Implementation of IMS-IBC Service Authentication Platform

Dingqi Yang@telecom-sudparis.eu
Mohamed Abid@telecom-sudparis.eu
Songbo Song@orange-ftgroup.com

Abstract

This research report presents an evaluation of a service authentication mechanism called IMS-IBC for users on top of IP Multimedia Subsystem (IMS) environment. In this mechanism, the users’ service authentication is based on Identity Based cryptography (IBC). This solution enhances the security level and also gives the opportunity to personalize the service. An evaluation of this mechanism has been conducted. The measurements give the cost of the basic functions involved in the authentication mechanism. The implementation experiments show the feasibility and the advantage of this mechanism.

Key words

IMS service authentication, Identity Based Cryptography (IBC), Elliptic Curve Cryptography (ECC)
# Table of Contents

1. **INTRODUCTION** .............................................................. - 1 -

2. **STATE OF THE ART** ............................................................... - 2 -
   2.1 Authentication in the IP Multimedia Subsystem (IMS) .................. - 2 -
   2.2 Client-server model ............................................................ - 4 -
   2.3 Identity Based Cryptography (IBC) ........................................ - 4 -
   2.4 Elliptic Curve Cryptography (ECC) ....................................... - 5 -
   2.5 MIRACL ............................................................................ - 7 -
   2.6 IMS-IBC ............................................................................ - 7 -

3. **IMPLEMENTATION OF THE PLATFORM** .................................... - 9 -
   3.1 Introduction ........................................................................ - 9 -
   3.2 API and process in C/C++ .................................................... - 10 -
      3.2.1 Socket ........................................................................ - 10 -
      3.2.2 Thread ......................................................................... - 10 -
      3.2.3 Mutex .......................................................................... - 10 -
   3.3 Design model ....................................................................... - 11 -
   3.4 The parameters of ECC .......................................................... - 12 -

4. **RESULTS AND DISCUSSIONS** ................................................ - 13 -
   4.1 Results ................................................................................ - 13 -
      4.1.1 The UE authenticates for the first time ................................. - 13 -
      4.1.2 No authentication required ................................................ - 15 -
   4.2 Discussion ........................................................................... - 16 -
      4.2.1 Execution time ............................................................... - 16 -
      4.2.1 Analysis ........................................................................ - 16 -

5. **CONCLUSION** ........................................................................ - 17 -

6. **REFERENCES** ......................................................................... - 18 -

APPENDIX .................................................................................. - 19 -
# Table of Illustrations

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>IMS architecture</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Generic Bootstrapping Architecture</td>
<td>3</td>
</tr>
<tr>
<td>Figure 3</td>
<td>IBE in e-mail scenario</td>
<td>5</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Elliptic Curve Addition</td>
<td>6</td>
</tr>
<tr>
<td>Figure 5</td>
<td>IMS-IBC Service Authentication</td>
<td>7</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Elliptic Curve parameters generation</td>
<td>13</td>
</tr>
<tr>
<td>Figure 7</td>
<td>The UE and the NAF in the first authentication</td>
<td>14</td>
</tr>
<tr>
<td>Figure 8</td>
<td>The BSF and the HSS in the first authentication</td>
<td>15</td>
</tr>
<tr>
<td>Figure 9</td>
<td>The UE and the NAF when no authentication required</td>
<td>15</td>
</tr>
<tr>
<td>Figure 10</td>
<td>The BSF and the HSS when no authentication required</td>
<td>16</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

IP Multimedia Subsystem (IMS) is an international, recognized standard which has been developed by the Third Generation Partnership Project (3GPP) and 3GPP2. IMS is a set of specifications that describes the Next Generation Networking (NGN) architecture for implementing IP based telephony and multimedia services. It is an overlay architecture for the provision of multimedia services (such as Voice over IP (VoIP), video conferencing, presence, push-to-talk etc.) on top of all IP networks as well as NGN (Next Generation Networks). IMS is thus promising in the future technology for the convergence of data, speech and mobile networks, thanks to providing easy and efficient ways to integrate different value added services and seamless integration of legacy services.

IP-based multimedia communications services must be safe for people to use (free from malware or malicious attacks) whether through their mobile or fixed terminals. Users will also want reassurance that others cannot gain unauthorized access to their personal services and information. In the security aspect, IMS provides a unique architecture for authentication and accounting of different services. Each subscriber uses an ISIM (IMS-SIM) card with a stored secret key to be able to authenticate to the IMS network and to be able to access the IMS services. The following section details the authentication process in IMS. Indeed, IMS authentication proved to have some security limitations such as server spoofing and off-line password guessing attacks. Another requirement provided in IMS, is the tight attachment of the subscriber authentication to the user equipment since the ISIM card is used as substrate and also authentication is restricted to one algorithm: Authentication and key Agreement (AKA). This fact limits the personalized access and hence the service personalization, since each user should have access through his own devices (having his own ISIM card) in order to be correctly identified.

