MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Biometrics Used for Random Number Generation

BACHELOR'S DEGREE THESIS

Shkodran Gerguri

Brno, Spring 2008

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Advisor: doc. RNDr. Václav Matyáš, MSc., PhD.

Acknowledgement

I would like to thank my supervisor, doc. RNDr. Václav Matyáš, MSc., PhD., for his continuing support, encouragement and supervision during my work on the thesis.

Special thanks goes to RNDr. Vladimír Ulman for kindly providing the image processing programs.

Abstract

A lot of recent research focuses on the field of biometric key generation and "locking", which promises to replace passwords by biometrics in the process of extracting a secret value from a secure storage. One of the major obstacles on the way to a reliable key generation technique is the variability in obtained biometric samples. This variability could be exploited for random number generation.

The purpose of this thesis is to give an overview of some of the interesting results in the field of biometric key generation and "locking", as well as to present a proposed method of generating random numbers from fingerprints and calculate the entropy of shift and rotation of fingerprints, which is seen as one of the contributing factors to the overall entropy of a fingerprint scanning.

Keywords

biometrics, fingerprint, fingerprint shift and rotation, generation, information entropy, pseudorandom number

Contents

Chapter 1

Introduction

Biometrics enjoy widespread use in today's security technologies. First deployed in authentication mechanisms, biometrics were intended to replace passwords. Strong passwords are hard to remember, and the use of a single password for multiple systems poses a security risk. Biometrics, on the other hand, cannot be attacked as straightforwardly as passwords, which promises the use of a person's biometrics across multiple system. Another advantage is that the user need not remember any value or passphrase. Biometrics as an authentication mechanism have become so popular that most recent laptops come equipped with a fingerprint reader and corresponding software to provide means to log in to a operating system by scanning a finger of a user.

More recently, the possibility of key generation and key "locking" using biometrics have been explored. Secret key material has been up to now usually protected by means of passwords, effectively decreasing the security strength of the key material down to the strength of the password, which can be often successfully attacked by dictionary attacks. As in authentication systems, this issue becomes more critical with multiple key material. The idea to use biometrics to access a secret value is appealing and many such schemes have been proposed. As a result, substantial effort is being put into research of reliable methods of (re)constructing a secret key from a user's biometrics.

One of the major obstacles on the way to reliable key reconstruction using biometrics is the inherent variability in biometrics. Two samples of a user's biometric are almost never identical, and so error-correcting methods have to be applied to compensate for the differences in the obtained samples to always obtain the correct key. On the other hand, the natural variability may make it possible to generate new cryptographic keys, instead of reconstructing them. Dr. Matyáš, Dr. Říha and Dr. Smolík recently proposed such a method, suitable for use on mobile devices with fingerprint reader, such as PDA, mobile phones and laptops.

The purpose of this thesis is to present an overview of some of the recent

and interesting results in the field of biometric key generation and reconstruction, introduce the proposed method of generating cryptographic keys using fingerprints, and estimate the entropy of shift and rotation of fingerprints, which is seen as a contributing factor to the variability in captured fingerprint samples.

The rest of the thesis is organized as follows: Chapter 2 covers pseudo-random number generators from the viewpoint of the NIST Special Publication SP 800-90 [1]. Chapter 3 gives a detailed overview of some of the recent methods in biometric key generation and reconstruction. Chapter 4 introduces the proposed method of generating random numbers from fingerprints. Chapter 5 covers the calculation of entropy of fingerprint shift and rotation and its results. The thesis concludes in Chapter 6.

Chapter 2

Pseudorandom Number Generators

Modern cryptographic algorithms follow the Kerckhoffs' principle: the algorithm itself is publicly known, while the key remains secret. As a result, the key space must be large enough to prevent brute-force attacks, and every key from the key space must be equally likely to be selected in order to prevent reductions of key space size. To ensure such property of the key space, the key-generating function must generate all keys from the key space with equal probability.

Formally, let X denote a discrete random variable, $X \in \{0, 1\}$. A bit b is called *random*, if $P(X = b) = 0.5$. Similarly, a bit sequence is called random, if $P(X = b) = 0.5$ for every b in the sequence and all the bits are statistically independent. A *random bit generator* is in this context a device or an algorithm that outputs random bit sequences. As numbers can be expressed by binary sequences, a *random number* and *random number generators* can be defined using random bit sequences and random bit generators. Random bit generators can be either hardware-based, measuring physical phenomena such as radioactive decay, or software-based, measuring various computer processes such as system clocks.

Let $\mathcal{M} = \{0, 1\}^m$ denote the set of bit sequences of length m, and $\mathcal{N} = \{0, 1\}^n$ denote the set of bit sequences of lenght n, where $n > m$. A *pseudorandom bit generator* is in this context an algorithm that maps a random bit sequence $x \in \mathcal{M}$ to a bit sequence $y \in \mathcal{N}$. The value x is called the *seed*, while y is called the *pseudorandom bit sequence*.

The important property of random bit sequences is that, given a bit subsequence, it is impossible to determine the next bit with probability higher than 0.5. The primary aim of pseudorandom bit generators is to produce bit sequences of higher length that appear to be random. To ensure this, it is necessary that x is sufficiently long to prevent brute force attacks on the set $\mathcal{M}$. Furthermore, it is required that the generated pseudorandom bit sequences are statistically indistinguishable from random bit sequences of the same length and the property of random bit sequences (the inability to determine the next bit with higher probability) holds for an attacker with

limited computational resources. [12]

It is possible to test the pseudorandom bit generator by *statistical tests for randomness*. These test sample pseudorandom bit sequences of the generator for various properties a random bit sequence possesses. Some statistical tests for randomness include the *Maurer's universal statistical test* [13] or the test suite specified in FIPS 140-1.

The rest of this chapter will introduce the architecture of a pseudorandom bit generator (PRBG), as described in the NIST Special Publication 800-90 [1].

2.1 Pseudorandom Bit Generators Architecture

From a high level perspective, a PRBG has two components – a *Mechanism* and a *Source of Entropy Input* (Source). A Source supplies the Mechanism with a bit sequence that contains entropy and the Mechanism performs computation upon it. The overall structure of a PRBG as defined in [1] is depicted in fig. 2.1.

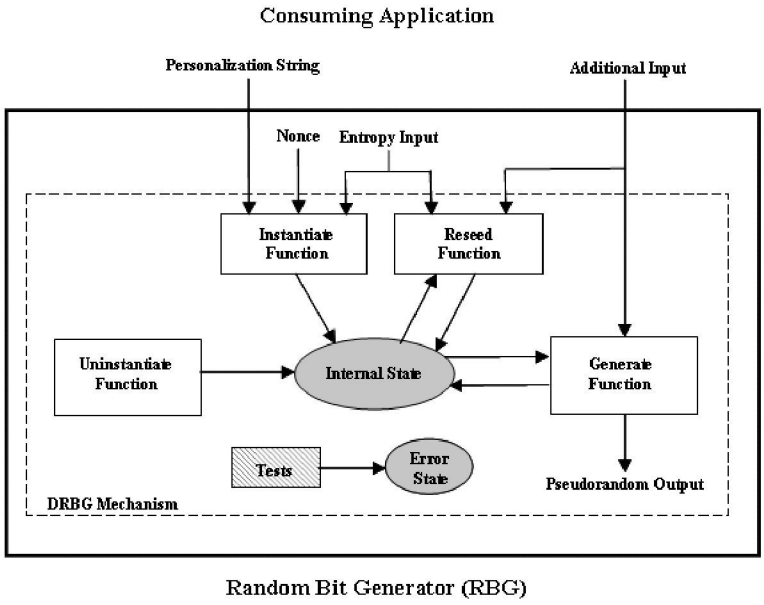


Figure 2.1: The overall structure of a pseudorandom bit generator, as specified in the NIST SP 800-90 (taken from [1]).

2.1.1 Mechanism Structure

When a consuming application requires pseudorandom data, it must either make use of an existing instantiation of the PRBG, or call the *Instantiate Function* to create a new one. The Instantiate Function creates an instantiation of a PRBG – a logical instance of the generator with its own internal state. Inputs to Instantiate Function include an *Entropy Input*, a bit sequence containing entropy; a time-varying value with a negligible chance of repeating, called *Nonce*; and optionally a *Personalization String*, a value that mainly distinguishes between existing instantiations of the PRBG and may or may not contain entropy. A *seed* is either derived from these values, or taken directly as the values put together. PRBGs are instantiated to one of the four supported *security strengths* – 112, 128, 196 or 256 bits. Security strength determines the maximum entropy a pseudorandom bit sequence, generated by the instantiation, may contain.

When an instantiation of a PRBG is no longer required, a call to the *Uninstantiate Function* discards that instantiation and frees all the resource that particular instantiation has been using.

Internal State is a set of values in internal variables of a Mechanism, containing one or more values derived from a seed; a *request counter*, that keeps track of the number of requests for pseudorandom bit sequences; and various other parameters of the generator instantiation. A subset of variables from Internal State, containing the values derived from a seed and the counter, forms the *Working State*.

Working State is used for generation of pseudorandom data by the *Generate Function*. Requests for pseudorandom bit sequences result in new Internal and Working States.

To ensure secure operation, PRBG instantiations need to be supplied with new seeds after a number of calls to the Generate Function has been made. The *Reseed Function* obtains new Entropy Input and possibly some additional input and reseeds the instantiation with the seed obtained from these values.

The PRBG instantiation is periodically checked for secure and correct operation by the *Health Tests*. If an instantiation is found to be erroneous or insecure, the instantiation transitions to an *Error State*.

A Mechanism can be divided into multiple parts (components) and distributed, each part providing partial functionality of the Mechanism. Secure communication between the distributed components must be ensured in such a case.

