j) > 0)
                if (Curves_M_transform(i, 2*j-1) > nn1) |
(Curves_M_transform(i, 2*j) > mm1)
                    disp ('ERROR');
                    disp (Curves_M_transform(i, 2*j-1));
                    disp (Curves_M_transform(i, 2*j));
                else
                    fprintf(fid, formatSpec, Curves_M_transform(i, 2*j-1),
Curves_M_transform(i, 2*j),
M_transform(round(Curves_M_transform(i, 2*j)),
round(Curves_M_transform(i, 2*j-1))));
                end %% end of inner if
            end %% end of outer if
        end %% end of for i
        fclose (fid);
    end %% end of for j
%++++++++++

%    Curves_with_noise = ['./results/' short_name '_curve_w_noise.txt'];
%    dlmwrite(Curves_with_noise, Curves_M_transform);

    thisIm = imread (['./Sample_Fingerprints/' short_name '.tif']);
    f=figure('visible', 'off'), imshow (thisIm);
    hold on;
    axis on;
    [m n] = size (Curves_M_transform);
    Curvature_Values = zeros(m, n/2);
    for i=1:n/2
        x = round(nonzeros(Curves_M_transform(:,2*i-1)));
        y = round(nonzeros(Curves_M_transform(:,2*i)));
        plot (x, y, '-.g');%, '-.og');
        hold on;
        %This is where we create a matrix with curvature values.
        for j=6:size (x) - 5
%We subtract 5 from the size of x so we do not
%exceed the number of points when we use j+5
            theta_left = M_transform(y(j-5), x(j-5));
            theta_right = M_transform(y(j+5), x(j+5));
            alpha = theta_left - theta_right;
            Curvature_Values(j,i) = 1 - cos(alpha);
        end
        %We have the OFFC curvature values.
    end

    Curve_Vals = ['./results/' short_name '/Curvature_Vals_Vert.txt'];
    dlmwrite(Curve_Vals, Curvature_Values);
```

```
%%%
    %We need to remove some false high curvature values
    Filtered_Curvature_Values = Curvature_Values;
    for i=1:n/2
        while (1)
            [Filtered_peaks_max, Filtered_index] =
max(Filtered_Curvature_Values(:,i));
            filtered_y = nonzeros(Curves_M_transform(:,2*i));
            Filtered_index = round(Filtered_index);
            if (Filtered_peaks_max == 0)
                break
            end
            if ( (filtered_y(Filtered_index) >
filtered_y(Filtered_index - 1)) &
(filtered_y(Filtered_index) >
filtered_y(Filtered_index + 1)) )
                break
            elseif ( (filtered_y(Filtered_index) <
filtered_y(Filtered_index - 1)) &
(filtered_y(Filtered_index) <
filtered_y(Filtered_index + 1)) )
                break
            else
                Filtered_Curvature_Values(Filtered_index, i) = 0;
            end
        end
    end
    Curve_Vals = ['./results/' short_name
'/FILTERED_Curvature_Vals_Vert.txt'];
    dlmwrite(Curve_Vals, Filtered_Curvature_Values);
    %False values have been removed now.

    %Initialize our High Curvature Points matrix
    High_Curvature_Points = zeros(1, 3); %Initialize array to store 1 HCP
    for i=1:n/2
        [HCP_peaks, HCP_index] = max(Filtered_Curvature_Values(:,i));
        HCP_index = round(HCP_index);
        x = nonzeros(Curves_M_transform(:,2*i-1));
        y = nonzeros(Curves_M_transform(:,2*i));
        for j=1:length(HCP_index)
            HCP_x = x(HCP_index(j));
            HCP_y = y(HCP_index(j));
            HCP_omega = HCP_peaks(j);
            %Based on DJ04 p 750 we are not going to include
%HCP less than .3
            if (HCP_omega >= .3)
                next_row = [HCP_x HCP_y HCP_omega];
                High_Curvature_Points = [High_Curvature_Points ; next_row];
```

```
            end
        end
    end
    [r c] = size(High_Curvature_Points);
%    for i=2:r
%        plot (High_Curvature_Points(i,1),
%  High_Curvature_Points(i,2), '*r');
%        hold on;
%    end
    HCP_Vals = ['./results/' short_name '/HCP_vert.txt'];
    dlmwrite(HCP_Vals, High_Curvature_Points);
%%%%
    newFilename1 = ['./results/' short_name '/OFFC_vert.tif'];
    print (f, '-dtiff', newFilename1);

end
disp ('Done');
```

highcurvaturepoints_total.m

```
% HIGHCURVATUREPOINTS
%
% Function to d
%
% Usage:
%
% Argument:
%
% Returns:

% Patrick Perry
%
% James Madison University
%
%
%
% October 2012

fvc02_files = dir(fullfile('./results/', '*-fvc02_orient.txt'));

for a = 1:size(fvc02_files)
    currentFile = fvc02_files(a).name;
[pathstr, nameWOext, ext] = fileparts(currentFile);
disp (nameWOext);
    short_name = strrep(nameWOext, '-fvc02_orient', '');
    short_nameWext = [short_name '.tif'];
    Horiz = dlmread(['./results/' short_name '/HCP_horiz.txt']);
```

