TAMPERE UNIVERSITY OF TECHNOLOGY

Caroline Hay
Gerti Dhima

# Implementing a secured IMS-based Identity exchange

Master of Science Thesis

Examiner: Professor Jarmo Harju,
Associate Professor Isabelle Augé-Blum
Supervisor: Lic.Tech Seppo Heikkinen
Examiner and topic approved in the
Faculty of Computing and Electrical
Engineering Council
meeting on 8 September 2010

# Abstract

TAMPERE UNIVERSITY OF TECHNOLOGY

**HAY, CAROLINE; DHIMA, GERTI**: Implementing a secured IMS-based Identity exchange
Master of Science Thesis, 89 pages, 13 Appendix pages
September 2010
Major subject: Communications Engineering
Examiner(s): Professor Jarmo Harju, Lic. Tech. Seppo Heikkinen, Associate Professor Isabelle Augé-Blum
Key words: IMS, SIP Signaling, Application Server, HIT, Identity exchange

With the continuous development of telecommunications, networking and the ubiquitous computing the necessity of higher bandwidth and better quality of services is always one of the most important user requirements. In this background, IP Multimedia Subsystem (IMS) is becoming very important for the Next Generation Networking (NGN) and all-Internet Protocol (all-IP) infrastructure. This new tendency provides opportunities for new operators and service providers to enter the market and to be competitive. These developments will generate new challenges related to the user identity assurance. It will be more difficult to rely on the old paradigms of the static operator relationships guaranteeing end-to-end the identity of the users. In this case there is crucial need to find new mechanisms to provide to the end points assurance about the identity of their counterparts.

In this work we implemented a solution that establishes a trust between two end points by taking advantage of IMS in a roaming scenario where the visited access network may not be entirely trustworthy. In essence, this means establishing an identity association so that the parties can have operator provided assurance regarding the used identities. This allows local trust decisions and does not rely on the existence of global Public Key Infrastructure (PKI).

Concretely in this work we have modified the Session Initiation Protocol (SIP) "INVITE" messages by adding new SIP headers such as the identity and the signature of the SIP entities taking part in a multimedia conversation. Every SIP entity has to add its own identity and signature and also has to verify those of its counterparts in a typical SIP "INVITE" exchange.

By this work we show that establishing this kind of identity association is feasible but some scalability issues have to be taken into account such as the time delay or the size of the new messages. In order to accomplish this master thesis work, we have used the *Open Source IMS Core* (OSIMS) platform developed by *FOKUS*, *SailFin project* as the Application Server (AS) and *IMS Communicator* as the IMS client.

# Preface

This master thesis is part of the research activities of *Networks and Protocols Group;* one of the researching groups in the Department of Communications Engineering at "Tampere University of Technology". The work was accomplished during our academic exchange of spring 2010 at Tampere University of Technology. At the same time this master thesis will be the *End of Study Project* for our home university "Institut National des Sciences Appliquées de Lyon", France.

We would like to thank the people who made this thesis possible. First of all we would like to thank Professor Jarmo Harju and Lic. Tech. Seppo Heikkinen, our supervisors at "Tampere University of Technology", for the subject they found, the guidance and the advices they have provided to us during these months of work.

Then we would like to thank Associate Professor Isabelle Augé-Blum our examiner at "Institut National des Sciences Appliquées de Lyon" for the support on our research.

Tampere, September 2010

**Caroline HAY** **Gerti DHIMA**

# Contents

# Abbreviations and notations

| | |
|---|---|
| ACK | Acknowledgement |
| ADSL | Asymmetric Digital Subscriber Line |
| AKA | Authentication and Key Agreement |
| ALL-IP | All Internet Protocol |
| AN | Application Network |
| AoR | Address of Record |
| API | Application programming interface |
| AS | Application Server |
| BGCs | Breakout Gateway Control Functions |
| B2BUA | Back-to-back user agent |
| CN | Core network |
| CPU | Central Processing Unit |
| CSCF | Call Session Control Function |
| DBMS | Database Management System |
| DNS | Domain Name System |
| EAR | Enterprise archive |
| ETSI | European Telecommunications Standards Institute |
| FHoSS | FOKUS Home Subscriber Server |
| GGSN | Gateway GPRS Support Node |
| GPRS | General Packet Radio Service |
| GSM | Global System for Mobile Communications |
| GUI | Graphical User Interface |
| HIT | Hash Identity Tag |
| HLR | Home Location Register |
| HN | Home Network |
| HSS | Home Subscriber Server |
| IDE | Integrated development environment |
| IETF | Internet Engineering Task Force |
| iFC | Initial Filter Criteria |
| IMS | IP Multimedia Subsystem |
| IP | Internet Protocol |
| ISC | IMS Service Control interface |
| IPsec | Internet Protocol Security |
| ITU | International Telecommunication Union |
| ITU-T | Telecommunication Standardization Sector |
| I-CSCF | Interrogating CSCF |

| | |
|---|---|
| JAIN-SIP | Java APIs for Integrated Networks |
| JDK | Java Development Kit |
| JMF | Java Media Framework |
| JRE | Java Runtime Environment |
| J2DK | Java Development Kit 2 |
| LGPL | GNU Library General Public License |
| LTE | Long Term Evolution |
| MD5 | Message Digest 5 |
| MRF | Media Resources Functions |
| MRFC | Media Resources Function Controllers |
| MRFP | Media Resources Function Processor |
| NDS | Network Domain Security |
| NGN | Next Generation Network |
| OSIMS | The FOKUS Open Source IMS Core |
| PDA | Personal digital assistant |
| PKI | Public Key Infrastructure |
| PSTN | Public Switched Telephone Network |
| P-CSCF | Proxy-CSCF |
| QoS | Quality of Service |
| RAM | Random Access Memory |
| R&D | Research and Development |
| SAR | Sip ARchive |
| SDP | Session Description Protocol |
| SER | SIP Express Router |
| SFC | Secondary Filter Criteria |
| SGW | Signaling Gateway |
| SHIPNET | Service Handling on IP Networks |
| SIP | Session Initiation Protocol |
| SLF | Subscriber Location Function |
| SPT | Service Point Triggers |
| Ssh | Secure Shell |
| SVN | Subversion |
| S-CSCF | Serving CSCF |
| TEL URI | Telephone Uniform Resource Identifier |
| THIG | Topology Hiding Inter-network Gateway |
| TISPAN | Telecoms & Internet converged Services & Protocols for Advanced Networks |
| TP | Trigger Point |
| UA | User Agent |

| | |
|---|---|
| UE | User Equipment |
| UMTS | Universal Mobile Telecommunications System |
| URI | Uniform Resource Identifier |
| USIM | Universal Subscriber Identifier Module |
| VoIP | Voice Over IP |
| VN | Visited Network |
| xDSL | x Digital Subscriber Line (contains DSL, ADSL, HDSL, SDSL, VDSL) |
| xG | Different generations of Mobile telecommunications technology |
| WAR | Web Archive |
| WLAN | Wireless Local Area Network |
| 2G | 2nd Generation: GSM |
| 2.5G | 2.5nd Generation: GPRS |
| 3G | 3 rd Generation: UMTS |
| 3G+ | 3.5 rd Generation: HSDPA |
| 3GPP | 3rd Generation Partnership Project |
| 3GPP2 | 3rd Generation Partnership Project 2 |
| 4G | 4th Generation: LTE |

## Table of Figures

# 1 Introduction

Nowadays, we are at a real turning point in the history of networks. During a long time the different networks used to individually develop on their own, while now is the time for the convergence. This tendency logically arose to follow the users growing requirements. They demand access to any service, anywhere and at any time with the same user experience. As it is becoming possible to interconnect heterogeneous networks, the market is becoming very open and allows almost everybody to provide access services of their own. This leads to the increase of the amount of mini-operators, and affects the development of new ubiquitous networks. In such a diverse environment, the security needs to be revised. Indeed, the operator relationships will not be anymore based on pre-established roaming agreements but they will be more likely to be dynamically generated. In this future context, the security requirements and in particular, the identities of the users and statements are becoming even more critical.

Still, it can be envisaged that users trust their own operators who provide authentication procedures for them through secured infrastructures and billing services. It can be envisaged that in the future networks, the relationships between users and their own operators would remain trusted. In this case, users' identity would be provided by their operator. In order to increase their revenues and to develop their customers' loyalty by providing more profitable multimedia services, the operators invest a lot in new solutions. For example, as the next step of the Third Generation (3G), Ip Multimedia Subsystem (IMS) possibilities are carefully studied.

In this project, we envisage a solution that takes advantage of IMS architecture in a roaming scenario to signal the needed identity parameters between two communication endpoints that rely on their own operators to establish their trust evaluation. In other words, we propose to establish an identity association so that the operators can provide assurance regarding the used identities, in a scenario where the visited access network may not be entirely trustworthy. It is assumed that the users trust their own operators and that the home operators of the users are linked with a roaming agreement. It is also assumed that every entity is in possession of a secure cryptographic identifier. This one enables to justify of one's identity by providing a proof of possession. This solution does not rely on the existence of global Public Key Infrastructure (PKI) and is based on local trust decisions. Thus, in essence, this work proposes to use IMS infrastructure to carry the necessary signaling needed to establish a call between two end points, visiting not entirely trustworthy networks.

This thesis is organized as follows. In the next chapter we will introduce the theoretical background related to this work. Notions such as IMS, Session Initiation Protocol (SIP) and principles of cryptography will be presented and explained in details. We will deal with the security problems of IMS technology and a mechanism to improve it will be presented.

In the third chapter we will present the software environment that we have used and some important tools. We will explain the installation and configuration of the *Open Source IMS Core* and the integration of *SailFin* as an Application Server (AS). We will explain the configuration in the case of one IMS domain and in the case of two IMS domains. This part will be finished by testing and analyzing via *Wireshark* a simple call between two IMS clients. Firstly, the clients are registered at the same domain and in a second time they are registered in two different IMS domains.

The fourth chapter will be dedicated to the implementation of our solution that will be based on the development of an IMS client open source code and then on the development of a SIP application based on the SIP Servlet technology that will be deployed on the SIP application server. We will describe the classes and the methods developed in order to implement the mechanism presented. In the fifth chapter we will discuss the results and present some improvements and future work on the topic. And finally, the sixth chapter will conclude the work and present some future work recommendations.

# 2 Theoretical background

In this chapter we will present general overview of the background technologies that we have used in this thesis. We will introduce the theoretical part of these technologies and in some occasions we will explain in more details those subjects that have been needed for the comprehension of the topic and for the implementation of the project. After a short introduction to the paradigms of the communications networks over the last years, we will deal with the introduction of IMS, the architecture and its components. The security challenges of IMS will be introduce later and some technical mechanisms to correct them will be presented. As the last part of this chapter we will expose some key notions that have been very useful during the project such as the SIP protocol and the structure of the SIP messages, SIP Servlet technology and the general principles of cryptography.

## 2.1 Evolution of mobile networks

Since they appeared, the mobile telecommunication technologies have always known a continuous and fast growth. International Telecommunication Union (ITU) even admits that it has been the “most rapidly adopted technology in history” and that “it is today the most popular and widespread personal technology on the planet” [1]. With over one billion users worldwide, the cellular technology has become the most important part of communication in business and general public use. As a consequence, to answer to the growing customers demands, the wireless telephone technology was developed quite rapidly through its different generations. Each generation was marked by a decisive evolution enabling higher data rates, the use of new frequency bands and non-backwards compatible transmission technologies. The second generation (2G) technologies proposed circuit-switched voice (with the well-known Global System for Mobile Communications (GSM)) and packet-switched data (introduced by the General Packet Radio Service (GPRS)). The 3G marked a turning point by adding the packet-switching for the voice and data IP to the existing 2G infrastructures. By adding the rapidity of IP packet transmissions to the everywhere virtually coverage provided by the mobile phones infrastructures, the 3G enabled the emergence of broadband services with consistent user experience.

Supported by the technology Universal Mobile Telecommunications System (UMTS), the 3G was developed by the 3rd Generation Partnership Project collaboration (3GPP) standardization body. The 3G is based on a mixed core network using circuit and packet switched technologies. It increases bit rates in downlink and uplink, offers a fast access to Internet which enables to provide new multimedia services such as Mobile Television and Video on Demand [2]. All these new services introduced by the 3G were significantly adopted by the costumers which resulted in a dramatic increase in data traffics as shown in Figure 2-1.

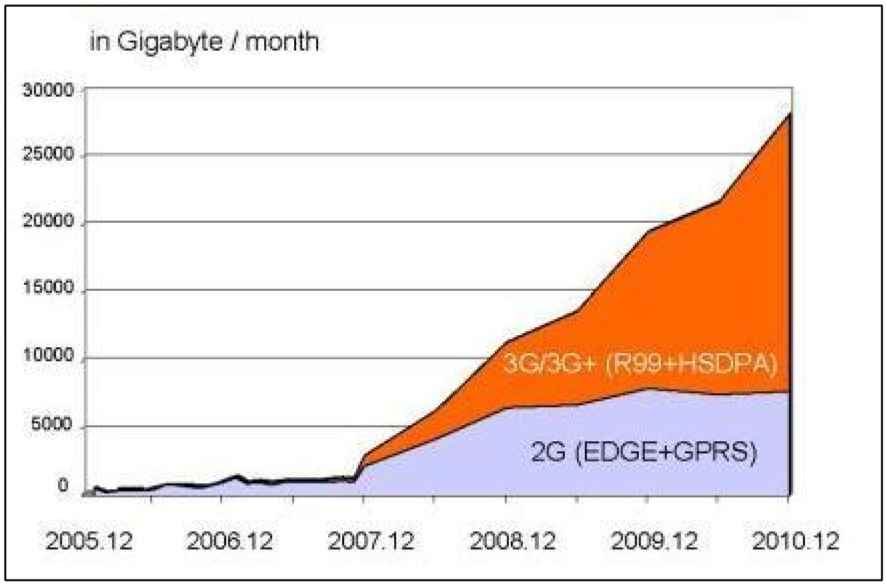


***Figure 2-1.*** *Data traffic: 3G/3G+ vs. 2G [3]*

This data traffic was so important that new standards still had to be developed. Indeed, in spite of the good performance of the 3G, users still asked for more services, with the same Quality of Service (QoS), regardless the network they are attached to. It is in this perspective that nowadays, the 3GPP develops the NGN, working in collaboration with some other organizations like:

- The NGN group in Telecommunication Standardization Sector (ITU-T)
- The Telecoms & Internet converged Services & Protocols for Advanced Networks (TISPAN) group in European Telecommunications Standards Institute (ETSI)

The approaching 4G mobile communication systems aim at solving the still remaining problems in 3G networks and at providing new services. The 4G, a network of networks which provides seamless service to mobile users anywhere, at anytime, is “a must” so far. The 4G architecture separates the Application Network (AN) and the Core Network (CN) and focuses on seamlessly integrating the existing wireless technologies including GSM and Wireless Local Area Network (WLAN). The future 4G infrastructures will be composed of a set of various networks using IP as a common protocol. NGNs are defined not only to be multiservice, multiprotocol, multi-access, IP-based networks, but also secured, reliable and trusted. [4]

## 2.2 Introduction to IMS and its components

In order to offer higher bandwidths, lower operational costs and new generation of services, wireless standards organizations are focused on building the next generation all-IP network infrastructure. The IMS was created as a key part of the third generation network to solve these needs. It was introduced during the 3G development, in UMTS release 5. In its first version, the objective was to "support applications involving multiple media components per session in such a way that the network would be able to dissociate different flows with potentially different QoS characteristics associated to the multimedia session." [4]. IMS was later extended by the ETSI, during its work on the NGN, the 4G. Indeed, IMS was standardized as a subsystem of NGNs by TISPAN. The Figure 2-2 shows a simplified view of the architecture proposed by IMS. We can distinguish the main functions and nodes that will be better presented in the *General Overview of IMS* section.

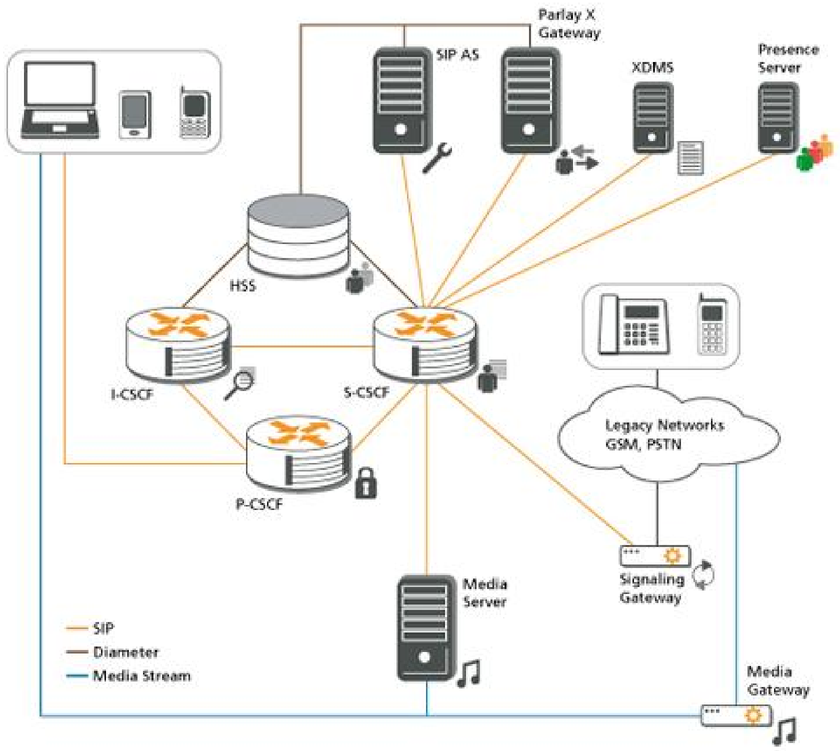


***Figure 2-2.*** *Simplified IMS architecture [5]*

The IMS consists in a set of new entities dedicated to the handling of the signaling and user traffic flows related to the multimedia applications. All IMS entities are located in the IMS Core Network. According to the standards, IMS is defined in the form of reference architecture to enable delivery of next-generation communication services of voice, data, video, wireless, and mobility over an IP network [6]. It exists as part of an entire network.

IMS by itself needs others components such as an access network to complete its functionalities as a system for multimedia service delivery.

### 2.2.1 **Why do we need IMS?**

Nowadays, it is possible to access via a mobile phone any internet services. This is possible due to the development of wireless technologies and the available higher data rates available. The evolution from circuit-switched domain to packet-switched domain improves the user experience in Internet services access through the cellular network. So the question, “why do we need IMS for?”, still remains. According to [7], some of the reasons why we need IMS are:

- One of the most important reasons is the Quality of Service (QoS). This is very important in real time multimedia sessions. The packet-switched domain provides best effort service without really considering QoS. Although IP networks have had their own QoS mechanisms with DiffServ and IntServ. Consequently, it might become a nightmare for users who want to use Voice Over IP (VoIP) or other real time services. Therefore, one of the reasons for creating the IMS was to provide the QoS required for enjoying rather than suffering real time multimedia sessions.
- Another reason for creating IMS was to charge multimedia sessions appropriately. Generally in a network without IMS, 3G operators have no possibility to know what kind of service the user is using.
- The third main reason is to provide integrated services. IMS offers the possibility for 3G operators to provide to the users their own services but also other services provided by third parties. The operators can integrate or combine these third party services to their own services in order to offer completely new services to users

In other words, the IMS aims to [7]:

- combine the latest trends to technology
- make the mobile Internet paradigm come true
- create a common platform to develop diverse multimedia services
- create a mechanism to boost margins due to extra usage of mobile packet-switched networks

There are some requirements that led to design of the 3GPP IMS [7]:

- support for establishing IP Multimedia Sessions
- support for a mechanism to negotiate QoS
- support for interworking with the Internet and circuit-switched networks
- support for roaming

- support for strong control imposed by the operator with respect to the services delivered to the end-user
- support for rapid service creation without requiring standardization

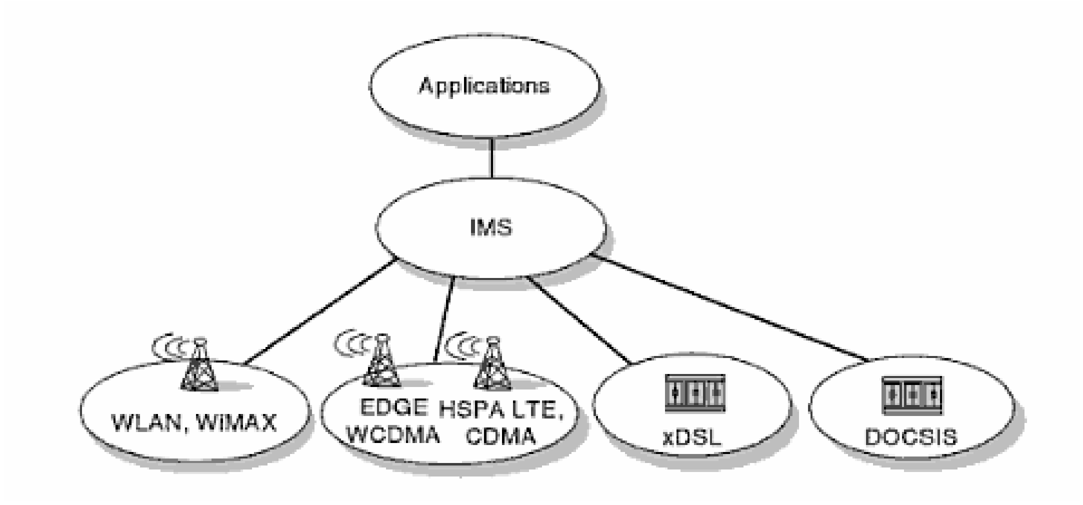


***Figure 2-3.*** *Access Networks Independence [8]*

As shown in the Figure 2-3, another very attractive characteristic of IMS is the access network independency. This means that telecommunication operators can integrate IMS in their network independently of the access network they are providing to theirs users, e.g., GPRS, UMTS or Long Term Evolution (LTE). IMS can even be integrated with wired networks such as Cable Networks, x Digital Subscriber Line (xDSL) or Public Switched Telephone Network (PSTN).

### 2.2.2 General overview of IMS components

First of all, we have to keep in mind that 3GPP standardize functions instead of nodes. This means that the IMS architecture is a collection of functions linked by standardized interfaces [7]. Depending on the operator implementation, we can find two or more functions into a single node or one function into two or more nodes. For a closer view of the IMS architecture standardized by 3GPP, have a look at the Figure 2-4.

In this figure we can distinguish different logical components, interfaces and networks. On the left side, there are different IMS terminals such as Personal Digital Assistants (PDAs) or computers that can be used as IMS clients. These devices referred as User Equipments (UE) can be connected to the IMS core network via wired or wireless access networks such as Asymmetric Digital Subscriber Line (ADSL), GPRS, UMTS, etc. In the IMS Core Network we have different SIP servers, databases and gateways through other networks.

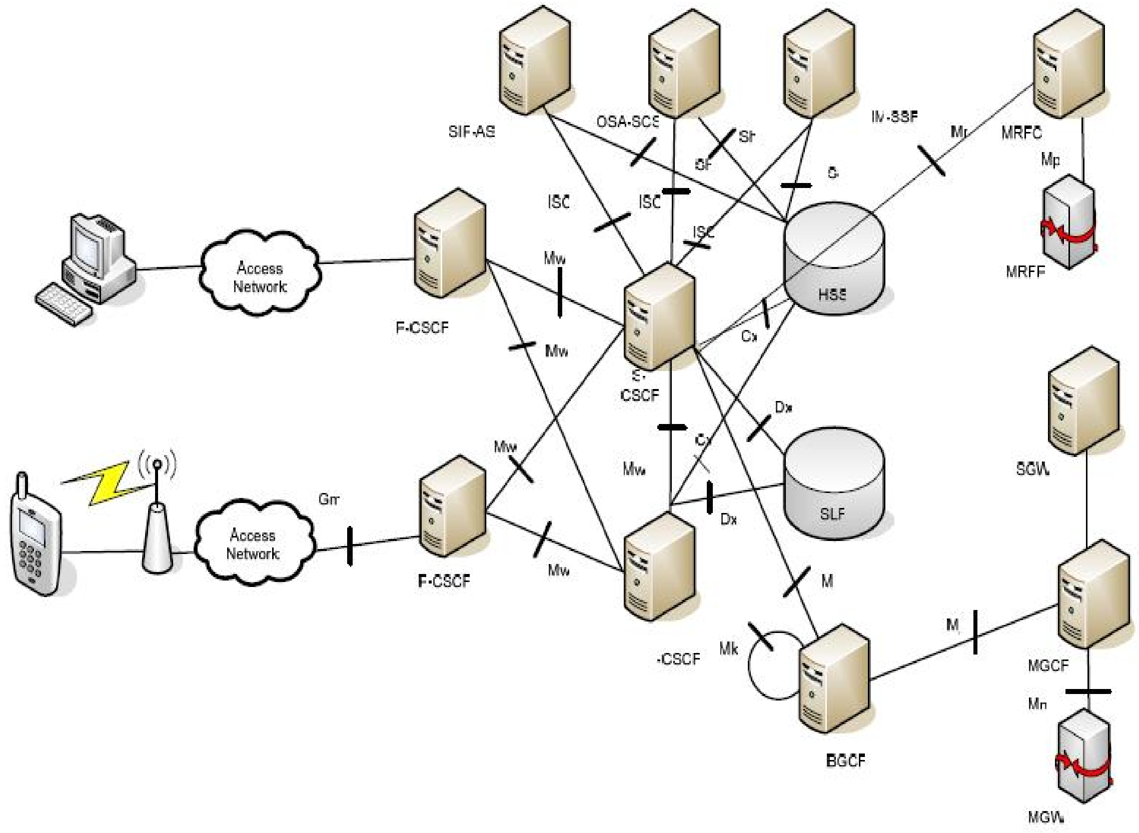


***Figure 2-4.*** *IMS architecture [7]*

Figure 2-4 shows all the nodes included in the IP Multimedia Core Network Subsystem. These nodes are [7]:

- One or more user databases called Home Subscriber Servers (HSS) and Subscriber Location Functions (SLF)
- One or more SIP Servers known as Call Session Control Functions (CSCF)
- One or more Application Servers (Ass)
- One or more Media Resources Functions (MRFs), each one further divided into Media Resources Function Controllers (MRFCs) and Media Resources Function Processors (MRFPs)
- One or more Breakout Gateway Control Functions (BGCFs)
- One or more PSTN gateways, each one decomposed into a Signaling Gateway (SGW)

In this thesis we were more interested in some of these nodes. We will introduce their role in the next sections. Fortunately, one does not need to be expert in all the nodes in order to install the platform and develop applications. The other nodes which are less important for the project will not be introduced in this work. The reader can find more information in [7].

### 2.2.3 User identity in IMS

Identification is one of the most important abilities of a network. Users have to be identified in any kind of network, such as when calls can be directed to the appropriate user. IMS also has its own way to identify users. In IMS, there are two key notions concerning the user identity. These are the *Private User Identity* and the *Public User Identity*.

#### Private User Identity

Each IMS subscriber has one *Private User Identity*. The Private User Identity takes the format of NAI (Network Access Identifier)

```
username@operator.com.
```

The Private User Identity is used for subscription identification and authentication. It is stored in a smart card and there is no need for the user to know it.

#### Public User Identity

Each IMS user can be allocated one or more *Public User Identity*, which is used to route SIP signaling in IMS. The Public User Identity can be a SIP URI or a TEL URI. If Public User Identity is a SIP URI, the form is as follow:

```
sip: first.last@operator.com
```

When a PSTN subscriber makes a call to IMS terminal or receives a call from IMS user, the TEL URI is always needed, because the identification in PSTN can only be presented as numbers. So, TEL URI uses an international way to present a phone number.

### 2.2.4 The user databases: the HSS and the SLF

The HSS is the main user database of IMS. It is used to store the authentication information of users and subscription-related information also known as user profiles. Technically, the HSS is an evolution of the Home Location Register (HLR). A network may contain more than one HSS, but in any case, all of the data related to a particular user is stored in a single HSS. In the Figure 2-5 the reader can find a schematic representation of the HSS.

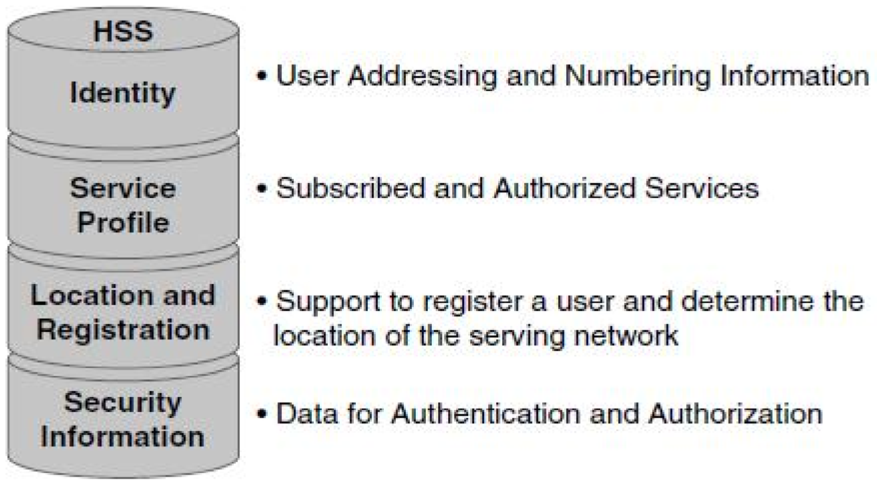

HSS
Identity
• User Addressing and Numbering Information
Service Profile
• Subscribed and Authorized Services
Location and Registration
• Support to register a user and determine the location of the serving network
Security Information
• Data for Authentication and Authorization

***Figure 2-5.*** *Information in the HSS [6]*

The HSS supports the CSCF functions by [6]:

- identifying the address of the CSCF that should be handling the session
- storing the user's registration and location information
- supporting the authentication and authorization by providing the integrity and ciphering keys
- providing an access to a service profile, for which the subscriber has been provisioned

If multiple HSSs are used in the network, a SLF is needed. This is a database used to map the users' address to HSSs. Both the HSS and the SLF can implement the DIAMETER protocol which is used to exchange AAA-related information [9]. The HSS and the SLF are always located in the home network.