2.1.2 Source of Entropy Input Structure

A Source of Entropy Input supplies the Mechanism with Entropy Input. A Source can be any of the following:

1. random bit generator (RBG)
2. pseudorandom bit generator, or a chain of PRBGs; in such a case, the (first) PRBG must be seeded with output from the RBG
3. a suitable entropy source

Entropy source is one possible Source of Entropy Input for PRBG. It consists of a *noise source, digitization function, health test function* and an optional *conditioning function*. A noise source provides the raw data that contain entropy. The digitization function converts the obtained raw data into a bit sequence. The resulting bit sequences, however, may not have a uniform distribution, so the optional conditioning function may be employed to retrieve a bit sequence of shorter length and uniform distribution. Finally, the resulting data are assessed for entropy, and optionally processed to sequences with full entropy. [1]

2.1.3 Supported Mechanism Algorithms

NIST SP 800-90 specifies four Mechanism algorithms suitable for pseudorandom number generation, based on:

- approved hash functions,
- AES or TDEA in CTR mode,
- HMAC with an approved hash function,
- dual elliptic curves.

In the case of hash functions, a hash function in counter mode is used to generate the pseudorandom data. When pseudorandom data are requested, a secret value in Internal State is iteratively changed to generate each successive n-bit block of requested output, with n being the number of bits in an output block of the hash function. All SHA-family functions are allowed, i.e., SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512. A derivaton function is used to obtain a seed from input to the Instantiation Function. This

1. A bit sequence has *full entropy* if its entropy equals its bit length.

approach is faster than using HMAC, and computation can be parallelized. The maximum number of requested bits is 2^{19} , and the maximum number of requests before reseeding is required is 2^{48}.

HMAC Mechanism uses an Approved hash function in a HMAC construction. An initial value V and a secret key K are used to produce the pseudorandom data:

$$V = \mathrm{HMAC}(K, V).$$

A new K and V is generated in the process. This approach is slower than using purely hash functions, but has smaller requirements on security of the underlying hash function. It is parallelizable, and the number of requests before reseeding and the maximum number of requested bits is the same is in generators based purely on hash functions.

Mechanisms based on block ciphers in CTR mode may be used as well. One advantage of this approach is that the security of Mechanisms based on CTR is effectively reduced to the security of the underlying block cipher. Another advantage is that the computation can be parallelized. In case of AES, up to 2^{48} requests for pseudorandom data can be made, each one of maximum 2^{19} bits. In case of TDEA, the smaller block size results in smaller upper bounds on requests and pseudorandom bit sequences length (2^{32} and 2^{13} , respectively). A CTR-based Mechanism can opt not to use a derivation function to create the seed and take the raw Entropy Input as a seed itself.

Mechanisms based on Dual Elliptic Curves generate pseudorandom outputs by extracting bits from elliptic curve points. In most cases, these Mechanisms are the slowest of the four types described in this section. The maximum number of requests before reseeding is required is 2^{32} , with a maximum of either $2^{32} \times 240$, $2^{32} \times 368$ or $2^{32} \times 504$ bits, depending on the curve used.

Chapter 3

Biometrics and Cryptography

Traditional biometric authentication systems use a database of templates against which a freshly obtained biometric sample is matched. Such a centralized storage of biometrics from multiple users presents a risk if the biometrics are used in other systems as well. To remedy the situation, multiple schemes that do not require a template to be stored were proposed. The techniques used either try to derive a key directly from the biometrics' characteristics, or use biometrics as means to access a secret key, either by combining the secret value and the biometrics together or by storing the secret key in a protected form and reconstructing it every time the user authenticates successfully. This chapter gives an overview of these three techniques. Additional schemes of interest are covered in Appendix A.

An initial overview of the field was written by Ondřej Bitto [2]. The overview gives an outline of 5 techniques and one related thesis; three of the five methods are also covered in this chapter. One of the other methods, the so-called "Password hardening" [15], is, however, completely unrelated to biometric key generation, and merely serves as a basis for later research performed by the authors on key generation from biometrics using speech recognition. The thesis covered by the overview also deals with fingerprint recognition in general, instead of key generation using fingerprints.

The text was used as a starting point for initial overview of the field. This chapter presents an enhanced and more thorough overview of the field, including some very recent results.

3.1 Key Retrieval Techniques

Key retrieval techniques refer to methods of retrieving a secret key, which is either stored in a protected form, or combined with the biometrics to produce a "hardened" template from which the secret key can be obtained when a biometric sample of a legitimate user is obtained.

Contents of this section:

- Fuzzy Commitment Scheme
- Biometric Key Binding
- Combining IrisCode with Cryptographic Keys

3.1.1 Fuzzy Commitment Scheme

Juels and Wattenberg propose a new cryptographic scheme in [10]. A commitment scheme is a mapping $Commit : \{0,1\} \times X \rightarrow Y$. When Alice wants to commit a value she computes $y = Commit(b, w)$, known as *blob*, and sends y to Bob. The values b and w represent the selected bit and a random value called a *witness*. Bob then makes a guess as to which value Alice has committed, and afterwards Alice tells Bob whether he was right or wrong. She also provides the values b and w so that Bob can decommit the blob (i.e., obtain the committed value) and verify whether Alice told him the truth.

A commitment scheme is said to be *binding* if it is infeasible to decommit the blob with an incorrect bit. A commitment scheme is *concealing* if the probability of Bob guessing the committed bit is not significantly greater than 0.5. [10]

The *fuzzy commitment scheme*, as proposed by Juels and Wattenberg, is an extension of the commitment scheme – the function is defined as $Commit : C \times X \rightarrow Y$, where $C = \{0,1\}^n$, $n \in \mathbb{N}$ or C is equal to a string of some other values. Furthermore, in the fuzzy commitment scheme, decommitment of the blob y is achievable using the witness x only, and the scheme has the property of *strong binding* – blob decommitment using two different witnesses x and x' is impossible. Finally, for any two witnesses whose distance is within some boundary under a suitable metric it holds that either witness can successfully decommit the blob y. The underlying construction uses both hash functions and error-correcting codes to achieve such properties.

Internals

Scheme compounds:

- committed value $c \in C$, $C \subseteq \{0,1\}^n$, $n \in \mathbb{N}$
- witness $x = c \oplus \delta$, $\delta \in \{0,1\}^n$

- decoding function $f : \{0,1\}^n \rightarrow C \bigcup \{\perp\}$
- hash/one-way function $h : \{0,1\}^n \rightarrow \{0,1\}^l, n, l \in \mathbb{N}, n \neq l$
- commitment function $F : (\{0,1\}^n, \{0,1\}^n) \rightarrow (\{0,1\}^l, \{0,1\}^n)$, defined as follows:
  $F(c,x) = (h(c), c \oplus x)$

The committed value is a codeword of an error-correcting code. In certain cases, we may be unable to transform a message into a codeword, such as in the case of a fingerprint reader. We may consider the error-prone fingerprint reader as the noisy channel and the fingerprint as the message. Because it is impossible to transform the message in such case, the fingerprint/witness is treated as a corrupted codeword instead and no mapping between the message space and codeword space takes place.

Decoding function f takes a binary string and tries to map it to the nearest codeword. If such a mapping is impossible, the string is mapped to $\perp$.

Since the witness is a binary string of length n, it can be expressed as a XOR of the committed value c and an offset δ. When committing a secret value c, $F(c,x) = (h(c), x \oplus c) = (\alpha, \delta)$is computed. When a user wants to decommit the blob using a witness x' he simply substracts δ from x' and applies f to the result. Intuitively, the witness x' may be viewed as $x' = c \oplus \delta'$, and by subtracting δ we get $f(c \oplus (\delta' \oplus \delta)) = c'$. If $h(c') = \alpha$, the blob has been successfully decommitted and the secret retrieved, with $c' = c$.

If the code used has a correction threshold (number of bits it can correct) of $\frac{qn}{100}$, where n is the length of the codewords, then F is q % resilient, i.e. can correct up to q % of bits in the commitment. Security of the proposed scheme is proved in [10].

Applications

The scheme is applicable to a wide range of scenarios. The paper presents protocols for authentication, challenge-response and encryption, of which the Fuzzy Authentication protocol is described as an example below. All the protocols may use a biometric sample as a witness without modification with a suitable error-correcting code. It is important to note that due to the construction of the scheme, no biometric template needs to be stored.

Fuzzy Authentication:

- *Enrollment*
  1. Enroll the witness x - a fingerprint.
  2. Choose $c \in C$ randomly.
  3. Compute $y = F(c, x)$, and store y.
- *Authentication*
  1. User claiming an identity presents a witness x'.
  2. Look up y for the claimed identity.
  3. Check whether x' yields successful decommitment.
  4. If decommitment is successful, authenticate user under the claimed identity, if not, reject user.

3.1.2 Biometric Key Binding

An approach where the secret key is combined with biometrics such as a fingerprint to produce a secure template is presented in [20, 19]. The technique, called *Biometric Key Binding* (BKB) and developed by Soutar *et al.*, combines the biometric template, obtained during enrollment, with the secret key to produce a *user record* which is stored on a nonvolatile storage. The user record is resilient to attacks in the sense that neither key nor the biometric template can be obtained from the record itself. When the user wants to release the secret key, the biometric image reader obtains a sample of the user's biometric, which is combined with the user record by the BKB algorithm to retrieve the secret key, if the user is legitimate. One of the advantages of this scheme is that the biometric template is not directly stored and the key can be easily canceled. In fact, the scheme has been successfully turned into a commercial product offered by Bioscrypt, Inc.

Enrollment and verification processes are explained in the following two sections. Fingerprints are used as the model biometric.

Enrollment

The enrollment process can be broken into three stages:

1. fingerprint enrollment and transformation
2. secret key encryption and hashing
3. fingerprint/secret key linking

Fingerprint enrollment: A series of fingerprints is obtained, upon which Fourier transform is applied to obtain *signal magnitude* and *signal phase*. *Signal conjugate phase* is computed from signal phase and a *random phase* is generated, and the product of these is stored in the user record as a filter[1]. Finally, product of the four components (signal magnitude, signal phase, signal conjugate phase, random phase) is computed and inverse Fourier transform is applied to the result, producing the *output pattern*.