```
    if ( Horiz(1,1) == 0 )
        Horiz(1,:) = [];
    end
    Vert = dlmread(['./results/' short_name '/HCP_vert.txt']);
    if ( Vert(1,1) == 0 )
        Vert(1,:) = [];
    end
    Total = [Horiz; Vert];
    HCP_Vals = ['./results/' short_name '/HCP_total.txt'];
    dlmwrite(HCP_Vals, Total);
    thisIm = imread (['./Sample_Fingerprints/' short_name '.tif']);
    f=figure('visible', 'off'), imshow (thisIm);
    hold on;
    axis on;
    %We draw the high curvature points (horiz, vert & total).
    %vert
    [r c] = size(Vert);
    for i=1:r
        plot (Vert(i,1), Vert(i,2), '*r');
        hold on;
    end
    newFilename1 = ['./results/' short_name '/HCP_VERT.tif'];
    print (f, '-dtiff', newFilename1);
    %horiz
    thisIm = imread (['./Sample_Fingerprints/' short_name '.tif']);
    f=figure('visible', 'off'), imshow (thisIm);
    hold on;
    axis on;
    [r c] = size(Horiz);
    for i=1:r
        plot (Horiz(i,1), Horiz(i,2), '*r');
        hold on;
    end
    newFilename1 = ['./results/' short_name '/HCP_HORIZ.tif'];
    print (f, '-dtiff', newFilename1);
    %total
    thisIm = imread (['./Sample_Fingerprints/' short_name '.tif']);
    f=figure('visible', 'off'), imshow (thisIm);
    hold on;
    axis on;
    [r c] = size(Total);
    for i=1:r
        plot (Total(i,1), Total(i,2), '*r');
        hold on;
    end
    newFilename1 = ['./results/' short_name '/HCP_TOTAL.tif'];
    print (f, '-dtiff', newFilename1);
end
```

```
disp ('Done');
```

Appendix B

Code for Correlation Analysis Between Minutia and Helper Data within a Rectangular Zone

hcprectanalysis.m

```
HCP_total_files = dir(fullfile('F:\THESIS\MANUAL_FILTERED_HCP\',
'*-HCP_total.txt'));
Results = zeros(size(HCP_total_files),6);
Output = ['F:\THESIS\MANUAL_FILTERED_HCP\HCP_ZONE_ANALYSIS.txt'];
fid = fopen(Output, 'w');
formatSpec = '%s\t %f\t %f\t %f\t %f\t %f\t %f\n';

for a = 1:size(HCP_total_files)
    currentFile = HCP_total_files(a).name;
[pathstr, nameWOext, ext] = fileparts(currentFile);
disp (nameWOext);
    short_name = strrep(nameWOext, '-HCP_total', '');
%    short_nameWext = [short_name '.tif'];
    HCP = dlmread(['F:\THESIS\MANUAL_FILTERED_HCP\' short_name
'-HCP_total.txt']);
    XYT = dlmread(['F:\THESIS\MANUAL_FILTERED_HCP\' short_name
'nist.xyt']);
    rect_info = dlmread(['F:\THESIS\MANUAL_FILTERED_HCP\'
short_name 'myfv2002datab.txt'], '\t', 0, 1);
%HCP = dlmread('F:\THESIS\MANUAL_FILTERED_HCP\94_7-HCP_total.txt');
%XYT = dlmread('F:\THESIS\MANUAL_FILTERED_HCP\94_7nist.xyt');

%display(HCP);

    [row, col] = size (HCP);
    [row_xyt, col_xyt] = size (XYT);
    regression = zeros(row, 5); %columns is 5 for x, y, xy, x^2, y^2
    regression(:,1) = HCP(:,1);
    regression(:,2) = HCP(:,2)*(-1);
%multiply by -1 to adjust for the fact we are really
%working in quadrant 4, y's increase as you go down the axis
    for i = 1:row
        regression(i,3) = HCP(i,1) * HCP(i,2) * (-1);
        regression(i,4) = HCP(i,1) * HCP(i,1);
        regression(i,5) = HCP(i,2) * HCP(i,2);
    end
    %display(regression);
```

```
    sums = sum(regression);

    y_intercept = ((sums(2)*sums(4)) - (sums(1)*sums(3))) /
((row*sums(4)) - (sums(1)*sums(1)));
    slope = ((row*sums(3)) - (sums(1)*sums(2))) /
((row*sums(4)) - (sums(1)*sums(1)));

    [hcp_bottom_y, hcp_bottom_y_index] = max(HCP(:,2));
    [hcp_top_y, hcp_top_y_index] = min(HCP(:,2));

    hcp_bottom_x = HCP(hcp_bottom_y_index,1);
    hcp_top_x = HCP(hcp_top_y_index,1);
    hcp_length = sqrt( (hcp_bottom_x-hcp_top_x)*
(hcp_bottom_x-hcp_top_x) +
( (hcp_bottom_y-hcp_top_y)*(hcp_bottom_y-hcp_top_y) ) );
    rect_width = .25 * hcp_length;
    rect_length = .25 * hcp_length;
    inside_hcp_zone = 0;

    for j = 1:row_xyt
        y = XYT(j,2);
        x = XYT(j,1);
        if ( y >= -1 * ((-1/slope) * (x-hcp_top_x) - hcp_top_y) )
%subtracting y value to account for working in the 4th quadrant
            if ( y <= -1 * ((-1/slope) * (x-hcp_bottom_x) - hcp_bottom_y - rect_length)
                if ( slope >= 0 )
                    if ( y >= -1 * ((slope) * (x+rect_width) +
y_intercept) )
                        if ( y <= -1 * ((slope) * (x-rect_width) +
y_intercept) )
                            inside_hcp_zone = inside_hcp_zone + 1;
                        end
                    end
                else
                    if ( y <= -1 * ((slope) * (x+rect_width) +
y_intercept) )
                        if ( y >= -1 * ((slope) * (x-rect_width)
+ y_intercept) )
                            inside_hcp_zone = inside_hcp_zone + 1;
                        end
                    end
                end
            end
        end
    end