## User Profile

All the data related to a user are stored in the HSS in a data structure called *user profile.* The Figure 2-6 shows the components of this structure. The *Private User Identity* and one or more *service profiles* are stored in the *user profile*. Each *service profile* contains one or more *Public User Identities* and zero or more *initial Filter Criteria* (iFC). The *user profile* contains information about the media types that the user is authorized to use, and about the services that are to be applied to the user. This is about the application servers that will need to be contacted whenever the user issues a request.

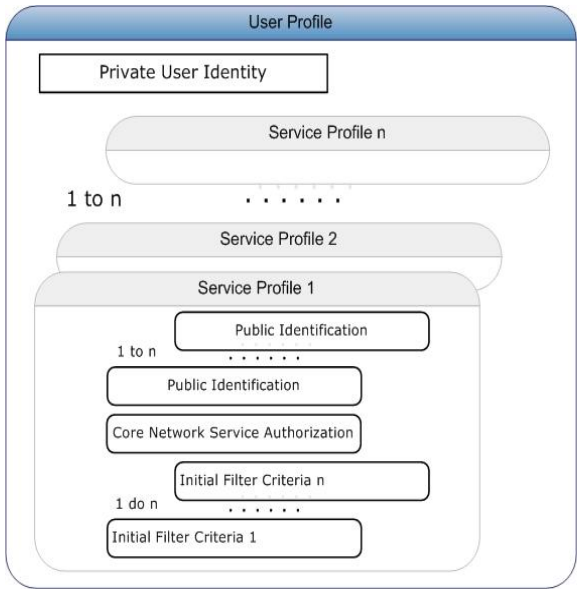


***Figure 2-6.*** *User Profile*

When the user registers to a particular Serving CSCF (S-CSCF) and the authentication is confirmed, the *user profile* including the iFC is downloaded from the HSS by the S-CSCF and is available for the whole period that the user is registered.

## Initial Filter Criteria

Besides other information, in the service profile there is also initial Filter Criteria. It contains user information that indicates the S-CSCF when to involve a particular Application Server to provide the service. Filter Criteria is a very important concept in IMS because it determines the services that will be provided to each user. An iFC defines a set of conditions that, when met, will force the S-CSCF to delegate the control to an AS whose Uniform Resource Identifier (URI) is also part of the iFC. The conditions can be based on different combinations of values of SIP methods, SIP headers and so forth, present in the incoming request. For more information on SIP the reader can refer to the section **2.3**.

The iFCs are always evaluated with SIP initial requests that initiate a dialog or are stand-alone requests like `”INVITE”, “REGISTER”, “SUBSCRIBE MESSAGE”`. Subsequent messages like `“BYE”, “NOTIFY, “INVITE”` inside a dialog are not evaluated. The iFC can be considered as the grouping between a Trigger Point (the logical expression of Service Point Triggers (SPT) matching a message) and an AS [10]. It is composed of groups and each group is formed of the logical atoms, the SPT. There are 5 distinct types of SPTs, based on what each of them is used to check in the SIP message:

- Request-URI *equals* <value>
- SIP Method *equals* <value>

- SIP Header *matches* <regular expression>
- Session Case *is one of* [originating, terminating, terminating to unregistered user]
- SDP Line [<line name>] *matches* <regular expression>

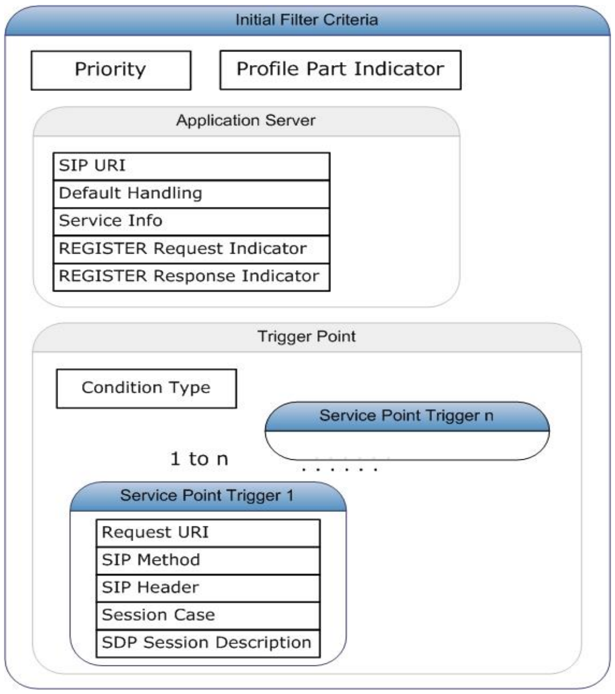


***Figure 2-7.*** *Initial Filter Criteria*

The components of an iFC are presented in the Figure 2-7. As we note, the iFC is composed by one or more Service Point Trigger and some fields such as Priority or information related to the AS.

## 2.2.5 **SIP servers: Call Session Control Function**

The CSCF provides the central control function in the IMS Core Network to set up, establish, modify, and tear down multimedia sessions. This is basically a SIP server which processes and is responsible for all SIP signaling in the IMS. The CSCF function is distributed across three types of functional elements based on the specialized function they perform [7]. These three elements are the Proxy-CSCF (P-CSCF), Interrogating-CSCF (I-CSCF), and the Serving-CSCF.

### *P-CSCF*

The P-CSCF is an edge access function and is the entry point for a UE to request services from an IMS network. The P-CSCF is allocated to the IMS terminal during IMS registration and does not change throughout the whole registration time. The role of this

SIP server is to function as a proxy by accepting incoming requests and forwarding to the entity that can service them. From the SIP point of view, the P-CSCF is acting as an inbound/outbound proxy server. The incoming requests are either the initial registration or an invitation for a multimedia session. A request of the UE to register for a service is normally forwarded to a session controller or to one with the capability to interrogate for it. Session invitation requests are forwarded by the P-CSCF to a S-CSCF.

The P-CSCF also performs some important edge functions. Since this is the first-hop access, it establishes and maintains a number of Internet Protocol Security (IPSec) security associations with the IMS terminal. It also provides compression of SIP signaling to minimize latency over the air interface. SIP messages can be large due to the text based nature of SIP protocol. In order to reduce the time to transmit SIP messages, a mechanism to compress and decompress the message is used.  It provides a policy function by initiating support for IP flow control and authorization of traffic-bearer resources. The P-CSCF is also capable of handling emergency call sessions. [7] [8] [11]

The P-CSCF may be located in the home network or in the visited network. The location of the P-CSCF is directly related with the location of the Gateway GPRS Support Node (GGSN) in the 2.5G networks. Once the IMS reaches the mass market the operators will migrate the configuration and the P-CSCF and the GGSN will be definitely located in the visited network.

### *I-CSCF*

The I-CSCF is a SIP proxy located on the edge of an administrative domain [9]. The Domain Name System (DNS) records of the domain store the address of the I-CSCF. The SIP server obtains the address of an I-CSCF of the destination domain when it tries to find the next SIP hop for a particular message.

The I-CSCF is responsible for determining which Serving CSCF should be assigned for controlling the session requested by the UE. A request to the I-CSCF may come from the home network or a visited network through the Proxy CSCF. The I-CSCF obtains the request for the address of the S-CSCF from the HSS during a registration request, and provides it to the P-CSCF for subsequent multimedia requests.

The I-CSCF has an interface to the HSS and SLF, which is based on the Diameter protocol. It can obtain the address of S-CSCF from HSS. And it could also retrieve user location information and route a SIP request received from another network towards the appropriate destination, such as the S-CSCF. The I-CSCF may perform transit routing functions. When the I-CSCF determines that the destination of the session is not in the IMS, it may forward the request or return with a failure response. In some cases the I-CSCF can encrypt some parts of SIP messages that contain sensitive information about the domain such as the DNS servers, their names or their capacity or information about the route the SIP message has taken. This functionality is known as Topology Hiding Inter-network Gateway (THIG).The I-CSCF is usually located in the home network. However, for some special cases, for example, an I-CSCF THIG, it may also be located in visited network.

#### *S-CSCF*

The S-CSCF is responsible for conducting both registration and session control for the registered UE's sessions. In addition, it is responsible for interfacing with the Application Servers in the Application Plane. As a registrar, it enables the network location information of the UE to be available through the HSS. It maintains a binding between the user location (the IP address of the terminal the user is registered with) and the used SIP address of record (also known as Public User Identity). It makes the decision to allow or deny service to the UE. Its role is to execute the session request by locating the destination endpoint and conducting the signaling toward it. The S-CSCFs maintain a full state of the sessions and have the capability to originate and terminate a session on behalf of a requesting endpoint. Like I-CSCF, the S-CSCF also implements a Diameter interface to the HSS. The reason is to download from the HSS the authentication vectors of the user who wants to access IMS. These vectors are used to authenticate the user. The S-CSCF also downloads from the HSS the user profile which contains a set of triggers that may route the SIP message through one or more AS.

In addition to acting as a registrar, the S-CSCF is also responsible for routing all SIP messages to the AS.  To do this, the S-CSCF uses information obtained from the HSS in the form of iFC that acts as triggers against inbound session establishment requests.  The iFC includes rules that define how and where SIP messages should be routed to the various AS that may reside in the Application Plane.  A network may include a number of S-CSCFs in the sake of scalability and redundancy. Each of them serves a number of IMS terminals. The S-CSCF is always located in the home network.

### 2.2.6 **Application Servers**

The SIP servers that we already presented don't provide any added-value services. IMS applications are implemented in SIP AS with the help of media servers [11]. Application server is a service creation and execution platform that interacts with the S-CSCF using SIP via IMS Service Control (ISC) interface. The Figure 2-8 presents the ISC interface between the AS and the S-CSCF and the basic principle of the relation between them.

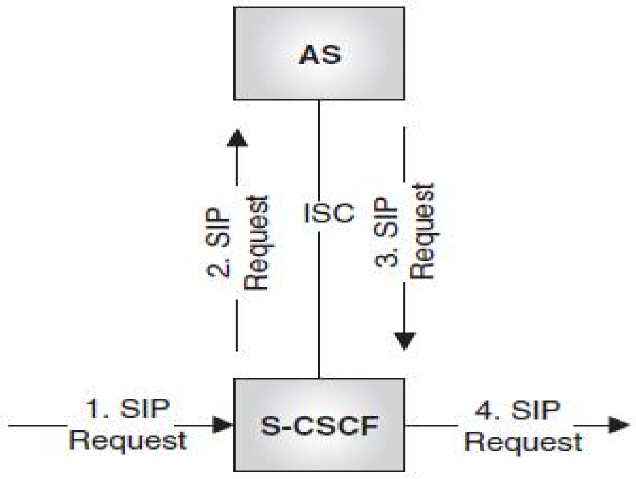


***Figure 2-8.*** *ISC Interface*

The AS can operate on a SIP User Agent (UA) mode or Back-to-back User Agent (B2BUA) mode. The ASes are responsible for hosting and executing the services. A single AS may host multiple services, for example, telephony and messaging services. One advantage of this flexibility is to reduce the workload of the control layer. There are many application servers providing different services, such as presence service or instant messaging service etc.

The service profile downloaded from the HSS contains a list of ASs, identified by their URI, and some filtering rules. The filtering rules allow the S-CSCF to determine when received SIP requests need to be routed to a particular AS. In this way, customized services can be applied to the users, and a clear separation is enforced between the basic SIP functions (S-CSCF) and the enhanced service logic (AS) [11]. According to [7], we have several kind of AS. We will cite only the most useful for this work, the **SIP-AS**. This AS, is the service part in IMS, supporting well defined signaling and administration interfaces. This AS hosts and executes IP multimedia services based on SIP. It is triggered by the S-CSCF and it comprises filter rules for choosing applications for the handling of the session.

An AS can be located in the home network or in an external third-party network. If located in the home network, it can query the HSS with the Diameter Sh or Si interfaces (for a SIP-AS).

## 2.3 Overview of SIP

The Session Initiation Protocol is an application-level signaling protocol defined by the Internet Engineering Task Force (IETF) in RFC 3261 [12] for creating, modifying, and terminating sessions with one or more participant over an IP network. Sessions are described using the Session Description Protocol (SDP) defined in RFC 4566 [13]. The

SDP sessions are carried in the SIP messages and allow participant to agree on a set of parameters such as transport addresses or media types [11]. In addition, SIP deals with the location of the users. For this reason, every user that wants to be able to receive multimedia communications has to be registered. This is done through the registration procedure, during which the user has to communicate its present location (expressed as a "location" SIP URI), together with its public identity (expressed as a "logical" SIP URI), to its home SIP server (its registrar server), which will then maintain a table with the mapping.

SIP follows the well-known client-server model like many other protocols developed and used by IETF. SIP is a text-based protocol that borrows most of its principles from SMTP (Simple Mail Transfer Protocol) and HTTP (Hypertext Transfer Protocol); two of the most successful protocols on the Internet. This means that it is easy to extend, debug and use to build services [7].

SIP URIs are used to identify users in the SIP architecture. In general they identify communication resources. SIP URI follows the general rules for URIs defined in RFC 3986 [14] as shown in the following example: `"sip:name.surname@example.com"`. This URI is made of user part which identifies a particular resource and host-port part which identifies the source providing the resource. This might be a Fully Qualified Domain Name (FQDN) or an IP address plus an optional port value. As we will see later, a SIP URI may represent [11]:

- The public identity of a user. This is the identifier that anyone could use to contact that user. Example: `sip:gerti.dhima@openims.com`
- A user at a specific host or location. For example in the following SIP URI the user bob is located at location `192.168.10.33`. Example: `sip:bob.brown@192.168.10.33`
- A SIP server such as in the example : `sip:proxyA.openims.com`
- A group of users such as in the example: `mobileUsers@openims.com`. This might represent all the mobile phone users of a company

As we can see, some of the SIP URIs point to logical identities and some others indicate locations. SIP URIs that point to locations, can be directly resolved to the corresponding IP address, port, and transport via DNS queries. Other URIs can be used to identify communication resources for example the Telephone Uniform Resource Identifier (TEL URI). We will not introduce them in this work but the reader can find more about them in [11].

### *SIP Messages*

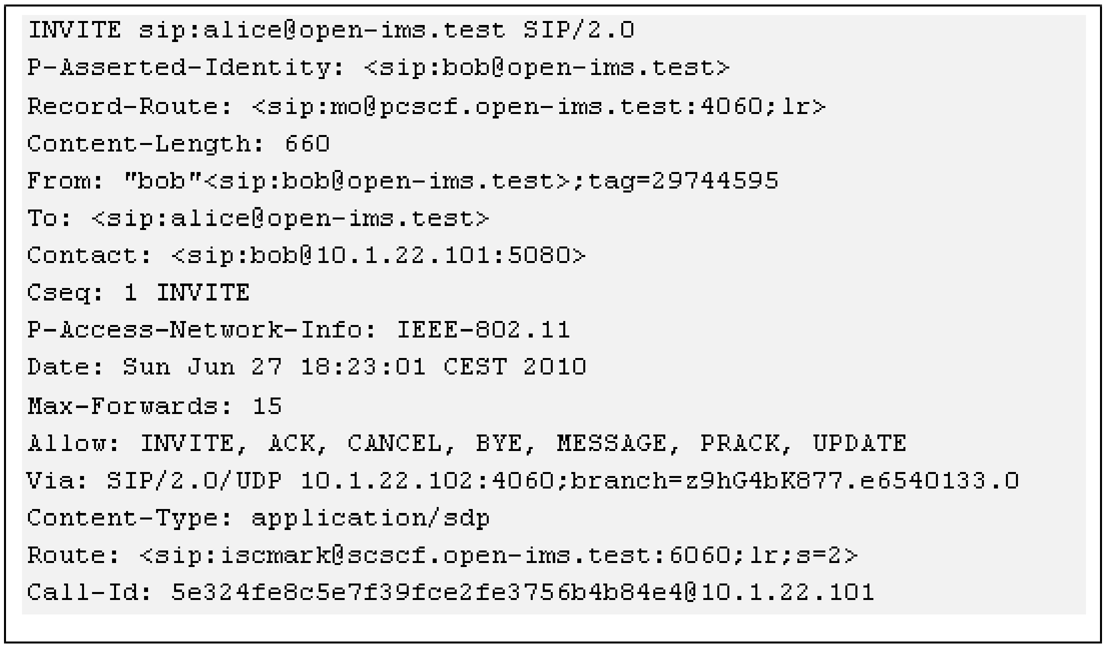
The communication using SIP means a couple of messages. These messages can be, either SIP Requests sent by a client to a server or SIP Responses sent by a server to the client as a response to the SIP Request. The messages are transported by the underlying network protocols.

- Requests are used in order to start a communication or to inform the receiver about a request
- Responses are used in order to confirm that a request has been received and treated by the server or to indicate the status of the treatment.

The general format of a SIP messages is:

```
request-line
message-header
message-body]
Request-line = Methods Request-URI SIP-Version
```

The methods can be: `"REGISTER"`, `"INVITE"`, `"ACK"`, `"CANCEL"`, `"BYE"`, `"OPTION"`. The Figure 2-9 presents a simple `INVITE` request sent by the SIP user `bob@open-ims.test` to start a conversation with the SIP user `alice@open-ims.test`:



```
INVITE sip:alice@open-ims.test SIP/2.0
P-Asserted-Identity: <sip:bob@open-ims.test>
Record-Route: <sip:mo@pcscf.open-ims.test:4060;lr>
Content-Length: 660
From: "bob"<sip:bob@open-ims.test>;tag=29744595
To: <sip:alice@open-ims.test>
Contact: <sip:bob@10.1.22.101:5080>
Cseq: 1 INVITE
P-Access-Network-Info: IEEE-802.11
Date: Sun Jun 27 18:23:01 CEST 2010
Max-Forwards: 15
Allow: INVITE, ACK, CANCEL, BYE, MESSAGE, PRACK, UPDATE
Via: SIP/2.0/UDP 10.1.22.102:4060;branch=z9hG4bK877.e6540133.0
Content-Type: application/sdp
Route: <sip:iscmark@scscf.open-ims.test:6060;lr;s=2>
Call-Id: 5e324fe8c5e7f39fce2fe3756b4b84e4@10.1.22.101
```

***Figure 2-9.*** *Invite message*

Through this example we have shown the most important SIP headers of the simple `INVITE` message. We will see in the implementation chapter of the project that some other SIP headers have been added. These new headers contain the Identity of different SIP entities and their respective signature. Later on we will discuss more in detail about these aspects. In the Table 1 we present the most important SIP headers and their role.

| **SIP Header** | **Role** |
|---|---|
| Call-ID | The Call-ID header uniquely identifies a particular invitation or all registrations of a particular client. |
| CSeq | The CSeq header (command sequence) uniquely identifies transactions within a dialog. |
| Contact | The Contact header contains a list of URLs used to redirect future requests. |
| Content-Length | The Content-Length header indicates the size of the message-body in decimal number of octets |
| Date | The Date header field reflects the time when the request or response was first sent. |
| From | The From header indicates the initiator of the request. |
| Max-Forwards | The Max-Forwards header is used to limit the number of proxies or gateways that can forward the request. |
| Route | The Route header is used to store the route set of a transaction. |
| Record-Route | The Record-Route headers are used to establish a route for transactions belonging to a session. |
| To | The To header specifies the "logical" recipient of the request. |
| Via | The Via header indicates the path taken by the request so far. |
| Event | The Event header is used to indicate the event or class of events the message contains or subscribes. |
| Supported | The Supported header enumerates all the capabilities of the client or server. |
| Path | The Path header field is a SIP extension header field with syntax very similar to the Record-Route header field. |
| Content-Type | The Content-Type header indicates the media type of the message-body sent to the recipient. |

***Table 1.*** *The most important Sip headers*

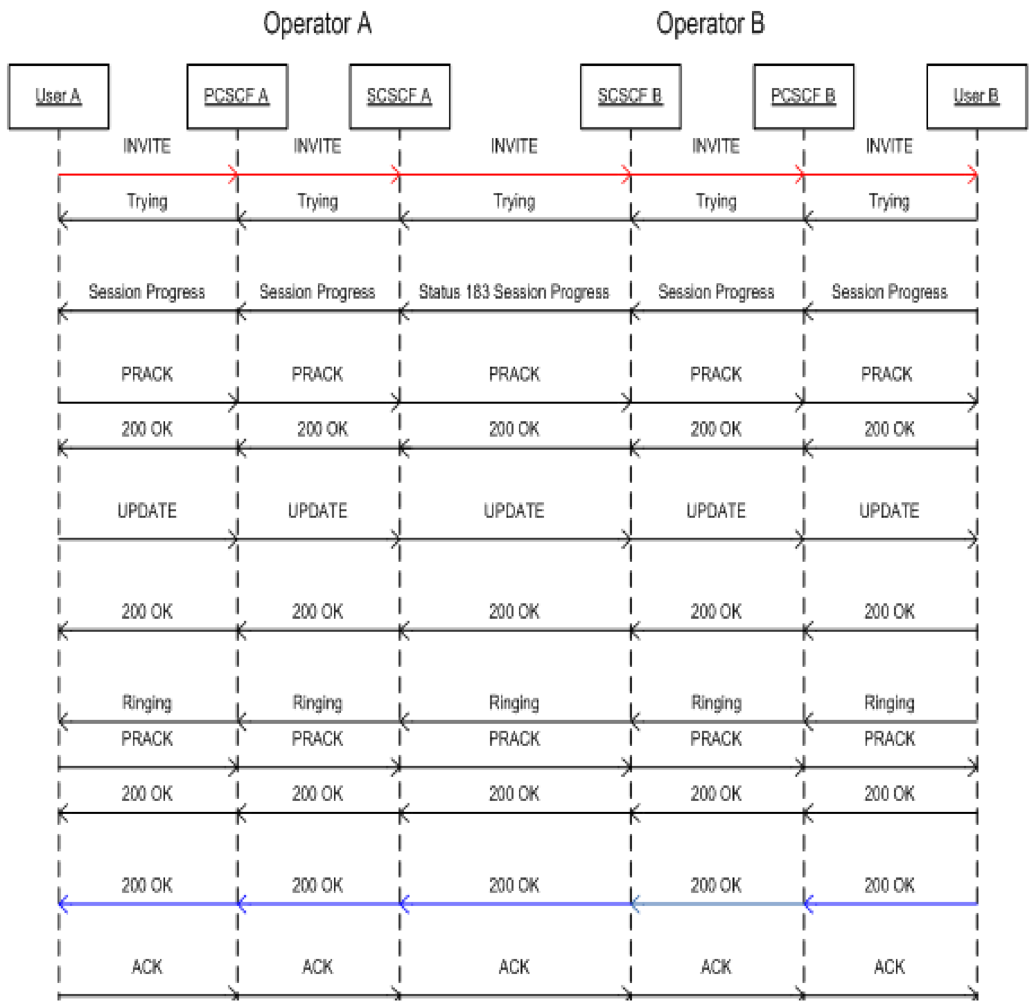


***Figure 2-10.*** *INVITE call flow*

As the mechanism of enhancing the identity assurance is implemented on the basic `INVITE` exchange, we will introduce here the call flow of this exchange as described by the standards, presented in the Figure 2-10. For more details one can be referred to [7].

In this figure, in red is presented the first `INVITE` request sent by *UserA* when he wants to start a multimedia conversation with another IMS user (*UserB*) that can be registered in another IMS domain as presented in this case. Before arriving at *UserB* the request pass through different SIP servers of the domain A and then B as per a normal `INVITE` call flow. Once this request arrives at the *UserB*, some other responses are exchanged up to the moment *UserB* decides to take the call. In this moment, a SIP response "`200 OK`" is generated by *UserB*, represented in blue in this figure, and is sent to *UserA*. In this work we modify only the first "`INVITE`" request and then the last "`200OK`" response.

## 2.4 SIP Servlet Technology

A SIP Servlet is a server-side component of Java programming language that performs SIP signaling. SIP Servlet was designed to simplify SIP development and attract developers to the SIP protocol. They are managed by SIP Servlet containers, which typically are part of a SIP-enabled application server [15]. SIP Servlets interact with clients by responding to incoming SIP requests and returning corresponding SIP responses. This is comparable to the "request/response" model known from HTTP Servlets. They are built using the generic servlet Application Programming Interface (API) provided by the Java Servlet Specification.

The Figure 2-11 shows a general overview of the SIP Servlet API and compares it with the Servlet API. SIP Servlet are based on Java APIs for Integrated Networks (JAIN SIP), a low level Java API specification for SIP Signaling. [16]

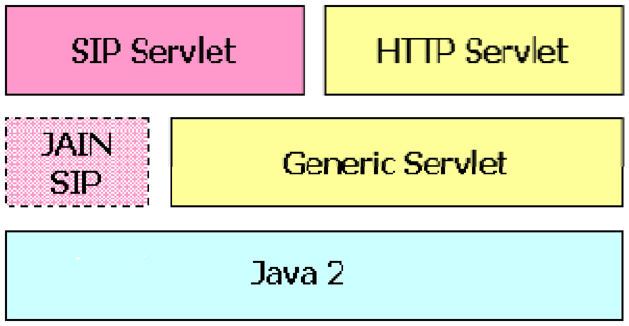


***Figure 2-11.** Servlet API and SIP Servlet API*

SIP Servlet is a Java programming language class that extends the "`javax.servlet.sip.SipServlet`" class, optionally overriding SipServlet's methods [15]. These methods are named along the model of "`doRequest`" where "`Request`" is a SIP request name. For example, the "`doINVITE`" method will respond to incoming SIP "`INVITE`" requests.

The SIP Servlet API (JSR 116) is a server-side interface describing a container of SIP components or services. For more details the reader can find this API in [17]. The lifecycle of SIP Servlets are managed by the SIP Servlet container, which at the same time enables network communication for SIP requests and responses by listening on a particular listening point. A typical SIP application consists of the following components:

- one or more SIP Servlets
- optional utility and helper classes such as SIP listeners
- static resources used by the classes
- metadata and configuration files

One of the configuration files is "`sip.xml`" deployment descriptor, which is used by the SIP Servlet container to process deployed SIP applications and configure the runtime to properly respond to incoming SIP requests [15]. SIP applications are packaged in either SIP archive (SAR) or web archive (WAR) files. SIP containers will recognize either the "`.sar`" or "`.war`" extensions when processing SIP applications. The SAR format includes the use of the presence of the "`WEB-INF`" folder that contains class files and deployment descriptors. SIP application that has been packaged in a SAR or WAR may be packaged itself within Enterprise archive (EAR).

We have used SIP Servlet API to develop the application, which will be responsible for checking the identity of one SIP entity when an "`INVITE`" `message` arrives to the S-CSCF and then adding the identity and the signature of the SIP server to the message. We delegated this task to the application server because in this way the S-CSCF is not overloaded with this extra task. This will be better explained when we will discuss the implementation of the project in the section **4.3**.

## 2.5 Principles of cryptography

In the computer era, cryptography has become a necessity for the exchange of critical messages. Indeed, Internet has become an essential communication tool both for business and private needs. It appears that the transactions can be intercepted by malicious people for several purposes: just to get some information (passive listening), to stop exchanges, or even to modify it. So, there is an obvious need for a security system or mechanism.

The cryptography is a technique which aims at protecting the confidentiality, integrity and authenticity of the information by ciphering the messages. By applying arithmetic calculations to the original message (plain text), it creates a new message unreadable without applying a specific action (cipher text). Most of the time, we use a cipher key to cipher a message, and a decipher key to decipher a message. There are two kinds of encryption: the symmetric and the asymmetric ones. The next sections will present them and also the principle of digital signatures which derives from them.

### 2.5.1 Symmetric encryption

The symmetric encryption uses symmetric keys or "secret keys". A representation of the symmetric mechanism is given in the Figure 2-12. The sender and the receiver share a common secret key. The sender ciphers the message with this secret key, and the receiver deciphers it with the same secret key. As a consequence, the confidentiality of the message is ensured.

This is a very efficient method as, if anyone intercepted the message, he couldn't get the information because he wouldn't have the secret key. The problem with this method is that, it is assumed that the two users share a common secret. In the Internet world, as people communicating with each other can be at the two sides of the world, the key exchange may need to be done through the network. Thus, we come back to the initial situation where we need to secure this exchange. Rather than doing so, a second encryption method, the asymmetric cipher, exists to avoid these problems: the asymmetric cipher.

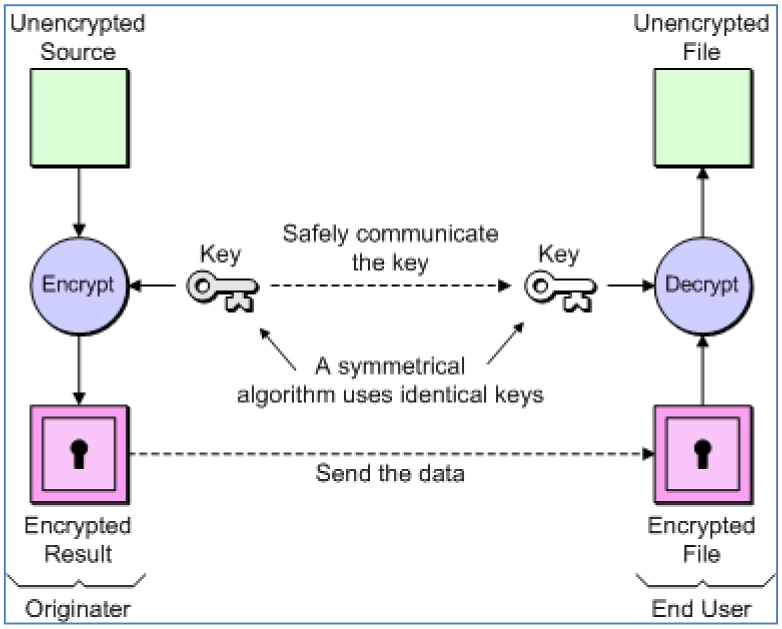


***Figure 2-12***. Symmetrical encryption and decryption *[18]*

### 2.5.2 **Asymmetric encryption**

Asymmetric encryption uses asymmetric keys which are a pair of private/public keys. This method can solve the problem of key distribution caused by the symmetric encryption. Indeed, rather than having a single secret key, each user is going to have a pair of cryptographic keys: a private key and a public key. The private key is kept secret while the public key can be widely spread. Contrary to the symmetric encryption, as we can see in the Figure 2-13, the encryption and decryption are done with different keys. Every time the Originator wants to send a message to the End User and wants to make sure he is going to be the only one who can read it, he has to get the End User's public key, encrypt the message with it and send it to him. When the End User receives the message, he decrypts it with his private key. Thus, the Originator can be sure that nobody else than the End User can open his message, because he is the only one who has the appropriate private key. Consequently, the confidentiality of the message is assured.

