A Security Mechanism to Increase Confidence in M-Transactions

David Pequegnot, Laurent Cart-Lamy, Aurélien Thomas, Thibault Tigeon, Julien Iguchi-Cartigny and Jean-Louis Lanet
Secure Smart Devices - XLIM Labs,
University of Limoges,
83 rue d'Isle, 87000 Limoges, France
{david.pequegnot, laurent.cart, aurelien.thomas, thibault.tigeon}@etu.unilim.fr
{julien.cartigny, jean-louis.lanet}@xlim.fr

***Abstract*—Currently, NFC phones are coming in the handheld market, providing facilities to perform m-transactions. Obviously, this type of operation requires special security precautions. Indeed, a malicious code could intercept and hijack the system, even if there is a smart card. For example, the amount of the payment displayed in the terminal can be hijacked by an attacker to fool the user, or user's credential can be stolen thanks to a *keylogger* (and thus malicious codes can perform unwanted m-transactions automatically).**

This paper describes a security mechanism based on a graphical Turing test to prevent m-transactions submission by malwares. Firstly it introduces current m-transactions solutions. Then it explains the security mechanism that we propose to tackle the problem of untrusted handheld devices. It also underlines a proof of concept we implemented, to test its feasibility on a SIM card. Finally, it gives information on performances corresponding to the implementation that we made.

***Keywords*-banking; information security; m-transactions; Java Card; cellular phones**

I. Introduction

During the past years, the featured phones and smartphones markets grew up quite heavily. The first quarter of 2011 confirms this trend with an increase of 83% compared to the same period in 2009 concerning the worldwide smartphone market [1].

Many people have adopted this new type of device for communication and computing. Main usages are games, Internet browsing, e-mails, pictures and videos, etc. A new trend is appearing, easing user daily life transactions: using smartphone as a smart card, but with the advantages of one device for many transactions and rich user interactions. This trend is accelerated by the production of Near Field Communication (NFC) technology. Applications examples are payment, home banking, ticketing, loyalty, etc. and they may be executed for proximity transactions (through NFC technology) or distant transactions (through mobile network technologies such as GSM/UTMS, or WIFI). Given the characteristics of the mobile phone market, the development of NFC-contactless technology in mobile phones could be fast and important.

This trend comes not only from the users themselves, but also from service providers, which consider mobiles as a new vector for enhancing customer relationship, or simplifying and reducing the cost of some tasks such as renewing a subscription for public transports on unmanned automatons or over the Internet, instead of manual transactions where the user has no queue in front of a booth for accessing to an employee.

However, security requirements for these types of applications are quite high and complicate to satisfy. Moreover, they may require limitations to the user experience which make these measures quite unpopular among users.

Section II reviews relevant works. Section III describes the security model we propose to make mobile transactions more secure. Section IV introduces a partial implementation we made. Sections V and VI detail this implementation. Section VII highlights the performances and section VIII the security of such a mechanism. Finally, section IX underlines the remaining work to realize.

II. Related Works

Nowadays, there is a growing interest in mobile banking. Many studies are related to the security of such a service. However, too few takes into account the possibility of an untrusted device.

Roßnagel and Muntermann [2] suggested the use of a SIM card as a security module in mobile devices to ensure accountability, confidentiality and integrity in real-time services (information and transaction services). Pohlmann et al. [3], and Hassinen et al. [4] also proposed to use a PKI-enabled SIM card based on a public key infrastructure. The objectives are to guarantee authentication, non-repudiation, integrity and confidentiality for an e-payment system. However, they did not discuss the problem of untrusted platforms.

The use of a graphical Turing test was already introduced by Xie et al. [5] to prevent malicious codes diffusion. They take as an example the case of a MMS-based malware. To address this issue, users have to valide the test before sending such a message. Thus, the malware is blocked by this "Turing test" and its diffusion could be stopped. Our work described in this paper is based on the same idea that malicious code cannot read an image.

More specifically, there is already a set of payment solutions on the market, each using its security mechanism. For instance, there are Google Wallet, Paypal, iZettle, Buyster or Kwixo. However, most of these solutions do not meet our requirements: they assume that they are running on a trusted mobile phone without malicious code, and especially without a *keylogging* software.

For example, most of them just rely on a set of credentials (mostly an identifier and a password) to perform transactions. Of course, we mean the existence of a secure network link between the mobile phone and service providers. Nonetheless, malicious codes may listen the mobile phone keyboard and retrieve the user's credentials (this security issue already exists on personal computers). It may also modify information displayed on the screen, such as the amount.

Thus, the security of these solutions must be improved to increase confidence in such a transaction and this is the subject of this paper.