Consequently, IMS authentication falls short in one hand to realize authentication in a personalized manner, which is an important prerequisite in new services such as social internet. The strong dependency on the ISIM card also limits the security performance and stands as an obstacle towards compatibility and evolution (considering that many operators do not support the smart card). On the other hand, using AKA in IMS proved to have some weakness, like short key for cryptographic purposes [1], [2]. Many solutions are proposed to strengthen IMS security like in [3] where the authors define a new AKA based on Elliptic Curve Cryptography (ECC) and [4] where the authors define a new AKA called one pass AKA for UMTS. Furthermore, Ring et al [5] tried to design a new AKA mechanism for SIP using IBC (Identity Based Cryptography) [6]. However, these works were only done for the subscriber authentication. And there are also some solutions proposed to enhance the service authentication like in [7] where Abid and his collaborators define a new method of service authentication in IMS using the Identity Based Cryptography (IBC) on Elliptic Curve Cryptography (ECC). Our research work is to test this solution and analyze its performance.
The rest of this research report is organized as follow. Section 2 is the state of the art. Section 3 gives the methodology and design model. In the section 4, we present the results and analysis. Finally, we give the conclusion in the section 5.

2. **STATE OF THE ART**

2.1 **Authentication in the IP Multimedia Subsystem (IMS)**

The IP Multimedia Subsystem (IMS) is an access-independent and IP connectivity based service control architecture that enables various types of multimedia services to end-users using common Internet-based protocols [8]. It was originally designed by the wireless standards body 3rd Generation Partnership Project (3GPP) [9] and was later extended by TISPAN as a subsystem of NGN [10]. IMS supports IP Multimedia applications such as video, audio and multimedia conferences and SIP was chosen as the signaling protocol for creating and terminating Multimedia sessions. The security of IMS services, authentication, authorization protocols and encryption/decryption procedures, have been defined and implemented [11-13].

Because IMS uses Session Initiation Protocol (SIP) for the control and signaling of sessions, its main architectural elements are SIP proxies, known as Call Service Control Functions (CSCF). Figure 1 illustrates the main entities constituting the IMS core. The CSCFs handle all the SIP session signaling and are divided into: P(Proxies)-CSCF, I(Interrogating)-CSCF and S(Serving)-CSCF.

1. P-CSCF is used as IMS contact points for end users within IMS.
2. I-CSCF is the contact point within the operator’s network and forwards connections to the appropriate destination.
3. S-CSCF is considered as the focal entity of the IMS since it is responsible for users’ authentication, registration and authorization, and also for managing the application servers (AS).

The Home Subscribe System (HSS) is another important entity in IMS which is a database for all subscribers and service-related data of the IMS. The main data stored in the HSS includes user identities, registration information, etc. During registration of a user, firstly the P-CSCF contacts the I-CSCF for acquiring the address of the S-CSCF corresponding to the user. The I-CSCF in turn contacts the HSS to assign an appropriate S-CSCF and forwards the registration request to it. When the S-CSCF receives the registration request, it downloads the user’s authentication data from the HSS and based on such authentication data, it generates a challenge to be sent to the user/UE (User Equipment).
Parallel to SIP traffic, a number of services might be accessed over HTTP. In order to allow the access to services over HTTP in a secure manner, IMS uses the Generic Bootstrapping Architecture (GBA) which consists of four entities: User Equipment (UE), Network Application Function (NAF), Bootstrapping Server Function (BSF) and Home Subscriber Server (HSS). The GBA performs authentication between the BSF and the UE, which is also based on AKA (AKA is a challenge-response based mechanism that uses symmetric cryptography).

As illustrated in the figure 2, the authentication starts with the UE that sends a message to the NAF for asking if the authentication is necessary. If the UE has already accessed to the service and the session hasn’t expired yet, the NAF replies the UE, the authentication is not necessary. Otherwise, the UE sends the challenge to the BSF to start the verification of the user’s identity. The BSF communicates with the HSS to obtain the parameters used for the authentication. After have the parameters, the BSF and the UE authenticate each other. If the authentication succeeds, the BSF will give a generated key to the UE and the NAF so that they can communicate in a secure way. The security protocol AKA is used in service authentication in IMS.
2.2 Client-server model

Client-server computing or networking is a distributed application architecture that partitions tasks or workloads between service providers (servers) and service requesters, called clients. The client-server model describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services.

Often clients and servers operate over a computer network on separate hardware. A server machine is a high-performance host that is running one or more server programs which share its resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await (listen to) incoming requests. In our case, we are going to simulate the client-server model on the same machine.

2.3 Identity Based Cryptography (IBC)

Identity-Based Cryptography (IBC) [6] is an important primitive of ID-based cryptography. As such, it is a type of public-key encryption system in which the public key of a user is some unique information about the identity of the user (e.g. a user's email address, the physical IP address, etc).

Identity-based systems allow any party to generate a public key from a known identity value such as an ASCII string. IBC cannot have any usage if not linked to a well known non-repudiable entity. A domain name and an IP address are examples of such a relation between public information and its non-repudiable domain. This domain is also known as Private Key Generator (PKG). To operate, the PKG first publishes a master public key, and retains the corresponding master private key (referred to as master key). Given the master public key, any party can compute a public key corresponding to the identity ID by combining the master public key with the identity value. To obtain a corresponding private key, the party authorized to use the identity ID contacts the PKG, which uses the master private key to generate the private key for identity ID.

In figure 3, we present an example of an e-mail scenario, where Alice sends mail to Bob at bob@b.com; she simply encrypts her message using the public key of Bob such as the string bob@b.com. There is no need for Alice to obtain Bob's public key certificate. When Bob receives the encrypted mail, he contacts a third party, which is called the Private Key Generator (PKG). Bob authenticates himself to the PKG in the same way he would authenticate himself to a Certificate Authority (CA) and obtain his private key from the PKG. Then, Bob can read the encrypted mail.
2.4 Elliptic Curve Cryptography (ECC)

Elliptic curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. As it is known, public-key cryptography is based on the intractability of certain mathematical problems. For elliptic-curve-based protocols, it is assumed that finding the discrete logarithm of an elliptic curve element is infeasible. The size of the elliptic curve determines the difficulty of the problem. It is believed that shorter keys, generated from elliptic curves, can be used to obtain the same level of security as RSA-based systems. Using shorter keys reduce storage and transmission requirements.