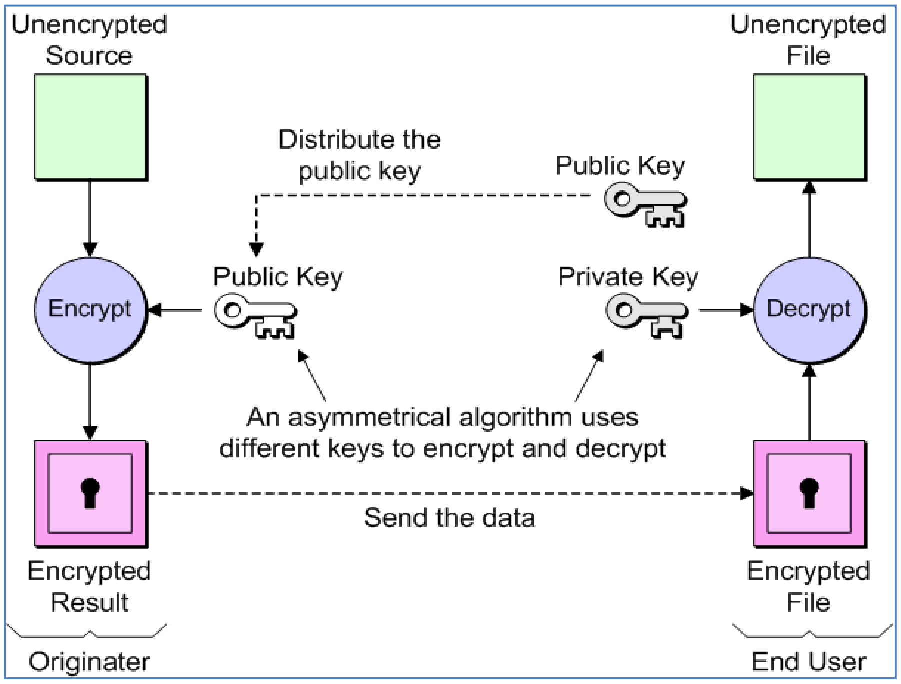


***Figure 2-13.*** *Asymmetrical encryption and description [18]*

The main advantage of this technique is to avoid the decryption key distribution problem, because the public keys can be freely sent. But there is a new challenge rising. The receiver has to be sure that the public key that he get from the sender is really this one corresponding to the person that he thinks. Still, it is easier to implement and more secured than the symmetric encryption. In reality the most systems use asymmetric keys to encrypt the symmetric key, which is used to encrypt the actual message.

### 2.5.3 **Digital signature**

The asymmetric encryption enables the creation of the digital signatures which aim at protecting the message. It consists of applying a hash function to a message and then encrypting it with the private key to obtain a signature. The generation of this signature is represented in the Figure 2-14. This one is a kind of DNA of the message, which is added to the original message to produce a signed message. Once this signed message received, the signature can be verified by the receiver as shown in the Figure 2-14:

- The end user gets the signature
- He deciphers it with the sender's public key
- And then, it compares the result with the result of the hash function on the message

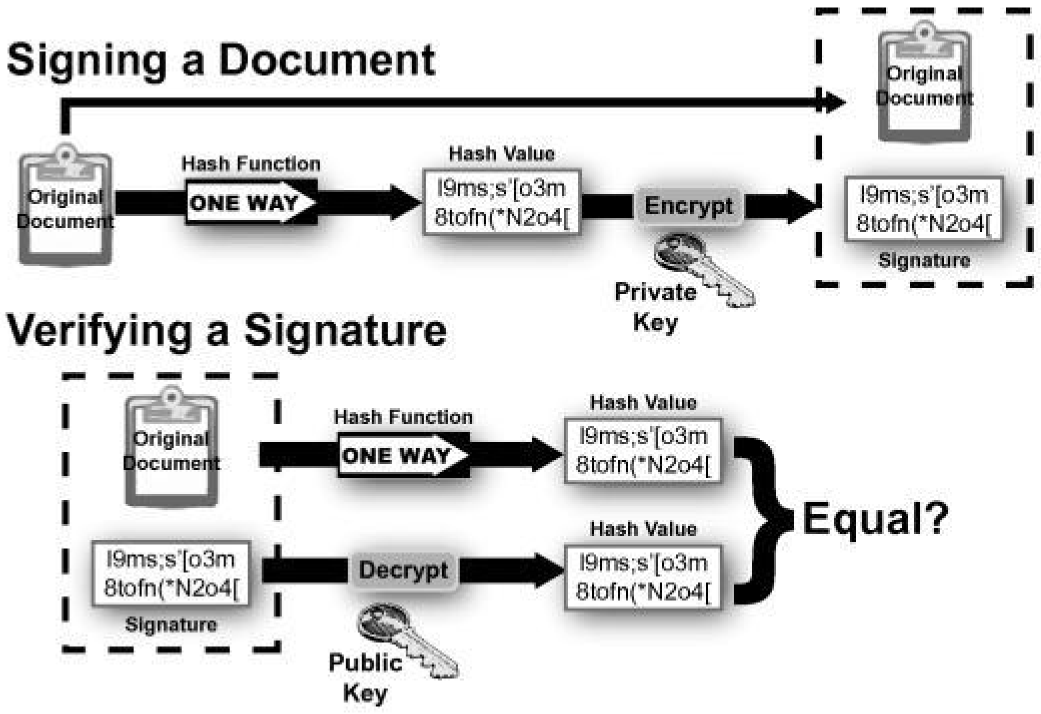


***Figure 2-14.*** *Generation and validation of the signature [19]*

A signature generated with a certain private key is validated by using the corresponding public key. Moreover, it should not be computationally feasible to generate a valid signature for a party who does not have the private key. This process guarantees the integrity of the message by creating a hash of it and its authenticity by encrypting this hash with the private key. [20]

## 2.6 The main security problems of IMS

In the above sections we gave a general overview of the IMS architecture. Now we will discuss some security problems related to IMS and more precisely to the signaling protocol it is using.

The IMS architecture presents significant security challenges that must be addressed by the carriers as IMS moves into widespread deployment. The generally open and distributed architecture creates the advantage of flexibility in implementation and deployment. It also creates a multitude of interface points that must be secured [21]. The security in IMS is an important and urgent issue and it is required to ensure that every SIP based Internet service can meet the corresponding security requirements.

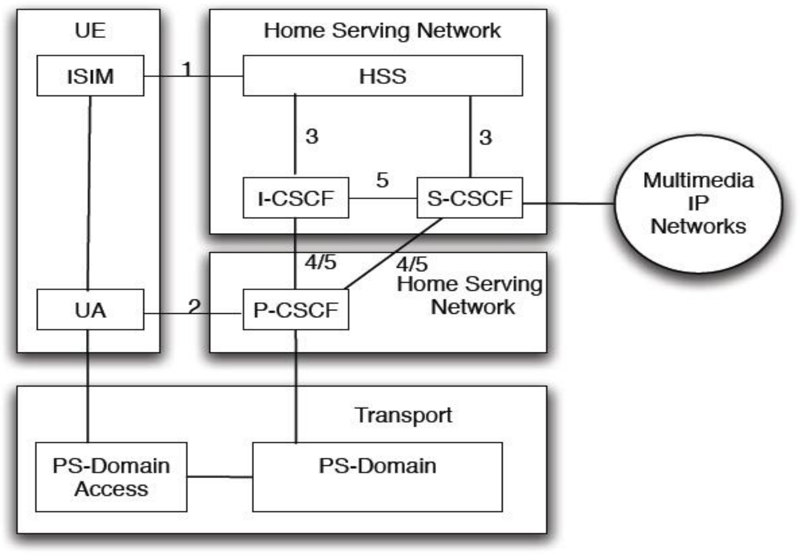


***Figure 2-15.*** *IMS Security Architecture [21]*

According to [21], in the ***Figure*** 2-15, five different security associations within the IMS are identified:

1. Mutual authentication between the UE and the IMS. The HSS delegates this to the S-CSCF but the HSS is responsible for generating the keys and challenges.
2. Secure link and security association between the UE and P-CSCF for authentication of data origin.
3. The Cx interface provides internal security for the link between the CSCF and HSS. This association plays an important role in securing the keys and challenges during the UE registration process.
4. Link which provides security between the P-CSCF and other core SIP services when the UE is roaming in a Visited Network (VN).
5. This link provides security between the P-CSCF and other core SIP services when the UE is operating in the Home Network (HN).

The main problems are also related to the underlying protocols that IMS and SIP are using. SIP protocol, according to RFC 3261 [12], utilizes transport protocols such as TCP and UDP. As a result, SIP inherits the vulnerabilities of these protocols. As noted in RFC 2617 [22], SIP authentication typically uses HTTP digest authentication, which is vulnerable to many forms of known attacks. According to [23], the HTTP digest authentication in SIP suffers from a major weakness when it is applied in SIP. It is the lack of securing all headers and parameters in SIP which would possibly need protection.

Since the current authentication mechanism is not providing security at an acceptable level, several new schemes are proposed to improve it. One can find a non-exhaustive list of these schemes in [24]. Some of the mechanisms suggested to increase the SIP security are S/MIME, IPSec and TLS. Moreover with the last developments in the networks and computing technologies, it becomes easier for the new players and third part service providers to enter the market. Therefore signaling and data packets can traverse untrustworthy network realms. According to the specifications [25], the security of IMS is based on the fact that the multimedia sessions are supposed to be taken place between trusted entities. Thus, the operator static relationship cannot rely anymore in old paradigms to guarantee the user identity end-to-end. Because of these new developments, there is a real need for new methods that provide better assurance about the identity of the entities taking part in a multimedia conversation. In order to provide the recipient of a SIP message with greater assurance of the identity of the sender, a cryptographic signature can be provided over the headers of the SIP request, which allows the signer to assert a verifiable identity as described in RFC 3893 [26].

In this particular technologic background and actual state of art, we have implemented a new mechanism which takes advantage of IMS architecture to establish a trust between the entities taking part in a multimedia session in a roaming scenario. We will better introduce the work in the next section.

## 2.7 Proposition of an secured identity exchange mechanism

In order to make the authentication mechanism security level meet the requirements, one solution can consist in exchanging secured identities before starting a multimedia session. Based on the paper [9], this thesis proposes to take advantage of the IMS's ability to easily and rapidly provide services developed in application servers to imagine a service which could implement this security mechanism. The home operators of the users guarantee the identities of their users and are assumed to trust each other thanks to a roaming agreement. This kind of agreement can link two telephone companies to outline the terms and conditions under which the participating companies will provide wireless service to their subscribers (it dictates the liabilities of the parties and the necessary security association). Roaming agreements are useful when companies can't offer a complete national or international coverage. It's assumed that every entity has a secure cryptographic identifier attributed. The possession of this secured identifier enables its owner to prove his identity. In this project, the necessary initial signaling needed to enhance the trust between the parties in the communication is carried by the IMS architecture.

The mechanism envisaged is based on local trust decisions and roaming agreements. It does not rely on the existence of a global PKI which avoids the problems relying to the future dynamic networks. There probably will not be any agreement about the common global trust roots. Thus in practice, such a security infrastructure would not be very practical to establish and to support.

The Figure 2-16 shows how the chain of trust between two SIP users *UserA* and *UserB* is established. The operators trust their respective users and the trust relationship between the operators allows *UserA* and *UserB* to trust each others. This leads to create a chain of trust between the two users.

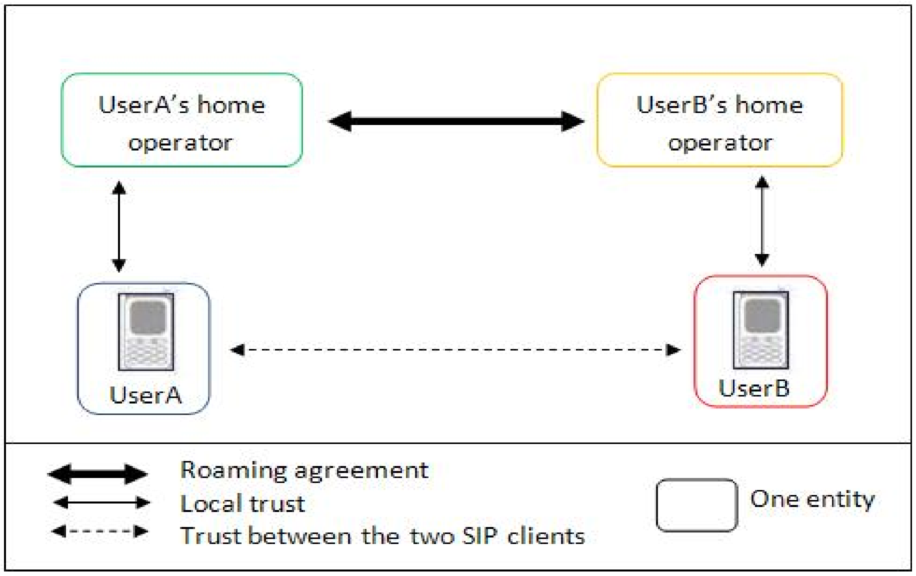


***Figure 2-16.*** *Chain of trust establishment between two SIP users*

In practice, the fact that two entities trust each others can mean that they share secured information. Every entity is in possession of his own secured information which represents his identity. Every time one wants to prove his identity, he has to show that it is in possession of this secured information. In this project, we chose to use Public Keys to constitute the entities' secured identity. The Figure 2-17 represents the creation of the cryptographic identifier which is based on a public Key. By applying hashing to the public key attributed to the entity, we can form a concise representation of this identity, Hash Identity Tag (HIT). Basically, HIT and public Key represent the same thing (the entity's cryptographic identity), the HIT is just more suited for protocols. Moreover it has to be noted that the term HIT that we use in this specific context should not be mixed with the same term used in Host Identity Protocol context. Even though the identity we presented here is based on the HIT representation, there are some differences that we have to keep in mind. The identity that we are using is not exactly the same as HIT in a typical HIP exchange. Actually in HIP it has certain structure as well to give it a IPv6 interpretation. In this work we haven't implemented HIP specific exchange, but just the assured identity exchange.

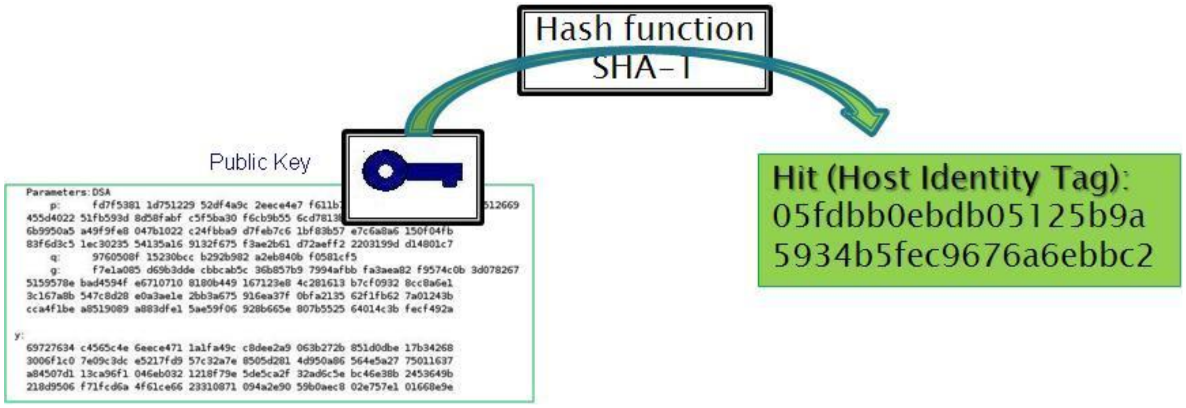


*Figure 2-17.* *Creation of the HIT based on the public Key*

In essence, when two entities want to establish a trust between them, they first exchange their identities HIT (or their Public keys with which they calculated the HIT, like in the Figure 2-18). Then, when they need to communicate in a trusted way, they just have to prove that they are in possession of their own identity (or of their Public Key).

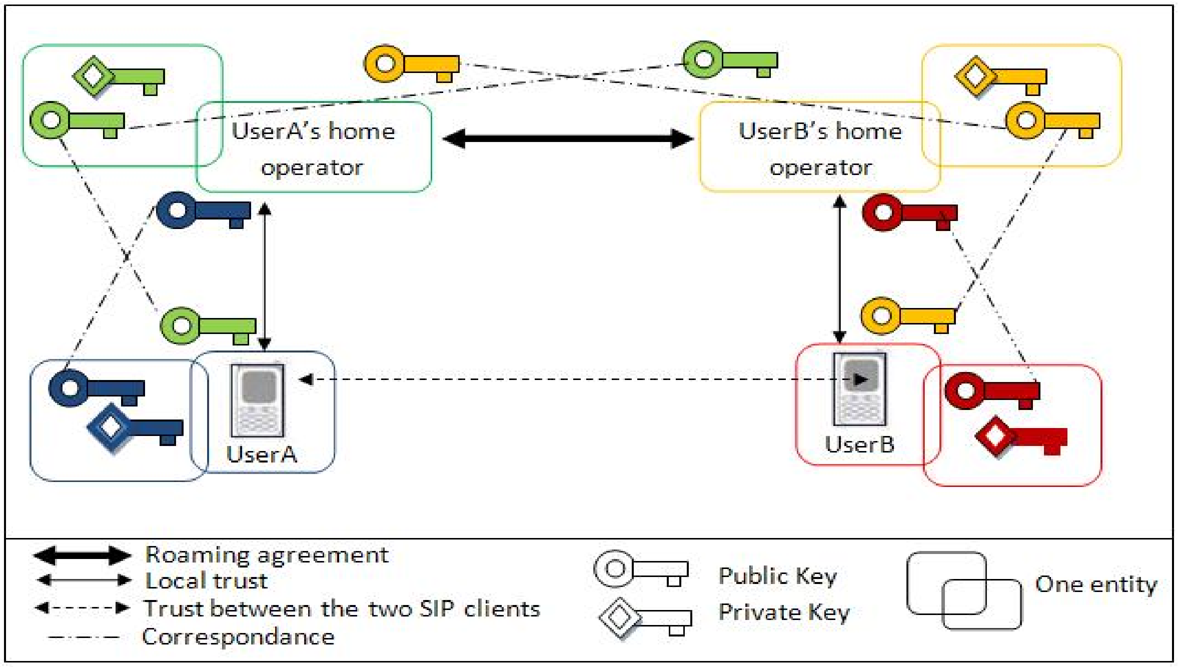
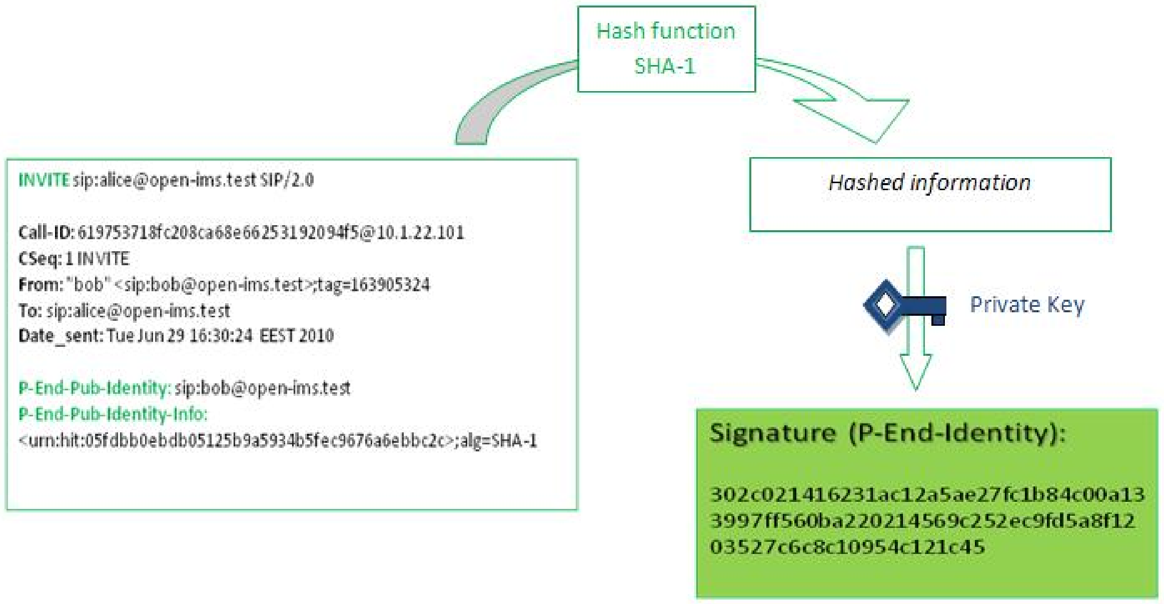


***Figure 2-18.*** *Distribution of the cryptographic keys*

When one entity wants to prove its identity when sending a message, it sends this cryptographic identifier HIT in the message, and protects it by a signature. As shown in the Figure 2-19, this signature is calculated by ciphering the hash of some fields selected in the message with the private key. By doing this, all these fields become protected, which means that the correspondent can be sure of their authenticity. Our interest lies on protecting the following headers: `From`, `To`, `Date_sent`, `Call-Id`, `Cseq`. For example, securing the header `Date_sent` protects the message from replay attacks, while the header `Call-Id` would prevent a third part from modifying the type of message (for example to transform an `INVITE` message in a `BYE` one in order to end the conversation). The new headers containing the HIT (`P-End-Pub-Identity-Info`) and the identity of the user (`P-End-Pub-Identity`) are also protected by the signature to prevent them from being falsified. Their description is defined in the section 4.1.1.



```
INVITE sip:alice@open-ims.test SIP/2.0

Call-ID: 619753718fc208ca68e66253192094f5@10.1.22.101
CSeq: 1 INVITE
From: "bob" <sip:bob@open-ims.test>;tag=163905324
To: sip:alice@open-ims.test
Date_sent: Tue Jun 29 16:30:24  EEST 2010

P-End-Pub-Identity: sip:bob@open-ims.test
P-End-Pub-Identity-Info:
<urn:hit:05fdbb0ebdb05125b9a5934b5fec9676a6ebbc2c>;alg=SHA-1
```

```
Signature (P-End-Identity):

302c021416231ac12a5ae27fc1b84c00a13
3997ff560ba220214569c252ec9fd5a8f12
03527c6c8c10954c121c45
```

***Figure 2-19.*** *Creation of the signature with the private key*

In practice, we modify the messages exchanged between the two users by adding this secured cryptographic identity, so that every user can prove his identity. We also protect the message with a signature. The next paragraphs show the different behaviors of each entity towards these specific messages.

In the first SIP "`INVITE`" message, shown in the Figure 2-20, *UserA* sends his identity information (HIT) and the algorithm used to obtain it (ALG). The contents and the relevant SIP headers are protected with a signature.

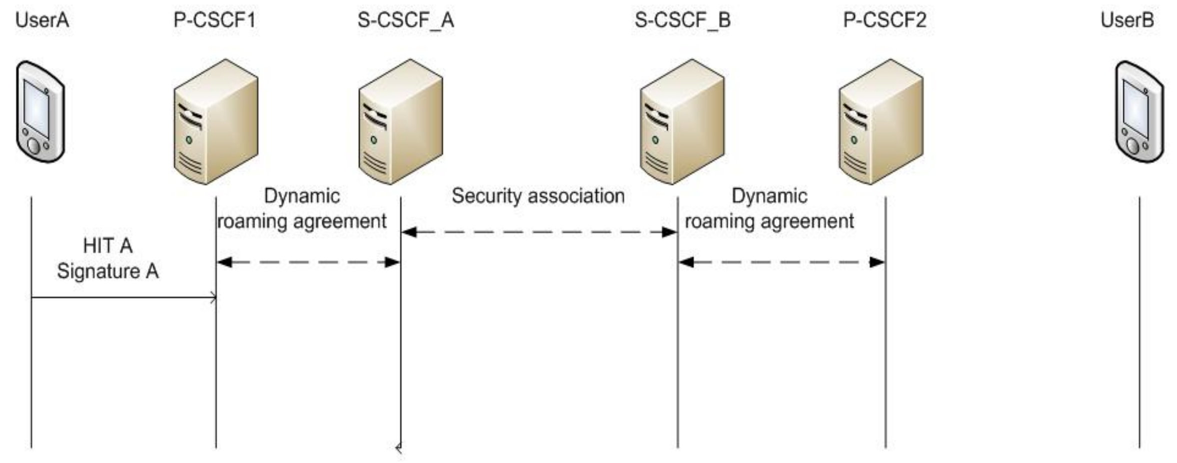


***Figure 2-20.*** *UserA sends the first SIP INVITE message*

In the next phase, presented in the Figure 2-21, the message is received by the proxy *P-CSCF1*. Depending on where the user is, *P-CSCF1* can belong to the user's home network, or to another operator. First, the proxy verifies the message fields, for example, it checks if the user is registered and if all the media parameters fit the current local policy. Then, the proxy does not do much else than forward the message to the user's home network: *S-CSCF_A*. However, it can add some new headers like "`P-Asserted-Identity`" (which represents its own definition of the identity of the user) or "`Record Route`" (where it indicates its own SIP URI) [7]. However, the reliability of this information depends on the existence of security association between the access (*P-CSCF1*) and home operators (*S-CSCF_A*) as it is generally expected that the IMS charging information is exchanged only between trusted networks. Then, the message is forwarded to the home network of *UserA*.

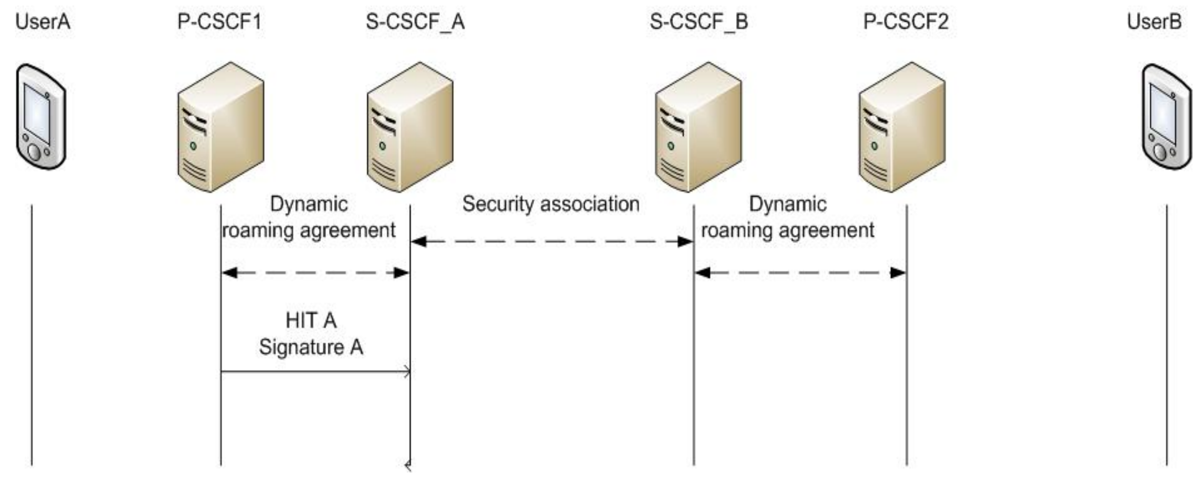


***Figure 2-21.*** *Message INVITE forwarded by the Proxy.*

In the Figure 2-22, when *S-CSCF_A* receives the message, it can check the correspondence of the registered user identities. Of course, it is assumed that the user has previously registered the identities he is using on this connection. He has already shared his cryptographic identity (HIT) as a secret with his operator. There are several ways to share this secret. Either, it can be already recorded in the Universal Subscriber Identifier Module (USIM) card of the SIP phone and in the database of the operator, either it can be shared during the "REGISTER" phase. This second solution is more secured as the shared secret changes every time the user performs a call, but the first one is the easiest to implement because it avoids all the key exchange problems. After this, and the verification of the signature, *S-CSCF_A* can sign user's HIT, which means to add some specific headers in the message (Cf. in the section **4.1.2**). By doing this, the home operator indicates that this user can be trusted at this expressed application level identity. Based on the receiver's Address of Record (AoR), the message is forwarded to the home operator of that entity (*S-CSCF_B*).