III. Security Mechanism Overview

A. Context

The security mechanism described in this article aims to tackle the problem of non-secured platforms when performing mobile transactions (i.e. using handheld devices).

Mobile devices (and especially smartphones) are quite similar to personal computers: they have an Operating System (OS) loaded after power-up through a boot process. Most of the time, the OS allows the user to launch any application installed in the device. Moreover, if smartphones are sold with a set of applications pre-installed by manufacturers, development environment are often provided for third parties to develop applications and enhance the choice of applications. Then these "third party applications" can be available through a market or the Internet.

Some issues can be predictable: OS may have security flaws, applications may contain malicious code (for instance Trojans, and especially with third party software), etc. It is really difficult to have an entirely secure platform, and thus ensuring a secure transaction in which the user can have confidence in his terminal, and the payment service in the transaction request itself.

On the one hand, when performing a mobile transaction, the payment service must be sure that the user agrees. Thereby, a malicious code loaded in the mobile phone must not validate a payment without the user being aware.

On the other hand, the user must trust the amount displayed in its mobile phone. Any modifications of this amount (in order to fool the user by displaying a lower amount than the real one) must be detected by him.

To address these issues, our security mechanism is able to add semantics while the transaction process to establish the desired confidence and to ensure a secure transaction.

B. Design

1) Secure Element: The security mechanism relies on a trusted component to ensure the transaction security. This component will be named Secure Element (SE).

In the GSM world, there is a component of trust which is common to all phones: the Subscriber Identity Module (SIM).

The SIM is a complex smart card using standards defined at the European Telecommunications Standards Institute (ETSI), and also the Java Card technology [6] and Global Platform [7] standards.

Such smart cards embeds a virtual machine, which interprets application byte code. It contains all necessary elements for multi-application support: the isolation between applications of different issuers, the possibility of securely downloading and installing new applications including their initial keys, removing applications, etc. The Global Platform specification defines also a protocol to control the ability to download code into the card: the owner of the code must have the necessary credentials to perform the action.

However, other forms of SE may exist. For instance, a specific chip embedded on the motherboard of the mobile, or a secure SD card, but with computing possibilities.

Fortunately, our security mechanism does not rely on the SE hardware implementation. Additionally, it assumes that the mobile phone contains a SE which provides security support. The component must handle all sensitive operations. This mechanism must contain all sensitive data and cryptographic keys to communicate with payment services and validate a transaction initiated by the user through his mobile phone.

2) A First Naive Implementation: A naive implementation (figure 1) consists on giving the user's Personal Identification Number (PIN) code, or a more complex password, to validate the transaction. For example, when a user wants to buy a product or a service with his handheld device (through a web page, a QR code [1], NFC, etc.) a payment request is sent to the SE. Then, the component asks the user for his PIN code to confirm the transaction.

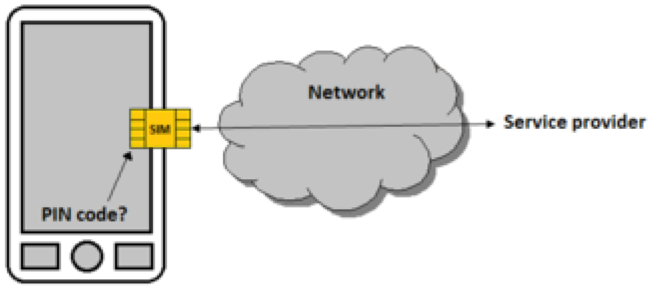

Fig. 1. A naive implementation in which the user has to enter his PIN code to validate the transaction

However, this implementation reveals two main security issues. First, the terminal can be compromised by malicious code which is able to steal the user's PIN code (for instance by listening the keyboard thanks to a *keylogger*), and thus to accept the transaction without the user's agreement. The second security issue consists on displaying a wrong amount to the user and doing a payment request to the SE with a larger amount than expected.

[1]A QR code is a two-dimensional barcode. It is a matrix with black squares on a white background which encodes data: an URL for example.

The security mechanism must be able to address these issues. The previous example demonstrates that using only a PIN code and trusting the amount displayed in the handheld device can be a weak solution concerning the mobile transactions security.

3) Introducing CAPTCHA in the Security Mechanism: "CAPTCHA" stands for "Completely Automated Public Turing test to tell Computers and Humans Apart". It is a test, mostly used in online forms to be protected against automatic and intensive submissions made by malicious robots.

Interesting properties come from CAPTCHA implementations which generate pictures.For instance, humans can read distorted text mixed with parasite curves and gradient background, but current computer programs cannot (in some extent).