    percent_min_in_zone = inside_hcp_zone / row_xyt * 100;
```

```
    Results(a,1) = percent_min_in_zone;
    Results(a,3) = 100 - percent_min_in_zone;
    Results(a,6) = row_xyt;

    vertex1_x = ( (-rect_width*slope*slope) - (slope*y_intercept) -
(slope*hcp_top_y) + hcp_top_x ) / ( 1+slope*slope );
    vertex2_x = ( (-rect_width*slope*slope) - (slope*y_intercept)
- (slope*hcp_bottom_y) - (slope*rect_length) + hcp_bottom_x ) /
( 1+slope*slope );
    vertex3_x = ( (rect_width*slope*slope) - (slope*y_intercept)
- (slope*hcp_top_y) + hcp_top_x ) / ( 1+slope*slope );
    vertex4_x = ( (rect_width*slope*slope) - (slope*y_intercept)
- (slope*hcp_bottom_y) - (slope*rect_length) +
hcp_bottom_x ) / ( 1+slope*slope );

    x1 = rect_info(1);
    x2 = rect_info(3);
    y1 = rect_info(2);
    y2 = rect_info(4);
    length = rect_info(5);
    width = rect_info(6);
    rect_area = length * width;

    %zone area
    vertex1_y = -1 * ((-1/slope) * (vertex1_x-hcp_top_x) -
hcp_top_y);
    vertex2_y = -1 * ((-1/slope) * (vertex2_x-hcp_bottom_x)
- hcp_bottom_y - rect_length);
    vertex3_y = -1 * ((-1/slope) * (vertex3_x-hcp_top_x)
- hcp_top_y);
    vertex4_y = -1 * ((-1/slope) * (vertex4_x-hcp_bottom_x)
 - hcp_bottom_y - rect_length);
    %Need to adjust coords for polyarea if top perpendicular
% hits the top of the red rectangle.  This appears to
% be the only potential problem case.
    if ( vertex1_y < y1 )
        zone_x1 = (-y1+hcp_top_y)*(-slope)+hcp_top_x;
        zone_y1 = -1 * ( slope*(x1+rect_width) + y_intercept );
        zone_x = [ zone_x1 x1 vertex2_x vertex4_x vertex3_x ];
        zone_y = [ y1 zone_y1 vertex2_y vertex4_y vertex3_y ];
    elseif ( vertex3_y < y1 )
        zone_x2 = (-y1+hcp_top_y)*(-slope)+hcp_top_x;
        zone_y2 = -1 * ( slope*(x2-rect_width) + y_intercept );
        zone_x = [ zone_x2 vertex1_x vertex2_x vertex4_x x2 ];
        zone_y = [ y1 vertex1_y vertex2_y vertex4_y zone_y2 ];
    else
        zone_x = [ vertex1_x vertex2_x vertex4_x vertex3_x ];
        zone_y = [ vertex1_y vertex2_y vertex4_y vertex3_y ];
```

```
    end

    area = polyarea(zone_x,zone_y);
    percent_area_in_zone = area / rect_area * 100;
    Results(a,2) = percent_area_in_zone;
    Results(a,4) = 100 - percent_area_in_zone; %area within HCP zone
    Results(a,5) = hcp_length;

    fprintf(fid, formatSpec, short_name, Results(a,:));