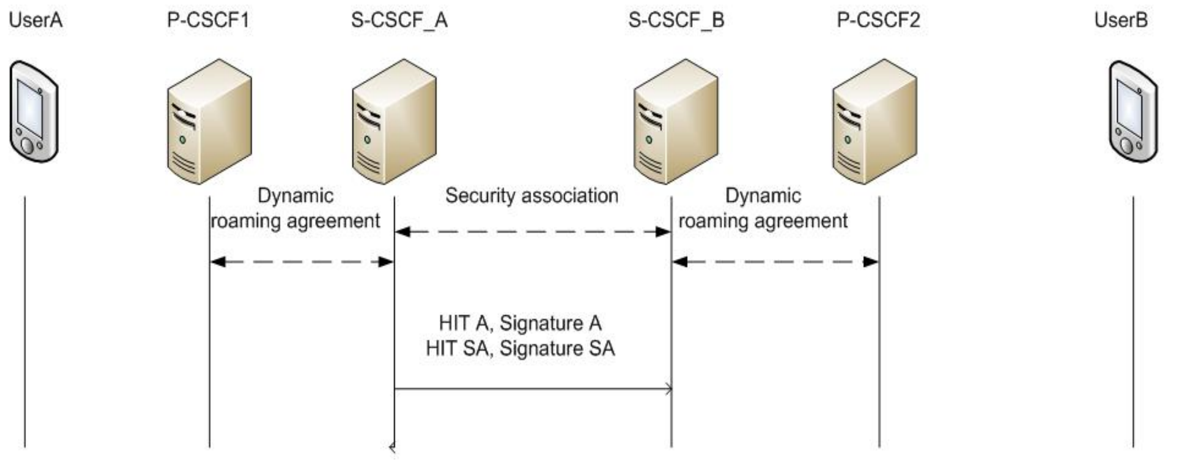


***Figure 2-22.*** *The message is forwarded to the home network of UserB*

The next step, presented in the Figure 2-23, is the reception of the message by the home network of *UserB* (*S-CSCF_B*). This one has a roaming agreement with the home network of *UserA* and has exchanged its identity with it. As a consequence, *S-CSCF_B* can check that the message from the server can be trusted. Then, because the message has been signed by the *S-CSCF_A*, the identity of the sender is assured. Thus, it is willing to forward traffic from this operator to its own subscribers. *S-CSCF_B* proceeds in including its own assurance to the identity of *UserA* as a token of trust it has on the established agreement.

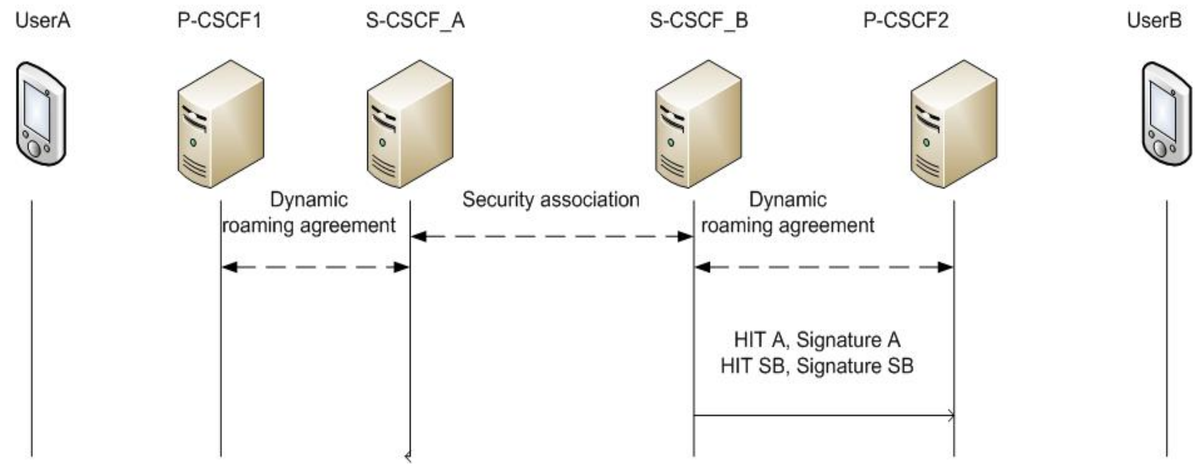


***Figure 2-23.*** *The home network of UserB forwards the message to UserB's proxy*

In the last phase as illustrated in the Figure 2-24, *UserB* receives the message containing the specific headers of trust from its own home operator. It is assumed that the user is in possession of the identity of his home operator, so he can check that he can trust this message by comparing the identity presented in the specific headers added by his home operator and the saved identity. Then, *UserB* can save *UserA's* identity for a later use, and sends the response to the request INVITE with his own configuration information and relevant identities protected by a signature. These new headers added in the message have a double goal: first, they assure *UserB*'s identity to his home operator, and then, they let know this identity to *UserA*, when the message will be forwarded to him.

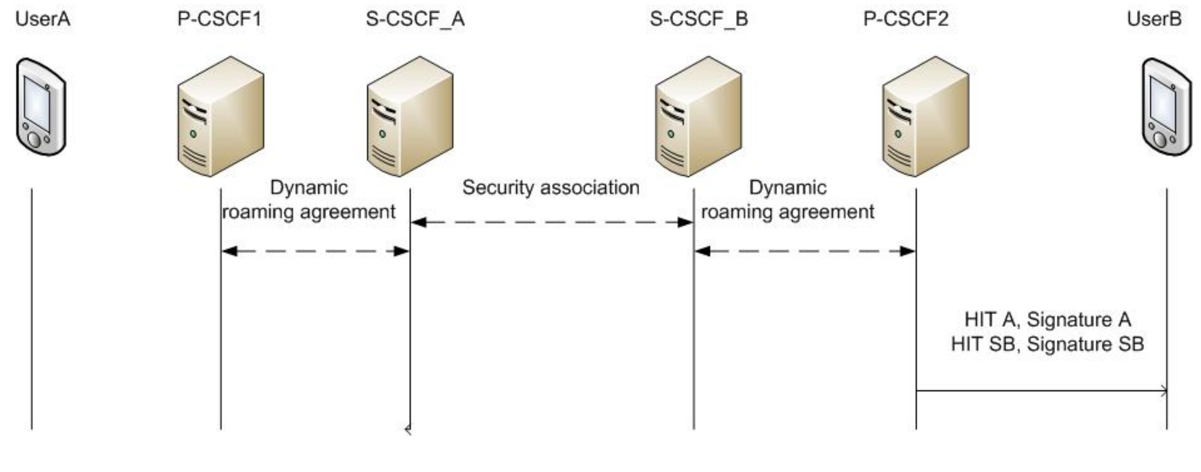


***Figure 2-24.*** *Reception of the message by UserB*

Then, the procedure described in the Figure 2-25 works to the opposite direction in a similar fashion. Indeed, *UserB*'s home operator assures *UserB*'s identity, and based on transitivity, *UserA*'s home operator asserts the received identities. The response is forwarded to *UserA*.

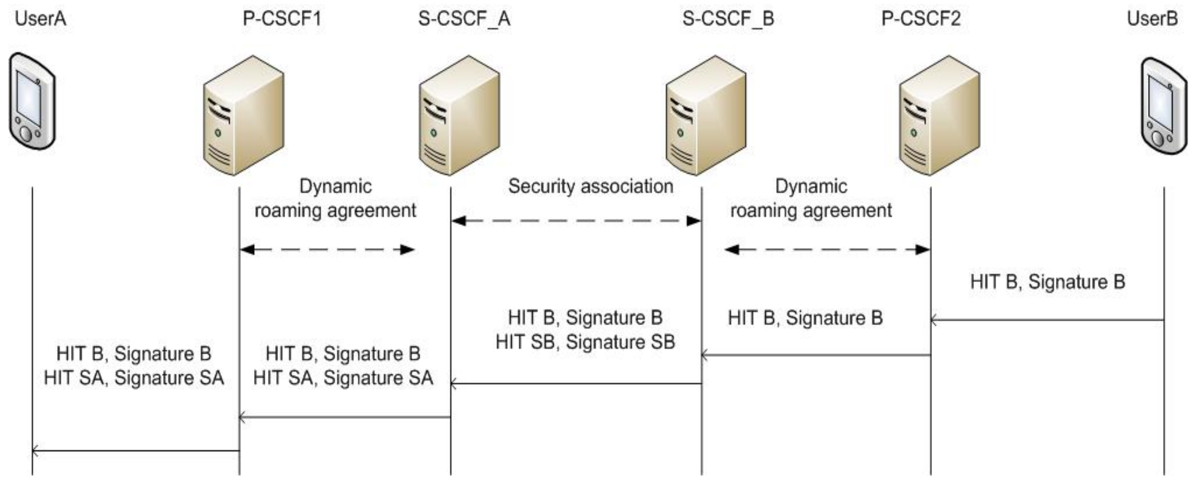


***Figure 2-25.*** *UserB's answer to UserA's INVITE forwarded to UserA*

The Figure 2-26 presents *UserA*'s behavior when receiving this answer. *UserA* concludes the transaction by sending an acknowledgment (ACK) message directly to *UserB* as in typical SIP transaction. In this ACK message, *UserA* includes the HIT representing his secured identity that *UserB* had previously saved when receiving the first message Invite (shown in the Figure 2-26). When *UserB* receives this ACK message, he can check the correspondence of the identities received and saved. When the test confirms *UserA*'s identity, he answers with a similar ACK message containing his own HIT that *UserA* can verify. After this, the identity association is realized; *UserA* and *UserB* can switch to communicate directly on IP layer with the negotiated session parameters as they are in possession of each other's assured identities.

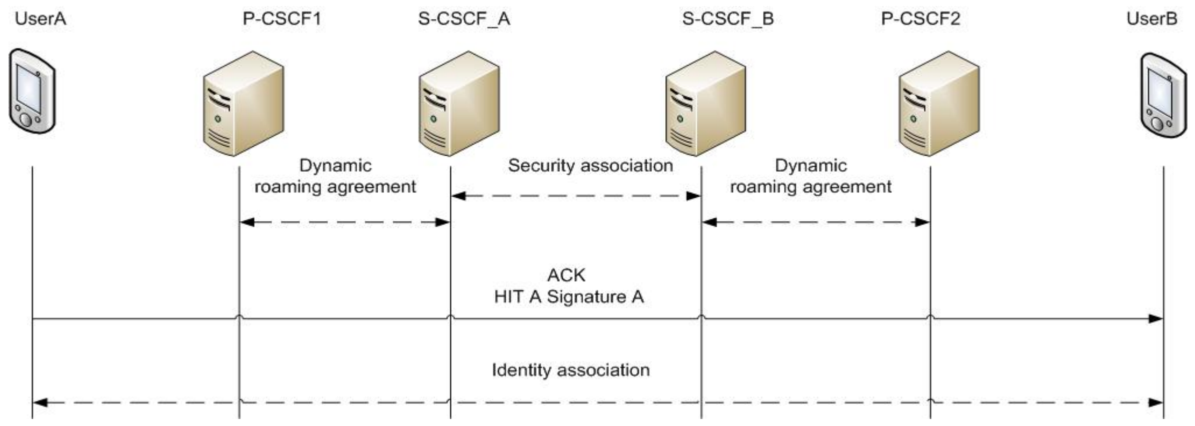


***Figure 2-26.*** *Identity Association*

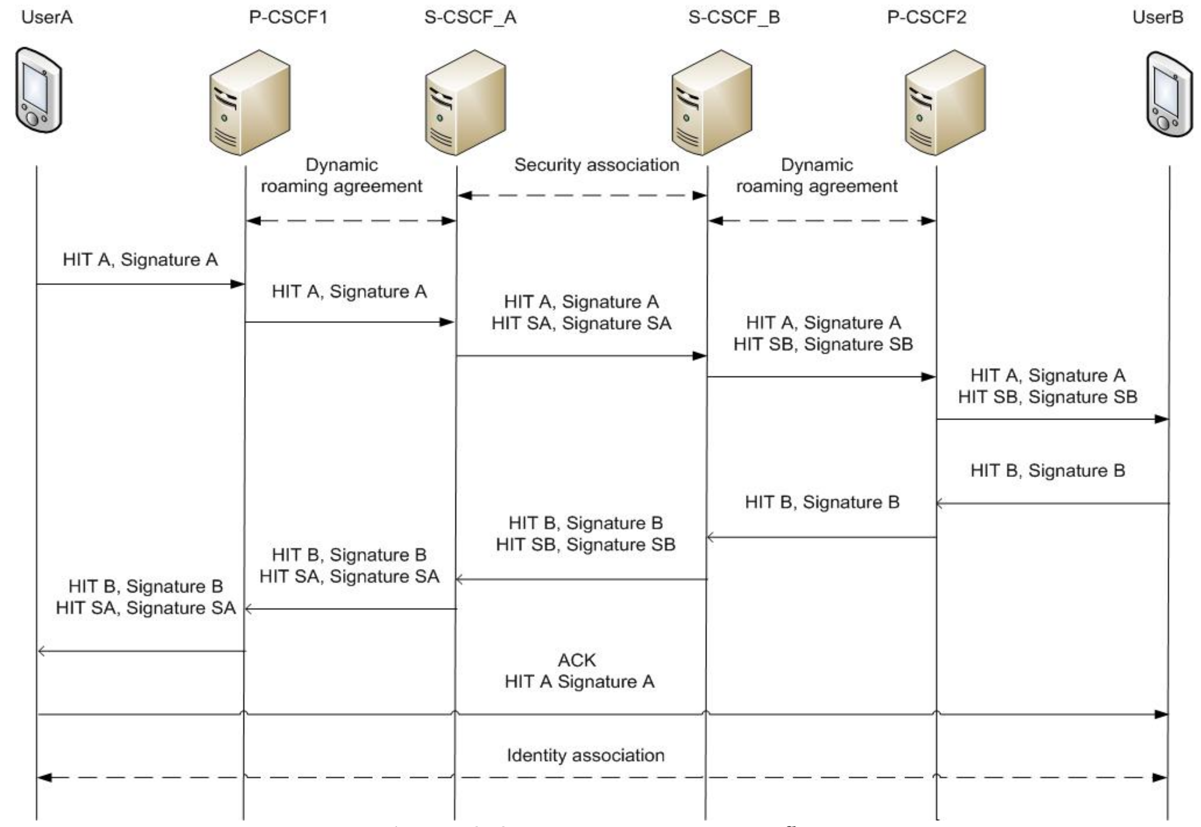


***Figure 2-27.*** *INVITE Message flow*

Finally, the whole mechanism presented in the Figure 2-27 enables to create an identity association between *UserA* and *UserB*. These users are roaming in networks different from their home networks, and can trust each others thanks to the secured identity exchange. This mechanism relies on local trusts and dynamic roaming agreements. It differs from a typical IMS setup by the fact that the relationships between operators are dynamically established, which contrasts with the actual IMS environments, where the relationships are static and trust is based on the reliability of the operator. Even if this scenario isn't yet a reality, it's expected to be more likely in the future ubiquitous environment. [9]

# 3 Software environments and tools used

In this chapter we will introduce different software environments that we have used as background technologies and platforms in order to develop, implement and test this project. The Open Source IMS architecture from FOKUS Institute will be introduced as well as the IMS clients and the application server that have been used to perform the tests. We will also introduce the background tools that we have used such as *Wireshark*, *Eclipse* Integrated Development Environment (IDE) and server access in distance via Virtual Private Network (VPN) and Secure Shell (Ssh).

## 3.1 IMS environment

The Open Source IMS core project has been used as an IMS platform for this work. This project offers all the basic components of an IMS core network that were needed during the thesis work. Even though nowadays there are already many Open Source projects established in the plain VoIP area for SIP clients, proxies, stacks and tools around the IETF SIP standards, but there are practically no Open Source projects with specific focus on the IMS. The *OSIMS* project aims to fill the currently existing IMS void in the Open Source software landscape with a flexible and extendable solution that has already proven its conformance and performance in several national and international Research and Development (R&D) projects. [5]

### 3.1.1 **Open Source IMS Core Network from Fokus**

The *OSIMS* is an Open Source implementation of IMS CSCFs, the central routing elements for any IMS signaling, and a lightweight HSS, which together form the core elements of all IMS/NGN architectures as specified today within 3GPP, 3GPP2 (3rd Generation Partnership Project 2), ETSI TISPAN [27]. It is based on the open source project SIP Express Router (SER) and has been developed by the *Fraunhofer Institute FOKUS* in Germany. The first versions appeared during 2006 and are designed for *Linux*-based platforms.

The sole purpose of this platform is to provide an IMS core implementation reference for IMS technology testing and IMS application prototyping for research purposes, typically performed in IMS test-beds.

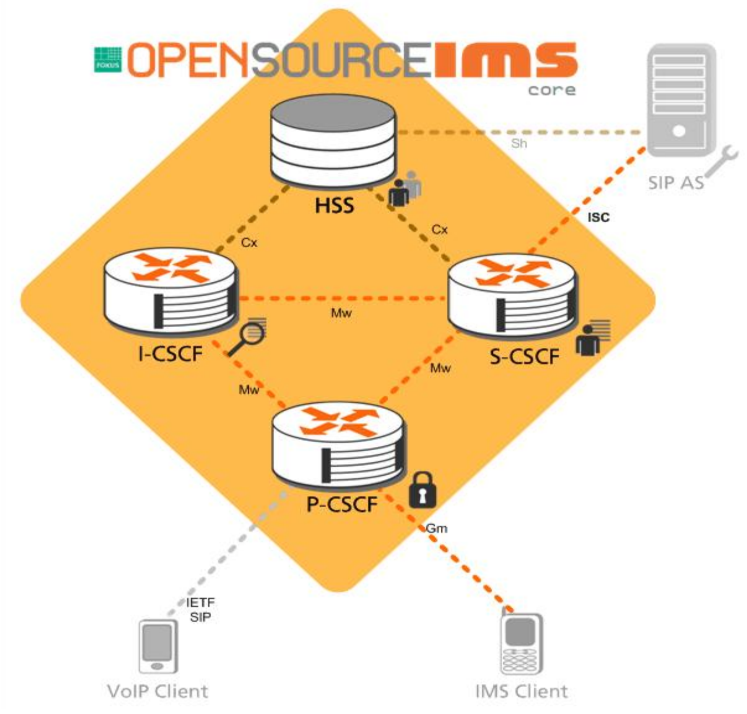


***Figure 3-1.*** *Fokus IMS architecture [5]*

The main functions and interfaces of the *OSIMS* are presented in the Figure 3-1. We will describe them more in detail in the next section.

### 3.1.2 **Implementation of IMS principles in Open Source IMS Core**

This implementation offers all the basic functions of an IMS core network that is, P-CSCF, I-CSCF and S-CSCF. It offers also the HSS function, which is responsible for provisioning users and the associated service profiles in order to deploy different services.

#### *P-CSCF*

In the current implementation of the *OSIMS*, the P-CSCF component is able to firewall the core network at the application level: only registered endpoints are allowed to insert messages inside the IMS network and the P-CSCF asserts the identity of the users. After a successful registration process to an IMS home network, subsequent user messages are forwarded based on DNS information towards the requested IMS home network. In the Figure 3-2, it is shown the modular architecture of the P-CSCF implemented by the *OSIMS*.

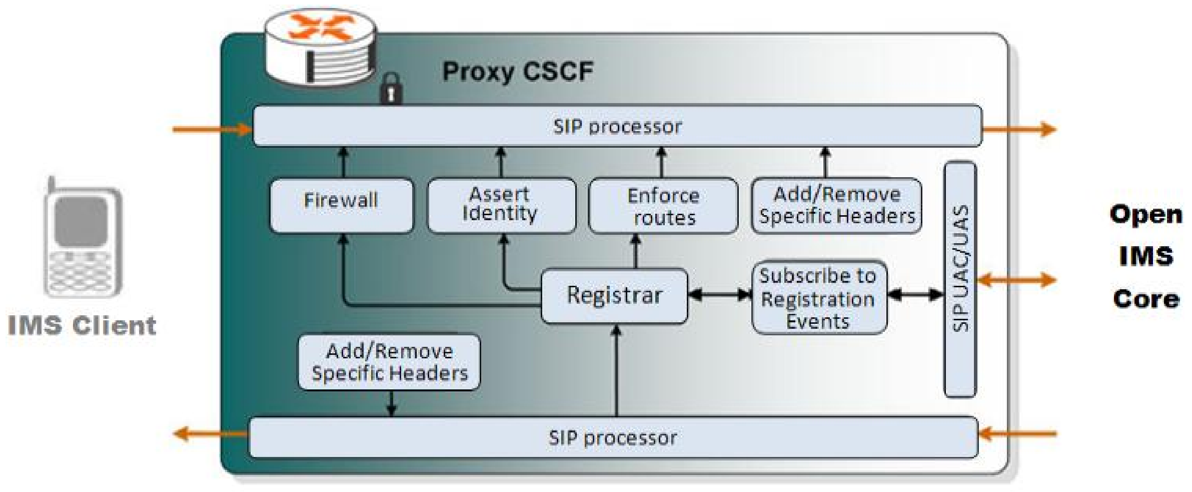


***Figure 3-2.*** *Proxy CSCF [28]*

## *I-CSCF*

The role of the I-CSCF is a stateless proxy that queries the HSS, by using the indicated public identities of the caller or the callee, and based on responses routes the message to the correct S-CSCF. It implements the Cx interface [29] of an I-CSCF to the HSS. Therefore it supports the required Diameter commands to locate the user-assigned S-CSCF or to select, based on capabilities, a new S-CSCF and check identities for roaming authorizations.

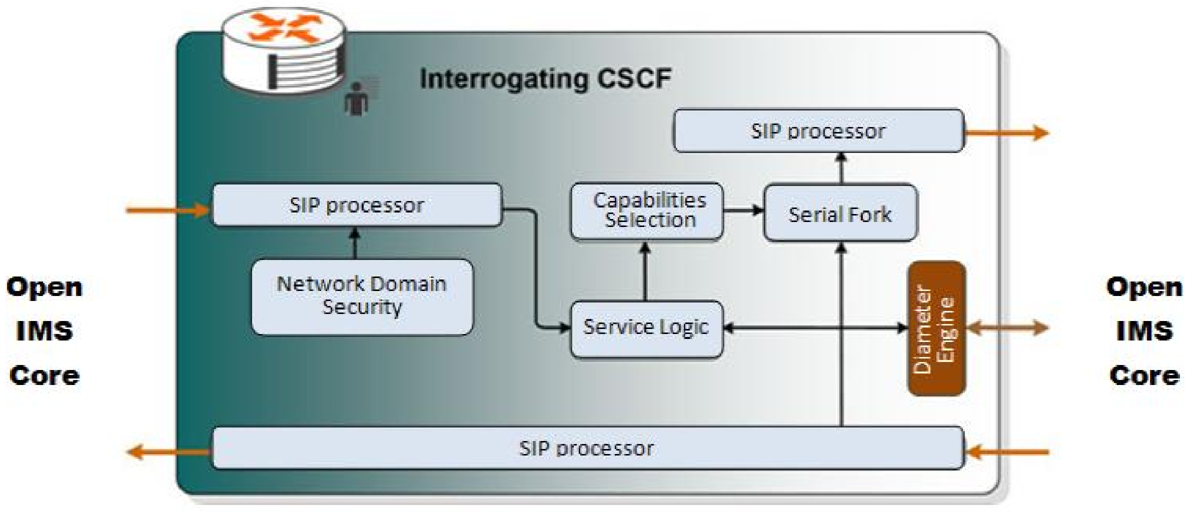


***Figure 3-3.*** *Interrogating CSCF [28]*

After receiving a successful answer for the Diameter query, the I-CSCF forwards the SIP messages in a transactional mode. To protect the home network, it has a firewalling capacity that only allows signaling messages coming from trusted networks. In the Figure 3-3 it is shown the modular architecture of the I-CSCF implemented by the *OSIMS*.

### *S-CSCF*

The S-CSCF implementation also communicates with the HSS using Diameter (over the Cx interface) to retrieve authentication vectors, update registration information and download the user profiles as specified in [29].The S-CSCF can apply specific iFC to enforce specific SIP routing rules. In the Figure 3-4, it is shown the modular architecture of the S-CSCF implemented by the *OSIMS*.

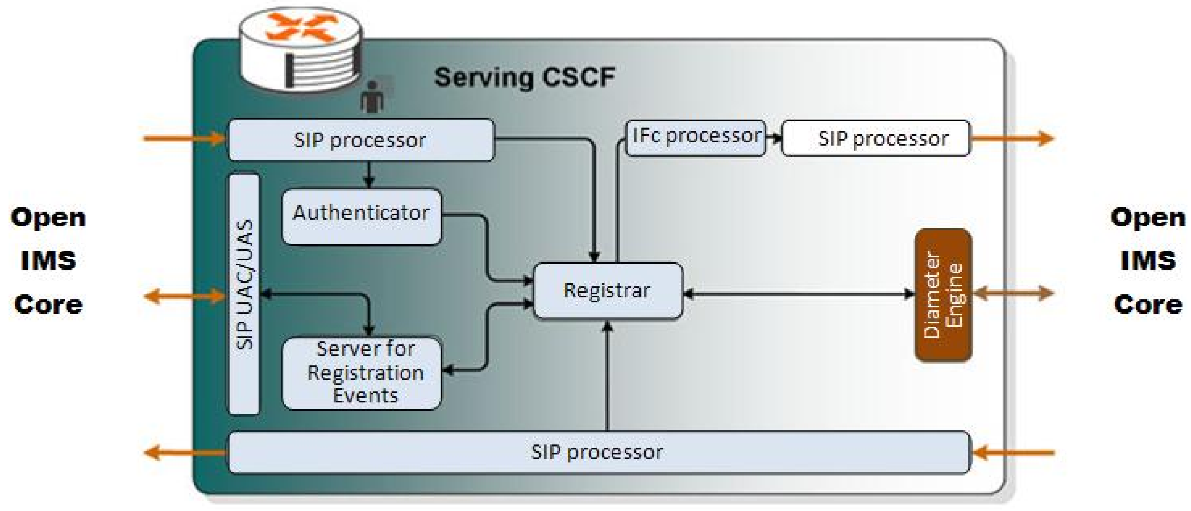


***Figure 3-4.*** *Serving CSCF [28]*

Some of the features of the *OSIMS* S-CSCF related to this project are [28]:

- Authentication through AKAv1-MD5(Authentication and Key Agreement version 1- Message Digest 5), AKAv2-MD5 and MD5
- Download of Service-Profile from HSS
- Initial Filter Criteria triggering
- ISC interface routing towards Application Servers

### *HSS*

The *OSIMS* would be incomplete without a Home Subscriber Server. FOKUS developed its own prototype, the FOKUS Home Subscriber Server (FHoSS), which is entirely written in Java and based upon Open Source software. The user data is kept inside a *MySQL* database. It is mostly a configurator for the Database Management System and the Diameter interfaces with the CSCFs and IMS application layer.

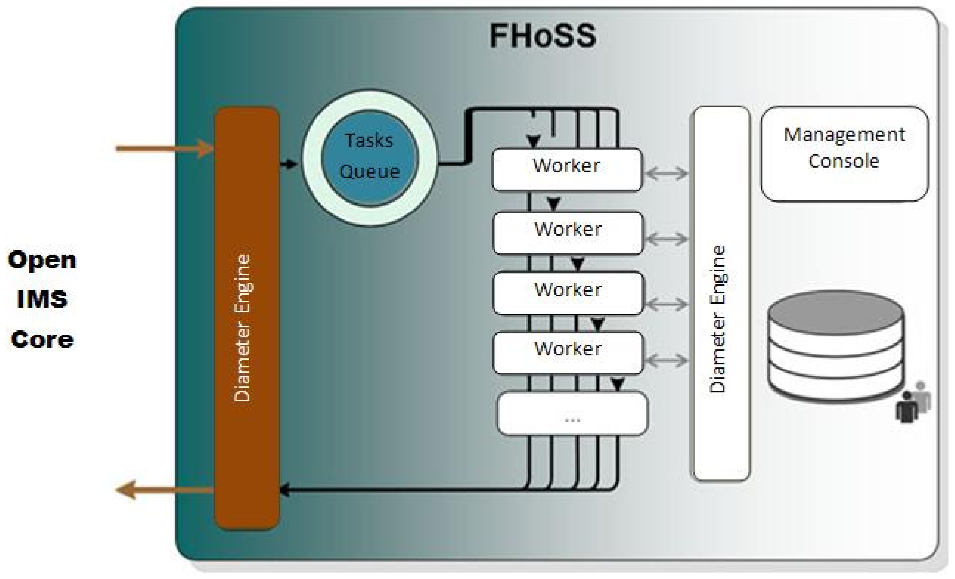


***Figure 3-5***. *FOKUS Home Subscriber Server [28]*

The FHoSS allows the generation of authentication vectors and it provides a HTTP-based management interface for easy management of user profiles and associated iFCs. In the

Figure 3-5 it is shown the modular architecture of the FHoSS implemented by the *OSIMS*. To manage and maintain the FHoSS, a web based management interface is provided. This provides a clear structure and separation of logic and Graphical User Interface (GUI) related tasks. The rendering is done by several Java Server Pages which can be found in the src-web folder.

### 3.1.3 **Prerequisites, installation and configuration of OSIMS**

The installation and configuration steps can be found in the website of the Open IMS project available in [30]. In order to install the key components of *OSIMS* platform some hardware, software and network prerequisites are required.

As hardware requirements, a current *Linux* desktop class machine should be enough but if we want to get ultimate performance, then it is better to have several Gigabytes of Random Access Memory (RAM) and as many Central Processing Units (CPUs)/Cores as needed.

As software requirements, we need to install several modules such as subversion in order to download the fresh code. Different packages such as *GCC3/4*, *make*, *Java Development Kit version 1.5 (JDK1.5)*, *ant* used for Java development are needed to be installed first. In this work we used *MySQL* as a Database Management System (DBMS) which is supported by FHoSS, I-SCSF and other functions requiring a DBMS. Development libraries *libxml* and *libmysql* are required. *Linux kernel 2.6* and *ipsec-tools* are needed to use *IPSec* security. In addition, we use *bind9* as a name server. And we choose *Google Chrome* which can connect to the box as a browser.

As network access requirements, a controllable DNS server should be enough. At the beginning we configured virtual IP interfaces over the loopback. In this way we had a better view while analyzing with *Wireshark*. This was when we configured everything in one *Linux box*. Then we configured two IMS domains and in this case we changed the configuration done previously and configured the two IMS domains over the *eth0* of that machine. We will better explain later the configuration needed in the two cases. The full list of the prerequisites can be found in the IMS Installation Guide [30].