Considering the "naive implementation", the kind of attacks described in the section III-B2 can be avoided by using this sort of graphical turing test. Indeed, malicious codes first have to "understand" the message in order to perform undetectable modifications. Moreover, these unwanted modifications need to keep the parasite curves and background continuity.

Both SE and CAPTCHA are the centre of our security mechanism. But the CAPTCHA implementation is a little bit different from the one used online. Indeed, it contains additional semantics to prevent automatic transaction approval by malicious code (and thus an untrusted mobile phone) if the PIN code is stolen.

C. Overview

1) CAPTCHA implementation: To be consistent with our security mechanism, everything cannot generate the CAPTCHA picture. Indeed, an untrusted component or an external device (for example a webserver) can create a fake picture in order to fool the user. So, the only component which must generate the CAPTCHA image is the SE defined in the section III-B1.

Moreover, the CAPTCHA implementation generates a picture which has a different semantics from those seen online. We have two objectives: the user must detect any modification of the amount displayed on his handheld device and the security issue corresponding to the user's PIN code theft must be avoided.

Concretely, the generated picture must contain at least two information:

- **the transaction amount**: the value supplied by the transaction request to the SE;
- **a Secure Code** (SC).

Both of them are included in the CAPTCHA image (the figure 2 is an example of such an image), making their recognition and modification by a computer program hard to achieve, and even more with the semantics we added.

The SC is an important item of our security mechanism. It is a random number generated for each picture rendering by the SE. To validate a transaction, the user must send the SC displayed in the CAPTCHA to the SE, in addition to his PIN code (figure 3).

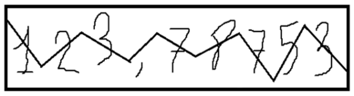

Fig. 2. CAPTCHA format in the security mechanism

The PIN code is still required to prevent anyone from being able to validate the transaction instead of the legitimate user. For instance, if someone has access to the mobile phone, he must not be able to make payments instead of its owner. Futhermore, the SC never leaves the SE, except its representation in the CAPTCHA picture.

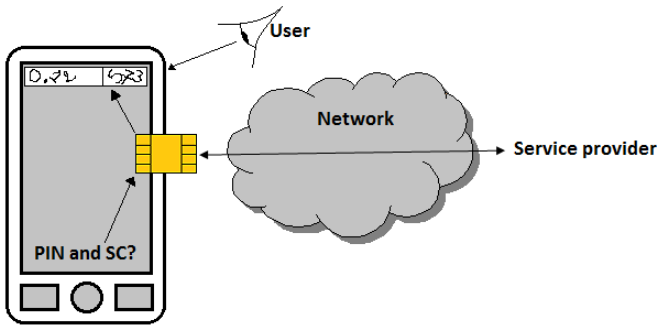

Fig. 3. The security mechanism including the CAPTCHA verification

To improve the security and forbid modifications, random curves are added through the entire CAPTCHA. A gradient background can be also implemented.

2) Semantics: In order to allow the user to detect any attempt to modify the generated picture, our security mechanism proposes to personalize the CAPTCHA generation parameters. The customization process can be done in an initialization step in which the user can define:

- **The font**: (optional) the SE asks the user for drawing each digit (from 0 to 9) and the decimal point. He can use his handwriting or any other symbols, like pictograms (but then he should be able to recognize the digits). As a result, while the CAPTCHA image is generated, the SE will use these drawing as the font. There is a predefined font in the SE, but it is strongly recommended to customize it in order to increase the image recognition difficulty.
- **Parasite curves**: the user may define parasite curves parameters: for example their shape, etc. These curves must cross the entire image.
- **Deformation parameters**: for instance, the minimal and maximal rotation of digits.

These parameters are the semantics defined by our security mechanism. Consequently, for each picture generation the user is able to check the CAPTCHA validity visually.

The use of a specific font is a good way to prevent any attempt from reading the image content by malicious code (people handwriting is rarely the same, using symbols instead of letters adds complexity, etc.).

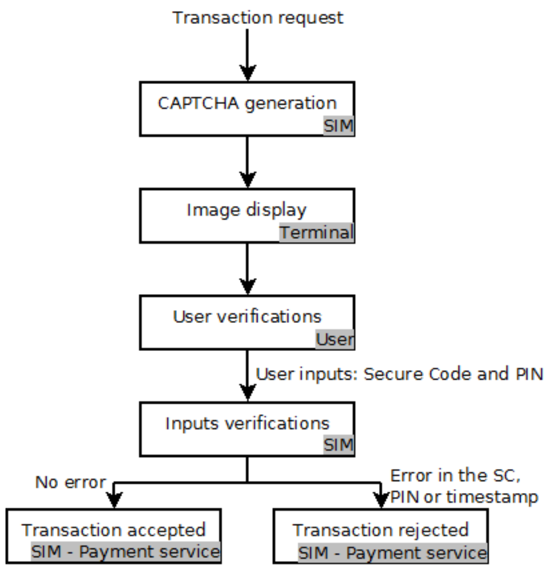

Fig. 4. CAPTCHA generation and verification steps

3) Protections: Considering an untrusted platform (a handheld device with its OS and applications) in which a user wants to perform a mobile transaction, our security mechanism is well suited. Indeed, the use of a CAPTCHA image blocks any attempt of a malware to perform an attack:

