Master semester project presentation

Securing biometric templates on smart cards

Jonathan Cheseaux - jonathan.cheseaux@epfl.ch

Supervisors : Andrzej Drygajlo - Leila Mirmohamadsadeghi

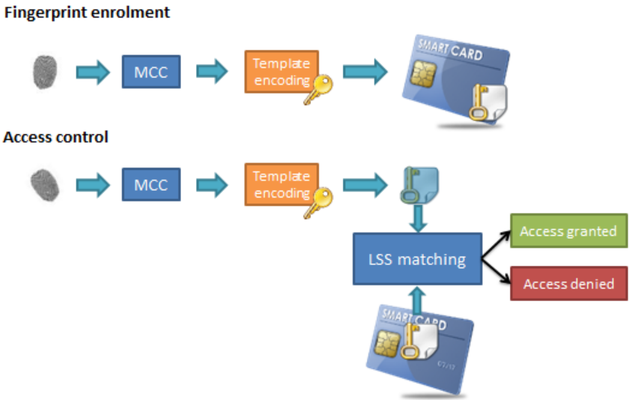
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

May 30, 2013

Outline

Introduction

Objectives


Fingerprint enrolment
MCC
Template encoding
SMART CARD
Access control
MCC
Template encoding
LSS matching
Access granted
Access denied
SMART CARD

Background

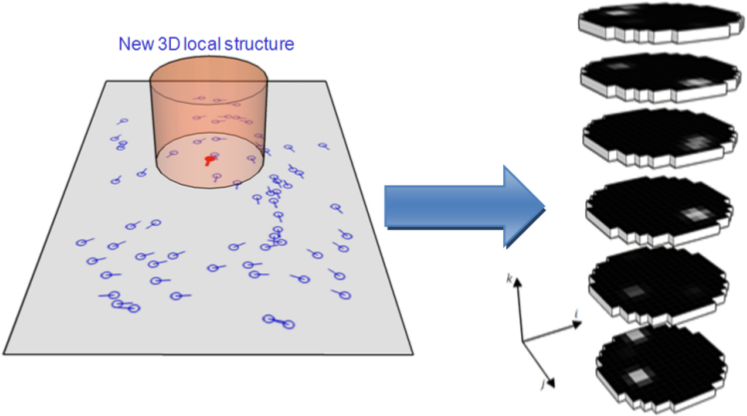
Fingerprints quality can be altered by different factors

- Physical pressure on the scanner
- Finger orientation, distortion
- Wet/Dry fingers
- Age of the user

Minutiae Cylinder-Code (MCC)

- ✓ Robust against feature extraction errors
- ✓ Does not depend on finger orientations/distortions
- ✓ Produces fixed-size descriptors

Recognition chain
Minutiae Cylinder-Code (MCC)



Recognition chain

Template transformation

Identity theft can have serious consequences

- Fingerprints are sensitive data !
- Biometrics templates should not be reversible
- Biometrics templates should be revocable
- **Privacy protection** needed

Properties of the template transformation

✓ Revocable (key)

✓ Irreversible (transformation + binarization)

Recognition chain

Template transformation

Algorithm 1: Biometric template privacy protection

Input: D, a set of minutia descriptors $\langle T_1, T_2, \ldots, T_n \rangle$
Input: H, the encryption key
Output: P, the transformed template

$P \leftarrow (0, 0, \ldots, 0)$;
index $\leftarrow 0$;
$A \leftarrow 5000$; *threshold* $\leftarrow 10^5$; \\\\ *Tuning parameters*
for *each descriptor T in D* **do**
    for $i \leftarrow 0$ **to** *length(T)* **do**
        if i **even then**
            $p \leftarrow (A * (T[H[i]] + T[H[i+1]]))^2$ **mod** n
            if $p >$ *threshold* **then**
                $P[index] \leftarrow 1$
            else
                $P[index] \leftarrow 0$
            index $\leftarrow$ *index* $+ 1$;

Figure : Biometric template privacy protection

Perfect fingerprint matching impossible

- No two transformed templates of the same finger are 100% identical
- Depends on feature extraction quality

Local similarity sort

- ✓ Compares distances of all 2-by-2 cylinders
- ✓ Based on angular differences of minutiae pairs
- ✓ Produces a similarity score from 0.0 (no match) to 1.0 (perfect match)