### ***Get the source code***

Once the prerequisites installed, we continue with the installation and the configuration of the FHoSS and then the CSCFs which are the core components of OSIMS. We start by downloading the fresh code of FHoSS and CSCFs at "`http://svn.berlios.de/svnroot/repos/openimscore`" and get the FHoSS source code at "`FHoSS/trunk`" and CSCFs source code at "`ser_ims/trunk`". The source code is pre-configured to work from a standard file path. For that, we create the directory "`/opt/OpenIMSCore`" and then go there to create two new directories "`FHoSS`" to checkout the HSS and "`ser_ims`" to checkout the CSCFs.

```
#>mkdir /opt/OpenIMSCore
#>cd /opt/OpenIMSCore
#>mkdir FhoSS
#>svn checkout
http://svn.berlios.de/svnroot/repos/openimscore/FHoSS/trunk FHoSS
#>mkdir ser_ims
#>svn checkout
http://svn.berlios.de/svnroot/repos/openimscore/ser_ims/trunk ser_ims
```

### ***Compile***

Before compilation, we must be sure we have a version of JDK superior or equals to 1.5. And then, we use *ant* to build and install the FHoSS. Starting from the installation directory of OSIMS we enter in FHoSS directory and then compile the sources of FHoSS and then install it.

```
#>cd FHoSS
#>ant compile
#>ant deploy
```

In similar way, we compile and install the code of CSCFs in the "`ser_ims`" directory.

```
#>cd ser_ims
#>make install-libs all
#>cd ..
```

## *Configure the environment if one IMS domain*

At the beginning, in order to test the project, we decided to configure all the SIP servers, IMS clients and AS in a single machine and the default domain we configured was "`open-ims.test`".

### Configuration of IP virtual addresses over loopback

According to the Installation Guide, the IP addresses of the SIP servers, the FHoSS and the IMS clients, by default, were configured over the loopback address "`127.0.0.1`" with different ports. In order to have a better view of the architecture, we have configured different virtual network interfaces over the loopback. So basically we differentiated the traffic over these virtual interfaces. Then we binded over these virtual interfaces the SIP servers, the FHoSS, the Application Server and the IMS clients according to the architecture presented in the Figure 3-6.

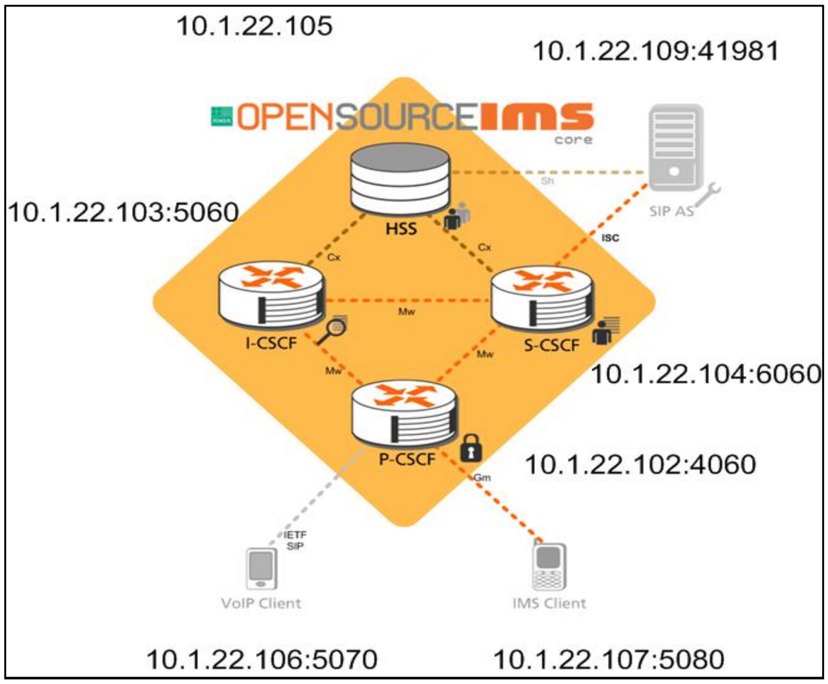


***Figure 3-6.*** *Address planning*

First of all, in order to configure nine virtual interfaces we start by editing the file "`/etc/network/interfaces`". The reader can find the configuration file in **Appendix A**. Here, we proceed by describing the configuration of the DNS server. The domain name that we have chosen is "open-ims.test" according to the Installation Guide. First of all we have to copy to the bind configuration directory "/etc/bind/" the DNS zone file accessible in the "/ser_ims/cfg/" directory.

```
#>sudo cp ~/ser_ims/cfg/open-ims.dnszone /etc/bind/open-ims.dnszone
```

Then we edit the file "named.conf" and insert the DNS zone file there.

```
#>cd /etc/bind/
#>sudo vim named.conf
zone "open-ims.test" IN {
  type master;
  file "/etc/bind/open-ims.dnszone";
  notify no; };
```

Then, the file "/etc/bind/open-ims.dnszone" has to be modified in order to assign the IP addresses to the SIP servers, IMS clients and Application Server. The reader can find in the **Appendix B** how this file should look like for the architecture we present.
Now the DNS server should be restarted and the modification will be taken into account.

```
#>sudo /etc/init.d/bind9 restart
```

In the same time we have to edit the file "/etc/resolv.conf" in order to point to the new installed DNS server.

```
search open-ims.test
nameserver 10.1.22.101
```

We can test if the different servers are resolvable with the command "dig".

```
#>dig pcscf.open-ims.test
```

## Configuring the MySQL

In order to use FHoSS, we needed a database. For this we used the sql scripts that can be found following the installation root directory. These scripts are used to create a *MySQL* database and then to add two users. Also we installed the icscf.sql into the *MySQL* database.

```
#>mysql -u root -p -h localhost < FHoSS/scripts/hss_db.sql
#>mysql -u root -p -h localhost < FHoSS/scripts/userdata.sql
#>mysql -u root -p -h localhost < ser_ims/cfg/icscf.sql
```

## Configure the IMS core

At this stage we should have the DNS and *MySQL* working. As the last step, we have to copy several files from the directory "/ser_ims/cfg" to "/opt/OpenIMSCore", the current directory.

```
#>cd /opt/OpenIMSCore
#>cp ser_ims/cfg/*.cfg .
#>cp ser_ims/cfg/*.xml .
#>cp ser_ims/cfg/*.sh .
```

All these files can be modified to our own preference in order to match the configuration we want to have. If one wants to use the default configuration then there is no need to change the parameters. In the directory "/FHoSS/deploy/" some of the files that can be modified are:

- "DiameterPeerHSS.xml": we need to modify the peer configuration here like the FQDN, Realm, Acceptor Port and Authorized identifiers.
- "hibernate.properties": what we can configure are the main properties for hibernate; implicitly is configured to connect to the *mysql* on the localhost (127.0.0.1:3306). The most relevant properties are: "hibernate.connection.url=jdbc:mysql://127.0.0.1:3306/hssdb" "hibernate.connection.username=hss" "hibernate.connection.password=hss"
- "hss.properties": Specify configuration like on which address the tomcat is listening (e.g. "host=localhost") and the relative path of the web interface of the FHoSS. (e.g. "appPath=/hss.web.console"). Other parameters like "operatorId", "amfId" and "defaultPsiImpi" can be specified here.
- "log4j.properties": Contains configuration for the logger. The most relevant things here are the output file path of the logger and the level of logging.

## *Changes to the Configuration if two IMS domains*

We described in the previous section the configuration related to one domain. As the aim of this project is to implement an assured identity exchange in a roaming scenario, we configured another IMS domain and then integrate it with the first one. We will continue by describing this configuration and explain all the difficulties.

The final version of the project was developed on two distant machines. In this way, the project could be available to everyone who wants to test it or to be base on it for future research. These two machines were in the same local network and their addresses were:

```
ims-a.rd.tut.fi (130.230.141.36)
ims-b.rd.tut.fi (130.230.141.37)
```

On the first machine (`ims-a`) we have installed and configured the first domain named "`open-ims.test`" and on the second machine (`ims-b`) we have installed the second domain named "`open-ims2.test`". In this case we will configure all the SIP servers of a domain on the IP address of that machine. This solution is not the best one for analyzing the traffic with *Wireshark*. This is the reason why we first configured on one box by creating virtual addresses binded over the loopback.

All the SIP server's configuration files should be modified in order to change the IP address configured by default on "`127.0.0.1`". It might be very useful to use the script `configurator.sh`. It can be found in "`opt/OpenIMSCore/ser_ims/cfg/`". This script is based on the function `seed` and takes as arguments the configuration files that we want to modify. The most important files to be modified are:

```
#>sh configurator.sh /opt/OpenIMSCore/ser_ims/cfg/*.xml
#>sh configurator.sh /opt/OpenIMSCore/ser_ims/cfg/*.sql
#>sh configurator.sh /opt/OpenIMSCore/ser_ims/cfg/*.cfg
#>sh configurator.sh /opt/OpenIMSCore/FHoSS/deploy/hss.properties
#>sh configurator.sh /opt/OpenIMSCore/FHoSS/deploy/DiameterPeerHSS.xml
#>sh configurator.sh /opt/OpenIMSCore/FHoSS/scripts/hss_db.sql
```

Once the script executed we enter the new domain name and the new IP addresses of the SIP servers. The script will modify the old domain name and IP addresses with the new ones. The basic configuration of the second domain is the same as the configuration we already described about the first domain. But still there are some changes related to the integration of the two domains. More specifically, particular attention should be paid to the configuration of the DNS. We configured the DNS on the "`ims-b.rd.tut.fi`". Here, we will present the changes done to integer the second domain with the first one.

On the file "`/etc/bind/named.conf`" of the "`ims-b.rd.tut.fi`" we enter two zones that correspond to the first and the second domain. Then in the zone file "`/etc/bind/open-ims.test`" and "`/etc/bind/open-ims2.test`" we configure the IP addresses related to every SIP server and the AS used respectively on the first and the second domain.

```
#>vim /etc/bind/named.conf
zone "open-ims.test" IN{
        type master;
        file "/etc/bind/open-ims.dnszone";
        notify no;
};
zone "open-ims2.test" IN{
        type master;
        file "/etc/bind/open-ims2.dnszone";
        notify no;
};
```

On the "ims-a.rd.tut.fi" we modified the file "/etc/resolv.conf" in order to indicate the new DNS configured on the second machine. On the "ims-b.rd.tut.fi" we modified the same file by indicating the DNS configured on that machine.

```
ims-a#> vim /etc/resolv.conf
search open-ims.test
nameserver 130.230.141.37

ims-b#> vim /etc/resolv.conf
search open-ims2.test
nameserver 130.230.141.37
```

With these basic configurations we were able to integrate two IMS domains. We can test this configuration with the command dig from the "ims-a.rd.tut.fi" as shown in the example. This command proves that the DNS configuration is correct.

```
ims-a#> dig scscf.open-ims2.test

; <<>> DiG 9.7.0-P1 <<>> scscf.open-ims2.test
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47396
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL:
1

;; QUESTION SECTION:
;scscf.open-ims2.test.          IN      A

;; ANSWER SECTION:
scscf.open-ims2.test.   86400   IN      A       130.230.141.37

;; AUTHORITY SECTION:
open-ims2.test.         86400   IN      NS      ns.open-ims2.test.

;; ADDITIONAL SECTION:
ns.open-ims2.test.      86400   IN      A       130.230.141.37

;; Query time: 0 msec
;; SERVER: 130.230.141.37#53(130.230.141.37)
;; WHEN: Mon Aug 30 13:09:37 2010
;; MSG SIZE  rcvd: 87
```

Some other issues should be taken into account when testing our project with two domains. There are some modifications done to the AS in order to specify the new IP address, here "130.230.141.37". Attention is required when configuring the iFC on the second domain. In general, the main procedure does not change.

### *Start the components*

The next step after installation is to start each module of CSCFs: `"pcscf.sh"`, `"icscf.sh"`, `"scscf.sh"` and at the same time. In order to start the FHoSS the `"startup.sh"` is used.

```
#>cd /opt/OpenIMSCore
#>sh ./pcscf.sh
#>sh ./icscf.sh
#>sh ./scscf.sh
#>sh ./startup.sh
```

We should make sure that each component can connect to HSS. And in this process, we could see periodical log messages with the content of the registrar and with the opened diameter links. After that, we can check the web interface on `"http://localhost:8080/"`. In order to connect as admin one should use as username: `"hssAdmin"` and as password: *`"hss"`*.

### *Configure Subscribers*

By default, FHoSS comes provisioned with a couple of sample users:

```
alice@open-ims.test
bob@open-ims.test
```

We can create new users using the web management interface on `"http://localhost:8080/hss.web.console/"`. In order to manage the FHoSS you should use `"hssAdmin"` as user login and `"hss"` as password. In order to create new users there are few easy steps to follow:

- create a Subscription
- create a Private Identity
- create a Public Identity
- link them

For more information follow the Installation Guide [30].

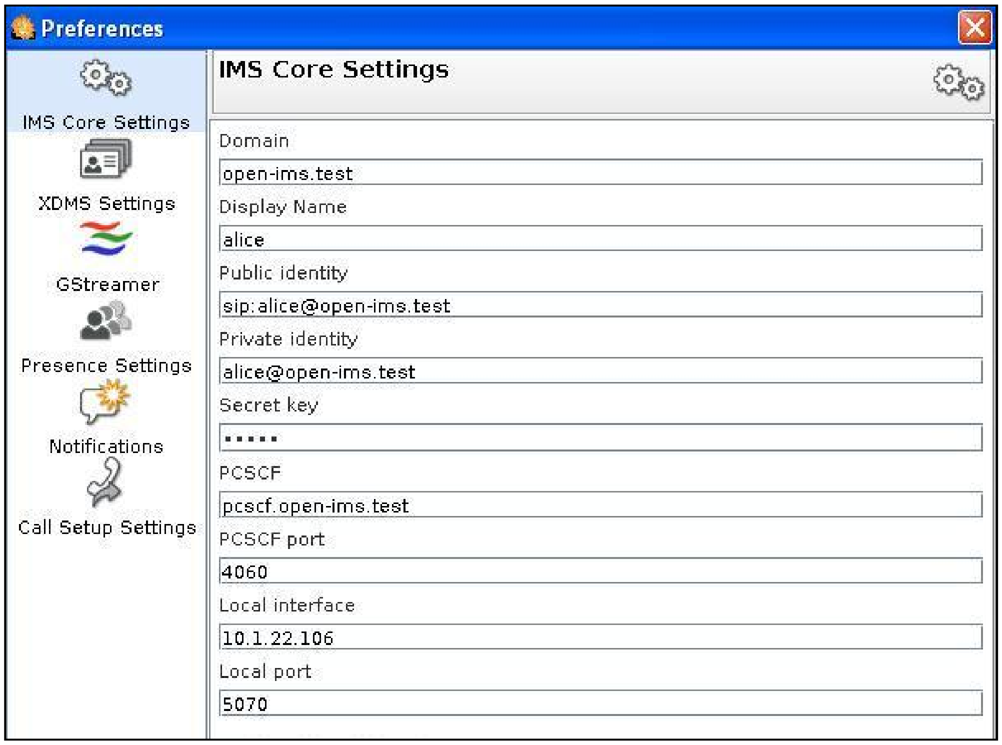
## 3.2 IMS Clients

The IMS Client chosen for this project has to meet some requirements: first, it has to be compatible with the IMS environment and then, it has to be open source, so that we can get and modify its code. The installation guide of *OSIMS* recommends several IMS clients such as *UCT*, *Monster* and *IMS Communicator*. The next sections present them, and explain which one we chose for this project.

### 3.2.1 Clients UCT and Monster

Concerning the IMS *UCT* [31] client, there was a problem of compatibility between the software and the machine on which we developed (i386), so we couldn't use this IMS Client. *Monster* [32] was interesting in the sense that it was developed by the same community that created *OSIMS* and it was totally compatible with the IMS environment. Moreover, it was a simple IMS client with all the basic functions and no complicated features that could lead to mistakes when first launched.

As a consequence, the IMS client *Monster* is ideal for the testing phase of the IMS environment. Even if it was not Open Source, this simple IMS client enabled us to see if the IMS environment was well configured (if the clients could register themselves and then initiate a call with another IMS client). The Figure 3-7 shows the configuration of the *Monster* client needed with the *OSIMS* environment.



IMS Core Settings

Domain: open-ims.test
Display Name: alice
Public identity: sip:alice@open-ims.test
Private identity: alice@open-ims.test
Secret key: •••••
PCSCF: pcscf.open-ims.test
PCSCF port: 4060
Local interface: 10.1.22.106
Local port: 5070

***Figure 3-7.*** *Configuration of the client Monster*

With *Wireshark*, we could follow the call flow and see to which entities the message was forwarded. After verifying that the IMS environment was well configured, we downloaded the sources of the third IMS Client possible: *IMS Communicator* [33]. The next section presents this last client.

### 3.2.2 **IMS Communicator**

*IMS Communicator* was created to fill the need to have an IMS client able to use and test all the new services and convergence scenarios made possible with SIP and the IMS architecture. It was originally developed by *PT Inovaçao* in order to support the development and testing of IMS/NGN components for its Service Handling on IP Networks (SHIPNET) IMS/NGN architecture. Now, it is open to the community as an open source project. It is licensed with the Apache Software License and the GNU Lesser General Public License (LGPL).

Based on a SIP project (SIP-Communicator softphone), *IMS Communicator* is built on top of the JAIN SIP RI, to which contributions were also made to support the IMS SIP extensions defined by 3GPP and IETF. *Java Media Framework* (*JMF*) API provides its media stack. This IMS client is multiplatform compliant and is written in Java.

To fit to the IMS requirements, several development efforts had to be made on the original *SIP Communicator* client. Indeed, it was needed to implement the IMS Registration and Authentication and the IMS Session establishment:

- IMS Registration and Authentication
  - support of IMPI
  - authentication algorithm AKAv1
  - subscription to the "`reg`" event package
- Security Agreement mechanism (no IPSec though)
  - IMS Session Establishment
  - Precondition Mechanism
  - Early Media
  - Call transfer

In practice, we can see that new functions and specific to IMS appear in the java sources (for example, methods to create the different IMS messages like `REGISTER` or `INVITE`). [33]

### 3.2.3 **Installation and configuration of IMS Communicator**

#### *Installation*

To install *IMS Communicator*, some requirements have to be filled:

- JRE 1.5 (Java Runtime Environment) installed
- J2DK 1.5 (Java 2 Development Kit) installed
- *Apache Ant*
- *JMF* API

To get the sources, we can download the *subversion* (*svn)* sources by opening a console and typing:

```
#>svn checkout svn://svn.berlios.de/imscommunicator/trunk/ims-
communicator
```

The *JMF* binaries are not shipped with the source code because of their size. Still, they are not necessary for the basic functioning of this project; they enable to realize voice calls. They can be downloaded as a separated package. As for the project sources, we just have to open a console and write:

```
#>svn checkout svn://svn.berlios.de/imscommunicator/trunk/ims-
communicator/trunk/lib
```

There's support for *Linux* and *Windows*. Thus, as we develop on a *Linux* platform (needed to install the IMS environment), we download the corresponding package and place it in the lib folder inside the *IMS Communicator* project as represented in the Figure 3-8.

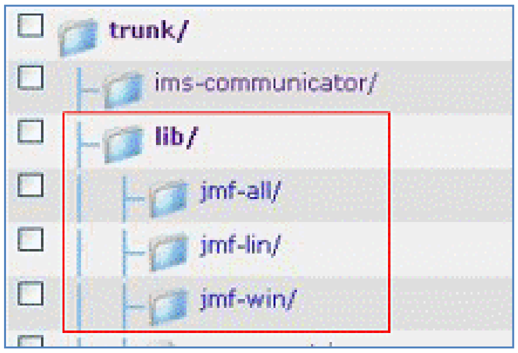


***Figure 3-8.*** *JMF libraries [33]*

## *Configuration*

The configuration of *IMS Communicator* consists in indicating the public identity of the user and the IMS environment that he belongs. All these values can be changed manually in the file "`ims-communicator.xml`". Most of the fields are already well configured by default; some particular fields have to be filled according to the IMS environment configured to enter the user identity and the address of the IMS Proxy.

*Sip:*

```
<PUBLIC_ADDRESS value=sip:bob@open-ims.test/>
<REGISTRAR_ADDRESS value="open-ims.test" />
<DEFAULT_DOMAIN_NAME value="open-ims.test" />
<DEFAULT_AUTHENTICATION_REALM value="open-ims.test" />
<OUTBOUND_PROXY value="pcscf.open-ims.test:4060/udp"   />
```

*Ims:*

```
<PRIVATE_ADDRESS value="bob@open-ims.test" />
<USE_IPSEC value="false" />
<AUTH_ALGORITHM value="MD5" />
```

*Common :*

```
<PREFERRED_NETWORK_INTERFACE value="lo" />
<PREFERRED_NETWORK_ADDRESS value="10.1.22.101" />
```

To launch the project, in the folder /ims-communicator launch:

```
#>ant rebuild
```

Some mistakes may appear which lead to a "BUILD FAILED". It may be necessary to open the files concerned add a library

```
#>vim /ims-
communicator/src/net/java/sip/communicator/sip/simple/XmlPresenceInfor
mationFactory.java

import net.java.sip.communicator.common.Console;

#>vim ims-
communicator/src/net/java/sip/communicator/sip/simple/storage/ContactL
istSerializer.java

import net.java.sip.communicator.common.Console;
```

After this, the command `ant rebuild` should lead to a "BUILD SUCCESSFUL". To launch the client, we just have to write:

```
#>ant run
```

A IMS Communicator window appears and we can register the user whom information is detailed in the file "ims-communicator.xml".

## 3.3 Application Servers

### 3.3.1 Different Application servers

An AS is a server that communicates through SIP and DIAMETER interfaces in order to provide services that can vary from basic telephony services to advanced multimedia services. There are different application servers. Some of them are open source and the others are developed by vendors. The most well known open source ASs are *Mobicents,* *OpenSIPs, Sailfin, Asterisk*. Some of them are C-based such as *OpenSIP* and some of them java-based such as *SailFin*. Basically, they are SIP application containers. They provide to the developers the possibility to deploy their application to be used by the IMS Clients through the IMS architecture. We have chosen as AS, *SailFin* project [34] because it is an open source, java-based AS and because we wanted to develop an application based on SIP Servlet technology. *SailFin* provides the possibility to deploy SIP applications.

### 3.3.2 **SailFin Application Server**

*SailFin* project is based on robust and scalable SIP Servlets technology on top of a deployment-quality, Java EE-based *GlassFish*. This project is contributed by Ericsson. *SailFin,* currently, provides *JSR 289* compatibility, high- availability, and clustering, and is integrated with the existing *GlassFish* services. It adds the SIP Servlet Container based on JSR 289 to *GlassFish*. Given the functionality, the *SailFin* codebase requires *GlassFish* as its underlying runtime. In the Figure 3-9 we can find the module architecture of *SailFin*. As we already mentioned, the Project *SailFin* provides a SIP Container for deploying SIP applications and a Web container for deploying different WEB applications.

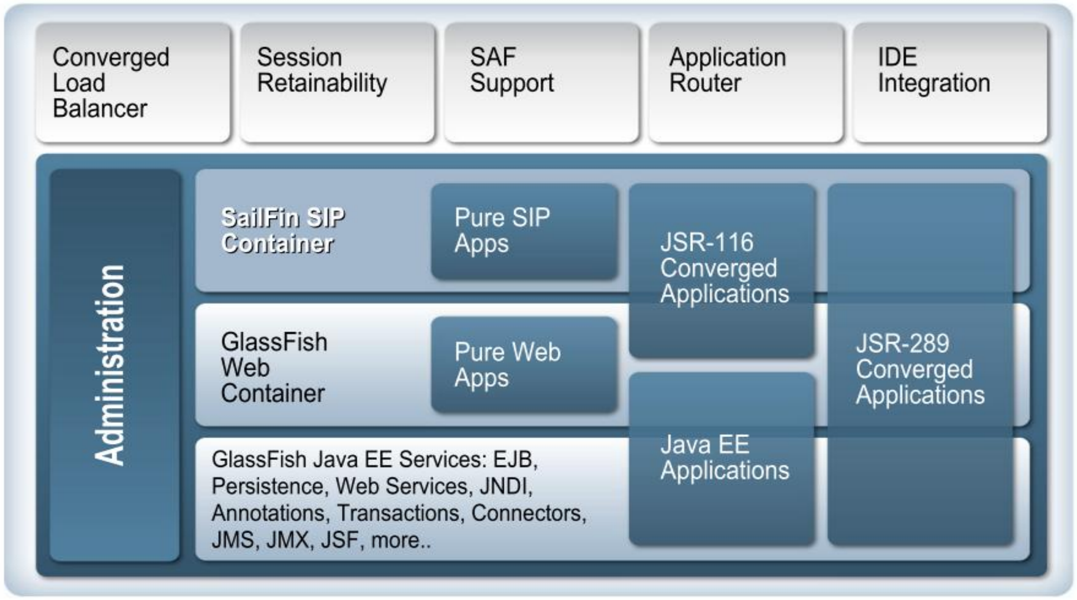
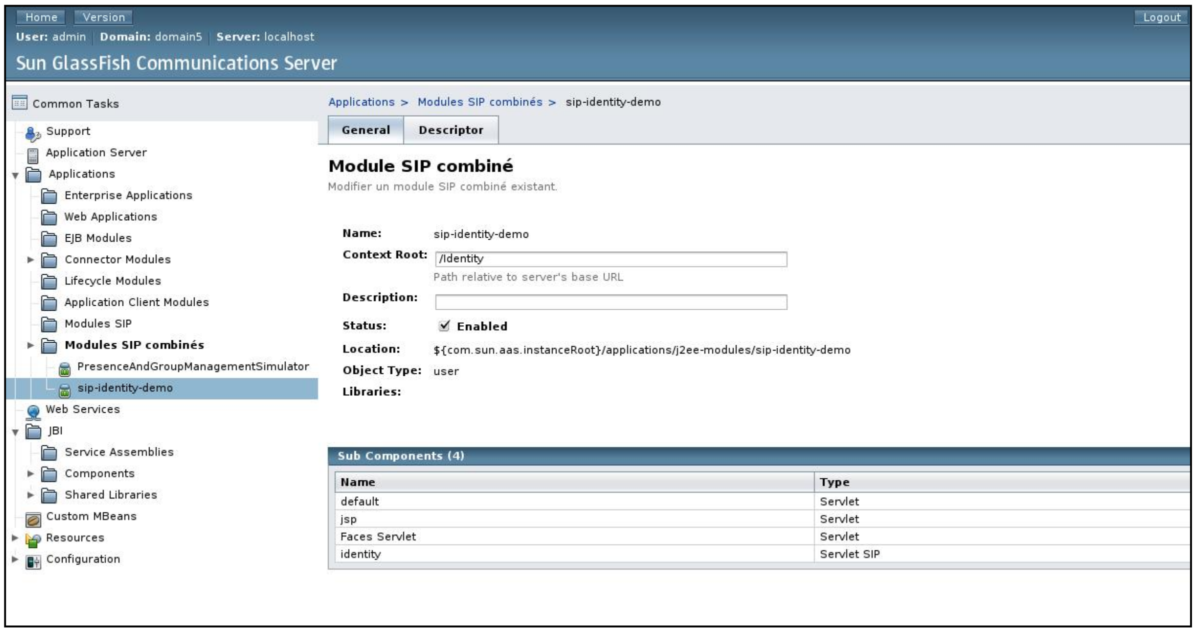


***Figure 3-9.*** *SailFin Architecture*

The *SailFin* project offers a Web console for administration purposes. This is shown in the Figure 3-10. This Web console offers the possibility to configure different parameters such as SIP listening ports and HTTP listening ports. Deploying a SIP Servlet Application becomes even easier with the Web administration console.



Module SIP combiné

Modifier un module SIP combiné existant.

| | |
|---|---|
| Name: | sip-identity-demo |
| Context Root: | /identity<br>Path relative to server's base URL |
| Description: | |
| Status: | ✓ Enabled |
| Location: | ${com.sun.aas.instanceRoot}/applications/j2ee-modules/sip-identity-demo |
| Object Type: | user |
| Libraries: | |

Sub Components (4)

| Name | Type |
|---|---|
| default | Servlet |
| jsp | Servlet |
| Faces Servlet | Servlet |
| identity | Servlet SIP |

***Figure 3-10.*** *WEB administration console*

## 3.3.3 Installation and configuration of SailFin

In this section we describe the general procedure how to provision any application server within *OSIMS* infrastructure. We describe how to install and configure *SailFin*. Later on, we explain how to deploy and run a SIP Servlet application on *SailFin*. First, we start by describing the *SailFin* installation and configuration. SailfinV2 can be found in [35]. Once we downloaded the source code we can extract the archive.

```
#>java -Xmx256m -jar sailfin-installer-v2-b31g-Linux.jar
#>cd sailfin
```

We can edit the file "setup.xml" according to our needs and then we can build and run *SailFin*. For instance, we have named the domain as "domain1" and have started it and the database as follows:

```
#>chmod -R +x lib/ant/bin
#>lib/ant/bin/ant -f setup.xml
#>./bin/asadmin start-domain domain1
#>./bin/asadmin start-database
```

The next step in the configuration of *SailFin* consists in changing the default SIP listener and external ports. SIP listener are ports on which SIP Servlets listen to incoming SIP requests and external ports are ports from which it sends outgoing requests. The ports for SIP traffic and HTTP traffic have to be chosen carefully in order not to be in conflict with the ports already dedicated to the IMS components. The configuration can be changed through web interface, by default on port 4848, or from the console with the usage of *asadmin* tool.

After having installed and configured *SailFin* we can easily deploy the SIP Servlet applications. All we need is to open a web interface and navigate through: `Common Tasks → Applications → Converged SIP Modules → Deploy` and specify the following:

- `Location` – path to the application war, on local file system
- `Application name` – any application name
- `Status enabled` – on, automatically starts the application
- `Run verifier` – on

We can use the command line version by using the `"build.xml"` application file. For this we enter in the application directory and use *ant* after having compile the application.

```
#>cd /sailfin/samples/sipservlet/Identity
#>ant compile
#>ant deploy
```

The *SailFin* instance hosts SIP Servlet application and is ready to be plugged into IMS infrastructure. We will describe the procedure to provision the application server supporting ISC interface within *Open Source IMS Core* infrastructure when we will discuss about the implementation of the project in the section **4.3**.