Secret key encryption and hashing: The secret key is input to an encryption algorithm which takes S bits of filter (the product of signal conjugate phase and random phase) as its encryption key. The resulting ciphertext is then fed as input to a hashing algorithm, producing an *identification code* id_{key}. Finally, the identification code is stored in the user record.

Fingerprint/secret key linking: A region of elements is taken from the output pattern and binarized. L elements from the region are then used to encode the N bits of the secret key. The elements are chosen so that the probability of an error in every bit of the key is minimal. The position of each element is entered in a look-up table, which is then stored in the user record.

The process of linking the output pattern to a secret key is shown in fig. 3.1. The enrollment stages are depicted in fig. 3.2.

Verification and Key Retrieval

The verification process is similar to the enrollment and goes as follows:

1. fingerprint collection and transformation
2. secret key retrieval
3. secret key verification

1. A phase-phase product is similar to one-time pad. Product of two phase terms is equivalent to addition (mod 2π).

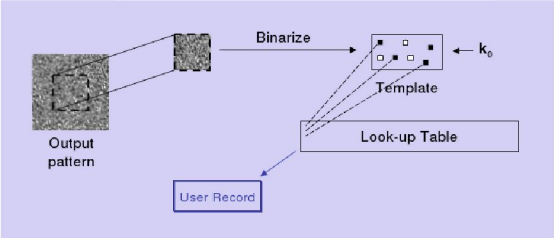


Figure 3.1: The process of linking the output pattern to a secret key in a Biometric Key Binding system (taken from [20]).

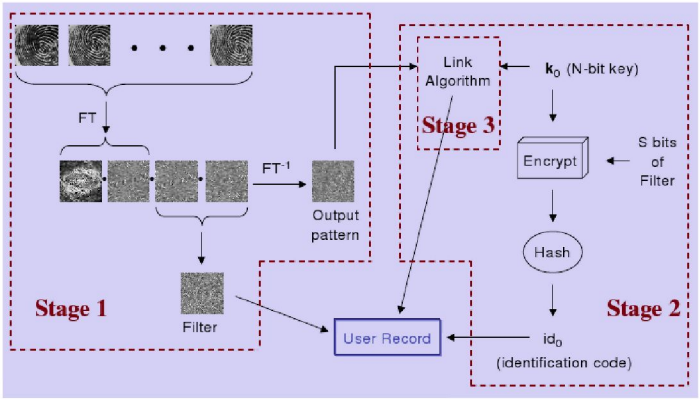


Figure 3.2: Enrollment in a Biometric Key Binding system (taken from [20]).

Fingerprint collection: As in the stage 1 of the enrollment process, a series of fingerprints is obtained. Fourier Transform is then applied, producing signal magnitude and signal phase. Filter is then retrieved from the user record. Next, the product of the filter, signal magnitude and signal phase are computed and inverse Fourier Transform is then applied to the result, creating an output pattern.

Secret key retrieval: Similarly to stage 3 of the enrollment process, a region is selected in the output pattern and binarized. The look-up table is retrieved from the user record, and elements of the binarized region of output pattern are selected according to the look-up table. Finally, the elements are translated[2] into bits, which constitute the secret key. The process of retrieving a secret key from the acquired output pattern and the user record is shown in fig. 3.3.

2. Error-correcting codes are used.

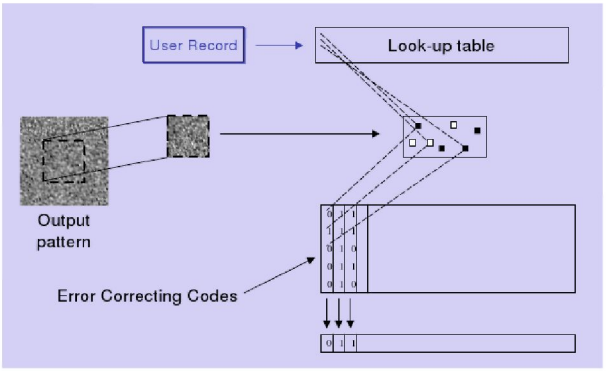


Figure 3.3: The process of retrieving a secret key in a Biometric Key Binding system (taken from [20]).

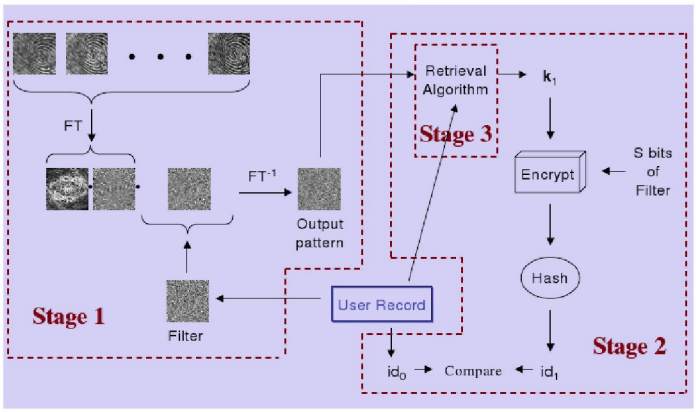


Figure 3.4: Verification and key retrieval in a Biometric Key Binding system (taken from [20]).

Secret key verification: The reconstructed key is input to the same encryption algorithm as in stage 2 of the enrollment process, and the same S bits of filter are taken as the encryption key. The resulting ciphertext is then fed to the same hash function as in stage 2 of enrollment process, producing $id_{retrieved}$. The identification code id_{key} is then retrieved and compared to the output of the hash function. If the output and the identification code are the same (i.e., $id_{key} = id_{retrieved}$), the secret key has been successfully reconstructed. The three stages of verification and key retrieval are depicted in fig. 3.4.

3.1.3 Combining IrisCode with Cryptographic Keys

A method of securely storing and accessing a secret key was recently proposed by Hao *et al.* in [6]. The approach the authors have taken slightly resembles both the fuzzy vault scheme and the Bioscrypt scheme. Unlike in fuzzy vault and Bioscrypt, however, a two-layered protection using error-correcting codes is used, with Reed-Solomon code [24] applied at the inner layer and Hadamard code applied at the outer one. The scheme is two-factor, using a token (such as smart card) and IrisCode, with an optional extension to three-factor scheme using a password.

The secret key K is generated as a sequence of bits and its hash value is computed. Next, it is encoded using Reed-Solomon and Hadamard code to produce a bit string Θ_{ps} of the same length as the IrisCode (2048 bits). Finally, the encoded key is XOR-ed with the user's reference IrisCode Θ_{ref}, and the resulting bit string Θ_{lock} is stored on a smart card along with the hash of the secret key. When the user wants to retrieve the key, he presents his iris sample, which is in turn XOR-ed with the locked key Θ_{lock} to retrieve the key K'. If $h(K) = h(K')$, then $K = K'$ and the user has retrieved the correct key.

The overall scheme is depicted in fig. 3.5. The key locking/unlocking procedure can be summarized as follows:

Key locking:

1. Compute Θ_{ps} by encoding K.
2. Compute $h(K)$.
3. Obtain Θ_{ref}.
4. Compute $\Theta_{lock} = \Theta_{ps} \oplus \Theta_{ref}$.
5. Store Θ_{lock} and $h(K)$ on a smart card

Key unlocking:

1. Obtain Θ_{sam}.
2. Compute $K' = \Theta_{lock} \oplus \Theta_{sam}$.
3. Compute $h(K')$.
4. If $h(K) = h(K')$, the key has been successfully unlocked.

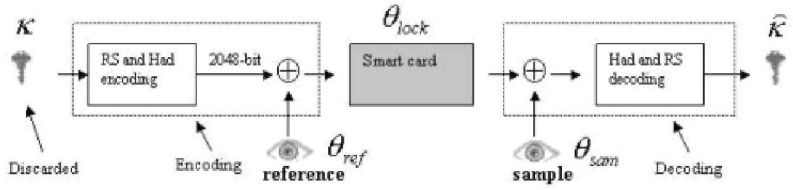


Figure 3.5: Key locking and release using IrisCode (taken from [6]).

Key Encoding

As described above, the freshly generated key is encoded using Reed-Solomon and Hadamard codes.

Hadamard codes are generated by a Hadamard matrix H, which is a square orthogonal matrix with elements 1 and -1 of size 1, 2, $4m$, where $m \in \mathbb{N}$. The authors chose the Sylvester method for generating Hadamard matrices, which recursively defines normalized matrices of size $m = 2^n$, $n \in \mathbb{N}$. The length of the codeword is m and the number of codewords is $2m$ (the remaining m codewords are rows from $-H$). Minimum distance is 2^{k-1} and the correction threshold is $2^{k-2} - 1$. [6] Hadamard codes are $[2^n, n+1, 2^{n-1}]$ codes.

Reed-Solomon code is defined as

$$C = \{(f(x_1), f(x_2), \ldots, f(x_n), f \in F[x], \deg(f) < k\}.$$

Reed-Solomon code is a $[n, k, n - k + 1]$ linear code. [24]

The secret key is divided into blocks which are first encoded using Reed-Solomon code to prevent burst errors and then using Hadamard code to compensate for the 10-20 % errors in IrisCodes belonging to the same user. This allows for the key K, which is locked in the 2048 bit sequence, to have length of 140 bits. With the parameters set to allow for 140-bit key, False Reject Rate is at 0.47 % and False Accept Rate is at 0 %.

Security and Three-Factor Extension

The scheme is resistant to attack scenarios involving an attacker with access to only the IrisCode or the token. Stealing a sample of the legitimate user's IrisCode will not yield the key unless the token is stolen as well. In such a case, the token needs to be tamper-resistant to prevent a successful attack. On the contrary, if the token is stolen while the IrisCode remains unknown and the token is not tamper-resistant, the attacker can obtain the locked

key Θ_{lock}. Experiments conducted by authors show that Θ_{lock} has 249 degrees of freedom and there is little systematic correlation between irises. The authors have estimated the lower bound for an attacker with perfect knowledge of the correlated bits to be 2^{44} computations. Knowledge of the correlated bits is, however, impossible without detailed knowledge of the iris sample itself. [6]

Security can be further enhanced when passwords are introduced into the scheme. A password can be used to encrypt the protected key or define permutations on the Hadamard matrix, leading up to 2^{592} of different matrices that can be obtained by permutation. [6]

3.2 Key Derivation Techniques

Key derivation techniques are schemes that allow for the same key to be generated every time a legitimate user presents his biometric to the system. Unlike key retrieval techniques, no secure template nor protected key is stored.