- Trying to modify the amount is really difficult thanks to the user-defined font, curves and CAPTCHA generation parameters. At each CAPTCHA generation, parasites curves are randomly drawn across the entire picture. Because the amount and the SC are concatenated, trying to modify the picture also means keeping the aspect and the continuity of the curves, knowing that the user is able to recognize them. Moreover, digits may overlap and have variable width to make their recognition difficult and prevent an attacker from finding the separation between the amount and the SC. As a consequence, a malware cannot carry out a “copy and paste” attack by cutting an amount from a previous CAPTCHA and re-presenting it in the current transaction.
- The use of a randomly generated SC prevents the PIN from theft security issue.
- The user-defined font prevents also the SC from being read by malicious code with the purpose of validating a transaction without the user being aware.

The security mechanism is summarized in figure 4.

Moreover, a timestamp mechanism may be implemented in the SE. It should limit the time granted to the user to enter the SC and PIN code. Thus, a malicious code would have less time to try to read data from the picture: this mechanism often exists in CAPTCHA implementation online for the same purpose. But the time must be read from a trusted source (for example from a time server using certificates or the carrier). Indeed, it makes no sense reading the time from untrusted devices or components, like the terminal.

IV. Implementation Principles

The security mechanism described in the section III relies on the Secure Element in order to generate the CAPTCHA picture. But generating such a complex picture in a low-performance but secure component (like SIM cards) can be really challenging. Additionally, users would not wait too long between the beginning of the transaction and the validation by the SE and the payment service.

This is the rationale of the first design. But the security mechanism has not been fully implemented: we just wanted to highlight the SE possibilities in order to generate a complex CAPTCHA picture. We aimed to generate an image in a reasonable time (in few seconds) with some deformations (rotation, scaling, etc.).

A. The Secure Element in our Implementation

Considering the most common trusted component in the GSM world, we chose to implement the CAPTCHA generation in a SIM card which uses the Java Card technology.

Thanks to this, we are able to code applications using a subset of the Java programming language in a secure environment in an embedded device, but with limited memory and processing capabilities. We also have to consider that Java Card applications (applets) are 2 to 4 times slower than native ones in a smart card, but it is a good compromise between the speed of development and execution.

Moreover, we created an application in a handheld device to test our Captcha implementation in the SIM card. It uses the Mobile Information Device Profile (MIDP) which is a Java 2 Micro Edition profile. A MIDP application is named “MIDlet”. To communicate with the SIM card with a third-party application, the Security And Trust Services API (SATSA) is available, but for security reasons, the MIDlet must be signed by a trusted authority in order to allow the communication.

It is also possible to implement such an application in smartphones which does not support MIDP. For example, there is the “Secure Element Evaluation Kit API for the Android platform” API (seek-for-android) [8] to communicate with the SIM card in the Android operating system.

B. Implementation Choices

1) Image Format: First of all, we chose to use our own image format. Indeed, well-known formats (i.e. bitmap, etc.) share more information than needed in our implementation and we target a constrained environment. Moreover, we have to consider data exchanges between the SE and the application. In the Java Card context, Application Protocol Data Unit (APDU) frames exchanged take time to be sent and received.

Indeed, the maximum APDU size is 256-byte long. But the generated CAPTCHA picture size is much greater (from 868 bytes to 1116 bytes in our implementation): many APDU frames are needed in order to send the entire CAPTCHA to the terminal.

Our image format is really simple: it is an array of bytes where each bit corresponds to a pixel (a bit at 0 means white, 1 means black). We also considered some optimizations: we

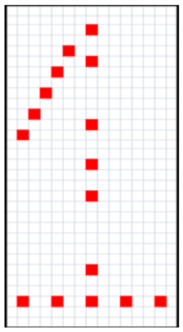
Fig. 5. Digit “1” storage representation

```
public static final byte [] ONE_X = { −6,
    −3, 0, 3, 6, 0, 0, 0, 0, 0, 0, −2,
    −3, −4, −5, −6 };
public static final byte [] ONE_Y = {
    −13, −13, −13, −13, −13, −10, −3, 0,
    4, 10, 13, 11, 9, 7, 5, 3 };
```

Listing 1. Storage of the digit “1”

TABLE I
APDU FRAME CORRESPONDING TO THE AMOUNT 12.34

CLA	INS	P1	P2	Lc	Data				
80	30	00	00	05	02	01	02	03	04

thought about using the JPEG format or sparse matrix to decrease the image size, but the optimization process would take more time than directly sending the entire CAPTCHA picture with many APDU frames.