end

fclose (fid);
%dlmwrite(Output, Results);
```

Appendix C

Data Tables

Table C.1: 10% by 125% HCP Zone Rectangle Data

Image	**% area in HCP zone**	**Total minutia**	o	e	$o-e$	$\frac{(o-e)^2}{e}$
26_8	16.268	57	6	9.3	-3.3	1.155
83_7	6.951	61	2	4.2	-2.2	1.184
17_8	2.941	52	0	1.5	-1.5	1.529
70_1	2.143	61	0	1.3	-1.3	1.308
32_2	3.970	56	1	2.2	-1.2	0.673
95_1	1.576	75	0	1.2	-1.2	1.182
53_2	5.454	58	2	3.2	-1.2	0.428
40_2	2.469	43	0	1.1	-1.1	1.062
8_1	2.408	84	1	2.0	-1.0	0.517
31_8	1.589	64	0	1.0	-1.0	1.017
19_7	1.572	62	0	1.0	-1.0	0.975
31_1	1.689	56	0	0.9	-0.9	0.946
22_2	3.682	52	1	1.9	-0.9	0.437
72_2	3.543	54	1	1.9	-0.9	0.436
93_8	1.419	64	0	0.9	-0.9	0.908
19_8	1.190	61	0	0.7	-0.7	0.726
32_8	1.873	37	0	0.7	-0.7	0.693
23_2	0.886	75	0	0.7	-0.7	0.664
80_8	1.507	43	0	0.6	-0.6	0.648
40_7	1.448	43	0	0.6	-0.6	0.623
3_1	3.011	53	1	1.6	-0.6	0.222
51_8	1.217	47	0	0.6	-0.6	0.572
48_1	1.221	44	0	0.5	-0.5	0.537
25_8	2.720	56	1	1.5	-0.5	0.180
97_2	1.023	51	0	0.5	-0.5	0.522
12_2	1.337	37	0	0.5	-0.5	0.495
74_7	0.637	75	0	0.5	-0.5	0.478
74_1	0.725	58	0	0.4	-0.4	0.420
13_1	0.995	41	0	0.4	-0.4	0.408
13_7	1.156	35	0	0.4	-0.4	0.405
48_2	2.366	58	1	1.4	-0.4	0.101
72_7	2.104	65	1	1.4	-0.4	0.099
30_1	1.799	76	1	1.4	-0.4	0.099
71_2	2.466	54	1	1.3	-0.3	0.083
72_1	2.013	64	1	1.3	-0.3	0.065
40_1	2.392	53	1	1.3	-0.3	0.057
77_1	2.308	54	1	1.2	-0.2	0.049
20_7	4.945	65	3	3.2	-0.2	0.014
63_2	1.814	66	1	1.2	-0.2	0.032
70_2	2.008	59	1	1.2	-0.2	0.029
1_2	0.404	42	0	0.2	-0.2	0.170
21_1	2.031	57	1	1.2	-0.2	0.021
9_8	2.721	42	1	1.1	-0.1	0.018
11_1	2.038	56	1	1.1	-0.1	0.018
12_7	0.228	59	0	0.1	-0.1	0.135

Table C.2: 10% by 125% HCP Zone Rectangle Data

Image	**% area in HCP zone**	**Total minutia**	o	e	$o-e$	$\frac{(o-e)^2}{e}$
9_2	2.623	43	1	1.1	-0.1	0.014
23_1	0.151	73	0	0.1	-0.1	0.110
17_2	3.473	60	2	2.1	-0.1	0.003
33_2	1.602	65	1	1.0	0.0	0.002
89_1	0.046	65	0	0.0	0.0	0.030
12_8	0.056	47	0	0.0	0.0	0.026
53_1	6.384	63	4	4.0	0.0	0.000
59_8	1.899	52	1	1.0	0.0	0.000
95_8	2.741	72	2	2.0	0.0	0.000
30_2	1.376	70	1	1.0	0.0	0.001
7_1	2.427	39	1	0.9	0.1	0.003
3_2	1.868	50	1	0.9	0.1	0.005
57_2	1.824	51	1	0.9	0.1	0.005
79_2	1.707	54	1	0.9	0.1	0.007
59_1	2.007	45	1	0.9	0.1	0.010
31_7	1.433	62	1	0.9	0.1	0.014
6_1	7.213	26	2	1.9	0.1	0.008
7_2	2.186	40	1	0.9	0.1	0.018
10_1	1.873	46	1	0.9	0.1	0.022
31_2	1.533	47	1	0.7	0.3	0.108
72_8	2.862	60	2	1.7	0.3	0.047
60_2	1.291	54	1	0.7	0.3	0.132
58_8	1.792	38	1	0.7	0.3	0.150
20_2	8.718	53	5	4.6	0.4	0.031
35_7	1.495	39	1	0.6	0.4	0.298
3_7	2.805	55	2	1.5	0.5	0.135
6_2	1.370	38	1	0.5	0.5	0.442
32_1	3.025	50	2	1.5	0.5	0.157
83_1	6.446	70	5	4.5	0.5	0.053
7_7	1.337	35	1	0.5	0.5	0.605
20_1	7.167	62	5	4.4	0.6	0.070
13_2	0.911	48	1	0.4	0.6	0.725
37_8	2.936	48	2	1.4	0.6	0.247
35_2	0.614	65	1	0.4	0.6	0.903
53_7	8.443	52	5	4.4	0.6	0.085
9_1	2.902	46	2	1.3	0.7	0.332
76_2	1.892	70	2	1.3	0.7	0.344
4_7	0.952	30	1	0.3	0.7	1.787
37_2	2.191	56	2	1.2	0.8	0.487
69_1	1.792	68	2	1.2	0.8	0.501
90_2	1.838	66	2	1.2	0.8	0.510
22_7	0.447	42	1	0.2	0.8	3.515
33_1	1.971	60	2	1.2	0.8	0.565
55_1	1.689	70	2	1.2	0.8	0.566
19_2	2.131	55	2	1.2	0.8	0.585

Table C.3: 10% by 125% HCP Zone Rectangle Data

Image	**% area in HCP zone**	**Total minutia**	o	e	$o-e$	$\frac{(o-e)^2}{e}$
80_1	2.533	45	2	1.1	0.9	0.649
46_1	3.107	68	3	2.1	0.9	0.373
99_7	1.549	69	2	1.1	0.9	0.812
21_2	1.846	51	2	0.9	1.1	1.191
22_1	4.667	41	3	1.9	1.1	0.617
61_7	1.644	55	2	0.9	1.1	1.328
82_2	3.618	78	4	2.8	1.2	0.492