Recognition chain
Template matching

Algorithm 2: Local similarity sort

input: $T1$, enrolment protected template $\langle P_1^{(1)}, P_1^{(2)}, \ldots, P_1^{(n)} \rangle$
$T2$, verification template $\langle P_2^{(1)}, P_2^{(2)}, \ldots, P_2^{(n)} \rangle$
$M1$, enrolment minutiae directions $\langle M_1^{(1)}, M_1^{(2)}, \ldots, M_1^{(n)} \rangle$
$M2$, verification minutiae directions $\langle M_2^{(1)}, M_2^{(2)}, \ldots, M_2^{(n)} \rangle$
delta, threshold

Output: *score*, the matching score between 0.0 and 1.0

$P \leftarrow (0, 0, \ldots, 0)$; $index \leftarrow 0$; $gamma \leftarrow (0, 0, \ldots, 0)$
$minNP \leftarrow 3$; $maxNP \leftarrow 10$; $muP \leftarrow 30$; $tauP \leftarrow 0.4$;
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** n **do**
    $norm \leftarrow \| T_1^{(i)} \| + \| T_2^{(j)} \|$
    if $angularDiff(M_1^{(i)}, M_2^{(j)}) \leq delta$ **then**
      $gamma[index] \leftarrow 1.0 - \frac{\|\text{hammingDistance}(T_1^{(i)}, T_2^{(j)})\|}{norm}$
      $index = index + 1$
sort$(gamma)$
$z \leftarrow (1 + \exp(-tauP * (n - muP)))$
$nP \leftarrow minNP + \lfloor (z * (maxNP - minNP)) \rfloor$
$sum \leftarrow 0$
for $i \leftarrow 0$ **to** nP **do**
  $sum = sum + gamma[i]$
return sum/nP

Figure : Local similarity sort

	Java Card 2.2	Java Card 3.0
byte, short	✓	✓
int, long, char, String	✗	✓
Threads	✗	✓
Garbage collection	✗	✓
Networking	✗	✓
2D Arrays	✗	✓
CPU	8-bit CPU	32-bit CPU
RAM	8kB of RAM	24kB of RAM

Table : Comparison of Java Card 2.2 and Java Card 3.0

Mandatory header				Optional body		
CLA	INS	P1	P2	Lc	Data field	Le

Table : Command APDU structure

Optional body	Mandatory trailer	
Data field	SW1	SW2

Table : Response APDU structure

Using smart cards for recognition
Smart card simulator

Java Card Workstation Development Environment `JCWDE`

- Simulator that emulates a Java Card
- Communication through `APDU` exchanges
- ✗ **No debugging tools**

Debugging a Java Card application

- First developed in Java world, under Java Card restrictions
- Possibility to use the Java Debugger tool and the console display
- Switching the entire application on Java Card remains a **tough challenge**

32-bit integers not implemented

- Use 16-bit Short type instead
- Maximum value : $2^{15} - 1 = 32767$

Floating-point values not implemented

- Only integer values allowed (Short)

Garbage collector not implemented

- Impossible to instantiate Objects outside the constructor
- Be cautious with memory usage

Materials and methods

INS **code**	**Corresponding instruction**
0x01	Load verification template
0x02	Load verification minutiae
0x03	Enrol user's template
0x04	Enrol user's minutiae
0x05	Initiate fingerprint matching
0x06	Reset the card

Table : CLA codes for the project

Connection to the card

- Connexion to the simulator through a TCP socket.
- Applet loaded on the card is identified with a unique AID
- If fields SW1 and SW2 in the APDU response are 0x90 and 0x00 it means that selection is successful

CLA	INS	P1	P2	Lc	Data field
0x00	0xA4	0x04	0x00	0xA0	0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x00

Table : Command APDU content for the applet selection

Templates and Minutiae uploading

- APDU size limit is 255 bytes in total
- Templates and minutiae are broken into several chunks and sent separately.
- Fields P1 and P2 inform which packet is being sent

CLA	INS	P1	P2	Lc	Data field
0x00	0x03	Packet number		Packet size	Packet content

Table : Command APDU content for enrolling a user's template

Fingerprint matching

- Templates/minutiae loaded on the card
- APDU request sent to initiate matching process