An elliptic curve over Galois field GF (p) is a plane curve which consists of the points satisfying the equation:

$$y^2 = x^3 + ax + b \mod p$$

If $x^3 + ax + b$ contains no repeated factors, or equivalently if $4a^3 + 27b^2$ is not 0, then the elliptic curve $y^2 = x^3 + ax + b$ can be used to form a finite group of points. An elliptic curve group over real numbers consists of the points on the corresponding elliptic curve, together with a special point O called the point at infinity.

The operation of the calculation over elliptic curve is a geometric approach. For example, $P + Q = R$ is the additive property defined geometrically. We have two points $P$ and $Q$ on the elliptic curve. To add the points $P$ and $Q$, a line is drawn through the two points and there’ll be two cases:

1. If $PQ$ intersects the elliptic curve in one more point, we have the point $P + Q$ which is reflected in the x-axis to this point, as illustrated in the figure 4.
2. If $PQ$ is a parallel with the y-axis, it does not intersect the elliptic curve at a third point. We have $P + Q = O$ (point at infinity).
The other case is, if \( P = Q \), we take the tangent line at the point \( P \). We will use this rule to calculate the \( k \cdot P \). Consequently, we have the algebra expression for the addition of two points (the result is \( R (x_3, y_3) \)):

1. The two distinct points \( P (x_1, y_1) \) and \( Q (x_2, y_2) \):

\[
\begin{align*}
x_3 &= \left( \frac{y_2 - y_1}{x_2 - x_1} \right)^2 - x_1 - x_2 \\
y_3 &= -y_1 + \left( \frac{y_2 - y_1}{x_2 - x_1} \right) (x_1 - x_3)
\end{align*}
\]

2. Doubling the same point \( P (x_1, y_1) \):

\[
\begin{align*}
x_3 &= \left( \frac{3x_1^2 + a}{2y_1} \right)^2 - 2x_1 \\
y_3 &= -y_1 + \left( \frac{3x_1^2 + a}{2y_1} \right) (x_1 - x_3)
\end{align*}
\]

To compute \( k \cdot P \), we use the version of square and multiply in Elliptic Curve. The method consists on converting \( k \) to binary base. Then, we run the algorithm in many steps. For each step, if we treat bit 0, we multiply the last result by 2 and if the bit is 1, we multiply the point by 2 and we add \( P \). In the example where \( k = 77 = (1001101)_2 \).

\[
P \rightarrow 2P \rightarrow 4P \rightarrow 9P \rightarrow 19P \rightarrow 38P \rightarrow 77P
\]

\[
\begin{array}{cccccccc}
1 & 0 & 0 & 1 & 1 & 0 & 1
\end{array}
\]
2.5 MIRACL

MIRACL (Multiprecision Integer and Rational Arithmetic C/C++ Library) [14] is a large number library which implements all of the primitives necessary to design large number cryptography into your real-world application. It is primarily a tool for cryptographic system implementers. RSA public key cryptography, Diffie-Hellman Key exchange, DSA digital signature, are all just a few procedures calls away. The latest version offers full support for Elliptic Curve Cryptography over GF (p) and GF (2^m). Although implemented as a C library, a well-thought out C++ wrapper is provided, which greatly simplifies program development. Most example programs are provided in both C and C++ versions.

2.6 IMS-IBC

The proposed IMS-IBC Service Authentication in [7] defined by Abid and his collaborators is illustrated in Figure 5 and is explained in the following steps:

Figure 5: IMS-IBC Service Authentication
Step 1. (messages 1 and 2) The UE starts communication with the NAF without GBA parameters. If the NAF requires the use of shared keys obtained by means of the GBA, but the request from the UE does not include GBA-related parameters, the NAF replies with a bootstrapping initiation message.

Step 2. (messages 3, 4 and 5) The UE sends a HTTP request to the BSF (Bootstrapping Server Function) including the IMS private user identity (IMPI) and public user identity (IMPU). The BSF then retrieves from the HSS:

1. The complete set of GBA user security settings (GUSS),
2. An Authentication Vector (AV) containing the RAND and PKG parameters,
3. The private key $K_{\text{priv}}$ (generated using IMPU) from the PKG. $K_{\text{priv}}$ is encrypted with the shared key $sk$.

Step 3. (message 6) In order to demand the UE to authenticate itself, the BSF forwards AV (RAND and PKG parameters) and the encrypted $K_{\text{priv}}$ to the UE in the “401 (Unauthorized) message”. We add also the public key $k_{\text{pub}}$. We choose to send PKG parameters to the UE as HSS periodically change his parameters.