Contents of this section:

- Private Key Generation from On-line Handwritten Signatures
- Key Generation Using FingerCodes and Support Vector Machines
- Biometric Fuzzy Extractors and FingerCodes

3.2.1 Private Key Generation from On-line Handwritten Signatures

One of the first plausible methods of generating a cryptographic key from biometrics, called BioPKI, was proposed by Feng and Wah in [4]. The authors use an on-line handwritten signature as the biometric from which the key is derived. One of the key advantages of this approach is the use of a behavioral biometric, which can be easily changed – this is important, as the proposed method uses a database of biometric templates to authenticate the user before generation of the key takes place; if the signature was ever to be stolen from the database, it can be easily revoked. [3]

3. Because the key is directly derived from the stored template, revoking the template also means revoking the key. This may be considered a slight inconvenience but from the user's perspective, no further steps need to be taken to create a new key other than enrolling a new sample.

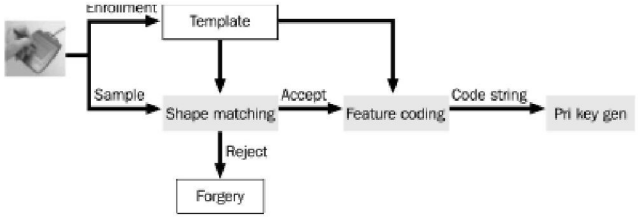


Figure 3.6: BioPKI enrollment/verification and key generation scheme (taken from [4]).

To establish a template within the system, a set of ten signatures is obtained from the user, from which a template is derived and a private/public DSA keypair generated. The private key is then discarded, while the public key is stored. Once a template has been established in the system, the user can regenerate his key by providing an on-line signature in three steps:

1. Shape matching – verification of the presented sample.
2. Feature coding – construction of a binary string from the template
3. Private key generation – derivation of the private key from the binary string constructed in step 2

The enrollment/verification phase is depicted in fig. 3.6.

Shape Matching

The goal of this stage is to compare the static features of the obtained sample with the stored template. Since the obtained sample and stored template differ, a technique called dynamic time warping is used during the matching process. Data on x,y coordinates (i.e., waveforms) of the obtained sample are shifted and correlation coefficients are computed. High correlation coefficients result in the sample being accepted. Experiments on a database with 25 registered users show a FAR of 62.8 % and FRR of 3.4 %.

Feature Coding

The goal of this stage is to derive a bitstring from the dynamic features of the signature. To deal with the variability in biometric samples, the space of feature values is first preprocessed for every feature and every user as follows:

1. Set $s_{U,x}$ of values of feature x for user U is calculated from the ten samples obtained during enrollment. The interval of values for a particular feature of all users forms the database interval d_x. The range of supported values (by the system) for feature x is denoted t_x.

2. *User boundary* for feature x and user U is calculated as $boundary_{U,x} = (\bar{T} - b \times \mathrm{std}_T, \bar{T} + b \times \mathrm{std}_T)$. b is a system parameter that defines the level of tolerance of variability in features.

3. The range of supported values t_x is segmented for user U as follows: the database interval d_x is segmented into intervals of length of the user boundary $boundary_{U,x}$ to the left and to the right of placement of the user boundary. The superfluous segments at either ends are extended into the remaining range of t_x. The segments are then labeled from left to right with integers starting from 0.

An example of determining a value for a sample feature is given in fig. 3.7. The computed user boundaries are stored with the user's template and include no further information related to segments or feature codes. The bit length of feature x is $\lambda = \log 2\mu$, where μ is the number of whole segments within the database interval d_x. The total bit length is taken as a sum of bit lengths of all the features. According to the authors, the average bit length for a group of 25 users is 40 bits.

Private Key Generation

The user's concatenated feature codes are used to derive the private key by simply computing the hash of the concatenated codes. The authors use SHA-1, which results in a 160-bit key. To verify the correctness of the private key, a signature is generated using the private key and verified using the public key that is stored with the template.

Results

Although this method is sound in its construction, it is problematic in practice. Experimental results show EER at 8 % for b a little over 8. For $b = 5$ the system reaches its limits of usability with FAR$= 1.2$ % and FRR$= 28$ %. FAR increases steeply as b decreases beyond 5.

Another issue is the keylength. While the private and public key both have 160 bits, the effective length of the private key is only 40 bits. The authors propose two possible directions of addressing the keylength problem

– either by increasing the number of features at the cost of a higher FRR, or by adding additional information derived, e.g., from passwords.

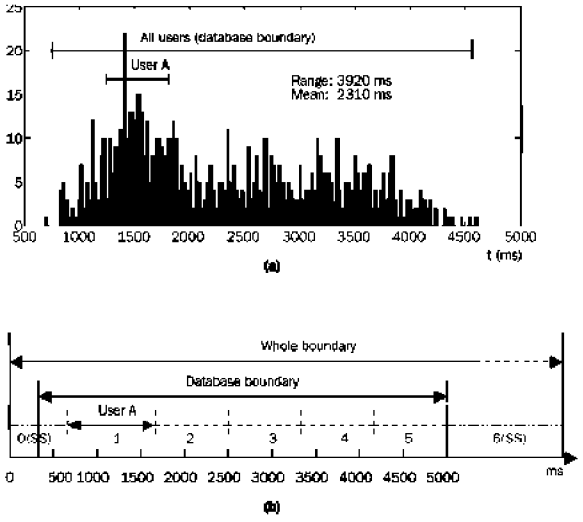


Figure 3.7: Determining value for a particular feature of a user's sample in BioPKI (taken from [4]).

3.2.2 Key Generation Using FingerCodes and Support Vector Machines

Instead of generating a cryptographic key from the characteristics of a biometric sample, Ramírez-Ruiz *et al.* proposed a method based on pattern recognition along with FingerCodes [16]. Each user is assigned a random binary number of fixed length and provides a number of fingerprint samples for the system to train on. For each bit position of the key one Support Vector Machine (SVM), shared by all the users, is trained using the fingerprints of the registered users. A trained SVM will then produce a bit for a recognized user, corresponding to the value of the user's number at the bit position it's been trained for. The length of the binary number that can be generated depends on the number of SVMs used.

When the user wants to regenerate his binary number, he presents a fingerprint sample to the system. The SVMs then classify the fingerprint and produce a sequence of bits. Correctness of the produced key can be verified by storing a hash of the number and comparing it to the one generated by SVMs. The process is depicted in fig. 3.8.

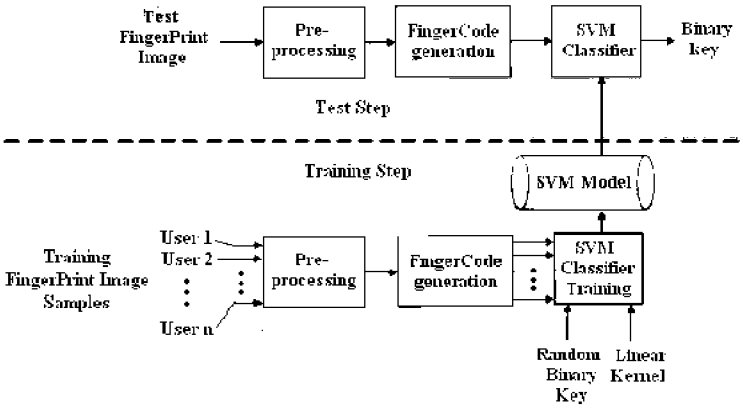


Figure 3.8: SVM training and key release (taken from [16]).

FingerCodes and Their Generation

Traditionally, most schemes involving fingerprint recognition used minutiae for identification and verification. Recently, Jain *et al.* proposed a new technique of fingerprint representation based on the texture of a fingerprint, which provides a more reliable method of fingerprint information representation, called *FingerCodes* [7]. Instead of using a set of points, a circular region around the center of a fingerprint is selected and partitioned into sectors concentric to the center of the fingerprint. The fingerprint is then normalized to eliminate the effects of pressure. Next, a set of Gabor filters with eight different directions is applied to the selected circular region. Finally, the FingerCode is constructed as a vector with 640 components taken as the average of absolute value of the standard deviation on each filtered sector, quantized to integers in the range $0, 1, \ldots, 255$. [16] The whole process is visualized in fig. 3.9.

Support Vector Machines

Support Vector Machines are supervised learning methods for classification and regression, belonging to a family of generalized linear classifiers. [25] It is a technique for separating data into groups in such a way, that the difference between the two groups is maximized. Details on SVMs can be found on Wikipedia. For the purposes of this scheme, SVMs are used to classify FingerCodes into one of two categories, corresponding to the binary 0 and 1.

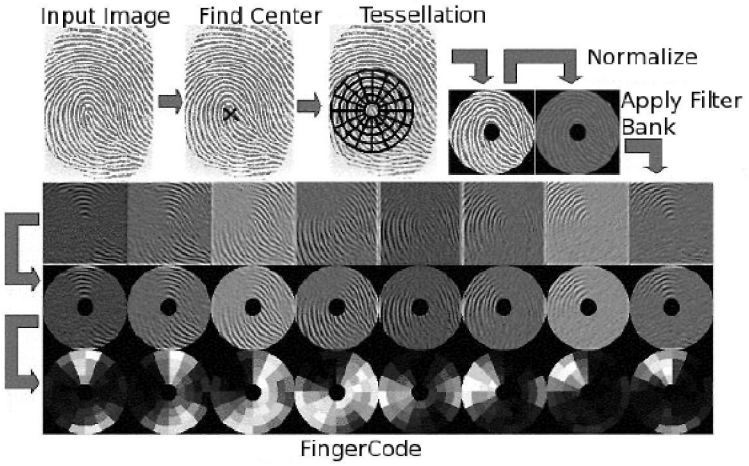


Figure 3.9: FingerCode extraction process (taken from [16]).