2) Picture Modifications and Semantics: Furthermore, our implementation supports these digit modifications: rotation, scaling and deformation.

On the other side, like introduced in IV, we do not implement the full security mechanism. The font is not specific to the user. We are using our own font with a specific format (see IV-C1), to make the digit modifications easier to implement. The curves are randomly generated: it is a line going up for a digit, then down for the next one, etc. but in a continuous way.

Finally, the Secure Code is a three-digit random number. According to the security mechanism, the font and the SC are only stored in the SIM card.

C. Initial Data Storage

This section describes how data are stored in the smart card according to our implementation. We must face the resource and performance problems of the SIM card. The digits should not need too much space to be stored, and drawing algorithms must be adapted to the environment.

1) Digits and decimal point storage: All the numbers from 0 to 9 and the decimal point are stored in the smart card. We are using our own font to make the implementation (and digits modifications) easier to proceed.

The image can be seen as a grid of pixels with two axes. A digit or a decimal point is represented by a set of points where each point has a coordinate in this grid. But they can be numerous, and applying coordinates transformations for each point of the digit (in order to modify the digit appearance) would take too much time. So, only the coordinates of some points are stored to represent partially the digits or the decimal point.

The figure 5 represents graphically how the digit “1” is stored in the card, while the listing 1 shows how the digit is effectively stored.

Then, we are able to draw the entire figure thanks to the Bresenham algorithm [9]. It allows us to find the shortest path between two points and to draw the corresponding segment. It is a very efficient algorithm adapted to our constrained environment.

Usually, no more than 20 coordinates are needed for a digit. Because a coordinate requires two bytes to be stored in our implementation (one for each axis), the storage of the font would not take more than 440 bytes in the secure element.

2) Sine and Cosine Functions Storage: To perform geometric transformations (i.e. digits rotation), we use sine and cosine functions. But those are not present in the Java Card API. So we had to store 22 values of sine and cosine corresponding to a rotation between $-\Pi/4$ and $\Pi/4$.

But we met a second problem: the calculated values are between -1 and 1. Moreover, Java Card does not support decimal values. So, the sine and cosine values have been multiplied by 1000 to store them as short values.

V. CAPTCHA GENERATION PROCESS

This section describes the Java Card applet behaviour to generate the CAPTCHA picture. We aimed to generate and send the image in few seconds to be consistent with our security mechanism and objectives.

A. Amount Reception by the Secure Element

When a user wants to buy a service or a product with our security mechanism, a transaction request must be sent to the Secure Element. It is the component in charge of validating or rejecting the transaction. So, an APDU frame must be sent to the smart card with a specific format.

In our implementation, the data field must contain (in order): the number of digits before the decimal point and each digit corresponding to the amount (excluding the decimal point). The maximum is 999.99. The table I is an example of such an APDU for the amount 12.34. `CLA`, `INS`, `P1`, `P2` and `Lc` fields correspond to the header of the frame.

The timestamp is retrieved at this step. Moreover, the image size is defined here with the number of digits which will be displayed in the picture.

B. Secure Code Generation

The Secure Code consists on 3 digits randomly generated one by one for each picture rendering, with the secure implementation of the “random” function supplied by the Java Card