Step 4. (message 7) The UE extracts its private key $K_{\text{priv}}$ using the shared key $sk$ which is stored in the ISIM card. Then, the UE hashes the RAND, generates a signature using Elliptic Curve Digital Signature Algorithm (ECDSA), where the key generation is carried out as follows:

- The UE already has the PKG parameters, $K_{\text{pub}}$ and $K_{\text{priv}}$. The number of points of the elliptic curve should be divisible by a large prime $n$.
- The UE computes an integer $d = k_{\text{priv}} \mod (n-2)$ and computes $Q = d \cdot K_{\text{pub}}$. Then, UE’s public parameters are the quadruple $(E, K_{\text{pub}}, n, Q)$ and UE’s private key is $d$.
- The UE generates the ECDSA signature when signing the value RAND as follows:
  1. Select a statistically unique and unpredictable integer $k$ in the interval $[1, n-1]$.
  2. Compute $k^* K_{\text{pub}} = (x_1, y_1)$ and $r = x_1 \mod n$. If $r = 0$, then go to 1. (This is a security condition: if $r = 0$, then the signing equation: $s = k^{-1}(h(RAND) + d \cdot r) \mod n$ does not involve the private key $d$).
  3. Compute $k^{-1} \mod n$.
  4. Compute $s = k^{-1}(h(RAND) + d \cdot r) \mod n$ where $h$ is the Secure Hash Algorithm (SHA-1). If $s = 0$, then go to 1. (If $s = 0$, then $s^{-1} \mod n$ does not exist; $s^{-1}$ is required in iteration 2 of the signature verification.)
  5. The signature for the value RAND is the pair of integers $(r, s)$.
- The UE sends $\text{Sig} (RAND) = (r, s), n$ and $Q$ to the BSF in an HTTP request in order to authenticate itself.
- To verify the UE’s signature $(r, s)$ on RAND, the BSF should do the following:
  1. The BSF already has the PKG parameters and receives $n$ and $Q$ from the UE. It verifies that $r$ and $s$ are integers in the interval $[1, n-1]$.
  2. Compute $w = s^{-1} \mod n$ and $h(RAND)$. 

- 8 -
3. Compute $u_1 = h(\text{RAND}) \cdot w \mod n$ and $u_2 = r \cdot w \mod n$.

4. Compute $u_1 \cdot k_{\text{pub}} + u_2 \cdot Q = (x_0, y_0)$ and $v = x_0 \mod n$.

5. Accept the signature if and only if $v = r$.

If the verification phase is successful, then the user is authenticated.

**Step 5. (message 8)** After the successful verification, the BSF generates B-TID (Bootstrapping ID) and stores it with the IMPU and GUSS. The BSF then sends to the UE a "200 OK message" including the B-TID encrypted with UE's public key $k_{\text{pub}}$. After receiving the message, the UE retrieves the B-TID using $k_{\text{priv}}$. In our solution, there is no key material $K_s$ stored in the UE and the BSF. Our system is based on asymmetric cryptography. The shared key $K_s$ between the UE and the HSS is used to encrypt the UE's $k_{\text{priv}}$. The BSF cannot retrieve this key and has to encrypt BTID using UE's $k_{\text{pub}}$.

**Step 6. (message 9)** The following steps apply Elliptic Curve Diffie-Hellman (ECDH) Protocol. This key agreement protocol will be used to generate the $K_s$-NAF key. The UE and the NAF first have to agree whether to use the shared keys obtained by means of the GBA. The UE chooses a random value 'a' to generate ‘$a \cdot K_{\text{pub}}$’ and provide the IMPU, B-TID, ‘$a \cdot K_{\text{pub}}$’, and a signature of B-TID to the NAF to allow it to retrieve the corresponding keys from the BSF.

**Step 7. (message 10)** The NAF sends the IMPU, the NAF-ID and the signature value to the BSF to request for GUSS and PKG parameters. NAF-ID is used by the BSF to verify that the NAF is authorized to use that hostname.

**Step 8. (message 11)** First of all, the BSF verifies the signature using $K_{\text{pub}}$. Then, it retrieves the GUSS and PKG parameters using B-TID and IMPU. Finally, it supplies to the NAF the GUSS, IMPU and PKG parameters.

**Step 9. (message 12)** The NAF checks the authentication and the authorization of the IMPU to the services according to the received GUSS and then generates a random value ‘b’ and send to the UE ‘$b \cdot K_{\text{pub}}$’. After receiving the message, the UE and the NAF will have the same $K_s$-NAF = $a \cdot b \cdot K_{\text{pub}}$. Once the execution of the protocol is completed, the UE and the NAF will communicate in a secure way.

We are going to explain in the following section the detailed implementation of this system.

### 3. IMPLEMENTATION OF THE PLATFORM

#### 3.1 Introduction

According to the schema of IMS-IBC, we start from the 4 principal entities: the UE, the NAF, the BSF and the HSS. Each of them should be an independent sub-system, and they exchange the message between each other to accomplish the authentication process. Generally, we know that the UE works as a client and it should execute once in a process of authentication. The other entities such as the NAF and the BSF can be client or server. And the HSS is a demon server which waits for request form the BSF. However, when a challenge arrives, the servers will create a
sub-process called thread that works specially for this challenge and the main process should wait for the next challenge. Here, in order to build a client-server model, we use “socket” in C++. The other point is that we have to synchronize some messages which are exchanged between them. For this, we use the notion of “Mutex”. The mutex are also used to synchronize the threads.

Moreover, the work should be divided into the sub-functions used by the 4 entities. We will call these sub-functions in the principal functions when they are needed.

3.2 API and process in C/C++

3.2.1 Socket

We use the sockets to establish the communication between the client and the server.

To establish a socket on the client side, we need to follow these steps:

1. Create a socket with the socket() system call
2. Connect the socket to the address of the server using the connect() system call
3. Send and receive data. There are a number of ways to do this, but the simplest is to use the read() and write() system calls.