CLA	INS	P1	P2
0x00	0x05	0x00	0x00

Table : Command APDU content for initiating matching process

Data field	SW1	SW2
score (2 bytes)	90	00

Table : Response APDU

for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** n **do**
    $norm \leftarrow \| T_1^{(i)} \| + \| T_2^{(j)} \|$
    if $angularDiff(M_1^{(i)}, M_2^{(j)}) \leq delta$ **then**
      $gamma[index] \leftarrow 1.0 - \frac{\|\mathrm{hammingDistance}(T_1^{(i)}, T_2^{(j)})\|}{norm}$
      $index = index + 1$
sort$(gamma)$
$z \leftarrow (1 + \exp(-tauP * (n - muP)))$
$nP \leftarrow minNP + \lfloor (z * (maxNP - minNP)) \rfloor$
$sum \leftarrow 0$
for $i \leftarrow 0$ **to** nP **do**
  $sum = sum + gamma[i]$
return sum/nP

$P \leftarrow 1550$

for $i \leftarrow 1$ **to** n **do**

  for $j \leftarrow 1$ **to** n **do**

    $norm \leftarrow \| T_1^{(i)} \| + \| T_2^{(j)} \|$

    if $angularDiff(M_1^{(i)}, M_2^{(j)}) \leq delta$ **then**

      $gamma[index] \leftarrow P - \frac{P * \|\text{hammingDistance}(T_1^{(i)}, T_2^{(j)})\|}{norm}$

      $index = index + 1$

sort$(gamma)$

$z \leftarrow (1 + \exp(-tauP * (n - muP)))$

$nP \leftarrow minNP + \lfloor (z * (maxNP - minNP)) \rfloor$

$sum \leftarrow 0$

for $i \leftarrow 0$ **to** nP **do**

  $sum = sum + gamma[i]$

return sum/nP

Software

LSS algorithm (non Java Card version)

for $i \leftarrow 1$ **to** n **do**

  for $j \leftarrow 1$ **to** n **do**

    $norm \leftarrow \| T_1^{(i)} \| + \| T_2^{(j)} \|$

    if $angularDiff(M_1^{(i)}, M_2^{(j)}) \leq delta$ **then**

      $gamma[index] \leftarrow 1.0 - \frac{\|\mathrm{hammingDistance}(T_1^{(i)}, T_2^{(j)})\|}{norm}$

      $index = index + 1$

sort$(gamma)$

$z \leftarrow (1 + \exp(-tauP * (n - muP)))$

$nP \leftarrow minNP + \lfloor (z * (maxNP - minNP)) \rfloor$

$sum \leftarrow 0$

for $i \leftarrow 0$ **to** nP **do**

  $sum = sum + gamma[i]$

return sum/nP

Software

LSS algorithm for Java Card environment

$P \leftarrow 1550$
for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** n **do**
        $norm \leftarrow \| T_1^{(i)} \| + \| T_2^{(j)} \|$
        if $angularDiff(M_1^{(i)}, M_2^{(j)}) \leq delta$ **then**
            $gamma[index] \leftarrow P - \frac{P * \|\text{hammingDistance}(T_1^{(i)}, T_2^{(j)})\|}{norm}$
            $index = index + 1$
sort$(gamma)$
$nP \leftarrow LOOKUPTABLE[n]$
$sum \leftarrow 0$
for $i \leftarrow 0$ **to** nP **do**
    $sum = sum + gamma[i]$
return sum/nP

Demonstration

Results

Test databases

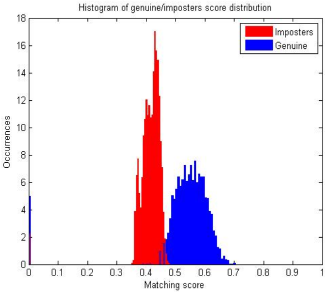
FVC 2000-2002-2004

- Databases provided by the Fingerprint Verification Competition
- Contains several users' fingers impression and also synthetic fingerprints
- Quality can vary a lot
- Features extracted with FingerJetFX software

Training set / Test set

- FVC2000 - DB1-2-3 for parameters tuning
- FVC2004 - DB1 for final results

Imposters/Genuine scores
Original recognition algorithm



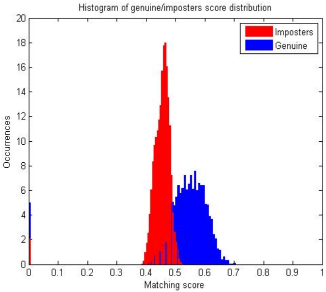


Table : Original matching - Histograms of imposters **(red)** /genuine **(blue)** score repartition over 5000 samples. On the left : unknown key scenario. On the right : stolen key scenario.