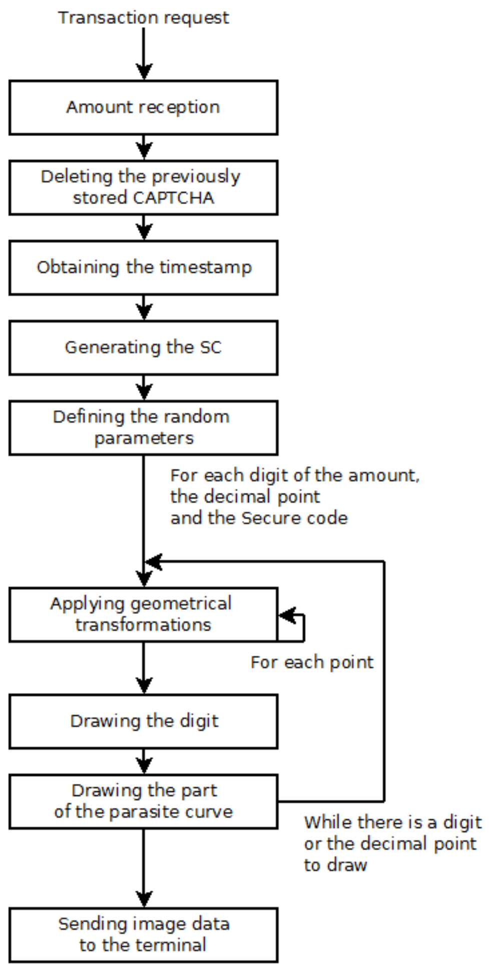
Transaction request
Amount reception
Deleting the previously stored CAPTCHA
Obtaining the timestamp
Generating the SC
Defining the random parameters
For each digit of the amount, the decimal point and the Secure code
Applying geometrical transformations
For each point
Drawing the digit
Drawing the part of the parasite curve
While there is a digit or the decimal point to draw
Sending image data to the terminal

Fig. 6. CAPTCHA generation process

API. It never leaves the card except in the CAPTCHA image. So when the user enters the CAPTCHA response, the applet is able to verify that the user-input SC corresponds to the one stored in the smart card.

The SC is stored in an byte array in the Random Access Memory (RAM) of the smart card in which each digit is an element of this array. Thus, when the SIM card is unplugged, the content of the array would be erased.

C. Drawing the Entire Picture

Drawing the picture needs a lot of optimization and tricks in order to meet our main objective: generating a CAPTCHA image in seconds. Indeed, if the SIM card is a Secure Element, it is also very constraint in performances and resources. For instance, any unnecessary array copy must be avoided.

The picture format is defined in the section IV-B. In the smart card, it is a linear array which is reset for each CAPTCHA generation. Digits are drawn one after the other. We know the decimal point position in the image thanks to the initial APDU frame corresponding to a transaction request (see the section V-A).

To draw a digit (or the decimal point), the following steps are performed:

1) Random parameters are generated (i.e. the angle to rotate, etc.).
2) For each pair of coordinate of the digit (the origin and the extremity of a segment):
    a) We apply the transformation with the parameters defined in 1).
    b) With these new coordinates and an offset corresponding to the position of the digit in the picture, we are able to draw the segment thanks to the Bresenham algorithm.
    c) We repeat the steps 2)a) and 2)b) for each pair of coordinates. At the end of the operation, we would obtain the desired digit.
3) Finally, the part of the parasite curve corresponding to the digit position is drawn.

Steps 2)b) and 3) mention a digit position. Indeed, we are using a fixed rectangle size to draw them. The space allocated for the drawing of a digit and the decimal point is 32 pixels wide and 31 pixels high. So, it is easy to found the offset corresponding to the x th digit. Our implementation can generate pictures 224 pixels wide (corresponding to a three-digit amount) to 288 pixels (a five-digit amount). The image height is fixed (31 pixels).

Moreover, it does not need any array copy. The digits are directly drawn in the linear array (it will be the final CAPTCHA picture). This is the point we worked on to optimize the CAPTCHA generation process: we are able to set directly the bit corresponding to the pixel in the linear array which contains the picture.

Thus, this implementation is able to generate a CAPTCHA picture in seconds according to our implementation principles defined in IV. It has been tested in a real SIM card supporting the Java Card technology.

VI. CAPTCHA Verification

On the other hand, the SE must verify the user inputs corresponding to our security mechanism (i.e. the PIN code, SC and timestamp). If something is wrong, the CAPTCHA becomes invalid and a new one must be regenerated. This protection prevents malicious codes from brute force attack against the SC. Moreover, to prevent other attacks (like disconnecting the SIM card), the CAPTCHA becomes invalid from the beginning of the verification.

A. PIN Code

The Java Card API provides the "`OwnerPIN`" class to manage PIN codes. It is a secure and reliable implementation able to protect against attacks based on program flow prediction. The class provides also some interesting features, and among them the possibility to define a maximum number of wrong tries and a method to securely compare the user input to the stored one.

Our implementation allows three tries before the PIN is locked, and thus before the application would no longer operate. It must be unlocked (for instance by a Mobile Network Operator, or a service provider) before new CAPTCHA pictures can be generated and m-transactions can be validated.

While the verification process, if the user-supplied PIN code is wrong, the CAPTCHA becomes invalid, and the number of remaining attempts is decremented. A new picture with a new Secure Code must be generated.

B. Secure Code

In our implementation, its value is stored in a byte array. Each byte of this array corresponds to a digit. The verification step consists only in a array comparison with the user input.