## 3.4 Tools used

### 3.4.1 Wireshark

*Wireshark* [36] is an open source network packet analyzer. It captures network packets and tries to display that packet data as detailed as possible. Some of its different features are:

- capturing live packet data from a network interface
- displaying packets with detailed protocol information
- opening and save packet data capture
- filtering packets on criteria (protocol, source, destination)
- searching for packets in many criteria
- showing the call flow of messages between different entities communicating

In this work, these tools were very useful because they enabled us to observe every SIP message sent between the different entities and help us to debug the program. The installation is traditional. We can download the software from the website or we can download it via *Linux* package synaptic manager:

```
#>apt-get update
#>apt-get install wireshark
```

Once the *Wireshark* is installed, we can launch it from a window. With *Wireshark,* one has the possibility to capture on one of the different network interfaces. As in this project we have installed everything in one box, we selected the loopback interface.

By default, *Wireshark* does not interpret the DIAMETER protocol. If the DIAMETER messages are not present on the traces then it is probably because the used ports in the *OSIMS* haven't been associated to the protocol. In order to associate them one has to go through:

```
Edit > Preferences > Protocols > Diameter
Set up the DIAMETER TCP ports = 3868,3869,3870
```

*Wireshark* offers the possibility to have a better view of the trace captured. This is the coloring rules. You can also write your coloring rules in a file and then import it to *Wireshark*.

```
View > Coloring Rules
```

## 3.4.2 **Eclipse IDE**

*Eclipse* [37] is a multi-language software development environment written in Java. It is composed of an integrated development environment and an extensible plug-in system. It helps us to develop applications in Java and other languages (C/C++, python, php) thanks to its numerous functions: syntax coloration, completion, debug, project management…). The installation of this software is also traditional: it can be found on the website or can be downloaded for console writing:

```
#>apt-get update
#>apt-get install eclipse
```

# 4 Implementing a secured identity exchange based on IMS

In this chapter, we will present the concrete implementation of the secured identity SIP exchange mechanism presented in the section **2.7.** We will first describe the way the work was organized and shared. Then, we will describe more in detail the different parts of the implementation. The first task deals with the modification of the IMS Client, which presents the different functions created in the *IMS Communicator* open source client. Then, we will present the principle of filter criteria on the HSS. We will finish with the development of a SIP Servlet application for securing the identity exchange in the AS. The reader can find in the **Appendix E** how to test the project on his own platform.

## 4.1 Organization of the work

This project was done as joint thesis. The most logical way to implement this project was to process step by step, making sure that the previous step worked before trying to develop the other step. We followed the message flow presented in the project in the Section 2.7. These are the main steps of this implementation. They are not ordered in time, because as previously mentioned, they were realized in a parallel way:

A. Modify the first `INVITE` message sent by the caller
B. Modify the behavior of the different SIP servers in the IMS Core Network towards the new headers present in the incoming message
C. Implement an SIP Servlet application responsible for assuring the identity exchange on the S-CSCF
D. Modify the behavior of the callee towards this modified message

A representation of this planning is presented in the Figure 4-1.

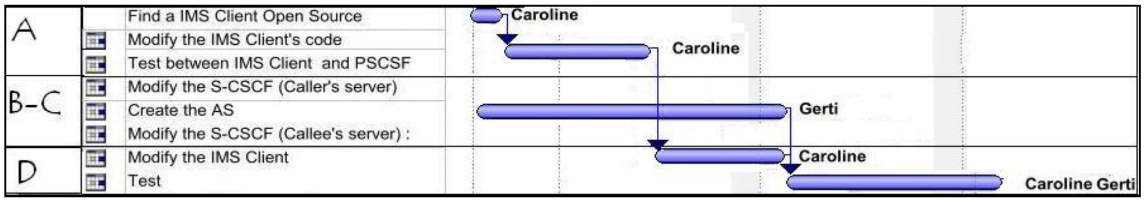


***Figure 4-1.*** *Planning of the project*

Concretely, we divided the work in two, so that we could work in parallel.

### 4.1.1 **Caroline: Modification of the IMS Client**

Caroline's work was about changing the behavior of the *IMS Communicator* used as the IMS client. She first implemented the exchange identity mechanism only between two IMS clients: *UserA* and *UserB*. In this first step, the IMS Core Network didn't check the identities and just forwarded the messages to the addressee. Moreover, in this simple case, *UserA* and *UserB* belonged to the same domain.

- **Task A:** Its representation is given in the Figure 4-2.

  a. *UserA* adds his identity and his signature in the first message "`INVITE`"
  b. The core IMS just forwards it to *UserB*

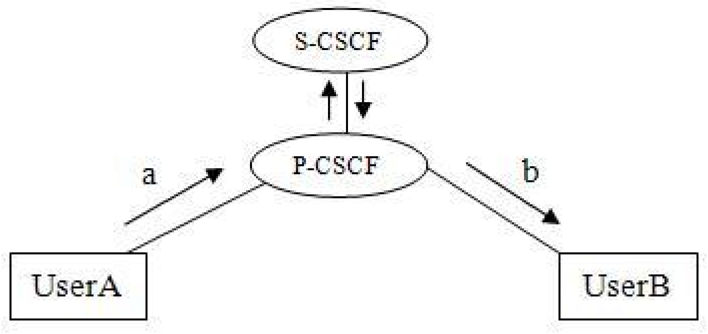


***Figure 4-2.*** *Task A: UserA sends a request INVITE to UserB*

- **Task D:** Its representation is given in the Figure 4-3.

  c. *UserB* verifies the identity and the signature sent by *UserA* and adds its own identity and signature in the SIP response message
  d. The core IMS just forwards it to *UserA* who checks *UserB*'s identity and saves it

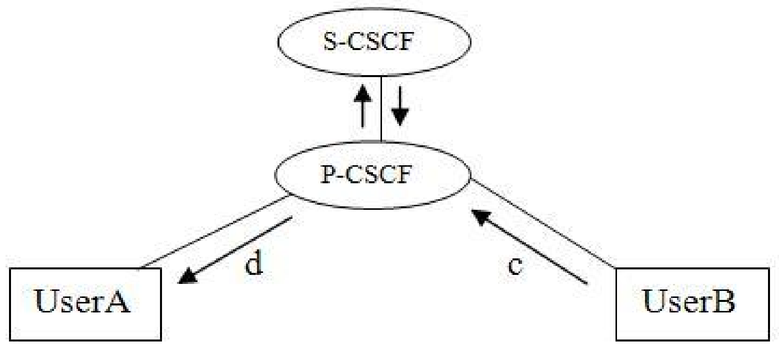


***Figure 4-3.*** *Task D: UserB answers to UserA's INVITE message*

In this case, the implementation is not final: *UserB* was not supposed to know the identity of *UserA*. It is the server which is supposed to check *UserA*'s identity and signature, and then add his own identity and signature that *UserB* checks. But, at this state of the project, the fact that *UserB* is able to check *UserA*'s identity is a good point. When the IMS Core Network would take charge of processing of *UserA*'s new message headers (after the steps B-C), it would very easy to switch *UserB* to verify the server's identity rather than *UserA*'s identity.

### 4.1.2 **Gerti: Modification of the Core Network and AS**

Gerti's work was about the modification of the IMS Core Network and the installation of an Application Server (tasks B-C). The AS is one element that does not exist by default in the *OSIMS* project, so he had to first familiarize himself with this entity, and the technologies associated. Firstly, he started by installing and configuring a basic AS which could communicate with the *OSIMS* (task D). The next step was the modification of the different filters at the level of the S-CSCF so that the SIP messages containing specific headers could be forwarded to the AS (task C). Once the task A was completed (the IMS Client sends a SIP `INVITE` with specific headers containing *UserA*'s identity), the test and sign functions specific to the servers could be implemented in a SIP application deployed in the AS. In this case, *UserA* and *UserB* still belonged to the same domain.

- **Tasks B & C**: Its representation is given in the Figure 4-4. The steps already implemented in the previous steps are represented in grey; the new implementations are in black.

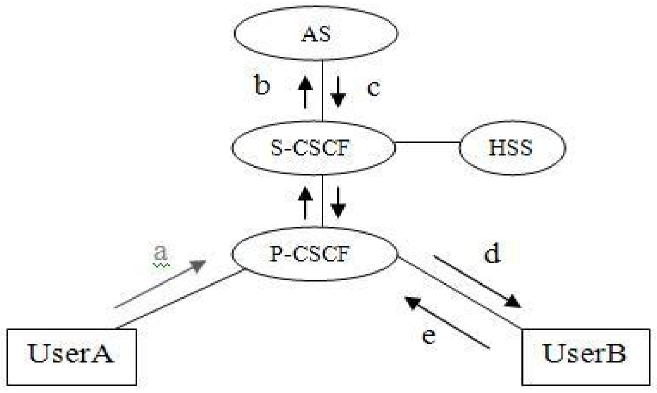


***Figure 4-4.*** *Tasks B & C: Implementation of the AS*

a. Based on the iFC, the S-CSCF detects the specific headers (containing the caller's identity and signature) and forwards the message to the AS.

b. The AS realizes the tests of user's identity and signature. When this user's information is verified, it adds its own identity and signature (or rather, the ones of the home operator) and forwards the message to the S-CSCF.
c. The server forwards it to the proxy which forwards it to *UserB*.
d. *UserB* is modified to check the server's identity and signature, and if they are verified, he saves *UserA*'s identity.

Then, the mechanism has a similar functioning in the opposite way: *UserB* creates the SIP response and adds his own identity and signature, forwards the message to the P-CSCF proxy, which forwards it to the S-CSCF. This one, detecting the specific headers, sends the message to the AS. This one checks *UserB*'s identity and signature, and then returns the message so that it can be forwarded to *UserA*: S-CSCF -> P-CSCF -> UserA.

### 4.1.3 **Next step: Creation of two domain operators**

Once the mechanism tested for one operator domain with two users, it is interesting to implement a scenario where the two SIP users belong to two different domains. In this case, the roaming agreement between the home operators is necessary to carry the SIP signaling through the two networks. We could implement a way to apply the principles of the roaming agreement: the servers have to add and verify their identities and signatures. So, the interesting thing in this scenario is that it comes closer to the real situation that we described in the chapter **4.4**.

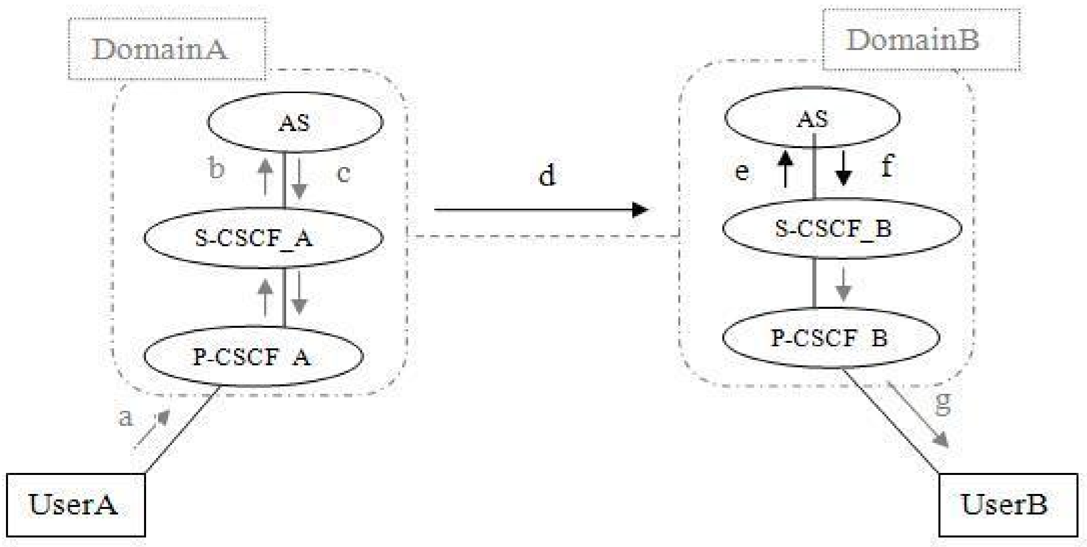


***Figure 4-5.*** *Request INVITE between two users belonging to different domains*

A representation of this implementation is given in the Figure 4-5.

a) After consultation of the I-CSCF, the server forwards it to the addressee's home network (*UserB* belonging to *DomainB*). The message goes through the network, reaches the proxy and is forwarded to the addressee's serving server: *S-CSCF_B.*
b) Based on the iFC, the server *S-CSCF_B* detects the specific headers (containing the caller's home operator identity and signature), and forwards the message to the second domain AS.
c) As the home operator of the addressee (*DomainB*) has a roaming agreement with the caller's home operator (*DomainA*), the AS (belonging to *DomainB*) can check its identity and its signature. Then, it returns the message to the *UserB*'s S-CSCF.
d) The server sends it to the proxy P-CSCF_B which forwards it to *UserB*.

Then, the process works exactly in the same fashion in the opposite way. This is presented in the Figure 4-6.

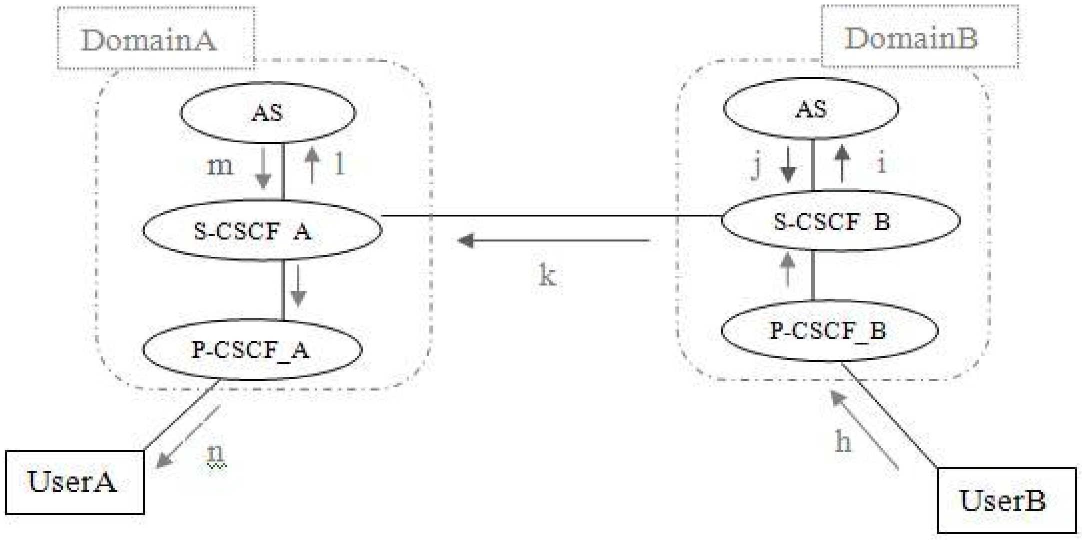


***Figure 4-6.*** *Answer to an INVITE between two users belonging to different domains*

a) *UserB* checks his server's identity and signature, and if they are verified, he saves *UserA*'s identity. Then he creates a new SIP response adding his own identity and signature.
b) *UserB*'s identity and signature are checked by the server in *DomainB*, and if they are verified, the identity and signature of *DomainB* are added.
c) The message is forwarded to *UserA*'s home operator, the *DomainA*.
d) The identity and signature of *DomainB* are verified, and if they are, the identity and signature of *DomainA* are added.
e) The message is forwarded to *UserA*, which checks his server's identity and signature and, if they are verified, he saves *UserB*'s information.

## 4.2 Modification of the IMS Client

As previously explained, we chose to develop the *IMS Communicator* client to implement the project. Before any modifications, the client was able to register to the IMS Core and have a multimedia call with another IMS client. All the features and functionalities proposed by this client can be found in [7].

In this project, we needed to modify the behavior of the client according to the secure exchange of identities mechanism previously described in [9]. "Modifying its behavior" means modifying the SIP messages that the client sends at different steps of its operation:

**1)** when sending an `INVITE` request (*UserA* in the Task A)
**2)** when receiving an `INVITE` request modified (*UserB* in the Task A)
**3)** when sending a response to the `INVITE` request (*UserB* in the Task D)
**4)** when receiving this response (*UserA* in the Task D)

For all of these steps, the same methodology was employed to implement the changes:

- understanding the IMS client way of working. This means which classes and methods are used in this step
- changing the code to apply the modification
- testing the new code and analyzing the messages sent using the log files and *Wireshark*

### 4.2.1 Sending an INVITE request

The first step was to find "where" the message `INVITE` was produced in the IMS client. This means, which classes and which methods were involved when a user wants to call another user. For this purpose, the easiest way was to analyze the log files. All the actions performed by *IMS Communicator* are recorded in a specific file that can be founded in the folder `"/log"`. However, to see all the debug information, we had to modify a parameter in the file `"ims-communicator.xml"` and set the trace level, which was fixed by default at 16 (stack log), to 32 (stack log + debug log).

```
<TRACE_LEVEL value="32" />
```

A new log file `"ims-communicator.debug.log"` appeared in the folder `"/log"`. We found out that when the client sends an `"INVITE"` request, the creation of the message was realized by the method `Invite` in the class `"/ims-communicator/src/net/java/sip/communicator/sip/CallProcessing.java"`.

We could see in the "INVITE" method that to create the different headers that compose the "INVITE" message, we always had to follow the same logic:

- create the header (for a new header use "javax.sip.header.ExtensionHeader")
- test if the header's syntax was correct
- add the header created to the object of type "javax.sip.Request" called "INVITE".

For the sake of commodity, we created a new java class which contains all the necessary functions to calculate and verify the different identities and the all the attributes of the client: "Security_ims_project_tools.java". This class is situated in the directory: "/ims-communicator/src/net/java/sip/communicator/sip/security"

We created two functions for calculating the HIT (GETHIT) and the signature (GETSIGNATURE), and some other auxiliary functions (GETPUBLICKEY, GETPRIVATEKEY) and parameters (String algorithm="SHA-1", PublicKey, PrivateKey):

### METHOD GETHIT

The method GETHIT takes in parameter the client's Public key, applies a SHA1 function and returns a String which contains the hash code generated: HIT. The HIT is defined as the cryptographic identifier, and is get from the hash on the public key.

We previously had generated different pairs of public/private keys for each entity: Alice, Bob, Server_Alice and Server_Bob. For example, for Bob, we created two files: "public_key_file_bob" and "private_key_file_bob" containing the corresponding cryptographic keys that we placed in the root folder ("/ims-communicator").

These cryptographic keys, which were saved in files, are then read by the function GETPUBLICKEY. This method takes in parameter the localization of the file containing the key, and returns an object of the type "PublicKey" defined in the package "java.security".

Then, we used the functions defined with the object "java.security.MessageDigest" to create the Hash of the public key. It is in the creation of this object that the variable "algorithm=SHA-1" was used to define the hash algorithm.

### METHOD GETSIGNATURE

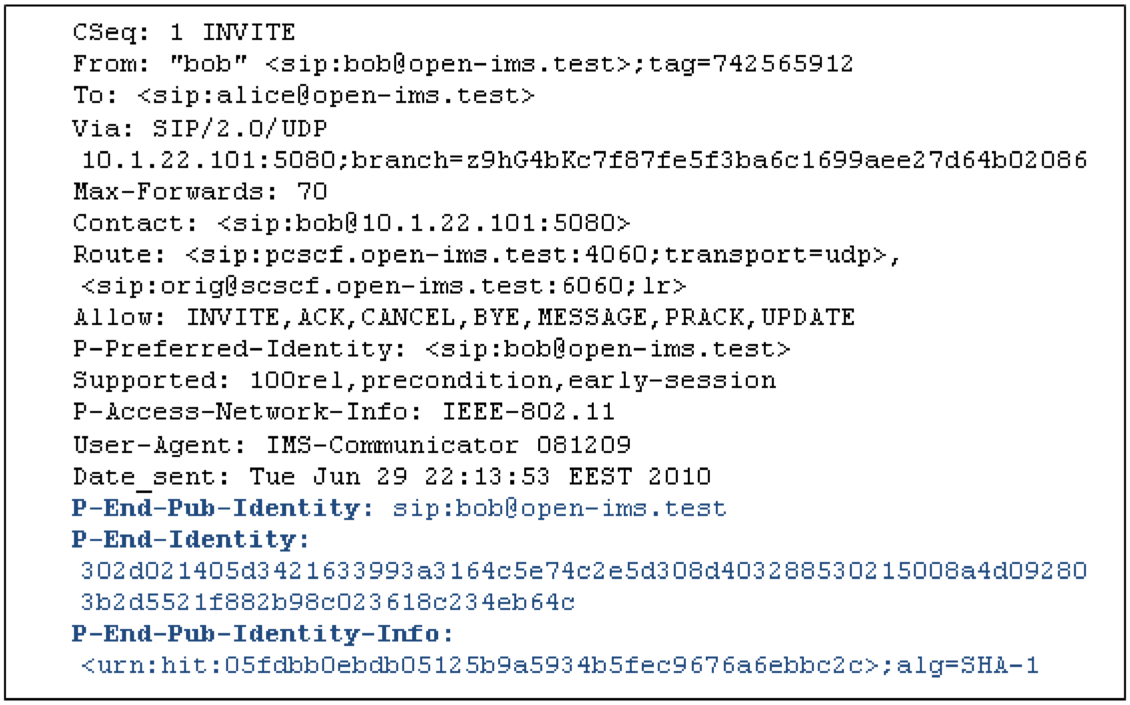
In the method GETSIGNATURE, we create the signature associated to the HIT. This signature is calculated creating a hash of several parameters present in the message, and then by ciphering it with the private key.

We use the Object "`java.security.Signature`", initialize it with the object "`java.security.PrivateKey`" obtained thanks to the method GETPRIVATEKEY (similar functioning than GETPUBLICKEY). Then, we update it with the different parameters so that it can generate a signature associated.

Once these functions defined, we can create the new headers (objects of type "`javax.sip.header.ExtensionHeader`") and add them in the message "`INVITE`". For each header, we just need the name and the value associated:

- "`P-End-Pub-Identity`": represents the identity that the client wants to communicate. Its value is copied from the header "`From`" (client's sip address)
- "`P-End-Pub-Identity-Info`": represents the cryptographic identity of the client, it contains the HIT. The value associated is obtained with the function GETHIT.
- "`P-End-Identity`": represents the signature associated to the HIT. Its value is obtained with the function GETSIGNATURE.

After these modifications, we can check with *Wireshark* and from the logs that the modifications have been taken into account. Indeed, we can see in the Figure 4-7 a message "`INVITE`" with these three new headers issued from the log files.



```
CSeq: 1 INVITE
From: "bob" <sip:bob@open-ims.test>;tag=742565912
To: <sip:alice@open-ims.test>
Via: SIP/2.0/UDP
 10.1.22.101:5080;branch=z9hG4bKc7f87fe5f3ba6c1699aee27d64b02086
Max-Forwards: 70
Contact: <sip:bob@10.1.22.101:5080>
Route: <sip:pcscf.open-ims.test:4060;transport=udp>,
 <sip:orig@scscf.open-ims.test:6060;lr>
Allow: INVITE,ACK,CANCEL,BYE,MESSAGE,PRACK,UPDATE
P-Preferred-Identity: <sip:bob@open-ims.test>
Supported: 100rel,precondition,early-session
P-Access-Network-Info: IEEE-802.11
User-Agent: IMS-Communicator 081209
Date_sent: Tue Jun 29 22:13:53 EEST 2010
P-End-Pub-Identity: sip:bob@open-ims.test
P-End-Identity:
 302d021405d3421633993a3164c5e74c2e5d308d40328853021500 8a4d09280
 3b2d5521f882b98c023618c234eb64c
P-End-Pub-Identity-Info:
 <urn:hit:05fdbb0ebdb05125b9a5934b5fec9676a6ebbc2c>;alg=SHA-1
```

*Figure 4-7. Request INVITE modified (sent in the Task A)*

This "`INVITE`" request is received by the SIP servers of the IMS Core, which forward it to the addressee. The reception of this request is described in the next section.

### 4.2.2 Receiving a modified INVITE request

Using the same methodology as previously, we discovered that it was the function "ProcessInvite" from the class "CallProcessing.java" which took in charge the message "INVITE" received. When the client (*UserB*) receives the message "INVITE" modified, he has to check the identity of *UserA* (his HIT) and the signature associated.

We created 2 new functions in the class "Security_ims_project_tools.java" (VERIFYHIT and VERIFYSIGNATURE), and the annex methods (EXTRACTHITUSER, EXTRACTALGUSER) and parameters (PublicKey public_key_*UserA*, Boolean hit_verified=false, Boolean signature_verified=false)

#### METHOD VERIFYHIT

The method VERIFYHIT takes in parameter the public key file containing the real saved identity of the sender (*UserA*) and the header containing the hit in the message "INVITE" received. It recalculates the real hit, and compares the hit received: if they correspond, it returns a Boolean "true".

First, the client *UserB* calculates the real HIT corresponding to the sender thanks to the public key of *UserA* previously saved in the folder "/ims-communicator" of *UserB*.

Then, he takes the value of the header "P_end_pub_identity_info" and applies the functions EXTRACTHITUSER and EXTRACTALGUSER to obtain the values of the HIT and the algorithm associated.

Finally, the function compares these two values (hit expected, hit received) and returns a Boolean "true" if they correspond each other.

#### METHOD VERIFYSIGNATURE

The method VERIFYSIGNATURE takes in parameter the public key file and several fields composing the header, it decodes the signature applying the public key on the data with which the signature was generated.

In this function, we take advantage of the methods proposed with the Object of type "java.security.Signature". Indeed, the function VERIFY enables to check if the Signature proposed really corresponds to this one expected. The function VERIFYSIGNATURE returns its result as a Boolean equals to "true" when the signature is verified.

If the result of these two functions equals to true, we could change the value of the two Booleans created in the file "CallProcessing.java": "hit_verified" and "`signature_verified`" to "`true`". These states would enable to later modify the answer to the `INVITE`, described in the next point.

Moreover, we save in a file the value of the header "`P_end_pub_identity`" which contains the identity that *UserA* wants to communicate to *UserB*. For example, if Bob sends the message "`INVITE`" to Alice, Alice creates a file "`sip:bob@open-ims.test`" in "`/ims-communicator`" which contains:

```
<urn:hit:05fdbb0ebdb05125b9a5934b5fec9676a6ebbc2c>;alg=SHA-1.
```

### 4.2.3 Answering to the modified INVITE request

For this step, the function SAYOK in the file "`CallProcessing.java`" is invoked. This function is called when UserB accepts to answer the call. The corresponding message is the fourth "`200 OK`" message sent. It is important not to add the secured identity in the previous "`200 OK`" as they are generated before accepting the call.

```
CSeq: 1 INVITE
From: "bob" <sip:bob@open-ims.test>;tag=742565912
To: <sip:alice@open-ims.test>
Via: SIP/2.0/UDP
10.1.22.101:5080;branch=z9hG4bKc7f87fe5f3ba6c1699aee27d64b02086
Max-Forwards: 70
Contact: <sip:bob@10.1.22.101:5080>
Route: <sip:pcscf.open-ims.test:4060;transport=udp>,
<sip:orig@scscf.open-ims.test:6060;lr>
Allow: INVITE,ACK,CANCEL,BYE,MESSAGE,PRACK,UPDATE
P-Preferred-Identity: <sip:bob@open-ims.test>
Supported: 100rel,precondition,early-session
P-Access-Network-Info: IEEE-802.11
User-Agent: IMS-Communicator 081209
Date_sent: Tue Jun 29 22:13:56 EEST 2010
P-End-Pub-Identity: sip:alice@open-ims.test
P-End-Identity:
302c0214335b5eef8f024529a69df25ee98e88811c5242d6021454e98142de3ab671
be683038f901c39d55ece62f
P-End-Pub-Identity-Info:
<urn:hit:719a95029058615c2b07e5b70f137ac11e95e163>;alg=SHA-1
```

***Figure 4-8.*** *Response to the request INVITE modified (Task D, c)*

As previously said, the *UserB* client first verifies the values of "`hit_verified`" and "`signature_verified`". If they are both fixed at `true`, which means that the identity of the sender is verified, *UserB* has create a new SIP message based on this one received. He copies and pastes all the headers (like "`From`", "`To`", "`Call-Id`") and replaces the value of the headers "P_end_pub_identity", "P_end_pub_identity_info" and "P_end_identity" filled by *UserA*'s information, by his own information. This is basically the same operation than this one that *UserA* did when he modified the "INVITE" Message.

After these modifications, we can check with *Wireshark* and from the logs that the modifications have been taken into account, and that the answer to the "INVITE" presents three new headers updated. This can be seen in the Figure 4-8, issued from the logs. This response is sent to the IMS Core which forwards it to the Caller *UserA*. Its behavior is described in the next section.

### 4.2.4 **Receiving the response to its request INVITE**

For this step, it is the function PROCESSINVITEOK in the file "CallProcessing.java" which is invoked. In this case, *UserA* does exactly the same than *UserB* has just done. This means, he verifies *UserB*'s identity and signature and save *UserB*'s identity. Then, to start the direct communication, he sends an ACK2 message to *UserB*, containing his HIT. This is a simple UDP message sent to the IP address of the *UserB* that contains the identities of the *UserA*. The reason of this message is to indicate the identity of the *UserA* directly to the *UserB*. Once receiving the UDP message, the *UserB* verifies the identity received in this message with the identity that he has already saved during the SIP exchange. This is another mechanism in order to improve the identity exchange and permit the user to have a better certainty about the identity of its counterpart.