Methods

All the protocols assume a system with n users, each one with m fingerprints.

Universal SVM: A single SVM per bit position is trained independently on the others so that it links FingerCodes of the same user to his assigned binary key. Users are assigned ranges of FingerCodes by the SVMs in the training stage to compensate for the variability in FingerCodes.

Possible weaknesses include attacks on the SVMs, which can be remedied by encrypting the SVMs, and fingerprint sample theft. [16] Because fingerprint samples are easy to obtain, the authors do not recommend this scheme beyond the use for public keys. Experiments conducted by the authors have shown this method to have a relatively low efficiency.

Separate SVM: Every user received his own SVM for every bit position of his key. The SVM is trained to link FingerCodes of the user it belongs to to his assigned bit and the rest of the users' FingerCodes to 0.

According to the authors, the method is susceptible to attacks. A disadvantage could be the increased number of SVMs in the system, although experiments conducted by the authors have shown this method performs very well in terms of efficiency.

Separate SVM with random perturbation: As in the case of separate SVMs, one SVM per key bit position is assigned to every user, with the SVM having the same properties. In addition to that, the values linked to the other users by a particular SVM are randomly set to 1. According to the authors, when the number of other users linked to 1 is nearing 0.5, the model performs similarly to the universal SVM model, while linking only a small number of the other users removes the drawbacks of the separate SVM model, almost achieving the performance of the separate SVM model. Possible attacks remain the same as in the universal SVM model.

Security and Performance

To overcome the problem of non-confidentiality of biometrics, the authors proposed an improved two-factor authentication model where a four-digit password was fed as an input to a pseudorandom number generator, and the output random sequence was used to permute the elements on the FingerCodes. This method should be safe to use for private key regeneration.

Experiments were performed using data from 30 different fingers, with eight samples per finger. In case of separate SVMs and separate SVMs with random perturbation, the correct key is regenerated in a little more than 90 % of all the cases, with the universal SVM model reaching up to 80 %. Results also show that for 32 impostors which the system has not been trained for a valid key has been released in 1 % of all the cases for both universal and separate models, while the separate SVM with random perturbations was able to reach roughly 0.5 %. Finally, the performance of the universal model decreases with the number of users the system is trained for, while at the same time increases for the other two models. The probability of impostors regenerating a valid key decreases for all the models with the number of legitimate users in the system.

3.2.3 Biometric Fuzzy Extractors and FingerCodes

Tong *et al* propose a scheme for key protection somewhat similar to [10] and [9] in [22]. The scheme uses FingerCodes for fingerprint representation and the construction of *fuzzy extractors* [3] to reconstruct the secret value from which the secret key is extracted.

To enroll a secret key consisting of $d+1$ numerals, a polynomial $p \in \mathbb{Z}[X]$ of degree d is chosen so that the $d+1$ integer coefficients of p correspond to the $d+1$ numerals of the secret key. Then a random set of n points on p is chosen, $n > d$, and the set is hidden using n subparts of the FingerCode. If

the user wants to retrieve the key, his newly obtained FingerCode is used to retrieve the set of points, from which the polynomial p is reconstructed using Lagrange interpolation.

Enrollment

First, a secret polynomial and a set of points are chosen as described above. The points are represented on l bits, which are encoded using Reed-Solomon code into codewords $c_0, c_1, \ldots, c_{n-1} \in 0, 1^{l+e}; l, e \in \mathbb{Z}$. Next, the user's FingerCode F is segmented into n parts – $F = f_0||f_1||\ldots||f_{n-1}$. Then the set of points $\{\delta_i = c_i \oplus f_i; 0 \leq i \leq n-1\}$ is computed. Finally, a *FingerKey* is created as a set of pairs $(\delta_i, H(c_i))$ using a suitable one-way function, for $0 \leq i \leq n-1$. The FingerKey is then made public.

Key Release

To retrieve the secret key, the FingerCode F' of the user is obtained and segmented, $F' = f_0||f_1||\ldots||f_{n-1}$. Then for every δ_i in the user's FingerKey $\delta_i \oplus f_i'$ is computed and transformed to the nearest codeword c_i'. If $H(c_i') = H(c_i)$ for at least $d+1$ points, the user can reconstruct the secret polynomial using Lagrange interpolation.

Results and Security

Experiments were conducted on a database of 1000 samples. The authors do not quantize the components of FingerCodes into 256 values, but rather round the components, which are in range $(0, 7)$ into values 0 (range 0-2), 1 (range 2-4) and 2.(range 4-7). This gives a keyspace of 3^{640}, and with 16 segments of FingerCode a FAR more than $60\ \%$. Using standard FingerCode matching as used in [7], the results showed FRR of $78\ \%$ and FAR of $0.1\ \%$. As the authors state, this is not on par with standard values of FingerCode matching, so the authors expect better results with further improvements. Security of the scheme is given in [22].

Chapter 4

Generating Random Numbers from Fingerprints

As stated in the first chapter, biometrics are inherently variable – it is extremely rare, if not impossible, to obtain two identical samples from two separate biometric measurements for the same user. While the observed features, such as the image of an iris, a fingerprint, or hand geometry, rarely change between two measurements, the measurement process itself is prone to errors. These can originate from the surrounding environment, the measurement device (e.g., a camera) or the user himself. The user may, for example, apply different fingerprint or hand pressure, or blink when capturing an image of his iris; the electronic nature of measurement devices may introduce noise into the captured biometric sample; finally, different environmental conditions such as temperature or light conditions may affect the measurement process.

This variability in captured samples could be potentially used to obtain truly random bit sequences, which could be used as a seed for PRNG, or as a random number by itself. This is the main idea behind a method recently proposed by Dr. Matyáš, Dr. Říha and Dr. Smolík. The proposed method uses fingerprints as the measured biometric and aims to provide a readily available means of generating random numbers on mobile devices equipped with fingerprint readers. The generation process goes as follows: first, a set of fingerprints is obtained from the user [1]. Next, the XOR operation is applied to all the fingerprints, resulting in a binary sequence of the same length as the fingerprints. Finally, the resulting binary sequence is input to a suitable hash function to produce the final (pseudo)random number. The whole process is summarized in the following steps:

1. Obtain a set of fingerprints

$$\mathcal{M} = \{fingerprint_1, fingerprint_2, \ldots, fingerprint_n\},$$

where n is a system parameter.

1. The number of fingerprints to obtain is a system parameter.

2. Compute $xor\_result = fingerprint_1 \oplus fingerprint_2 \oplus \ldots \oplus fingerprint_n$.
3. Compute $rn = hash(xor\_result)$, where $hash()$ is a suitable hash function.
4. Optional: Seed a PRNG with rn to generate pseudorandom bit sequences (numbers) of desired bit length.

The hash function works as a "randomness extractor" in this context and can be replaced by a more suitable, true randomness extractor.

Recently, we have discovered that an approach of generating random numbers from biometrics has been proposed by Szczepanski *et al.* in [21]. The main idea behind their method is the observance that the rightmost digits of values obtained from measurements of physical phenomena exhibit properties of random numbers. The random number generation method as proposed by the authors proceeds as follows: First, values $(x_0, x_1, \ldots, x_{n-1})$ from measurements of the phenomena are obtained. Next, the average value m and standard deviation s of the measurements are obtained. Values that lie outside the interval delimited by the average and the standard deviation multiplied by a system parameter r are discarded, i.e., values outside the range of $(m - rs, m + rs)$. Finally, the interval is divided into L equal subintervals, numbered $0, 1, \ldots, L-1$, which are associated with the values x_i they contain. Finally, the values are assigned a bit determined as $l_i \bmod 2$, where l_i is the number of the subinterval containing the value x_i.

The authors have tested their approach on neurophysical brain signals, measured during visual stimulation, and galvanic skin response, measured during avatar encounters in virtual reality. The generated sequences were tested using FIPS 140-2 statistical test suite as well as the Maurer's test, and the produced bit sequences generally performed well in both tests.

As can be seen from its construction, the method produces one random bit for every value x_i obtained by measuring the phenomena. Thus, the method is suitable for biometrics where the direct result of a measurement is an integer or real value, and the sampling rate is sufficient – in other words, biological characteristics that change in time. The approach proposed by Dr. Matyáš *et al.*, however, depends on a value (a fingerprint image) which generally has a low sampling rate, and is already in a binary form. This makes it suitable for biometrics where the obtained sample is an image, such as iris or fingerprint. The method proposed by Szczepanski *et al.* is generally unsuitable for such scenarios.

Chapter 5

Entropy of Shift and Rotation of Fingerprints

5.1 Motivation

Initial estimation of the entropy in a fingerprint for the proposed method of generating pseudorandom numbers was done by Dr. Říha. The overall entropy of a fingerprint was calculated from the fingerprint's image histogram by taking the level of gray in pixels as values in the system, computing the frequencies and entropy from them. The initial assumption, however, was that the pixels were "independent" – knowledge of a group of pixels would not necessarily yield knowledge of the neighboring pixels. As can be seen in [5], there is a strong correlation between pixels in a fingerprint, so the estimated entropy can only be taken as a highly optimistic upper bound.

We discussed how to approach this problem and obtain at least a rough estimate of the lower bound on entropy contained within fingerprints. An idealistic model of a fingerprint system would be such that the fingerprint reader would produce absolutely no noise and the user would always present his finger in such way that the reader would always extract exactly the same fingerprint. In such a case, should the attacker gain access to the fingerprint, he or she would know every future input to the system, which would render the proposed method insecure. Such a system is, however, unlikely to ever appear. Even if the reader would produce no noise, it is virtually impossible for the user to present his finger to the reader always on exactly the same position in the same angle, with the same pressure. Furthermore, the system may reject fingerprints which contain no difference from the reference one, guaranteeing some level of variability in the system.