Morevover, the SC is stored in the Random Access Memory in addition to a flag defining the state of the CAPTCHA (valid or invalid). Unplugging the smart card will reset their value, and the CAPTCHA will switch in an invalid state.

C. Timestamp

After all, the timestamp verification is a little bit more complex. Indeed, we were not able to obtain a timestamp from the network. To solve this problem, the value is supplied by the terminal in an APDU frame. But the timestamp is represented by 8 bytes and the Java Card API does not support 8-byte values. To allow the comparison of timestamps, the value has been converted into an array.

Considering the retrieved timestamp, there is also a problem to tackle: byte values are signed. But in the 8-byte array, they must be interpreted as unsigned ones. Therefore, we have used a short array and a mask (`0xFF00`) to make the "conversion".

Then we needed to implement the addition and subtraction operations in order to calculate the maximum timestamp value before the CAPTCHA becoming invalid. While the validation process, if the timestamp value supplied with the user inputs is greater than the calculated one, the CAPTCHA verification fails.

It is important to note that this timestamp implementation is not secure. Indeed, we just wanted to obtain a proof of concept. Clearly, we had better use a value provided by secure devices (for example a time server using certificates like in [10] or the network time).

VII. Performances

SIM cards embed a very constraint but secure environment. This is why implementing such a dynamic image generation in an acceptable time is really difficult, and even more in Java Cards which are slower than native ones. Our implementation described in the sections IV, and V targeted code optimization.

A. Number of Operations

To optimize our implementation, we tried to decrease the number of Java operations (and especially complex Java operations) needed to generate the CAPTCHA picture, and more precisely in two processes: the drawing of the digits and the decimal point and the geometrical transformations.

Drawing a digit (or the decimal point) returns to render all segments and thus defining all the pixels in the array which contains the picture. We can break up the number of Java operations in this manner:

- Setting a pixel in the linear array needs 22 operations with two complex operations (modulo),
- Rendering a segment costs $x \times (22 + 10) + 14$ operations where x is the number of pixels of the segment, and $x \times 10$ and 14 are the number of additional operations needed in order to render the segment,
- Drawing a digit will finally cost $y \times (x_i \times (22 + 10) + 14)$ operations where y is the number of segments of the digit and x_i the number of pixels of the segment i.

The geometrical transformations are a little bit costly. Indeed, calculations are more complex, even if they are not numerous. It needs $z * 47 + 16$ Java operations, where z is the number of points of the digits.

As a result, drawing the digit with its geometrical transformation will cost $x \times 47 + (x–1)(y_i \times 32 + 14)$ operations where x is the number of points of the digit, $(x - 1)$ the number of segments and y_i the number of pixels of the considered segment. These results take into account of our optimization work.

B. Metrics

During the development process, we have considerably improved our implementation. To test it in an environment close to reality, we used a GEMXplore 3G v3 SIM card.

At the beginning, we made a lot of copies of arrays: the digits were drawn in a temporary array before being copied in the one containing the CAPTCHA picture. Generating an 8-digit CAPTCHA picture (including the secure code) took in average 1'46" from the payment request to the image reception by the user in his terminal. It was far too long.

So we tried to optimize our code considering the number of operations needed to render such a picture. The last implementation is able to generate an 8-digit CAPTCHA image in only 16". Removing all geometrical transformations on digits brings a result of 5".

VIII. Security issues

As a result the security mechanism described in this paper is able to increase confidence in m-transactions thanks to protections introduced by our CAPTCHA picture.

There are also some issues which cannot be resolved by such a mechanism. Foremost, the SE is our Trusted Computing Base (TCB): if the component is compromised, then the entire mechanism may be compromised. Morevoer, if the mobile phone was stolen and the thief knows the needed credential (using social engineering techniques or whatever), he can perform any mobile transaction, because he is able to read the CAPTCHA. The user has to declare his phone stolen to his service provider to prevent unwanted transactions. However these issues are generic: it is the same thing with trusted component based systems as credit cards, SIM cards, etc.

Considering the generated CAPTCHA image, obviously malicious code able to read the CAPTCHA may also modify the amounts (of course, it needs to know how to modify correctly the image), or validate transactions by itself, only if it knows the needed credentials (PIN code, etc.). That is why characters deformation and curves are introduced, and characters personalization is strongly recommended, making character recognition harder to perform.

Finally, concerning the implementation described in this paper, power and timing analysis may also guess the generated characters. Indeed, each character does not need the same number of operations to be drawn in the picture. However, these last security issues are currently related to our implementation and are limited by the random deformations and curves.

IX. Future Works

Our next implementation will be fully compliant with our security mechanism, allowing the user to enter his own font and parameters in an initialization phase.