### 4.2.5 **Clients' final modifications**

As previously described, after these modifications, the mechanism was still basic between the two users; the server didn't take part in the process. However, the functions required only small modifications to make the users verify their operators' identities rather than the other SIP user's one. Indeed, in the code, there is just a variable which takes in consideration the value of the hit and another for the signature. Rather than taking the value of the hit of the user (in the header "P-End-Info-Identity"), we changed it so that it could now take the server's hit (in the header "Identity-Info") and signature (in the header "Identity"). We also realized three new methods: EXTRACTHITSERVER, EXTRACTALGSERVER, and EXTRACTNAMESERVER. In these headers, there is also information about the name of the home operator: we created a new function in the class "Security_ims_project_tools.java": VERIFYNAME. Finally, the modified IMS client uses new attributes and main methods, all defined in the file "Security_ims_project_tools.java" and described in the Table 2.

| IMS Client – *UserA* | |
|---|---|
| **Attributes:**<br><br>- (Public Key) Client itself - *UserA*<br>- (Private Key) Client itself - *UserA*<br>- (Public Key) *UserA*'s home operator<br>- (String) Algorithm used for the authentication<br>- (Boolean) hit_verified<br>- (Boolean) signature_verified<br>- (String) Name of home operator | **Methods:**<br>- String GetHIT<br>- String GetSignature<br>- Boolean VerifyHit<br>- Boolean VerifySignature<br>- Boolean VerifyName<br><br>**Auxiliary methods:**<br>- PublicKey GetPublicKey<br>- PrivateKey GetPrivateKey<br>- String extractHitServer<br>- String extractAlgServer<br>- String extractNameServer |

***Table 2*** *Attributes and Methods defined in "Security_ims_project_tools.java"*

These are the main modifications that we applied to the *IMS Communicator* client to implement the secured identity exchange mechanism. There are still some auxiliary methods that we needed to implement but those aren't very important for the comprehension of the project. These classes realize basic functions, as calculations or translations, and are used by more important classes. The next sections focus on the implementation related to the IMS Core (Tasks B&C).

## 4.3 Filter Criteria on the HSS

After the short introduction to the initial Filter Criteria concept presented in the chapter **2.2.4**, we continue by presenting the modifications that we have done to implement our own iFC. Of course before starting we should make sure that the *OSIMS* infrastructure is set up and running. We verified this by analyzing the traffic with a network analyzer such as *Wireshark*.

The purpose of this iFC would be to invoke an AS, as shown in the Figure 4-9, when the S-CSCF receives SIP `INVITE` requests that contain some particular headers such as "`Pub-End-Identity`", "`P-End-Identity-Info`", "`Identity`" or "`Identity-Info`". Actually these headers are respectively the signature of the SIP entities sending the message and their identity. We described earlier how these headers are calculated. The SIP entities can be SIP users or the S-CSCF.

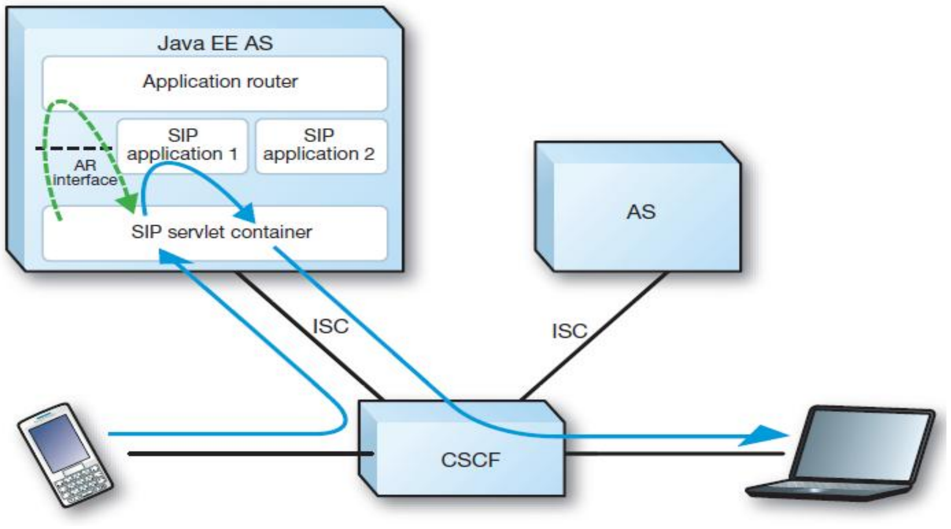


***Figure 4-9.*** *AS Invocation [38]*

As we mentioned before, the role of the SIP application that we developed is to verify the identity and the signature of the SIP entity when a SIP `"INVITE"` message arrives to the S-CSCF. Then, to add the identity of this S-CSCF in the `"INVITE"` message and fianly do the same task when the callee sends the SIP response to the previous SIP `"INVITE"` request.

For these reasons we created an iFC and then integrated it to the user profiles in order to benefit from this application. The FHoSS web interface, listening by default on port 8080, helped us to do all the configurations needed. We went through different steps in order to configure the iFC. All the configurations presented below needs to be done in admin mode through the FHoSS web interface available on *http://127.0.0.1:8080*.

Firstly, we added a new application server. For that we needed to specify its address in the form of SIP URI and the default behavior of the S-CSCF in case of connection failures. This is done through: `Services` → `Application Servers` → `Create` and specify the following values. As examples, we provide the names we have used in our iFC.

- `"Name"` – any unique name. Example: "`sailfin`"
- `"Server Name"` – it must be a valid SIP URI which resolves to application server address. Example `"sip:sailfin.open-ims.test:41981"`
- `"Diameter FQDN"` – fully qualified domain name. Example: `"sailfin.open-ims.test"`
- `"Default Handling"` – default behavior of the S-CSCF in case of connection errors. Example: `"Session - Continued"`
- `"Service-Info"` – required, if used by the application.

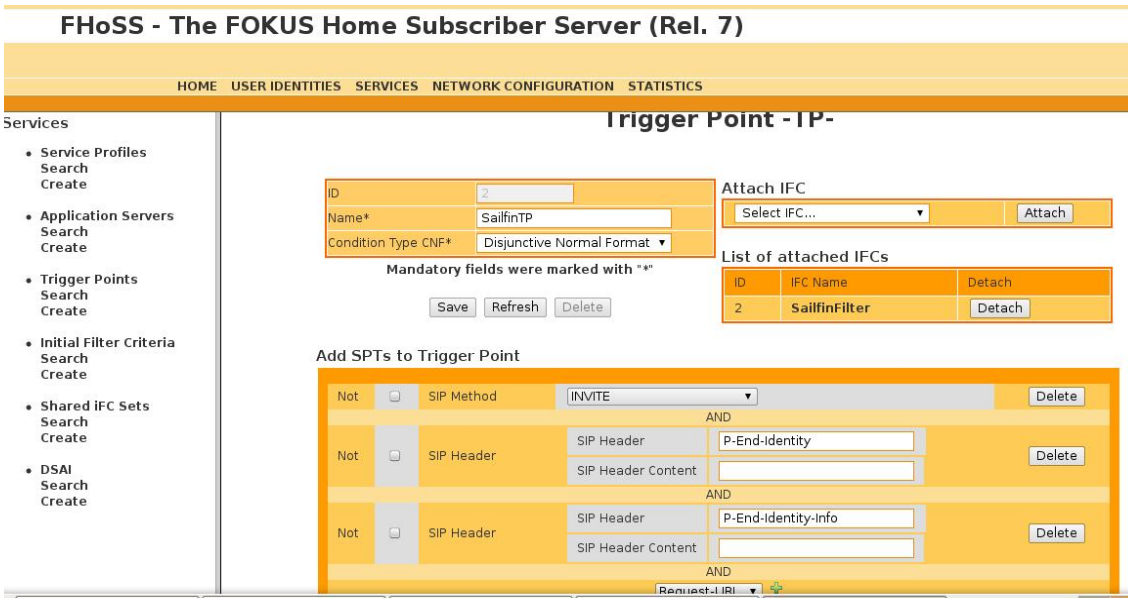
Secondly, we had to create a service profile. This step is a little bit more complicated than adding an application server. In order to create a service profile we need to perform the following three steps:

- create Trigger Point
- create Initial Filter Criteria
- create Service Profile itself

Trigger point defines a set of conditions under which particular SIP request is forwarded to the AS. Particular conditions are provided by Service Point Triggers (SPTs) in the form of the regular expression. The Trigger Point is created through: `Services` → `Trigger Points` → `Create` and specify the following values:

- “`Name`” – any unique name. Example: “`SailfinTP`”
- “`Condition Type`” – logic by which SPTs are evaluated. Example :“`Disjunctive Normal Forme`”

After this we specified the set of appropriate SPTs constituting this Trigger Point (TP) and provided optional list of already existing iFC to which this Trigger Point is attached. The Figure 4-10 shows how the TP of the iFC that we have configured in order to invoke the application we developed looks like.


FHoSS - The FOKUS Home Subscriber Server (Rel. 7)

HOME USER IDENTITIES SERVICES NETWORK CONFIGURATION STATISTICS

Services
- Service Profiles
  Search
  Create
- Application Servers
  Search
  Create
- Trigger Points
  Search
  Create
- Initial Filter Criteria
  Search
  Create
- Shared iFC Sets
  Search
  Create
- DSAI
  Search
  Create

Trigger Point -TP-

| | |
|---|---|
| ID | 2 |
| Name* | SailfinTP |
| Condition Type CNF* | Disjunctive Normal Format |

Mandatory fields were marked with "*"

Save Refresh Delete

Attach IFC

Select IFC... Attach

List of attached IFCs

| ID | IFC Name | Detach |
|---|---|---|
| 2 | SailfinFilter | Detach |

Add SPTs to Trigger Point

| | | | | | |
|---|---|---|---|---|---|
| Not | ☐ | SIP Method | INVITE | | Delete |
| | | AND | | | |
| Not | ☐ | SIP Header | SIP Header | P-End-Identity | Delete |
| | | | SIP Header Content | | |
| | | AND | | | |
| Not | ☐ | SIP Header | SIP Header | P-End-Identity-Info | Delete |
| | | | SIP Header Content | | |
| | | AND | | | |

Request-URI

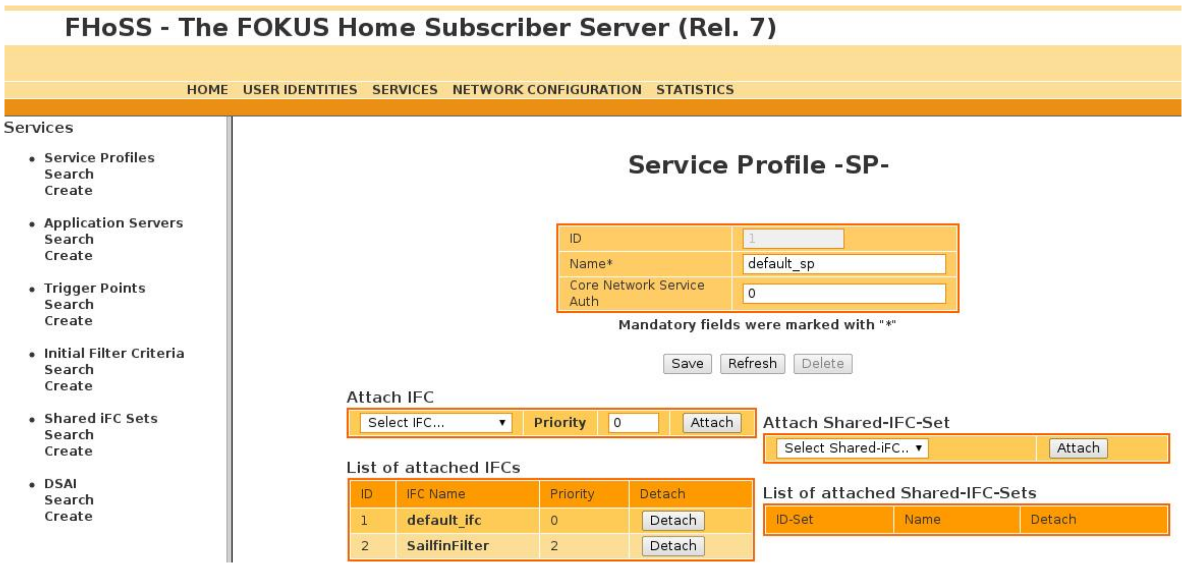
*Figure 4-10.* *Identity Filter Criteria*

Initial Filter Criteria defines a correlation between a set of Trigger Points and particular application server responsible for the execution of the associated service. The IMS standards specify that iFC can have from one to “n” attached Trigger Points. In the FOKUS OSIMS implementation there can be only one. So generally in order to create iFC we need to specify the previously created Trigger Point and desired application server address. For this we navigate through: `Services` → `Initial Filter Criteria` → `Create` and specify the following values:

- `"Name"` – any unique name, Example `"SailfinFilter"`
- `"Trigger Point"` – name of the already existing Trigger Point, Example `"SailfinTP"`
- `"Application Server"` – application server name, Example: `"sailfin"`
- `"Profile Part Indicator"` – specifies the registration state condition for criteria evaluation.

Then we created a Service Profile and assigned to it a prioritized list of iFC including the one just created. If no iFC is specified, then this Service Profile will not invoke any service logic and all requests would be processed without evaluation. In our case, the service profile is shown in the Figure 4-11. For creating a Service Profile we navigate through: `Services` → `Service Profiles` → `Create` and specify the following values:

- `"Name"` – any unique name, Example: `"default_sp"`
- `"IFCs"` – set of attached Initial Filter Criterias. Example: `"SailfinFilter"`



Service Profile -SP-

| | |
|---|---|
| ID | 1 |
| Name* | default_sp |
| Core Network Service Auth | 0 |

Mandatory fields were marked with "*"

Priority 0

List of attached IFCs

| ID | IFC Name | Priority | Detach |
|---|---|---|---|
| 1 | default_ifc | 0 | Detach |
| 2 | SailfinFilter | 2 | Detach |

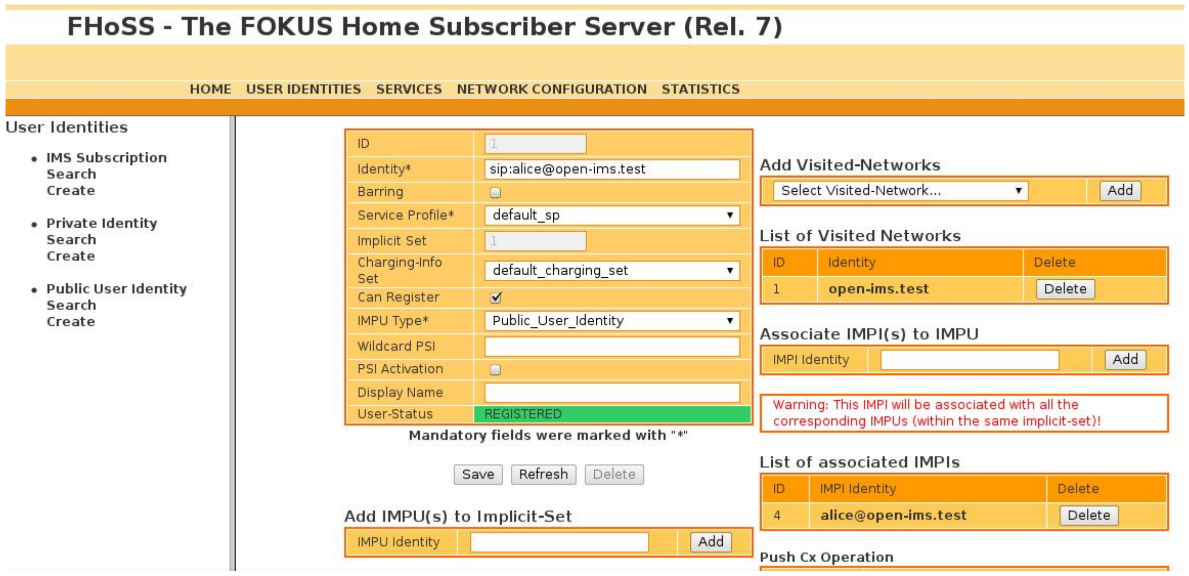
List of attached Shared-IFC-Sets

| ID-Set | Name | Detach |
|---|---|---|

***Figure 4-11.*** *Service Profile*

As the FHoSS, by default, comes provisioned with a couple of sample users, we didn’t create new user accounts. If this is needed, it can be done very easily through the FHoSS WEB interface.

Finally, we activated a Service Profile for a particular user by assigning it to one of his IMPU's (Public User Identity). The user can be found by navigating through User `Identities` → `Public User Identity` → `Search`. Then we just set the Service Profile field with the appropriate value. In the Figure 4-12 is shown the configuration in our example.



| | |
|---|---|
| ID | 1 |
| Identity* | sip:alice@open-ims.test |
| Barring | ☐ |
| Service Profile* | default_sp |
| Implicit Set | 1 |
| Charging-Info Set | default_charging_set |
| Can Register | ✓ |
| IMPU Type* | Public_User_Identity |
| Wildcard PSI | |
| PSI Activation | ☐ |
| Display Name | |
| User-Status | REGISTERED |

Mandatory fields were marked with "*"

Add IMPU(s) to Implicit-Set

IMPU Identity

Add Visited-Networks

Select Visited-Network...

List of Visited Networks

| ID | Identity | Delete |
|---|---|---|
| 1 | open-ims.test | Delete |

Associate IMPI(s) to IMPU

IMPI Identity

Warning: This IMPI will be associated with all the corresponding IMPUs (within the same implicit-set)!

List of associated IMPIs

| ID | IMPI Identity | Delete |
|---|---|---|
| 4 | alice@open-ims.test | Delete |

Push Cx Operation

***Figure 4-12**. User Profile*

Now that we have finished configuring an iFC we are ready to test it using the appropriate IMPU with the services hosted on the application server. In the next section we introduce the SIP Servlet application that we have developed.

## 4.4 Developing a SIP application for securing the identity exchange

The aim of this part of the project was to implement a module in the S-CSCF. We already have mentioned in the beginning of this chapter the role of this module. This module can be implemented directly by modifying the code of the S-CSCF or can be implemented as a SIP application and then deployed in the AS. Knowing that the S-CSCF is coded in C, if we had chosen this way of doing, then we would have had to code in C and would have spent a lot of time in understanding the sources code of this server. In other words, it was easier to develop a SIP application rather than to modify the code of the S-CSCF. At the same time, implementing this module directly in the S-CSCF would mean that this SIP server would have to support all the added burden coming from the amount of users that want to perform a call with identity exchange.

We decided to implement this module by developing a SIP application based on SIP Servlet technology and then deploying it on the *SailFin* AS. By this way we have a "plug and play" solution that can be deployed easily in another AS and be used in other IMS platforms. We chose *SailFin* as an application server because it is a Java-based AS that provides SIP Servlet container. Moreover, this solution helps to delegate the added burden to the AS and to maintain the performances of the S-CSCF.

### 4.4.1 **Overview of the SIP Servlet application**

In this section we will explain the SIP application and the java classes that we have developed. This application is based on a SIP Servlet and follows all the requirements of a SIP Servlet application. In the Table 3 the reader can find the file structure of the SIP application developed named "`identity`".

| **File** | **Description** |
|---|---|
| WEB-INF/ | Configuration and executable files of the SIP application are in this directory. |
| WEB-INF/sip.xml | The SIP Servlet-defined configuration file for the SIP application. |
| WEB-INF/web.xml | The Java EE standard configuration file for the Web application. |
| WEB-INF/classes | Store compiled class files in the directory. Both HTTP and SIP servlets can be stored in this directory. |
| WEB-INF/lib | Store class files archived as Jar files in the directory. |

***Table 3*** *SIP Application file structure*

Both "`sip.xml`" and "`web.xml`" have similar information except <`servlet-mapping`> setting. Another difference between the "`sip.xml`" and "`web.xml`" is that in "`web.xml`" we can specify a servlet associated with the file name portion of URL. But SIP has no concept of the file name. Filter conditions can be set using URI or the header field of a SIP request. The following "`sip.xml`" example shows that a SIP Servlet called "`identity`" is assigned all "`INVITE`" methods.

```
<sip-app>
    <display-name>SIP Identity</display-name>
    <description>SIP Identity application</description>
    <servlet>
        <servlet-name>identity</servlet-name>
        <description>Identity SIP servlet</description>
        <servlet-class>com.ericsson.sip.IdentityServlet</servlet-
class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>identity</servlet-name>
        <pattern>
            <or>
                <equal>
                    <var ignore-case="false">request.method</var>
                    <value>INVITE</value>
                </equal>
            </or>
        </pattern>
    </servlet-mapping>
</sip-app>
```

As the iFC we configured in the previous section, sends to the AS all "`INVITE`" SIP messages that contain the specific headers, the application that we developed should have a SIP Servlet responsible for receiving the "`INVITE`" messages. The SIP Servlet "`identity`" extends the "`javax.servlet.sip.SipServlet`" class and implements the "`javax.servlet.Servlet`" interface. It overrides the `DOINVITE` method. This application is responsible for:

- receiving all the "`INVITE`" requests forwarded by the S-CSCF according to the iFC
- verifying if the request comes from a SIP user or from a S-CSCF of another IMS domain
- verifying the sender identity of the message and his signature. If the sender is a SIP user that has already been registered in the domain, the application will have to identify its identity based on the Public Key of that user and his signature based on the headers present on the message. We assume that the S-CSCF is in possession of the Public Key of the user. If the message comes from another operator, based on the trust that the operators have between them, the identity and the signature of the S-CSCF of that operator is verified. The operators have previously exchanged their Public Identities.
- adding the S-CSCF identity and signature
- forwarding the message to the S-CSCF

The SIP Servlet application has to deal in the same fashion with the response of the "`INVITE`" message sent by the callee to the caller. The servlet we developed, also has to override the `DORESPONSE` method. In this case we don't need to create a specific iFC, because every response follows the same path as the request, so they are directly forwarded from the S-CSCF to the AS.

### 4.4.2 **Identity exchange SIP Servlet application**

After this brief introduction of the main functions of the SIP Servlet application, we go deeply in details of how we have coded it and present the main java classes and methods used. But first of all, we have to keep in mind that this case of study is based on a call between two IMS user. *UserA* sends an `"INVITE"` Request to *UserB*. We name *S-CSCF_A* the SIP server of *UserA* and *S-CSCF_B* the SIP server of *UserB*. We assumed the case where the two users have already been registered in two different IMS domains.

As already presented in the previous section, the *UserA* sends his modified `"INVITE"` Request to the *UserB*. In this Request there are some new headers such as `"P-End-Pub-Identity", "P-End-Identity-Info" and "P-End-Identity"`. When this Request arrives to the *S-CSCF_A*, after having passed through the *P-CSCF_A* and optionally through the *I-CSCF_A*, it is forwarded to the AS based on the iFC we already configured. As the `"sip.xml"` file indicates, the `"INVITE"` Request is received by the `"identity"` Servlet, the SIP application developed in this project. In the following code we present the structure of the SIP Servlet developed.

```
public class IdentityServlet extends SipServlet implements
   SipErrorListener,Servlet {

  @Override
  protected void doInvite(SipServletRequest request) throws
   ServletException, IOException {
    if(cp.verifyIdentity(request, public_key_file_user,
   "type_user")==true){
      cp.addIdentityS(request, public_key_file_server,
               private_key_file_server);
    }
    Proxy proxy = request.getProxy();
    proxy.setSupervised(true);
    proxy.proxyTo(request.getRequestURI());
  }

  @Override
  protected void doResponse(SipServletResponse response)throws
   ServletException, IOException {
    if(cp.verifyIdentity(response,
public_key_file_user,"type_user")==true)
    {
      cp.addIdentityS(response, public_key_file_serverA,
         private_key_file_serverA);
    }
    super.doResponse(response);
}  } }
```

All the auxiliary methods used for verifying or adding the identity and the signature of the user or the server are gathered in the "`Security_ims_project_tools.java`" class in the package "`com.ericsson.sip.security`".

The first thing that the Servlet has to do is to know, if the Request came from the SIP server of the other domain or if it came from one of its own users. Let's assume that the application server associated to *S-CSCF_A* has to check a message "`INVITE`" coming from *UserA* already registered in the *S-CSCF_A*. In this case, the AS has to check the identity of *UserA* (his HIT) and the signature associated. These verifications are done through the method VERIFYIDENTITY of the "`CallProcessing.java`" class in the package "`com.ericsson.sip`".

### METHOD VERIFYIDENTITY

This method takes as parameters:

- a SIP message
- the public key of the SIP entity the request comes from
- the type of the sender: it can be a IMS user or the S-CSCF of another domain

This method verifies the identity and the signature of the SIP entity with the two methods that we already presented in the SIP user code. These methods are VERIFYHIT and VERIFYSIGNATURE. They are both defined in the "`Security_ims_project_tools.java`" class. If the identity and the signature are verified then the VERIFYIDENTITY returns a Boolean "`true`".

Once the verification of the identity and signature are done, *S-CSCFA* has to add its identity and its signature to the SIP "`INVITE`" message. This is done through the method ADDIDENTITYS of the "`CallProcessing.java`" class.

### METHOD ADDIDENTITYS

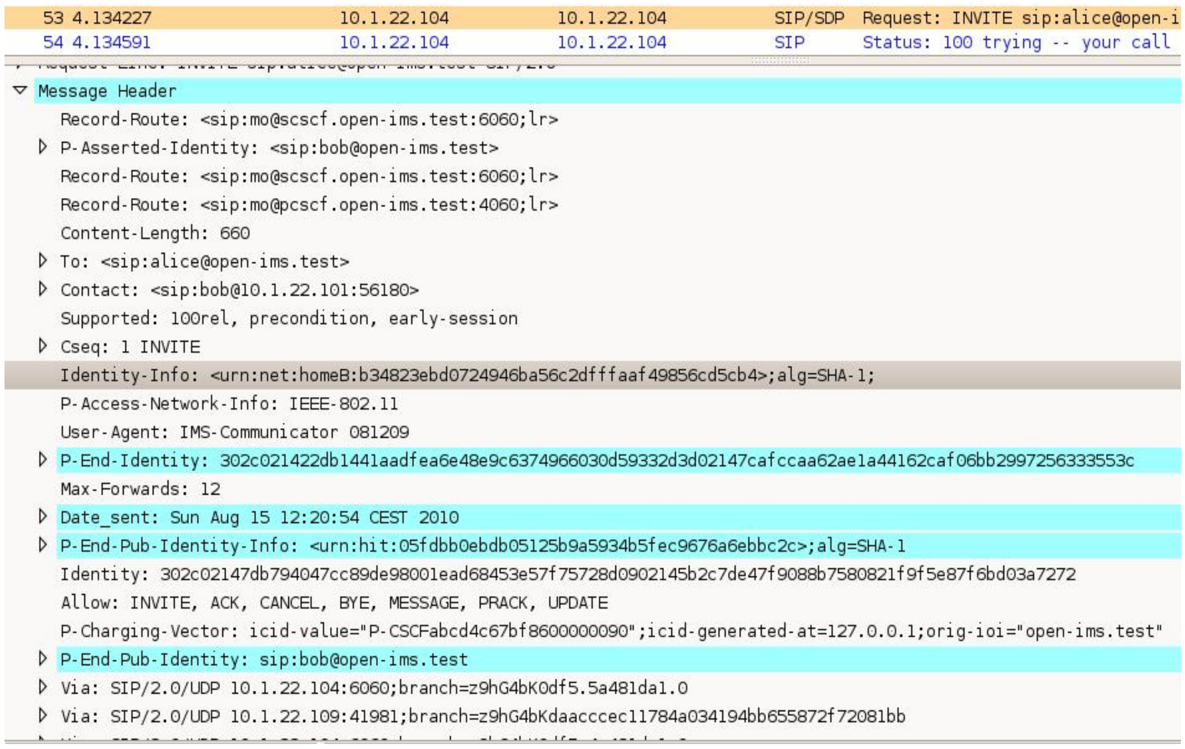
This method takes as parameters:

- a SIP message
- the public key of the SIP Server
- the private key of the SIP Server

This method uses the same methods that we already presented when describing how the IMS user adds his identity and his signature to the "`INVITE`" message. The methods

GETHIT and GETSIGNATURE used for creating the identity and the signature of the server are also coded in the "`Security_ims_project_tools.java`" class.

After this step, the AS has to forward the "`INVITE`" message to *S-CSCF_A* which will be responsible of forwarding it to *S-CSCF_B*, the SIP server of the other domain where the second IMS user is registered. The Figure 4-13 shows a typical message "`INVITE`" that is sent from the domain of *UserB=Bob* ("`domainB`") to the domain of *UserA=Alice* ("`domainA`"). We can see the identity and the signature of S-*CSCF_B* ("`homeB`") and also the identity and the signature of *`Bob`* who wants to start the conversation.



```
53 4.134227        10.1.22.104        10.1.22.104        SIP/SDP  Request: INVITE sip:alice@open-i
54 4.134591        10.1.22.104        10.1.22.104        SIP      Status: 100 trying -- your call
▽ Message Header
    Record-Route: <sip:mo@scscf.open-ims.test:6060;lr>
  ▷ P-Asserted-Identity: <sip:bob@open-ims.test>
    Record-Route: <sip:mo@scscf.open-ims.test:6060;lr>
    Record-Route: <sip:mo@pcscf.open-ims.test:4060;lr>
    Content-Length: 660
  ▷ To: <sip:alice@open-ims.test>
  ▷ Contact: <sip:bob@10.1.22.101:56180>
    Supported: 100rel, precondition, early-session
  ▷ Cseq: 1 INVITE
    Identity-Info: <urn:net:homeB:b34823ebd0724946ba56c2dfffaaf49856cd5cb4>;alg=SHA-1;
    P-Access-Network-Info: IEEE-802.11
    User-Agent: IMS-Communicator 081209
  ▷ P-End-Identity: 302c021422db1441aadfea6e48e9c6374966030d59332d3d02147cafccaa62ae1a44162caf06bb2997256333553c
    Max-Forwards: 12
  ▷ Date_sent: Sun Aug 15 12:20:54 CEST 2010
  ▷ P-End-Pub-Identity-Info: <urn:hit:05fdbb0ebdb05125b9a5934b5fec9676a6ebbc2c>;alg=SHA-1
    Identity: 302c02147db794047cc89de98001ead68453e57f75728d0902145b2c7de47f9088b7580821f9f5e87f6bd03a7272
    Allow: INVITE, ACK, CANCEL, BYE, MESSAGE, PRACK, UPDATE
    P-Charging-Vector: icid-value="P-CSCFabcd4c67bf8600000090";icid-generated-at=127.0.0.1;orig-ioi="open-ims.test"
  ▷ P-End-Pub-Identity: sip:bob@open-ims.test
  ▷ Via: SIP/2.0/UDP 10.1.22.104:6060;branch=z9hG4bK0df5.5a481da1.0
  ▷ Via: SIP/2.0/UDP 10.1.22.109:41981;branch=z9hG4bKdaacccec11784a034194bb655872f72081bb
```

***Figure 4-13.*** *SIP INVITE from "Domain_B" to "Domain_A"*

Once the "`INVITE`" message arrives to the S-CSCF_B, it is forwarded to the AS of this domain based on the iFC configured. Then the following procedures are in the same fashion as before. The difference is that now the S-CSCF_B has to verify the identity and the signature of *S-CSCF_A* and then add his own identity and signature to the headers of the message and forward it back to the *UserB*.