To establish a socket on the server side, we need to follow these steps:

1. Create a socket with the socket() system call
2. Bind the socket to an address using the bind() system call. For a server socket on the internet, an address consists of a port number on the host machine.
3. Listen for connections with the listen() system call
4. Accept a connection with the accept() system call. This call typically blocks until a client connects with the server.
5. Send and receive data

3.2.2 Thread

A thread results from a fork of a computer program into two or more concurrently running tasks. Multiple threads can exist within the same process and share resources such as memory, while different processes do not share these resources. In order to synchronize the different threads in a same process, we need the Mutex.

3.2.3 Mutex

A mutex is a program object that allows multiple programs or threads to share the same resource, such as file access, but not simultaneously. When a program is started, a mutex is created with a unique name. After this stage, any thread that needs the resource must lock the mutex from other threads while it is using the
resource. The mutex is set to unlock when the data is no longer needed or the routine is finished.

### 3.3 Design model

After giving the API process in the previous paragraph, we should notice that:

- the UE can be a client of the BSF and the NAF;
- the NAF can be a client of the BSF and a server for the UE;
- the BSF can be a client of the HSS and a server for the NAF and the UE;
- the HSS can only be a server for the BSF.

Now, we are going to analyze the sub-functions used in each principal program.

1. **aes_enc and aes_dec** : The functions for encrypting and decrypting a message by using Advanced Encryption Standard (AES). They are used when UE and HSS exchange the private key.

2. **ecsign and ecsver** : The functions for generating and verifying a signature of a message. We use the algorithm of Elliptic Curve Digital Signature Algorithm (ECDSA).

3. **ibe_ext** : IBC function for generating the public key and the private key using the predefined parameters of ECC. It will be used in the HSS for extracting the private key and the public key.

4. **menezes_enc a menezes_dec** : The functions for encrypting and decrypting a message by using the method of Menezes-Vanstone in ECC. They are used when the UE and the BSF exchange the B_TID.

5. **multi_a** : The function for calculating the product of a random number a and the point \( k_{pub} \) in ECC.

6. **multi_b** : The function for calculating the product of a predefined number b and a predefined point P in ECC.

7. **messagetohtml and htmltomessage** : The functions for translating the message in text to the form HTML and translating the message in HTML to the form of text. (The codes of these two functions are given by Orange lab)

Besides of the above functions, we have also some necessary files used by these functions.

1. **aes_key.aes** : The file that save the key of AES.

2. **final1.ibe et master1.ibe** : The parameters of ECC are in final1.ibe and the master key of PKG is in master1.ibe. (We have also another set of parameters and master key of PKG as final2.ibe and master2.ibe).

3. **identite.txt** : The file that save the identity information of the UE (the UE, IMPU and IMPI).

4. **naf_id.txt** : The file that save the identity of the NAF.
5. *users.txt*: The database of users in the HSS.
6. *users_temp.txt*: The database of the users who have already authenticated and whose sessions have not expired yet in the NAF.

After the previous analysis, we present, in a table, the relationship between the principal functions and the sub-functions.

<table>
<thead>
<tr>
<th>Principal function</th>
<th>Sub-function</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>The UE</td>
<td>aes_dec</td>
<td>identite.txt</td>
</tr>
<tr>
<td></td>
<td>ecsign</td>
<td>aes_key.aes</td>
</tr>
<tr>
<td></td>
<td>menezes_dec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>multi_a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>multi_b</td>
<td></td>
</tr>
<tr>
<td></td>
<td>message_to_html</td>
<td></td>
</tr>
<tr>
<td></td>
<td>html_to_message</td>
<td></td>
</tr>
<tr>
<td>The NAF</td>
<td>multi_a</td>
<td>naf_id.txt</td>
</tr>
<tr>
<td></td>
<td>multi_b</td>
<td>users_temp.txt</td>
</tr>
<tr>
<td></td>
<td>message_to_html</td>
<td></td>
</tr>
<tr>
<td></td>
<td>html_to_message</td>
<td></td>
</tr>
<tr>
<td>The BSF</td>
<td>ecsver</td>
<td></td>
</tr>
<tr>
<td></td>
<td>menezes_enc</td>
<td></td>
</tr>
<tr>
<td></td>
<td>message_to_html</td>
<td></td>
</tr>
<tr>
<td></td>
<td>html_to_message</td>
<td></td>
</tr>
<tr>
<td>The HSS</td>
<td>aes_enc</td>
<td>aes_key.aes</td>
</tr>
<tr>
<td></td>
<td>ibe_ext</td>
<td>final1.ibe</td>
</tr>
<tr>
<td></td>
<td></td>
<td>master1.ibe</td>
</tr>
<tr>
<td></td>
<td></td>
<td>users.txt</td>
</tr>
</tbody>
</table>

These are also some temporal files used by some functions that we haven’t mention in the table. For example, in BSF for each client we will create a private file that includes IMPU, IMPI, B-TID and GUSS.

### 3.4 The parameters of ECC

Before the implementation, we have to choose the appropriate parameters of the elliptic curve. As we have mentioned in chapter 2, the equation of the elliptic curve is $y^2 = x^3 + ax + b \mod p$. The points of this curve will define a finite field; their number must be a prime number.

In order to satisfy this condition, we fixed a prime number $p$ and the parameter $a$ in elliptic curve. And then we choose the parameter $b$ in elliptic curve that satisfy this condition. We used a function in MIRACL that can calculate the number of the points in a finite field. The principle of the algorithm is shown in the figure 6:
As shown in the schema, we begin with choosing a and p. And then we give an initial number to b and calculate n (the number of the point) on the elliptic curve. If n is a prime number, we have the proper result. If not, we increase b by 1 and calculate n again. We repeat this step until we find a prime number for n.