However, we still have to decide how the data entered by the user will be stored. Indeed, the implementation described in this paper makes the transformations applied to the characters easier. A first idea is that the user defines the points representing the characters corresponding to the one used in our implemention. From the user perspective, this solution is still complex and the drawing would not be necessarily representative of what the user expects. It is hardly conceivable.

Thus, we consider two solutions. In all cases the user draws the symbols entirely, as he wishes them to appear:

- All the points representing the characters are kept. However, the computing time of new coordinates for each point may be too costly: then, transformations may be removed.
- Just a set of points is kept, by analyzing the drawing of the user. This solution is more complex because it requires an anlysis of the user inputs, but it also helps to keep the transformations.

On the other hand, we wish to go further than a proof of concept on the timestamp management: we want to realize an implementation using timestamps provided by a trusted device.

X. Conclusion

NFC phones will certainly introduce quickly new services thanks to the payment facilities they propose. However, most of handheld devices have an operating system, possibly with a set of third party applications, which may contain flaws that malicious code could exploit. These security issues are very annoying, especially in the banking field. It is necessary to find a mechanism which allows making payments safely.

The security mechanism we describe in the section III is a simple solution overcoming the security issues brought by a non-secured platform. Moreover, this solution can be considered generic: it does not rely on an Operating System and its security, or on the type of Secure Element used in such a transaction. However, the SE must be able to make graphic computations, and thus to generate the CAPTCHA picture in a reasonable time. But it is often a low performance and resources component...

For this purpose, we have demonstrated the possibility to generate a CAPTCHA image in few seconds with the most common trusted component in mobile phones: the SIM card. The implementation needs code and algorithm optimizations. So, our first tests permitted us to render a CAPTCHA picture in nearly two minutes in such a trusted and secure component with a Java Card environment. Our optimization work has allowed us to obtain a more reasonable time: only a few seconds.

As a conclusion, we have a simple security mechanism in implementation and use terms. The terminal only has to display the generated image and the user to read its content. Therefore, the problem is to choose the SE which must be fast enough to generate the picture in reasonable time, but we proved that it is possible with a SIM card. Our next goal is to test a complete implementation in a native card in which performances are 2-4 times better than a Java Card.

Acknowledgements

We would like to thank Jean-Claude Paillès, GREYC laboratory, ENSICAEN-University of CAEN-CNRS, security consultant for this project, who has developed the security mechanism. We would also thank our anonymous reviewers for their comments and advice.

References

[1] P. Alto. Android increases smart phone market leadership with 35% share. Canalys. [Online]. Available: http://www.canalys.com/pr/2011/r2011051.html

[2] H. Roßnagel and J. Muntermann, "Introducing sim-based security tokens as enabling technology for mobile real-time services," in *Identity and Privacy in the Internet Age*, ser. Lecture Notes in Computer Science, A. Jøsang, T. Maseng, and S. Knapskog, Eds. Springer Berlin / Heidelberg, 2009, vol. 5838, pp. 163–178.

[3] N. Pohlmann, H. Reimer, W. Schneider, E. Trichina, K. Hyppönen, and M. Hassinen, "Sim-enabled open mobile payment system based on nation-wide pki," in *ISSE/SECURE 2007 Securing Electronic Business Processes*. Vieweg, 2007, pp. 355–366.

[4] M. Hassinen, K. Hyppönen, and K. Haataja, "An open, pki-based mobile payment system," in *Emerging Trends in Information and Communication Security*, ser. Lecture Notes in Computer Science, G. Müller, Ed. Springer Berlin / Heidelberg, 2006, vol. 3995, pp. 86–100.

[5] L. Xie, X. Zhang, A. Chaugule, T. Jaeger, and S. Zhu, "Designing system-level defenses against cellphone malware," in *Proceedings of the 2009 28th IEEE International Symposium on Reliable Distributed Systems*. IEEE Computer Society, 2009, pp. 83–90.

[6] Java card technology. Oracle. [Online]. Available: http://www.oracle.com/technetwork/java/javacard/overview/index.html

[7] GlobalPlatform. [Online]. Available: http://www.globalplatform.org/

[8] Secure element evaluation kit for the android platform. seek-for-android. [Online]. Available: http://code.google.com/p/seek-for-android/

[9] J. E. Bresenham, *IBM Systems Journal*. Riverton, NJ, USA: IBM Corp., March 1965, vol. volume 4 issue 1, journal Algorithm for computer control of a digital plotter.

[10] G. Starnberger, L. Froihofer, and K. M. Goeschka, "Using smart cards for tamper-proof timestamps on untrusted clients," Vienna University of Technology, 2010.