A practical fingerprint system will have a fingerprint reader that produces noise, and a user that produces different fingerprints of the same finger every time he presents his finger to the reader. In such a system, the entropy of a fingerprint comes from:

- fingerprint reader and environment noise (influenced by several factors),

- different position of a fingerprint inside the captured image,
- different structure of the fingerprint itself.

By "fingerprint reader noise" we understand random perturbations to the captured image containing the presented fingerprint. While some of the noise is inherent to the reader because of its underlying electronic architecture, environmental conditions can contribute to the reader producing different amounts of noise too, such as different levels of moisture, extreme temperatures or light conditions, to name a few. Since the user is unlikely to place his finger on the fingerprint reader the same way twice, the fingerprint will have different position inside the captured image. Finally, the varying pressure of the user's fingerprint, as well as the different portions of the surface of the finger help to ensure that no two fingerprints of the same finger are ever completely the same.

In an effort to estimate the minimum entropy of the fingerprint system, we decided to measure the entropy of its individual contributing components. In the first stage of entropy estimation, we decided to estimate the minimum and average entropy contributed by different position of the fingerprint inside the captured image.

To establish the entropy in a reasonable way, a suitable metric for describing the difference between two fingerprints of the same finger was needed. Ideally, the metric would be a set of numbers, or tuples of numbers, that would describe the shift and rotation of fingerprints. Consultations with Dr. Kozubek revealed that *Optical Flow Field* [1] vectors may be such a metric. Dr. Ulman proposed yet another approach using image alignment. Both these approaches were used to calculate the Shannon and min-entropy of the shift and rotation of fingerprints.

The remainder of this chapter will introduce the Shannon and min-entropy, the image processing methods and the accompanying methods of calculating entropy from the results of their output.

5.2 Information Entropy

The concept of *information entropy* was first introduced by Claude E. Shannon in 1948 in his paper "A Mathematical Theory of Communication" [18]. Formally, information entropy is defined as

$$H(X) = -\sum_{i=1}^{n} p_i \log p_i,$$

1. These will be refered to as "flow vectors" from now on.

where X denotes a random variable. [14] Informally, information entropy gives the amount of uncertainty in a system.

Min-entropy is a special case of a parametrized entropy measure called *Rényi entropy* [17]. Formally, it is defined as

$$H_\infty(X) = -\log\left(\max_{i \in I} p_i\right),$$

where

$$\max_{i \in I} p_i$$

denotes the highest probability in the system. [23] Informally, the min-entropy gives the minimum entropy contained within a system.

Both entropies do not have a logarithm function with fixed base. A base of 2 is used exclusively, however, because of its suitability to represent the binary nature of computer data.

5.3 Image Processing Methods

Due to the construction of the proposed key generation method, it is virtually impossible to calculate the precise amount of guaranteed entropy in the system. It is possible, though, to make an estimate based on a suitable metric, despite of the estimate being inaccurate. Intuitively, for two images containing a specific object (in our case the fingerprint inside the captured image) to be aligned on each other we would need a vector or a set of vectors that would describe the direction and length in which the object needs to be shifted and/or rotated inside that image. While a single, two-dimensional vector can only describe the shift, two vectors are sufficient to describe shift and rotation. This was discussed with Dr. Kozubek and Dr. Ulman, and they proposed two solutions:

- image alignment,
- Optical Flow Field.

Dr. Ulman kindly provided the programs for both methods. The remainder of this section will describe the programs and their underlying methods.

5.3.1 Image Alignment

The principal idea behind the image alignment method is to take an input image and try to align it to its reference image. The result is a vector describing the shift in position of the input image against the reference image

such that the shift yields the best match between the reference image and the input image. The implementation by Dr. Ulman takes two files containing the fingerprint images as input and outputs a single two-dimensional integer vector representing the shift that yields the best match. The process is depicted in fig. 5.1.

As is described above, a single two-dimensional vector cannot describe image rotation, so if the input fingerprint is also rotated in its position compared to the reference fingerprint, the program will not give correct results. In practice, this was not an issue, since the provided fingerprints were rotated only by small angles.

Figure 5.1: The first method takes two images and tries to align one to the other.

5.3.2 Optical Flow Field

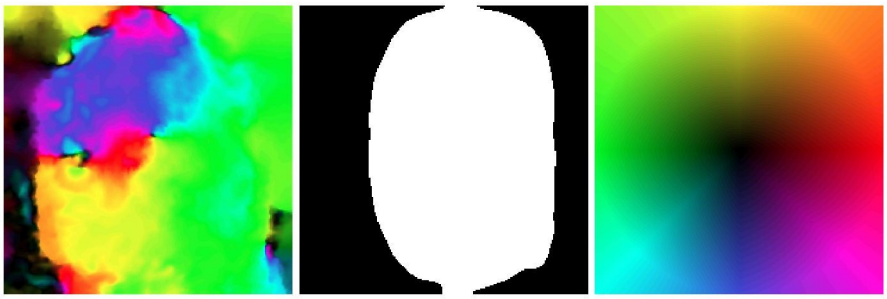
Optical flow field is a field of vectors that describe the change of position of a point inside an image. The method was originally developed to describe movement of objects in consequent images. When describing movement inside an image (i.e., change of position between two images), a vector is computed for every point, indicating the change of position of the point between the first and the second image. This is depicted in fig. 5.2a.

Since the different positions of a fingerprint in two images can be viewed as movement, the method is usable for description of change of position of two fingerprints. There are some drawbacks, however – the unique structure of fingerprints may cause many vectors to be incorrect. This problem only increases with the resolution of the image containing the fingerprint.

To overcome this problem, we decided reduce image resolution to decrease the number of vectors that are output. By making the images smaller,

very fine details that could "confuse" the method are eliminated, but the overall shape and structure of the fingerprint remains. This produces more consistent vectors that are in line with the actual shift and rotation of the fingerprint. Furthermore, to eliminate unnecessary vectors that describe regions outside the fingerprint (i.e. the white area round the fingerprint inside the image), a mask for each fingerprint is computed, and the vectors are computed only for the region delimited by the mask. A sample mask is shown in fig. 5.2b.

The original implementation by Dr. Hubený produces only a graphical output. Dr. Ulman adapted the program to output a set of real two-dimensional vectors instead, which are suitable for the purpose of entropy calculation. The implementation takes the two fingerprints, a mask corresponding to one of the fingerprints and some parameters as input. The number of vectors output for a pair differs based on the region delimited by the mask. This proved to be a slight problem during entropy calculation compared to the first method, as I will show in the next section.



(a) Flow vectors (b) Example mask (c) Flow vectors legend

Figure 5.2: A graphical representation of the vectors (a), an example mask computed for a fingerprint (b), and a legend to the graphical representation of the vectors (c). The color describes the direction of the movement between two fingerprints while the color intensity shows the displacement size. The white region inside the mask delimits the area of the fingerprint.

5.4 Entropy Calculation and Results

Entropy by itself is defined over random variables and their probabilities. For this reason, it is impossible to strictly adhere to its definition, and probability is instead replaced by relative frequency – number of occurrences of an event divided by the total number of events. The results therefore can-

not be absolutely accurate, but for a large dataset the relative frequency is similar to probability.

Calculating the entropy of different positions of fingerprints will thus require to obtain the relative frequency for each fingerprint position in the dataset. The available dataset had 219 fingerprints[2], which yields a total of 23871 possible pairs of fingerprints. We believe this is a large enough dataset for the relative frequency to reasonably correspond to probability.

5.4.1 Shift Entropy Calculation Using Image Alignment

To calculate the average and minimum shift entropy, a vector is computed for every possible pair of fingerprints from the dataset. The vectors and their frequencies are recorded, and the relative frequencies are computed subsequently. Finally, the relative frequencies are used as probability to calculate the Shannon entropy. Throughout the process, the vector with minimum frequency is chosen from the dataset and the min-entropy is calculated from its relative frequency. The algorithm can be summarized as follows:

1. Create an empty associative field. The keys will be the vectors, while the values will be the frequency of the vectors.
2. Obtain a vector for every possible pair of fingerprints.
3. If the vector is present in the associative field, increase its frequency by one, else insert the vector in the associative field and set its frequency to one.
4. Compute $p_i = \frac{freq_i}{pairnum}$ for every vector p_i, where $freq_i$ denotes the frequency of the i-th vector and $pairnum$ denotes the number of pairs in the dataset.
5. Compute $entropy = \sum_{i \in I} -p_i \log p_i$.
6. Choose the vector with highest frequency $freq_{max}$, then compute $min\_entropy = -\log p_{max}$, where $p_{max} = \frac{freq_{max}}{pairnum}$.
7. Output $entropy, min\_entropy$.

2. These belong to the same finger of the same user.

5.4.2 Shift and Rotation Entropy Calculation Using Optical Flow Field

Using flow vectors it is possible to obtain an estimate of fingerprint entropy of shift and rotation. Every flow vector coresponds to a position inside an image, so to calculate the overall entropy it is necessary to calculate the entropy of all positions inside the image and combine them together. The major obstacle, however, is that the number of vectors differs depending on the mask used. This can be solved by either taking a lower bound on the number of vectors and discarding the vectors that exceed it, or by fixing one fingerprint from the pair along with its mask.

The first solution requires an estimate of the guaranteed number of vectors for every pair of fingerprints. Then, for every pair of fingerprints, any additional vectors would be discarded. Entropies for positions corresponding to the undiscarded vectors would then be calculated and the average value from the entropies taken as the entropy of fingerprint shift and rotation. The problem with this solution, however, is that by discarding vectors loss of information occurs, as the relative frequencies of the discarded vectors are never taken into account when calculating the entropy.