4. RESULTS AND DISCUSSIONS

In this section, we present the results obtained after running the platform. We discuss also these results to check if we obtained the desired performances.

4.1 Results

In this section, we give the results of the simulation of the platform. We can see clearly that the exchanging of messages among the 4 entities and the output of the sub-functions. We just show some screen savers of the sub-functions. We choose 2 scenarios:

4.1.1 The UE authenticates for the first time

We present the screenshots of our platform in the case of an UE who authenticate for the first time. They contain the exchanges of every message.

4.1.1.1 The UE and the NAF

In the figure 7, the UE and the NAF finish the authentication process. All the steps in the authentication have been executed successfully.
Figure 7: The UE and the NAF in the first authentication

4.1.1.2 The BSF and the HSS
This scenario is for the first authentication of UE. In the program, we defined the SN (session number), MN (message number), sender and receiver. In figure 8, we present the sending and receiving threads by displaying the messages’ information in each sender and receiver.

Furthermore, we can observe that when the authentication finishes, the program UE also ends. But the NAF, the BSF and the HSS were still working as the servers waiting for the next user.

4.1.2 No authentication required

In this case, we suppose that the UE has already been authenticated.

4.1.2.1 The UE and the NAF

If the UE was already authenticated and the session didn’t expire yet, we are in the case where no authentication is required. In figure 9, when the UE sends the first message, the NAF checks that the UE is already in his temporal database and the authentication was not necessary. Consequently, the NAF would send the message to the UE.

And we noticed that the SN (session number) changed with every request so that we can distinguish the two processes of authentication.

4.1.2.2 BSF and HSS

As we have said, in the case where no authentication is required, the BSF and the HSS should do nothing. In figure 10, we can see that they didn’t receive any requests.
4.2 Discussion

In this section, we analyze the previous results from a performance point of view.

4.2.1 Execution time

In order to evaluate the performance of our solution, we tried to observe the execution time for an UE who wants to be authenticated for the first time.

The result we got is around 250 ms using a computer machine with theses configurations (Intel Centrino Duo P8700 2.53GHz, memory 2G in Linux Fedora12).

Moreover, we observed the execution time of some sub-functions, like:

1. Time needed to generate the PKG parameters (ibe_ext) for a client is around 26.3 ms.
2. Time needed to generate and verify a digital signature (ecsign and escver) are nearly 9.3 ms and 10.4 ms.
3. Time needed to encrypt and decrypt the private key (aes_enc and aes_dec) are around 5 ms and 5.1 ms.
4. Time needed to encrypt and decrypt the B_TID (menezes_enc and menezes_dec) are around 8.1 ms and 8.2 ms.
5. Time to calculate the product of a random number and a given point (multi_a) is nearly 4.8 ms.
6. Time to calculate the product of a given number and a given point (multi_b) is nearly 5.8 ms.
7. Converting the message between text and html needs nearly 2.7 ms.

Compared to the execution time for cryptography, the total execution time is much bigger. We can say that the most time-consuming part of our program is not in the cryptography. The exchange of messages and access to the file system (read and write a file) take a lot of time.

4.2.1 Analysis
However, there are still some points that we can change to enhance the execution time. For example, we can use the method that compiles each entity in only one executable file so that the program doesn’t need to call another binary executable. Another possibility to reduce the time is to improve the algorithm ibe_ext which is used to generate the private key and the public key, because it is the most time-consuming program in the whole program. To enhance the performance time, we can use the database in spite of simple file because the management of database is easier and faster.

5. CONCLUSION

In this research report, we presented the implementation of the platform of the IMS-IBC method which is a new IMS service’s authentication. We gave the general idea of the conception of the program structure, and the detail functionalities. Furthermore, we showed the result we have got and gave some idea to enhance the QOS (Quality of service) of this platform by reducing the time of the service’s authentication.

This platform could also be used as the open source for other program of IMS authentication.

Like a future work, we want to test our platform with many clients at the same time and we need also to compare IMS-IBC with the IMS-AKA method.
6. REFERENCES


[9] 3rd Generation Partnership Project (3GPP) http://www.3gpp.org/


APPENDIX

A. Implementation of the principal functions

The 4 principal functions are almost the same. We are not going to talk about the detail. We just talk about some difficulties and the solutions.

Here we take UE as an example. We know that UE must communicate with NAF and BSF, so we create two processes for UE to communicate with the other entities. But a question must be mentioned here is that we have to synchronize the message in two threads. Otherwise the messages will be sent out of order. The solution is Mutex. We defined a global variable in Mutex. When one thread should be stop for a while, we stop it and change the global variable in Mutex. Now we can start the new thread. If we want to return to the former one, we just stop the later one and release the global variable in Mutex. The codes examples (C++) in our case for UE are as follow:

We launch two processes NAF and BSF in UE.

```c
pthread_create(&th_id1,NULL,interact_NAF,(void *)&NAF_Client);
pthread_create(&th_id2,NULL,interact_BSF,(void *)&BSF_Client);
```

The variable global.

```c
int start_NAF_2 = 0;
int start_BSF_1 = 0;
#define NAF_2 1
#define BSF_1 1
int error2 = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

Here the start_NAF_2 and start_BSF_1 are for controlling the messages’ synchronization. The error2 is for error detection.

Lock the process BSF.