53_8	5.814	48	4	2.8	1.2	0.524
8_2	2.267	76	3	1.7	1.3	0.947
80_2	1.695	42	2	0.7	1.3	2.331
50_7	1.360	52	2	0.7	1.3	2.363
90_8	2.969	57	3	1.7	1.3	1.011
51_1	2.147	77	3	1.7	1.3	1.097
76_1	2.756	59	3	1.6	1.4	1.161
2_7	1.707	35	2	0.6	1.4	3.291
90_1	2.325	68	3	1.6	1.4	1.273
26_1	6.526	70	6	4.6	1.4	0.449
41_2	2.358	66	3	1.6	1.4	1.340
95_2	3.125	49	3	1.5	1.5	1.409
75_1	3.408	74	4	2.5	1.5	0.867
86_2	2.211	68	3	1.5	1.5	1.490
74_2	0.701	61	2	0.4	1.6	5.785
61_2	1.980	71	3	1.4	1.6	1.807
10_8	3.342	41	3	1.4	1.6	1.939
29_1	3.225	42	3	1.4	1.6	2.000
97_1	3.485	67	4	2.3	1.7	1.188
66_2	2.311	57	3	1.3	1.7	2.150
17_1	4.121	56	4	2.3	1.7	1.241
44_1	2.385	54	3	1.3	1.7	2.276
75_2	3.441	66	4	2.3	1.7	1.317
42_2	5.528	59	5	3.3	1.7	0.926
3_8	2.927	43	3	1.3	1.7	2.410
44_2	2.318	51	3	1.2	1.8	2.794
42_1	4.955	64	5	3.2	1.8	1.054
36_1	3.568	58	4	2.1	1.9	1.800
50_2	2.364	45	3	1.1	1.9	3.523
63_1	2.279	45	3	1.0	2.0	3.802
28_7	2.888	70	4	2.0	2.0	1.936
29_7	4.521	44	4	2.0	2.0	2.032
2_2	2.438	40	3	1.0	2.0	4.205
64_2	1.649	58	3	1.0	2.0	4.367
49_1	1.359	70	3	1.0	2.0	4.412
16_7	2.516	34	3	0.9	2.1	5.376
4_2	6.887	41	5	2.8	2.2	1.677
5_8	9.295	41	6	3.8	2.2	1.258

Table C.4: 10% by 125% HCP Zone Rectangle Data

Image	**% area in HCP zone**	**Total minutia**	o	e	$o-e$	$\frac{(o-e)^2}{e}$
38_7	4.018	45	4	1.8	2.2	2.657
10_2	5.395	51	5	2.8	2.2	1.838
19_1	1.147	65	3	0.7	2.3	6.820
28_1	2.779	59	4	1.6	2.4	3.397
5_7	1.709	36	3	0.6	2.4	9.240
25_2	4.384	58	5	2.5	2.5	2.375
82_8	5.446	65	6	3.5	2.5	1.709
24_2	1.882	76	4	1.4	2.6	4.618
84_2	3.148	76	5	2.4	2.6	2.843
16_2	2.581	53	4	1.4	2.6	5.065
10_7	3.372	40	4	1.3	2.7	5.212
46_2	2.099	64	4	1.3	2.7	5.253
29_8	5.441	43	5	2.3	2.7	3.025
78_1	1.640	81	4	1.3	2.7	5.375
34_8	2.692	49	4	1.3	2.7	5.451
22_8	3.340	39	4	1.3	2.7	5.585
86_7	1.652	78	4	1.3	2.7	5.704
76_7	2.918	78	5	2.3	2.7	3.260
93_7	2.704	82	5	2.2	2.8	3.493
79_7	1.874	59	4	1.1	2.9	7.580
84_1	4.428	70	6	3.1	2.9	2.714
34_7	2.141	50	4	1.1	2.9	8.019
43_1	2.456	84	5	2.1	2.9	4.180
28_2	1.954	54	4	1.1	2.9	8.216
35_1	1.855	56	4	1.0	3.0	8.442
41_7	1.593	65	4	1.0	3.0	8.485
24_1	2.454	81	5	2.0	3.0	4.563
14_2	3.105	62	5	1.9	3.1	4.912
25_1	3.477	55	5	1.9	3.1	4.984
5_1	2.146	41	4	0.9	3.1	11.061
5_2	1.921	38	4	0.7	3.3	14.645
83_2	6.189	58	7	3.6	3.4	3.240
100_1	2.205	69	5	1.5	3.5	7.953
86_1	2.013	72	5	1.4	3.6	8.700
15_2	1.964	60	5	1.2	3.8	12.392
94_7	5.074	62	7	3.1	3.9	4.722
45_2	3.862	55	6	2.1	3.9	7.072
64_1	2.147	52	5	1.1	3.9	13.513
41_1	1.807	61	5	1.1	3.9	13.788
4_8	4.778	43	6	2.1	3.9	7.577
54_1	3.300	59	6	1.9	4.1	8.439
44_7	3.612	51	6	1.8	4.2	9.384
58_2	3.781	45	6	1.7	4.3	10.859
82_7	4.040	64	7	2.6	4.4	7.537
11_2	2.536	62	6	1.6	4.4	12.465

Table C.5: 10% by 125% HCP Zone Rectangle Data

Image	**% area in HCP zone**	**Total minutia**	o	e	$o-e$	$\frac{(o-e)^2}{e}$
75_7	3.218	77	7	2.5	4.5	8.254
58_7	2.767	51	6	1.4	4.6	14.923
81_2	3.767	61	7	2.3	4.7	9.622
82_1	3.195	70	7	2.2	4.8	10.143
34_1	3.526	59	7	2.1	4.9	11.634
81_1	4.953	61	8	3.0	5.0	8.205
56_2	3.501	56	7	2.0	5.0	12.955
38_1	3.364	49	7	1.6	5.4	17.378
34_2	4.269	60	8	2.6	5.4	11.545
81_7	3.358	65	8	2.2	5.8	15.503
52_1	4.292	72	9	3.1	5.9	11.304
52_7	4.121	70	9	2.9	6.1	12.966
91_7	7.300	51	10	3.7	6.3	10.584
4_1	7.901	47	10	3.7	6.3	10.642
36_2	2.542	52	8	1.3	6.7	33.743
81_8	4.057	55	9	2.2	6.8	20.532
52_8	4.951	73	11	3.6	7.4	15.093
52_2	4.342	73	11	3.2	7.8	19.343
14_1	6.537	58	12	3.8	8.2	17.770
99_1	3.251	91	12	3.0	9.0	27.626

Table C.6: 25% by 125% HCP Zone Rectangle Data