The second approach overcomes this issue by calculating entropy from data obtained from all pairs of fingerprints where one fingerprint is fixed, then taking the average of those values as results. Flow vectors would be computed for pairs consisting of the fixed fingerprint and its mask and all the remaining fingerprints. Next, Shannon entropy and min-entropy would be computed from the vectors and then stored. This process would be repeated for all fingerprints, and the final entropy would be calculated as the average of all the entropies for fixed-fingerprint pairs. This was our preferred approach, as no information loss takes place. [3]

When calculating the entropy for a fixed fingerprint, entropy of each position is calculated. Experiments showed that adding these entropies together results in an unrealistic amount of entropy (in the order of thousands or tens of thousands of bits). Therefore, the average from all entropies of individual positions is calculated instead. The average of these Shannon entropies and min-entropies are then computed as the overall Shannon entropy and min-entropy of shift and rotation of fingerprints. The algorithm can be summarized as follows:

1. Create an empty list vec_list. The list items will be associative fields that will correspond to positions inside an image. The keys will be the vectors on those positions, while the values will be the frequencies of

3. Compared to the previous method.

the vectors.

2. Fix a fingerprint and its corresponding mask.

3. Compute the flow vectors for each pair of fingerprints with the fixed fingerprint.

4. Let v_i denote the i-th vector flow for a pair of fingerprints, $i \in I$, where $|I|$ denotes the number of flow vectors output for a given mask. If $vec\_list[i]$ contains v_i, increase the vector's frequency by one, else insert the vector to $vec\_list[i]$ and set its frequency to one.

5. For $vec\_list[i], i \in I$ compute the min-entropy and Shannon entropy for i-th position.

6. Compute the Shannon entropy and min-entropy of the fixed fingerprint as the average of all position entropies.

7. Empty the list vec_list and start from step 2 with the next fingerprint. Remember the calculated entropy for the fixed fingerprint.

8. Compute the overall Shannon entropy and min-entropy of the shift and rotation as the average of entropies of fixed fingerprints.

9. Output the computed Shannon entropy and min-entropy

Python source code of the above two algorithms (with commentaries) can be found in the accompanying CD.

5.4.3 Results

Shannon entropy and min-entropy of shift of fingerprints were calculated using the image alignment method. Using data obtained from the dataset of 219 fingerprints, Shannon entropy of fingerprint shift is 10.88 bits, with min-entropy of 6.17 bits. This shows a relatively high amount of entropy, as the highest possible Shannon entropy for a dataset of this size is 14.54 bits.

Surprisingly, the entropy of fingerprint shift and rotation using optical flow field did not show that much entropy. Shannon entropy is 5.30 bits, while min-entropy is 4.03 bits. Such a relatively low value is probably the result of rounding of vectors to integers, as well as the possibly repeating vectors on different positions for a pair of fingerprints.

We shall conduct further research in the area with the aim of refining the estimate of entropy contained within a fingerprint.

Chapter 6

Conclusion

Biometrics, as a widespread authentication technology, are now enjoying substantial research attention that promises to take the technology beyond the traditional authentication mechanisms. Many interesting schemes, targeting the problem of securely regenerating cryptographic keys, and thus eliminating the problem of insecure storage and access, have been proposed recently. Two such schemes are ready for enterprise deployment – Biometric Key Binding [20], based on fingerprints, has been already turned into a commercial product, while the IrisCode approach proposed by Hao *et al.* in [6], with its excellent FAR and FRR values and low computational complexity, is a promising technology that could replace passwords and PIN for cryptographic key locking on smart cards in the future.

As can be seen in [4], behavioral characteristics such as handwritten signatures can be successfully utilized for key regeneration as well, although there are issues with key length, as the key is effectively reduced to 40 bits. Fuzzy commitment scheme is a promising cryptographic construction with broad applications to biometrics; the main challenge in its use, however, is the selection of appropriate error-correcting code. A new fingerprint representation, the so-called FingerCodes, are enjoying a lot of applications in recent biometric reseach. Of the two schemes described here that use it, the scheme utilizing the support vector machines [16] has acceptable FAR and FRR values of less than 10 % and 1 %, respectively. Interestingly, FRR decreases with the number of users in the system for two of the three computational models proposed in [16].

Although not an explored area, it is also possible to use biometrics for random number generation. One such straightforward, low-cost method, based on fingerprints, has been introduced by Matyáš *et al.* Yet the major obstacle on the way to its practical deployment remains in estimating the amount of entropy the produced numbers contain. An effort has been made to estimate the entropy of fingerprint shift and rotation, which is seen as a contributing factor to the overall entropy in the system. It should be noted, however, that the measured entropy is only an estimate and its relation to

the overall entropy remains unknown.

Future work will concentrate on refining the estimate, determining the entropy of other contributing factors, and establishing the relation between them. We plan to explore the details of the image processing algorithms, estimate the amount of entropy from fingerprint reader noise, and measure the entropy of fingerprint shift using vectors describing the shift between reference points inside the fingerprints.

Appendices

Appendix A

Additional Schemes

This chapter presents two schemes that have been left out from the main text – a framework for cryptographic key generation, regeneration, and signatures; and a method for protecting fingerprints during their transfer from the fingerprint reader to the verification database over an untrusted network. The latter can be considered a *biometric cryptosystem*, where the key material, used for encryption of the biometric sample, is constructed from the biometric sample itself. Biometric cryptosystems do not use traditional cryptography; encryption and decryption algorithms are designed specifically for the type of biometric samples they are intended to protect. Another example of such scheme, based on fractional Fourier transforms, can be seen in [11].

A.1 A Framework for Cryptographic Key and Signature Generation

Je-Gyeong et al. proposed a framework for digital signature generation and encryption using fingerprints in [8]. The framework offers key generation and recovery functions as well as straightforward signature generation and encryption/decryption. The authors claim the proposed method of generating private keys is suitable for use with most encryption/signature cryptosystems. [1]

A.1.1 Overall Design

The system consists of the following components:

- capturing device
- matching function
- key extraction function

1. However, as will be discussed later, no convincing proof is given to support the claim.

- Biometric Certificate Authority (BCA)
- secure storage for biometric template and the encryption/signing key

The capturing device represents a fingerprint reader with normalization and noise reduction functions. The output (=sample) of the capturing device is a set of location and angle values of minutiae set. The matching function matches the obtained sample against the template stored in the BCA's database.

A *Biometric Certificate* is simply a confirmation of user identity, linked to the user's template and his public key. *Biometric Certificate Authority* holds the biometric certificates, as well as templates of registered users and public keys in a database. If the user is legitimate, the private key is extracted in the key extraction function from its protected form, which is stored in a secure storage, using a personal secret value and the user's template.

A.1.2 Key Generation

The user may opt to generate a key if he has not done so already. First, a sample of his fingerprint is obtained and matched against his template, stored in the Biometric Certificate Authority's database. If there is a match, the sample is combined with a secret value provided by the user, and hashed. The resulting value is stored as the secret key in a protected form on a secure storage – the authors propose using the construct of fuzzy vault to protect the key as follows:

1. A set of fake minutiae points is generated. The points are then inserted into a copy of the user's template.
2. Polynomials for real minutiae set and fake minutiae set (one for each) are constructed. The key is then projected to both polynomials.
3. The modified copy of user's template, obtained from step 1, as well as the results of key projection into the polynomials are stored on a secure storage.

The authors use ElGamal signature scheme for signing keys in their work, which means the public key is computed as follows:

1. Select a large prime p such that for the secret key s holds $0 < s < p-1$.
2. Select a generator g of $\mathbb{Z}_p$.

3. Compute $y \equiv g^a \mod p$. The public key is the triple (p, g, y).

After the public key has been computed, it is stored in the BCA's database. The proposed method of generating cryptographic keys is not, however limited to ElGamal and can be used with a wide range of cryptosystems. The key generation framework is depicted in fig. A.1.

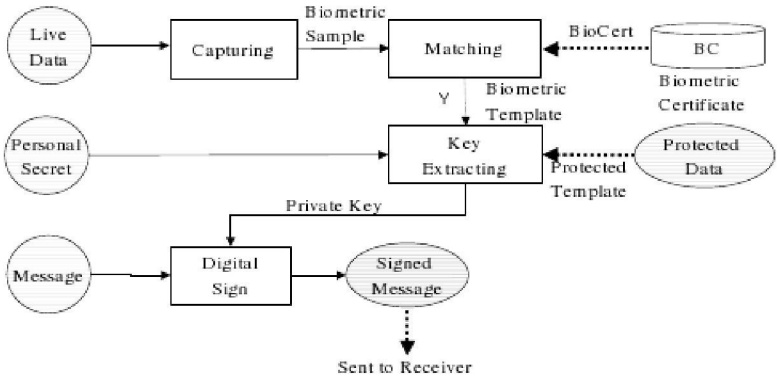


Figure A.1: Scheme of the proposed model for key generation (taken from [8]).

A.1.3 Signature Generation and Verification

Similarly to the key generation stage, a user's fingerprint sample is obtained and matched against his template in the BCA's database. If there is a match, the user extracts the signing key using the secret value and his fingerprint template/sample. He can then sign the message.

The receiver can verify the signature using the public key and the biometric certificate of the sender, obtained from the Biometric Certificate Authority.

A.1.4 Message Encryption

The process of encryption is almost identical to that of signature generation and verification. The sender who wishes to encrypt a message for a particular recipient first retrieves the recipient's public key from the BCA's database, then encrypts the message and sends it to the recipient. The recipient then retrieves his private key from the secure storage by first authenticating to the system, then using his secret value and fingerprint tem-

plate/sample to retrieve the key from its protected form. After that, the user may decrypt the message.

The proposed framework for message encryption and signature generation is shown in fig. A.2.