```c
pthread_mutex_lock(&mutex); // debut de la section critique
while (start_BSF_1 != BSF_1)
{
    pthread_cond_wait(&cond,&mutex);
}
pthread_mutex_unlock(&mutex); // fin de la section critique
```

Here start_BSF_1 != BSF_1 is the condition that we lock the process BSF. When start_BSF_1 = BSF_1, we restart the process.

Lock NAF and release BSF.

```c
pthread_mutex_lock(&mutex); // debut de la section critique
while (start_NAF_2 != NAF_2)
{
    start_BSF_1=BSF_1;
    pthread_cond_broadcast(&cond);
```
Here \( \text{start}\_\text{NAF} \_2 \neq \text{NAF} \_2 \) is the condition that we lock the process NAF and \( \text{start}\_\text{BSF} \_1 = \text{BSF} \_1 \) is to restart the process BSF.

### B. Implementation of the sub-functions

1. **aes_enc and aes_dec**

   The Advanced Encryption Standard (AES) is an encryption standard adopted by the U.S. government. We can find a lot of open source for this algorithm on internet. Here we find it out in the library MIRACL and we change the input and output to adapt it to our case. We use it to encrypt and decrypt the private key. So it encrypts the private.ibe to private.aes using the key stocked in aes\_key.aes. And the decryption is reversed.

2. **ecsign and ecsver**

   These two functions use the algorithm ECDSA (Elliptic Curve Digital Signature Algorithm). As we know the principle of ECDSA, we can implement them with the library MIRACL.

   To generate a signature of a message \( m \), we compute \( Q = d \ast \text{Kpub} \) (UE’s private key is \( d \)). Then, UE’s public parameters are the quadruple \( (E, \text{Kpub}, n, Q) \).

   1. We choose a random number \( k \) between 1 and \( n-1 \).
   2. We calculate \( k \ast \text{Kpub} = (x_1, y_1) \), and then \( r=x_1(\text{mod } n) \). If \( r=0 \) we go to step 1.
   3. We calculate \( k^{-1} \text{mod } n \).
   4. We calculate \( s=k^{-1}(h(m)+d*r) \) where \( h(m) \) is the result of a hash function \( \text{SHA}-1 \) (Secure Hash Algorithm) that hash the message \( m \). Here if \( s=0 \), we go to the step 1.
   5. The signature of the message \( m \) is a couple of number \( (r, s) \).

   To verify a signature of a message \( m \):

   1. We verify if \( r \) and \( s \) are in the interval \([1, n-1]\), otherwise the signature is invalid.
   2. We calculate \( w=s^{-1}(\text{mod } n) \).
   3. We calculate \( u_1=h(m)*w(\text{mod } n) \) and \( u_2=r*w(\text{mod } n) \).
   4. We calculate \( u_1 \ast \text{Kpub} + u_2 \ast Q = (x_2, y_2) \) and then \( v=x_2(\text{mod } n) \).
   5. The signature of the message \( m \) is verified if and only if \( r=v \).

   In our case the message is in the file ecdsa.txt and the signature is in the file ecdsa.ecs. And it also uses public.ibe, private.ibe and final1.ibe.

3. **ibe_ext**
This is the function that generates the public key and the private key from the IMPU of the UE by using the parameters of ECC and the master key. The original of this function is from MIRACL. But in order to adapt it to all the functions use on ECC, we change some functions of PKG in that. It output the private.ibe and the public.ibe by using final1.ibe, master1.ibe and IMPU.txt.

4 menezes_enc and menezes_dec

This is the function for encrypting and decrypting the massage by using algorithm Menezes-Vanstone. We use it for the B-TID which is in message 8. Before use this algorithm, we should divide the message B-TID into two part y and z.

To encrypt B-TID:

1. Generate a random number k.
2. We calculate:
   a) \(U = k \cdot P\)
   b) \((c,d) = k \cdot Q\) (c and d are the coordinate of the point \(k \cdot Q\) in ECC)
   c) \(v = c \cdot y \mod p\)
   d) \(w = d \cdot z \mod p\)
3. The encrypted message is \((U, v, w)\).

To decrypt B-TID:

1. \(D \cdot U = d \cdot (k \cdot P) = k \cdot (s \cdot P) = k \cdot Q = (c, d)\) (it calculate c and d)
2. \(c^{-1} \cdot v \mod p = c^{-1} \cdot c \cdot y \mod p = y\)
3. \(d^{-1} \cdot w \mod p = d^{-1} \cdot d \cdot z \mod p = z\)

As we have y and z here, we have the message B-TID.

This function output B_TID.ibe by using final1.ibe, public.ibe, private.ibe and B_TID.txt.

5 multi_a and multi_b

As we have mentioned in the above section, the multi_a is the function for calculating the product of a random number a and the point kpub in ECC. And the multi_b is the function for calculating the product of a predefined number b and a predefined point P in ECC. We can implement them easily with library MIRACL, so we will not go further on this issue.

The multi_a output multi_a.txt by using final1.ibe and public.ibe. The multi_b output multi_b.txt by using final1.ibe, point.txt and nombre.txt.

6 messagetohtml and htmltomessage

These are the converting functions that change the form of message between the form text and html. They are given by Orange Labs.

C. Integration