Image	**% area in HCP zone**	**Total minutia**	o	e	$o-e$	$\frac{(o-e)^2}{e}$
32_2	9.924	56	3	5.6	-2.6	1.177
72_7	5.259	65	1	3.4	-2.4	1.711
30_1	4.497	76	1	3.4	-2.4	1.710
26_8	40.670	57	21	23.2	-2.2	0.205
72_2	8.858	54	3	4.8	-1.8	0.665
17_1	10.303	56	4	5.8	-1.8	0.543
32_8	4.682	37	0	1.7	-1.7	1.732
6_1	18.032	26	3	4.7	-1.7	0.608
71_2	6.165	54	2	3.3	-1.3	0.531
12_2	3.342	37	0	1.2	-1.2	1.236
13_7	2.890	35	0	1.0	-1.0	1.012
95_1	3.939	75	2	3.0	-1.0	0.308
9_8	6.802	42	2	2.9	-0.9	0.257
17_8	7.353	52	3	3.8	-0.8	0.177
22_2	9.204	52	4	4.8	-0.8	0.129
32_1	7.563	50	3	3.8	-0.8	0.162
80_8	3.767	43	1	1.6	-0.6	0.237
33_2	4.005	65	2	2.6	-0.6	0.140
83_7	17.378	61	10	10.6	-0.6	0.034
31_8	3.973	64	2	2.5	-0.5	0.116
59_8	4.746	52	2	2.5	-0.5	0.089
19_7	3.929	62	2	2.4	-0.4	0.078
51_8	3.041	47	1	1.4	-0.4	0.129
48_1	3.053	44	1	1.3	-0.3	0.088
12_7	0.571	59	0	0.3	-0.3	0.337
97_2	2.558	51	1	1.3	-0.3	0.071
75_1	8.519	74	6	6.3	-0.3	0.015
6_2	3.424	38	1	1.3	-0.3	0.070
83_1	16.115	70	11	11.3	-0.3	0.007
23_1	0.378	73	0	0.3	-0.3	0.276
93_8	3.548	64	2	2.3	-0.3	0.032
70_1	5.359	61	3	3.3	-0.3	0.022
72_1	5.034	64	3	3.2	-0.2	0.015
17_2	8.683	60	5	5.2	-0.2	0.008
7_2	5.466	40	2	2.2	-0.2	0.016
36_1	8.921	58	5	5.2	-0.2	0.006
7_7	3.342	35	1	1.2	-0.2	0.025
10_1	4.682	46	2	2.2	-0.2	0.011
77_1	5.770	54	3	3.1	-0.1	0.004
89_1	0.115	65	0	0.1	-0.1	0.074
12_8	0.140	47	0	0.1	-0.1	0.066
90_2	4.596	66	3	3.0	0.0	0.000
35_2	1.536	65	1	1.0	0.0	0.000
63_2	4.535	66	3	3.0	0.0	0.000
3_1	7.527	53	4	4.0	0.0	0.000

Table C.7: 25% by 125% HCP Zone Rectangle Data

Image	**% area in HCP zone**	**Total minutia**	o	e	$o-e$	$\frac{(o-e)^2}{e}$
70_2	5.020	59	3	3.0	0.0	0.000
55_1	4.222	70	3	3.0	0.0	0.001
4_7	2.380	30	1	0.7	0.3	0.115
75_2	8.601	66	6	5.7	0.3	0.018
23_2	2.215	75	2	1.7	0.3	0.069
40_2	6.172	43	3	2.7	0.3	0.045
40_7	3.621	43	2	1.6	0.4	0.126
38_7	10.045	45	5	4.5	0.5	0.051
22_7	1.117	42	1	0.5	0.5	0.600
35_7	3.738	39	2	1.5	0.5	0.202
10_8	8.354	41	4	3.4	0.6	0.097
1_2	1.010	42	1	0.4	0.6	0.781
30_2	3.440	70	3	2.4	0.6	0.146
7_1	6.067	39	3	2.4	0.6	0.170
21_2	4.614	51	3	2.4	0.6	0.178
9_1	7.254	46	4	3.3	0.7	0.132
34_8	6.729	49	4	3.3	0.7	0.150
61_7	4.110	55	3	2.3	0.7	0.242
59_1	5.017	45	3	2.3	0.7	0.244
40_1	5.980	53	4	3.2	0.8	0.218
13_2	2.276	48	2	1.1	0.9	0.754
74_2	1.752	61	2	1.1	0.9	0.812
37_2	5.477	56	4	3.1	0.9	0.284
8_1	6.020	84	6	5.1	0.9	0.176
82_2	9.045	78	8	7.1	0.9	0.127
90_1	5.813	68	5	4.0	1.0	0.277
19_1	2.867	65	3	1.9	1.1	0.693
82_8	13.615	65	10	8.9	1.1	0.149
80_1	6.333	45	4	2.8	1.2	0.464
60_2	3.227	54	3	1.7	1.3	0.907
99_7	3.871	69	4	2.7	1.3	0.661
44_7	9.030	51	6	4.6	1.4	0.422
63_1	5.697	45	4	2.6	1.4	0.805
37_8	7.341	48	5	3.5	1.5	0.619
48_2	5.915	58	5	3.4	1.6	0.718
64_2	4.122	58	4	2.4	1.6	1.083
10_7	8.429	40	5	3.4	1.6	0.786
4_2	15.536	41	8	6.4	1.6	0.417
3_2	4.670	50	4	2.3	1.7	1.187
66_2	5.778	57	5	3.3	1.7	0.885
72_8	7.154	60	6	4.3	1.7	0.679
46_1	7.768	68	7	5.3	1.7	0.559
22_8	8.350	39	5	3.3	1.7	0.933
86_7	4.131	78	5	3.2	1.8	0.981
44_1	5.963	54	5	3.2	1.8	0.984

Table C.8: 25% by 125% HCP Zone Rectangle Data

Image	**% area in HCP zone**	**Total minutia**	o	e	$o-e$	$\frac{(o-e)^2}{e}$
28_1	6.948	59	6	4.1	1.9	0.881
28_7	7.220	70	7	5.1	1.9	0.749
13_1	2.488	41	3	1.0	2.0	3.842
44_2	5.796	51	5	3.0	2.0	1.413
33_1	4.927	60	5	3.0	2.0	1.413
95_8	6.853	72	7	4.9	2.1	0.865
19_2	5.328	55	5	2.9	2.1	1.462
42_1	12.388	64	10	7.9	2.1	0.541
21_1	5.077	57	5	2.9	2.1	1.533
41_2	5.894	66	6	3.9	2.1	1.144
54_1	8.249	59	7	4.9	2.1	0.935