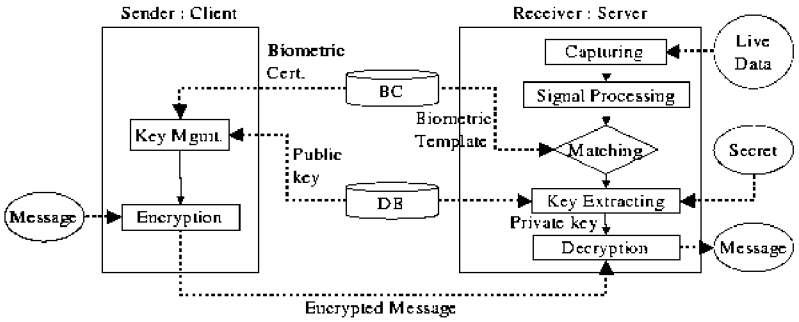


Figure A.2: Proposed framework for signature generation/verification and message encryption (taken from [8]).

A.1.5 Soundness and Security

Basically, this is a very convenient construction that fits well into existing infrastructure as it allows for conventional encryption and signature algorithms to be used. A Biometric Certificate Authority can be established from the existing Certificate Authority by simply storing the template along the key(s) and using biometric certificates instead of the conventional ones.

However, the authors give no details on the variability of angle and location values of minutiae points, which constitute the biometric template from which the key is partially generated/retrieved. The limitations of template and the personal secret value may also cause the generated key to contain less bit information, as is the case with [4]. Another, even more pressing, issue occurs during signature generation (key release). If the impostor can obtain a valid sample of the user's fingerprint, the security of the protected key storage reduces to the personal secret value, which is easy to obtain using exhaustive search. In this case it may be better to include a liveness test in the capturing device or monitor the exact value of presented sample and not accept a sample with the same values of minutiae points within a given time period.

A.2 Encryption Using Fingerprint Indexing

Biometric authentication schemes can be decentralized, involving transfer of a sample over an insecure channel. To prevent eavesdropping and sample tampering, samples need to be protected by encryption. The samples can be viewed as blocks of bits that can be encrypted using symmetric cryptography. Another approach is to encrypt the image using novel algorithms that are not based on traditional symmetric cryptography.

A novel protocol for fingerprint encryption has been proposed in [5]. Han *et al.* present an approach that combines pixel permutation and gray level substitution using extended Baker map, deriving the encryption key directly from the image itself. One of the greatest advantages is the low computational cost and key management, as the key is only temporal and thus is not stored anywhere.

A.2.1 Extended Chaotic Baker Map

The chaotic Baker map is a two-dimensional map defined on two-dimensional vectors of real numbers in the range of 0 to 1. A discretized Baker map, from which the extended Baker map has been developed by the authors, is defined for the pixel (r, s) and an $N \times N$ image as follows:

$$B_{(n_1,n_2,\ldots,n_k)}(r,s) = (\frac{N}{n_i}(r - N_i) + s \mod \frac{N}{n_i}, \frac{n_i}{N}(s - s \mod \frac{N}{n_i}) + N_i)$$

Here $N \in \mathbb{N}$ and $n_1, n_2, \ldots, n_k \in \mathbb{N}; n_1|N, n_2|N, \ldots n_k|N, n_1+n_2+\ldots+n_k = N$ are parameters of the Baker map, $N_i = n_1+n_2+\ldots+n_{i-1}$ and (r, s) must be such that $N_i \leq r < N_i + n_i, 0 \leq s < N$.

As mentioned above, along with pixel permutation a gray level substitution is applied pixel-wise. During the process, every pixel is assigned a new gray level. The substitution needs to be invertible, i.e. if $g(x, y, l) = (x, y, l')$ is the definition of the gray level substitution function, where the third argument to the function is the current gray level of the pixel (x, y), it must hold that $g(x, y, l) \neq g(x, y, k)$ for all $l, k \in \mathbb{N}, l \neq k$.

The discretized Baker map may be viewed as an example of a substitution and permutation network, with the parameters $n_1, n_2, \ldots, n_k$ as keys. The requirement $n_1|N, n_2|N, \ldots, n_k|N$, however, greatly reduces the security of the map. To lift the restrictions on the parameters, *Han et al.* have improved the discretized Baker map to work with integers that need not necessarily divide N. This greatly increases the parameter space, making the extended Baker map suitable for use as a cryptographic algorithm.

First, the image is divided into k columns of $N \times n_i$ pixels. If $n_i|N$ for $1 \leq i \leq k$, the substitution is similar to that of the standard discretized Baker map, otherwise the pixels of each column are divided into n_i boxes of N pixels each. Substitution then follows a different formula. Details can be found in [5].

A.2.2 Key Generation and Fingerprint Encryption

With the Baker map improved for direct use as a symmetric cryptosystem, the encryption process is now straightforward.

Components of encryption/decryption system:

- extended Baker map, used as the encryption algorithm
- asymmetric cryptosystem, used for key exchange between the communicating parties
- fingerprint to be transmitted

After five rounds of encryption, the image has an almost uniform histogram, completely scrambling any information. Encryption keys for all the rounds of encryption are derived from the image in its unchanged form. First, a few pixels are picked at random and their position and gray level values processed as part of the secret key. Then the fingerprint image is preprocessed and some of its features are extracted and converted into a series of positive integers in the range 0,1,...,99, which form the rest of the secret keys [2]. The fingerprint is encrypted using the newly generated key and the computed keys are encrypted using the asymmetric cryptosystem. Next, the encrypted fingerprint is sent to the receiving side, along with the encrypted keys. The receiving side then retrieves the secret keys and decrypts the fingerprint, ready to identify/verify the requesting user.

Note on the encryption key: Due to the nature of the key generation, the key may also be suitable for traditional cryptographic algorithms, once processed. Further research into the exact methods of key generation and variability of key components will be required, however, to prove it.

2. The key for a particular round of encryption is a multidimensional vector of values, which are part of the generated series.

Appendix B

CD Contents

The accompanying CD has three directories, the contents of which are described below. The image processing programs were placed on the CD with the consent of RNDr. Vladimír Ulman, who is the author of both of them.

python_programs – This directory contains the two entropy calculation programs and instructions for their use.

document – This directory contains a hyperlinked version of the thesis in PDF and a separate bibtex file containing the referenced bibliography.

image_processing – This directory contains the two image processing programs, necessary libraries, and installation instructions.

Bibliography

[1] Nist special publication 800-90:recommendation for random number generation using deterministic random bit generators.

[2] Ondřej Bitto. Review of the state-of-the-art: the use of biometric data for generation of cryptographic keys. document draft. 2006.

[3] Y. Dodis, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *Lecture Notes in Computer Science*, 3027:523–540, 2004.

[4] H. Feng and C.C. Wah. Private key generation from on-line handwritten signatures. *Information Management & Computer Security*, pages 159–164, 2002.

[5] F. Han, J. Hu, and X. Yu. A biometric encryption approach incorporating fingerprint indexing in key generation. *Lecture Notes in Computer Science*, 4115:342–351, 2006.

[6] F. Hao, R. Anderson, and J. Daugman. Combining crypto with biometrics effectively. *IEEE Transactions on Computers*, 55(9):1081–1088, 2006.

[7] A.K. Jain, S. Prabhakar, L. Hong, and S. Pankanti. Fingercode: A filterbank for fingerprint representation and matching. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2, 1999.

[8] J.G. Jo, J.W. Seo, and H.W. Lee. Biometric digial signature key generation and cryptography communication based on fingerprint. *Lecture Notes in Computer Science*, 4613:38–49, 2007.

[9] A. Juels and M. Sudan. A fuzzy vault scheme. *Proceedings of IEEE International Symposium on Information Theory*, 2002.

[10] A. Juels and M. Wattenberg. A fuzzy commitment scheme. *Proceedings of the Sixth ACM Conference on Computer and Communications Security*, 1999.

[11] M.K. Khan and J. Zhang. An intelligent fingerprint-biometric image scrambling scheme. *Lecture Notes in Computer Science*, 4682:1141–1151, 2007.

[12] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*, chapter 5 Pseudorandom Bits and Sequences, page 171. CRC Press LLC, 1996.

[13] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*, chapter 5 Pseudorandom Bits and Sequences, pages 183–184. CRC Press LLC, 1996.

[14] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*, chapter 2 Mathematical Background, page 56. CRC Press LLC, 1996.

[15] F. Monrose, M.K.Reitel, and R.Wetzel. Password hardening based on keystroke dynamics. *Proceedings of the Sixth ACM Conference on Computer and Communications Security*, pages 73–82, 1999.

[16] J.A. Ramírez-Ruis, C.F. Pfeiffer, and J.A. Nolazco-Flores. Cryptographic keys generation using fingercodes. *Lecture Notes in Computer Science*, 4140:178–187, 2006.

[17] A. Rényi. On measures of information and entropy. *Proceedings of the 4th Berkeley Symposium on Mathematics, Statistics and Probability 1960*, pages 547–561, 1961.

[18] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423/623–656, 1948.

[19] C. Soutar, D. Roberge, A. Stoianov, R. Gilroy, and B.V.K. V. Kumar. Biometric encryption™. Available at `http://www.bioscrypt.com/assets/Biometric_Encryption.pdf`.

[20] C. Soutar, A. Stoianov, and R. McIver. Biometrics and digital keys. Available at `http://www.biometrics.org/bc2004/CD/PDF_PROCEEDINGS/MicrosoftPowerPoint-8_SoutarBrief.ppt[Read-Only].pdf`, 2004.

[21] J. Szczepanski, E. Wajnryb, J. M. Amigó, M.V. Sanchez-Vives, and M. Slater. Biometric random number generators. *Computers & Security*, 23, February 2004.

[22] V.V.T. Tong, H. Sibert, J. Lecoeur, and M. Girault. Biometric fuzzy extractors made practical: A proposal based on fingercodes. *Lecture Notes in Computer Science*, 4642:604–613, 2007.

[23] Wikipedia. Min-entropy — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Min-entropy`. [Online; accessed 18-May-2008].

[24] Wikipedia. Reed-solomon error-correction — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Reed-Solomon_code`. [Online; accessed 18-May-2008].

[25] Wikipedia. Suport vector machine — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Support_Vector_Machine`. [Online; accessed 18-May-2008].