Imposters/Genuine scores
On-card matching

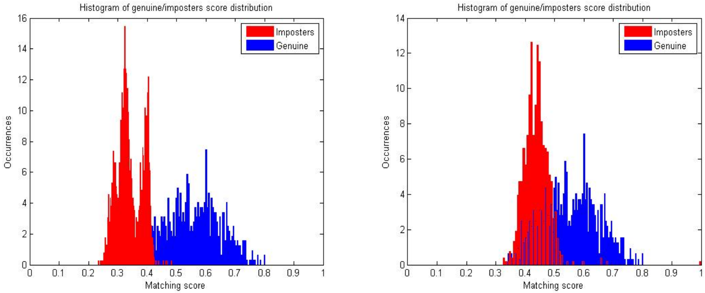
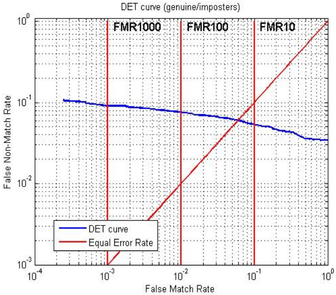


Table : On-card matching - Histograms of imposters **(red)** /genuine **(blue)** score repartition over 2500 templates. On the left : unknown key scenario. On the right : stolen key scenario.

False match rate (FMR) and False non-match rate (FNMR)
Original recognition algorithm



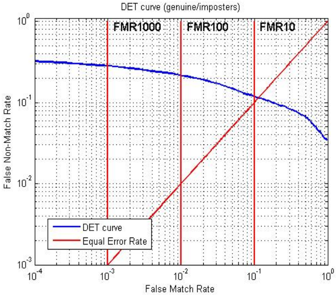


Table : Original matching - EER **(red)** and DET curve **(blue)**. On the left : unknown key scenario. On the right : stolen key scenario.

Master semester project presentation

Introduction
Background
Materials and methods
Results
Conclusion
Questions ?

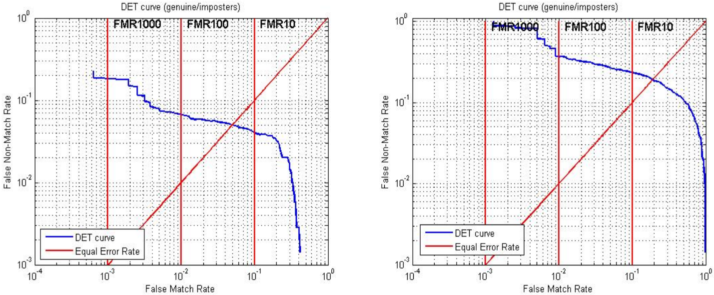


Table : On-card matching - EER **(red)** and DET curve **(blue)**. On the left : unknown key scenario. On the right : stolen key scenario.

Performance comparison

Original algorithm			
Unknown key		Stolen key	
FAR	**FRR**	**FAR**	**FRR**
10%	5.34%	10%	11.52%
1%	7.07%	1%	19.34%
0.10%	7.55%	0.10%	21.55%
EER : 6.41%		EER : 11.97%	

Java Card implementation			
Unknown key		Stolen key	
FAR	**FRR**	**FAR**	**FRR**
10%	4.00%	10%	23.29%
1%	5.86%	1%	32.71%
0.10%	6.71%	0.10%	37.00%
EER : 4.63%		EER : 19.34%	

Table : `FRR` values for fixed `FAR` rates for two scenarios (key unknown and key stolen)

	Genuine		Imposters			
			Unknown key		*stolen key*	
	mean	**deviation**	**mean**	**deviation**	**mean**	**deviation**
Original	0.53	0.11	0.37	0.05	0.41	0.05
On-card	0.56	0.09	0.34	0.04	0.44	0.04

Table : Mean and standard deviation of the score distributions for on-card matching and original algorithm

Conclusion

Future directions

Testing on real hardware

- Goal partially achieved :
  - Implementation works fine with simulator
  - Not tested on real devices
  - Compatibility with real devices not ensured

Precision loss

- Precision loss occurs because of integer division
- Solution : implement manually 32-bits floating-point values
- May not be suitable for smart cards

Future directions

Encryption

- Sensitive informations
  - Minutiae angular directions
  - Protected template
  - Transformation key
- RSA encryption method already implemented in Java Card

Fine tuning parameters

- Several "empirical" parameters
  - Precision parameter, Template binarization threshold
  - Score threshold (accept/reject)
- Machine learning for optimizing choice of parameters

Questions ?