3_7	7.014	55	6	3.9	2.1	1.190
11_1	5.096	56	5	2.9	2.1	1.614
25_8	6.800	56	6	3.8	2.2	1.262
80_2	4.237	42	4	1.8	2.2	2.770
50_7	3.400	52	4	1.8	2.2	2.817
86_2	5.527	68	6	3.8	2.2	1.337
81_2	9.418	61	8	5.7	2.3	0.885
34_7	5.352	50	5	2.7	2.3	2.019
50_2	5.911	45	5	2.7	2.3	2.059
81_1	12.382	61	10	7.6	2.4	0.793
20_2	21.795	53	14	11.6	2.4	0.519
5_7	4.274	36	4	1.5	2.5	3.938
2_7	4.268	35	4	1.5	2.5	4.204
2_2	6.094	40	5	2.4	2.6	2.693
16_2	6.452	53	6	3.4	2.6	1.947
31_1	4.223	56	5	2.4	2.6	2.937
78_1	4.099	81	6	3.3	2.7	2.163
76_2	4.731	70	6	3.3	2.7	2.182
79_2	4.269	54	5	2.3	2.7	3.151
31_7	3.584	62	5	2.2	2.8	3.473
74_7	1.593	75	4	1.2	2.8	6.586
75_7	8.044	77	9	6.2	2.8	1.271
3_8	7.317	43	6	3.1	2.9	2.589
74_1	1.812	58	4	1.1	2.9	8.276
83_2	15.472	58	12	9.0	3.0	1.020
24_1	6.136	81	8	5.0	3.0	1.847
42_2	13.345	59	11	7.9	3.1	1.241
95_2	7.812	49	7	3.8	3.2	2.629
9_2	6.557	43	6	2.8	3.2	3.587
19_8	2.976	61	5	1.8	3.2	5.587
31_2	3.833	47	5	1.8	3.2	5.679
64_1	5.367	52	6	2.8	3.2	3.691
25_1	8.693	55	8	4.8	3.2	2.167
84_1	11.070	70	11	7.7	3.3	1.364

Table C.9: 25% by 125% HCP Zone Rectangle Data

Image	**% area in HCP zone**	**Total minutia**	o	e	$o-e$	$\frac{(o-e)^2}{e}$
58_8	4.480	38	5	1.7	3.3	6.389
86_1	5.032	72	7	3.6	3.4	3.148
35_1	4.637	56	6	2.6	3.4	4.460
82_1	7.989	70	9	5.6	3.4	2.077
41_7	3.983	65	6	2.6	3.4	4.493
24_2	4.704	76	7	3.6	3.4	3.280
93_7	6.759	82	9	5.5	3.5	2.156
82_7	10.100	64	10	6.5	3.5	1.935
81_7	8.395	65	9	5.5	3.5	2.300
26_1	16.314	70	15	11.4	3.6	1.122
34_2	10.674	60	10	6.4	3.6	2.019
29_1	8.062	42	7	3.4	3.6	3.858
46_2	5.248	64	7	3.4	3.6	3.948
25_2	10.959	58	10	6.4	3.6	2.089
57_2	4.560	51	6	2.3	3.7	5.807
4_1	19.753	47	13	9.3	3.7	1.487
58_2	9.453	45	8	4.3	3.7	3.299
90_8	7.422	57	8	4.2	3.8	3.359
34_1	8.815	59	9	5.2	3.8	2.775
5_1	5.366	41	6	2.2	3.8	6.563
16_7	6.290	34	6	2.1	3.9	6.972
4_8	11.944	43	9	5.1	3.9	2.907
51_1	5.368	77	8	4.1	3.9	3.618
20_1	17.917	62	15	11.1	3.9	1.363
76_1	6.889	59	8	4.1	3.9	3.810
53_1	15.959	63	14	10.1	3.9	1.549
69_1	4.479	68	7	3.0	4.0	5.133
84_2	7.869	76	10	6.0	4.0	2.701
53_7	21.108	52	15	11.0	4.0	1.475
15_2	4.910	60	7	2.9	4.1	5.578
56_2	8.752	56	9	4.9	4.1	3.428
94_7	12.684	62	12	7.9	4.1	2.175
29_8	13.603	43	10	5.8	4.2	2.945
43_1	3.365	84	7	2.8	4.2	6.161
5_2	4.803	38	6	1.8	4.2	9.549
22_1	11.667	41	9	4.8	4.2	3.717
41_1	4.516	61	7	2.8	4.2	6.541
28_2	4.886	54	7	2.6	4.4	7.210
81_8	10.143	55	10	5.6	4.4	3.505
58_7	6.917	51	8	3.5	4.5	5.670
49_1	3.397	70	7	2.4	4.6	8.982
38_1	8.409	49	9	4.1	4.9	5.778
20_7	12.363	65	13	8.0	5.0	3.066
29_7	11.303	44	10	5.0	5.0	5.081
11_2	6.341	62	9	3.9	5.1	6.535

Table C.10: 25% by 125% HCP Zone Rectangle Data

Image	**% area in HCP zone**	**Total minutia**	o	e	$o-e$	$\frac{(o-e)^2}{e}$
53_2	13.636	58	13	7.9	5.1	3.278
10_2	13.488	51	12	6.9	5.1	3.813
100_1	5.512	69	9	3.8	5.2	7.099
76_7	7.295	78	11	5.7	5.3	4.954
5_8	23.237	41	15	9.5	5.5	3.144
61_2	4.951	71	9	3.5	5.5	8.559
45_2	9.655	55	11	5.3	5.7	6.096
91_7	18.249	51	15	9.3	5.7	3.482
53_8	14.535	48	13	7.0	6.0	5.200
97_1	8.712	67	12	5.8	6.2	6.508
79_7	4.684	59	9	2.8	6.2	14.074
8_2	5.667	76	11	4.3	6.7	10.400
36_2	6.355	52	10	3.3	6.7	13.567
14_2	7.762	62	13	4.8	8.2	13.930
52_1	10.729	72	17	7.7	9.3	11.137
14_1	16.343	58	19	9.5	9.5	9.563
52_7	10.302	70	17	7.2	9.8	13.288
99_1	8.129	91	18	7.4	10.6	15.198
52_2	10.855	73	19	7.9	11.1	15.480
52_8	12.377	73	21	9.0	12.0	15.843

Bibliography

[1] E.-C. Chang and Q. Li. Hiding secret points amidst chaff. In S. Vaudenay, editor, *Advance in Cryptology - EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 59–72, 2006.