Since we have chosen this structure of the program, the integration of the system is not difficult. We compile the sub-functions and we call them in the principal programs when we need. Generally this way can simplify the debug. We can test each function and make sure there is no problem with it. But in our case a problem come out in the step of integration.

As we have mentioned in the section above, the functions aes_enc and aes_dec are for the encryption and the decryption of the private key. They ensure that the distribution of the private key is in a secure way. When we tested them separately, the two processes worked very well. But after integration, the decryption didn't work well any more. So we started to analyze every step between the encryption and decryption. We found out that the problem was that the encrypted private key was the garbled codes. If we treated the garbled codes like a chain of characters in the scenario of exchanging message among the entities, a problem turned out. There were ‘/0’, NULL or ‘/n’ in the garbled codes which were the special codes in the meaning of the characters. So if we operated them, it would give a part of the garbled codes instead of the whole encrypted private key. So when the decryption program tried to decrypt it, it would give a worry result.

The solution we found out was to use the conception of the framework. That meant we converted the ‘sensible’ characters like ‘/0’, NULL or ‘/n’ to a chain of characters normal so that the exchanging of message wouldn’t have a problem. In the receptor, we just tested this chain of characters and we convert them to the characters like ‘/0’ or ‘/n’ if there are. The codes of this scenario are as follows:

1. The converting program in HSS after encryption:

   ```c
   while(i<len)
   {
     h=fgetc(f);
     if(h=='\n')
     {
       mes5->kpriv_sk[i++]='d';
       mes5->kpriv_sk[i++]='e';
       mes5->kpriv_sk[i]='f';
       len=len+2;
     }
     else if(h=='\0')
     {
       mes5->kpriv_sk[i++]='a';
       mes5->kpriv_sk[i++]='b';
       mes5->kpriv_sk[i]='c';
       len=len+2;
     }
     else mes5->kpriv_sk[i]=h;
     i++;
   }
   ```

7. The reverse converting program in UE before decryption:

   ```c
   while(mes6->kpriv_sk[i]!='\0')
   {
     if(mes6->kpriv_sk[i]=='d')
     {
   ```
i++;
if((mes6->kpriv_sk[i]=='\0')&&(mes6->kpriv_sk[i]=='e'))
{
    i++;
    if((mes6->kpriv_sk[i]=='\0')&&(mes6->kpriv_sk[i]=='f'))
    {
        fprintf(f,"%c",'\n');
    }
    else {
        fprintf(f,"de");
        i=i-1;
    }
}
else {
    fprintf(f,"d");
    i=i-1;
}
}
else if(mes6->kpriv_sk[i]=='a')
{
    i++;
    if((mes6->kpriv_sk[i]=='\0')&&(mes6->kpriv_sk[i]=='b'))
    {
        i++;
        if((mes6->kpriv_sk[i]=='\0')&&(mes6->kpriv_sk[i]=='c'))
        {
            fprintf(f,"%c",NULL);
        }
        else {
            fprintf(f,"ab");
            i=i-1;
        }
    }
    else {
        fprintf(f,"a");
        i=i-1;
    }
}
else fprintf(f,"%c",mes6->kpriv_sk[i]);
i++;
### Rapports de recherche parus récemment au sein du département (Réseaux et Services Multimédia Mobiles)

| RR 10 003 RS2M | Implementation of IMS-IBC Service Authentication Platform  
D. YANG, M. ABID, S. SONG |
| RR 10 001 RS2M | Quantifying Spatiotemporal Stability by means of Entropy : Motivations & Approach Major Specificities  
M.H. ZAYANI, V. GAUTHIER, D. ZEGHLACHE |
| RR 09 012 RS2M | Delay and capacity analysis of structured peer-to-peer networks for IP telephony  
J. GHIMIRE, M. MANI, T. SANGUAKOTCHAKORN, N. CRESPI |
| RR 09 011 RS2M | A Novel node connectivity index for wireless adhoc networks  
J. GHIMIRE, M. MANI, N. CRESPI, T. SANGUAKOTCHAKORN |
| RR 09 006 RS2M | A Trade-off between Identity-Based and Certificateless Cryptography (TIBC) for Publish/Subscribe systems  
A. AHMAD, A. BIRI, H. AFIFI |
| RR 09 002 RS2M | Optimization of File Allocation for Video Sharing Servers  
E. ABD ELRAHMAN, W. BENAMEUR, H. AFIFI |
| RR 08 007 RS2M | Resource matching, selection and binding for virtual network embedding  
I. HOUIDI, W. LOUATI, W. BEN-AMEUR, D. ZEGHLACHE |
| RR 08 001 RS2M | Efficient physical layer security for LDR Sensor Networks  
A. AHMAD, H. AFIFI, H. CRUICKSHANK, S. MIRZADEH |
| RR 07 010 RS2M | Autonomic Virtual Routers System  
W LOUATI, I. HOUIDI, D. ZEGHLACHE |
| RR 07 009 RS2M | A Distributed and Autonomic Virtual Network Mapping Framework  
I. HOUIDI, W LOUATI, D. ZEGHLACHE |
| RR 07 007 RS2M | Securing BGP  
A. BIRI |
| RR 07 005 RS2M | Etat de l’art sur les systèmes Crypto Biométriques  
M. ABID, H. AFIFI |
| RR 07 004 RS2M | A Management Framework for Personal Networks using P2P- based Publish/Subscribe Service Naming Systems  
W. LOUATI, D. ZEGHLACHE |
| RR 06 010 RS2M | Expérimentations et programmations des réseaux de capteurs IEEE 802.15.4.  
A. DE SANTIS, H. AFIFI, B. MAINAUD |
| RR 06 008 RS2M | SEPSA : A Scalable and Efficient P2P- based service Discovery Architecture  
W. LOUATI, D. ZEGHLACHE |