[2] S. Dass and A. K. Jain. Fingerprint classification using orientation field flow curves. In *Proc. Indian Conf. Computer Vision, Graphics and Image Processing*, pages 650–655, 2004.

[3] Y. Dodis, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In C. Cachin and J. Camenisch, editors, *Advance in Cryptology — EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 523–540, 2004.

[4] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM Journal on Computing*, 38 (1):97–139, 2008. URL `http://link.aip.org/link/?SMJ/38/97/1`.

[5] X. Jiang and W.-Y. Yau. Fingerprint minutiae matching based on the local and global structures. In *Proceedings of the 15th International Conference on Pattern Recognition*, volume 2, pages 1038–1041, 2000.

[6] A. Juels and M. Sudan. A fuzzy vault scheme. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT 2002)*, Lausanne, Switzerland, 2002.

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

[8] K. Ko. User's guide to NIST biometric image software (NBIS). NIST Interagency/Internal Report (NISTIR) - 7392, January 21 2007. URL `http://www.nist.gov/manuscript-publication-search.cfm?pub_id=51097`.

[9] A. J. Kotlarchyk, A. S. Pandya, and H. Zhuang. Simulation and experimental studies on fuzzy vault fingerprint cryptography. *International Journal of Knowledge-based and Intelligent Engineering Systems*, 12(5-6):305–317, 2008.

[10] P. Li, X. Yang, K. Cao, X. Tao, R. Wang, and J. Tian. An alignment-free fingerprint cryptosystem based on fuzzy vault scheme. *Journal of Network and Computer Applications*, 33:207–220, 2010.

[11] S. Lin and D. J. Costello. *Error Control Coding*. Prentice Hall, second edition, April 1 2004. ISBN 0130426725.

[12] D. Maltoni, D. Maio, A. K. Jain, and S. Prabhakar. *Handbook of Fingerprint Recognition*. Springer, 2nd edition, 2009. ISBN 978-1-84882-253-5.

[13] Moxie Marlinspike. Divide and conquer: Cracking ms-chapv2 with a 100 CloudCracker::Blog, July 2012. URL `https://www.cloudcracker.com/blog/2012/07/29/cracking-ms-chap-v2/`.

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

[15] M. Purser. *Introduction to Error-Correcting Codes*. Artech House, 1995. ISBN 0-89006-784-8.

[16] B. Rodes and X. Wang. Security analysis of a fingerprint-protected USB drive. In *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC '10)*, pages 89–96, New York, NY, USA, December 06-10 2010. ACM.

[17] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[18] M. Tico and P. Kuosmanen. Fingerprint matching using an orientation-based minutia descriptor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8): 1009–1014, 2003.

[19] X. Wang, P. D. Huff, and B. Tjaden. Improving the efficiency of capture-resistant biometric authentication based on set intersection. In *Proceedings of the 24th Annual Computer Security Applications Conference (ACSAC 2008)*, pages 140–149, Anaheim, CA, December 8-12 2008. IEEE Computer Society Press.

[20] X. Wang, X. Zhang, and H. McMillen. On the risks of a fingerprint fuzzy vault. Submitted, June 2012.

[21] C. I. Watson, M. D. Garris, E. Tabassi, C. L. Wilson, R. Mi.McCabe, S. Janet, and K. Ko. User's guide to NIST biometric image software export control(NBIS-EC). NIST Interagency/Internal Report (NISTIR) - 7392, January 21 2007. URL http://www.nist.gov/customcf/get_pdf.cfm?pub_id=51096.