The SIP Response follows the same path as the SIP "`INVITE`" Request and the same methods were applied in order to verify and add the identities and the signatures of the SIP entities. Once the *SailFin* is installed and configured we can deploy the application through the WEB interface or through the "`build.xml`".

# 5 Results and discussion

In this part, we will present the results of the implementation of the project. At the beginning we will give general results that have been revealed during this work and then we will go through the relevant *Wireshark* screen shots to analyze the different SIP messages. We will analyze step by step the messages transiting in the network through the different entities when an IMS user wants to call another IM Client. Finally we will give some improvements to be considered for future research.

## 5.1 General results

The mechanisms presented on this work have been evaluated according to their scalability aspects. Adding new headers on the SIP messages augment the size of these messages. The process of verifying the identity and the signature every time the SIP messages pass through a SIP entity generate new calculations and therefore augment the time delay. The main problem that the size of the `"INVITE"` messages can cause is the possibility to switch from a UDP datagram to TCP if the UDP datagram is too large. This can be prevented by encoding properly the packet. We have analyzed the SIP messages containing the new headers. This has shown that the total size of the SIP message does not overpass the maximum size of the UDP datagram.

The result of this work is that the high level architecture presented in order to enhance the security aspects of multimedia communications based on IMS architecture and SIP exchange is implementable. Nevertheless some scalability aspects such the size of the new SIP packets and the time delay has to be taken into account.

## 5.2 Packet analysis

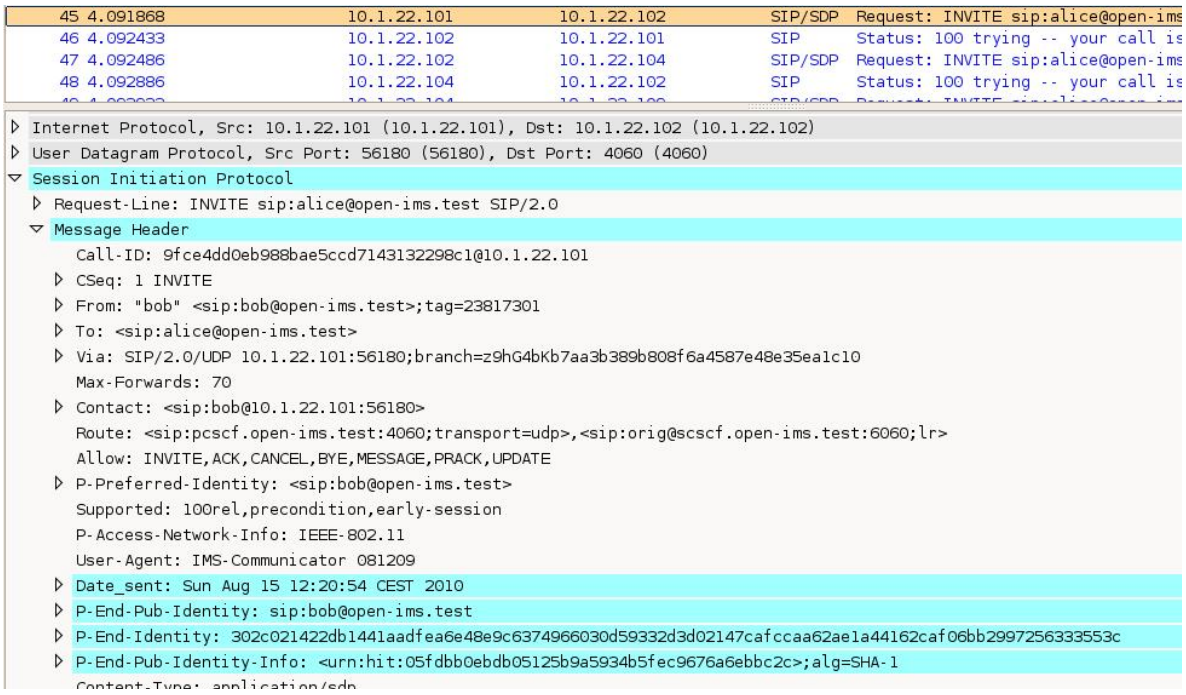
In our test runs, it is Bob (“`sip:bob@open-ims.test`”) who calls Alice (“`sip:alice@open-ims.test`”). They both belong to the same domain (“`open-ims.test`”) and are already registered. These are the different IP addresses of the entities:

- Bob: `10.1.22.101`
- Alice: `10.1.22.101`
- P-CSCF: `10.1.2.102`
- I-CSCF: `10.1.2.103`
- S-CSCF: `10.1.2.104`
- AS: `10.1.2.109`

The Figure 5-1 shows the first `"INVITE"` message sent by Bob to his proxy (P-CSCF). We can see the IMS-specific headers composing the SIP message like `"Call-ID"`, `"CSeq"`, `"From"`,`" To"`, `"Via"`, `"Contact"`, `"Route"` and `"Allow"`. There are also the identity-specific headers that we added with the different methods described in the previous section:

- `"Date_sent":` Contains the date when the message was produced. In fact, *IMS Communicator* didn't create this header by default, so we had to add it.
- `"P-End-Pub-Identity":` Contains the identity that Bob wants to communicate to his interlocutor, Alice.
- `"P-End-Pub-Identity-Info":` Contains Bob's cryptographic identifier (HIT) and the algorithm to use to decipher it (ALG)
- `"P-End-Identity":` Contains the Signature protecting the message.

Then, the proxy just forwards the message to the I-CSCF which consults the database HSS to know the S-CSCF associated to Bob. Then it forwards the `"INVITE"` message to it. These two SIP servers don't do much more than forwarding the message. The headers that we use to calculate the signature are not removed nor modified. The proxy just modifies the header `"Route"` to indicate its address for the future answer to pass by him, and add some headers that we did not use in this project. As a consequence, the messages forwarded by P-CSCF and I-CSCF are quite the same than this one sent in the Figure 5-1.



```
   45 4.091868        10.1.22.101     10.1.22.102     SIP/SDP  Request: INVITE sip:alice@open-ims
   46 4.092433        10.1.22.102     10.1.22.101     SIP      Status: 100 trying -- your call is
   47 4.092486        10.1.22.102     10.1.22.104     SIP/SDP  Request: INVITE sip:alice@open-ims
   48 4.092886        10.1.22.104     10.1.22.102     SIP      Status: 100 trying -- your call is

▷ Internet Protocol, Src: 10.1.22.101 (10.1.22.101), Dst: 10.1.22.102 (10.1.22.102)
▷ User Datagram Protocol, Src Port: 56180 (56180), Dst Port: 4060 (4060)
▽ Session Initiation Protocol
  ▷ Request-Line: INVITE sip:alice@open-ims.test SIP/2.0
  ▽ Message Header
      Call-ID: 9fce4dd0eb988bae5ccd7143132298c1@10.1.22.101
    ▷ CSeq: 1 INVITE
    ▷ From: "bob" <sip:bob@open-ims.test>;tag=23817301
    ▷ To: <sip:alice@open-ims.test>
    ▷ Via: SIP/2.0/UDP 10.1.22.101:56180;branch=z9hG4bKb7aa3b389b808f6a4587e48e35ea1c10
      Max-Forwards: 70
    ▷ Contact: <sip:bob@10.1.22.101:56180>
      Route: <sip:pcscf.open-ims.test:4060;transport=udp>,<sip:orig@scscf.open-ims.test:6060;lr>
      Allow: INVITE,ACK,CANCEL,BYE,MESSAGE,PRACK,UPDATE
    ▷ P-Preferred-Identity: <sip:bob@open-ims.test>
      Supported: 100rel,precondition,early-session
      P-Access-Network-Info: IEEE-802.11
      User-Agent: IMS-Communicator 081209
    ▷ Date_sent: Sun Aug 15 12:20:54 CEST 2010
    ▷ P-End-Pub-Identity: sip:bob@open-ims.test
    ▷ P-End-Identity: 302c021422db1441aadfea6e48e9c6374966030d59332d3d02147cafccaa62ae1a44162caf06bb2997256333553c
    ▷ P-End-Pub-Identity-Info: <urn:hit:05fdbb0ebdb05125b9a5934b5fec9676a6ebbc2c>;alg=SHA-1
```

***Figure 5-1.*** *Request INVITE : Bob -> P-CSCF*

The Figure 5-2 shows the "`INVITE`" message received by the S-CSCF associated to Bob. It, according to the iFC, forwards the specific "`INVITE`" message to the Application Server. In the AS is the SIP application responsible for securing the identity exchange. We can notice that once again the message hasn't been modified. Then, the SIP application on the AS performs its tasks: check Bob's identity, and check the authenticity of the message. It can accomplish these tasks with the help of the value of the HIT, the algorithm contained in the header "`P-End-Pub-Identity-Info`", the signature contained in "`P-End-Identity`", and the other headers used to calculate the signature. In the Figure 5-3, the application server has finished its controls, and has added its specific headers as the user has been identified:

- "`Identity-Info`": Contains the server's name domain, own HIT (net), and algorithm uses in order to obtain the hit.
- "`Identity`": Contains the corresponding signature.

All the headers previously added are still present in the message. After several communications with the I-CSCF of Alice in order to find the C-SCSF associated to Alice, Bob's S-CSCF forwards it to the S-CSCF of Alice. It is assumed that Bob and Alice home operators have a pre-established roaming agreement.

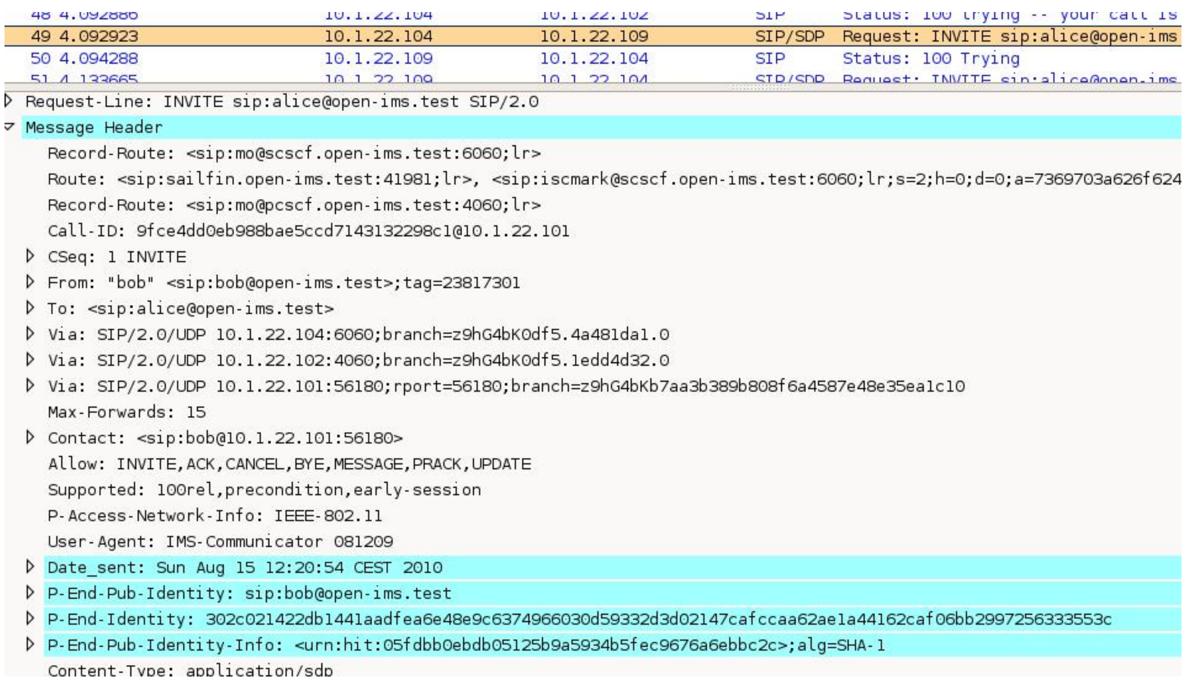
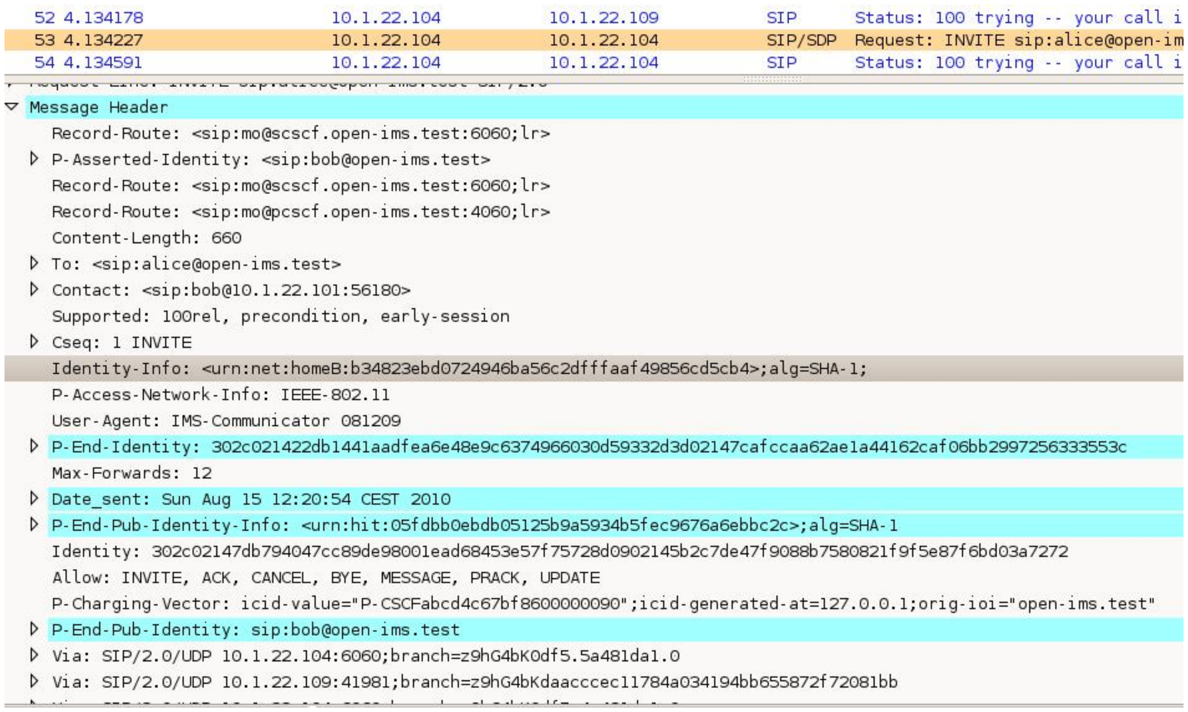
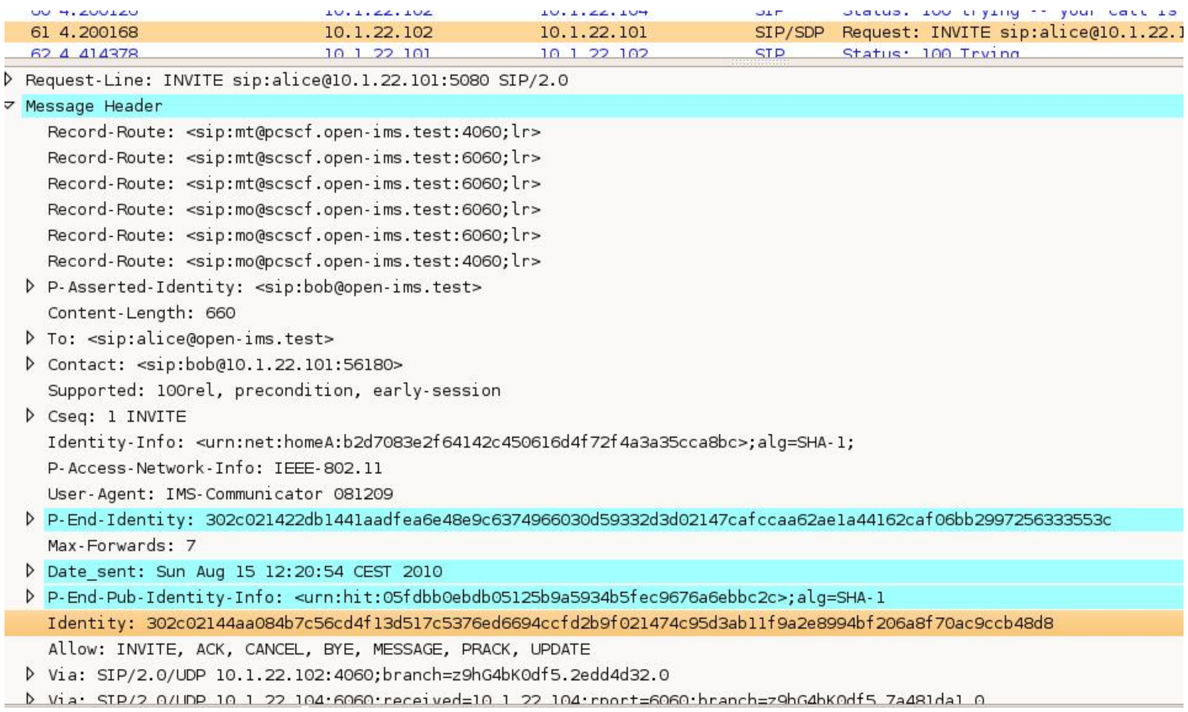

```
49 4.092923        10.1.22.104        10.1.22.109        SIP/SDP   Request: INVITE sip:alice@open-ims
50 4.094288        10.1.22.109        10.1.22.104        SIP       Status: 100 Trying
▷ Request-Line: INVITE sip:alice@open-ims.test SIP/2.0
▽ Message Header
    Record-Route: <sip:mo@scscf.open-ims.test:6060;lr>
    Route: <sip:sailfin.open-ims.test:41981;lr>, <sip:iscmark@scscf.open-ims.test:6060;lr;s=2;h=0;d=0;a=7369703a626f624
    Record-Route: <sip:mo@pcscf.open-ims.test:4060;lr>
    Call-ID: 9fce4dd0eb988bae5ccd7143132298c1@10.1.22.101
  ▷ CSeq: 1 INVITE
  ▷ From: "bob" <sip:bob@open-ims.test>;tag=23817301
  ▷ To: <sip:alice@open-ims.test>
  ▷ Via: SIP/2.0/UDP 10.1.22.104:6060;branch=z9hG4bK0df5.4a481da1.0
  ▷ Via: SIP/2.0/UDP 10.1.22.102:4060;branch=z9hG4bK0df5.1edd4d32.0
  ▷ Via: SIP/2.0/UDP 10.1.22.101:56180;rport=56180;branch=z9hG4bKb7aa3b389b808f6a4587e48e35ea1c10
    Max-Forwards: 15
  ▷ Contact: <sip:bob@10.1.22.101:56180>
    Allow: INVITE,ACK,CANCEL,BYE,MESSAGE,PRACK,UPDATE
    Supported: 100rel,precondition,early-session
    P-Access-Network-Info: IEEE-802.11
    User-Agent: IMS-Communicator 081209
  ▷ Date_sent: Sun Aug 15 12:20:54 CEST 2010
  ▷ P-End-Pub-Identity: sip:bob@open-ims.test
  ▷ P-End-Identity: 302c021422db1441aadfea6e48e9c6374966030d59332d3d02147cafccaa62ae1a44162caf06bb2997256333553c
  ▷ P-End-Pub-Identity-Info: <urn:hit:05fdbb0ebdb05125b9a5934b5fec9676a6ebbc2c>;alg=SHA-1
```

***Figure 5-2.*** *Request INVITE: S-CSCF_Bob -> AS*



```
52 4.134178     10.1.22.104     10.1.22.109     SIP       Status: 100 trying -- your call i
53 4.134227     10.1.22.104     10.1.22.104     SIP/SDP   Request: INVITE sip:alice@open-im
54 4.134591     10.1.22.104     10.1.22.104     SIP       Status: 100 trying -- your call i
▽ Message Header
    Record-Route: <sip:mo@scscf.open-ims.test:6060;lr>
  ▷ P-Asserted-Identity: <sip:bob@open-ims.test>
    Record-Route: <sip:mo@scscf.open-ims.test:6060;lr>
    Record-Route: <sip:mo@pcscf.open-ims.test:4060;lr>
    Content-Length: 660
  ▷ To: <sip:alice@open-ims.test>
  ▷ Contact: <sip:bob@10.1.22.101:56180>
    Supported: 100rel, precondition, early-session
  ▷ Cseq: 1 INVITE
    Identity-Info: <urn:net:homeB:b34823ebd0724946ba56c2dfffaaf49856cd5cb4>;alg=SHA-1;
    P-Access-Network-Info: IEEE-802.11
    User-Agent: IMS-Communicator 081209
  ▷ P-End-Identity: 302c021422db1441aadfea6e48e9c6374966030d59332d3d02147cafccaa62ae1a44162caf06bb2997256333553c
    Max-Forwards: 12
  ▷ Date_sent: Sun Aug 15 12:20:54 CEST 2010
  ▷ P-End-Pub-Identity-Info: <urn:hit:05fdbb0ebdb05125b9a5934b5fec9676a6ebbc2c>;alg=SHA-1
    Identity: 302c02147db794047cc89de98001ead68453e57f75728d0902145b2c7de47f9088b7580821f9f5e87f6bd03a7272
    Allow: INVITE, ACK, CANCEL, BYE, MESSAGE, PRACK, UPDATE
    P-Charging-Vector: icid-value="P-CSCFabcd4c67bf8600000090";icid-generated-at=127.0.0.1;orig-ioi="open-ims.test"
  ▷ P-End-Pub-Identity: sip:bob@open-ims.test
  ▷ Via: SIP/2.0/UDP 10.1.22.104:6060;branch=z9hG4bK0df5.5a481da1.0
  ▷ Via: SIP/2.0/UDP 10.1.22.109:41981;branch=z9hG4bKdaacccec11784a034194bb655872f72081bb
```

***Figure 5-3.*** *Request INVITE: S-CSCF_Bob -> S-CSCF_Alice*

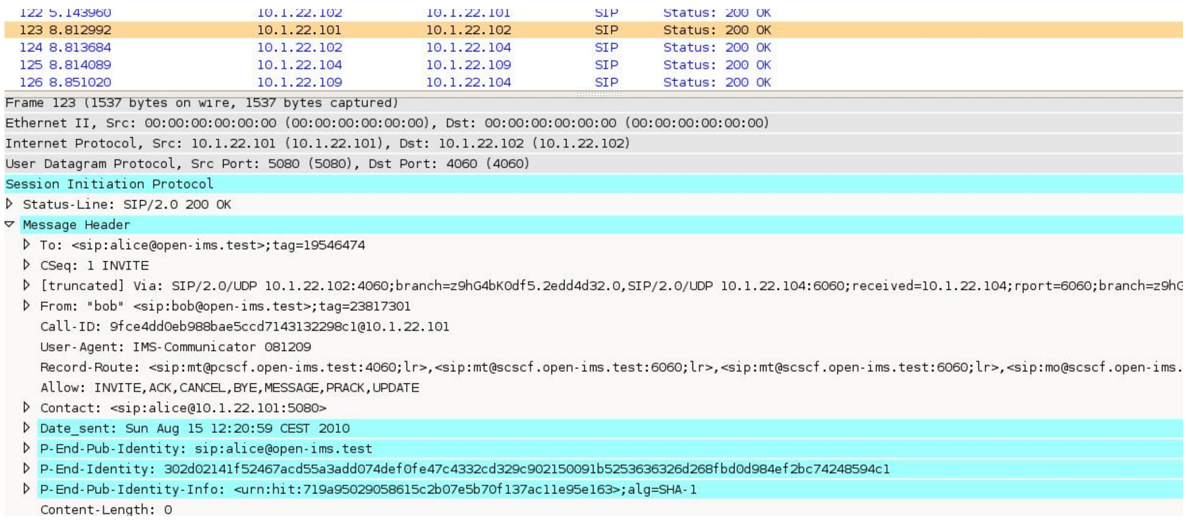


```
61 4.200168     10.1.22.102     10.1.22.101     SIP/SDP   Request: INVITE sip:alice@10.1.22.1
62 4.414378     10.1.22.101     10.1.22.102     SIP       Status: 100 Trying
▷ Request-Line: INVITE sip:alice@10.1.22.101:5080 SIP/2.0
▽ Message Header
    Record-Route: <sip:mt@pcscf.open-ims.test:4060;lr>
    Record-Route: <sip:mt@scscf.open-ims.test:6060;lr>
    Record-Route: <sip:mt@scscf.open-ims.test:6060;lr>
    Record-Route: <sip:mo@scscf.open-ims.test:6060;lr>
    Record-Route: <sip:mo@scscf.open-ims.test:6060;lr>
    Record-Route: <sip:mo@pcscf.open-ims.test:4060;lr>
  ▷ P-Asserted-Identity: <sip:bob@open-ims.test>
    Content-Length: 660
  ▷ To: <sip:alice@open-ims.test>
  ▷ Contact: <sip:bob@10.1.22.101:56180>
    Supported: 100rel, precondition, early-session
  ▷ Cseq: 1 INVITE
    Identity-Info: <urn:net:homeA:b2d7083e2f64142c450616d4f72f4a3a35cca8bc>;alg=SHA-1;
    P-Access-Network-Info: IEEE-802.11
    User-Agent: IMS-Communicator 081209
  ▷ P-End-Identity: 302c021422db1441aadfea6e48e9c6374966030d59332d3d02147cafccaa62ae1a44162caf06bb2997256333553c
    Max-Forwards: 7
  ▷ Date_sent: Sun Aug 15 12:20:54 CEST 2010
  ▷ P-End-Pub-Identity-Info: <urn:hit:05fdbb0ebdb05125b9a5934b5fec9676a6ebbc2c>;alg=SHA-1
    Identity: 302c02144aa084b7c56cd4f13d517c5376ed6694ccfd2b9f021474c95d3ab11f9a2e8994bf206a8f70ac9ccb48d8
    Allow: INVITE, ACK, CANCEL, BYE, MESSAGE, PRACK, UPDATE
  ▷ Via: SIP/2.0/UDP 10.1.22.102:4060;branch=z9hG4bK0df5.2edd4d32.0
  ▷ Via: SIP/2.0/UDP 10.1.22.104:6060;received=10.1.22.104;rport=6060;branch=z9hG4bK0df5.7a481da1.0
```

***Figure 5-4.*** *Request INVITE: P-CSCF_Alice -> Alice*

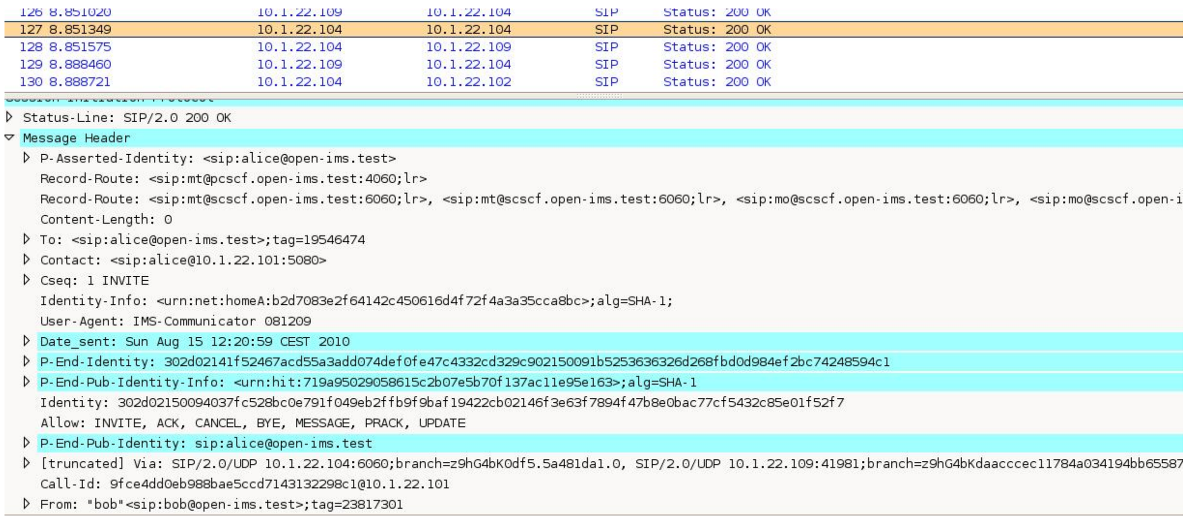
In the Figure 5-4, we can see that the SIP Server corresponding to Alice has replaced the headers "`Identity`" and "`Identity_Info`" with its own information. The information related to Bob’s identity has not been changed. This message is forwarded to Alice who checks the headers related to her home operator. Actually, she checks her operator’s hit, name, and signature.

If the identity of the server of Alice is verified by Alice, it means that the message can be trusted. The assurance issued by her server about the carried Bob’s identity, enables her to trust it too. As a consequence, she saves Bob’s information (hit and address) and answers to Bob’s request in the 4th `“200 OK”` as shown in the Figure 2-10. This message corresponds to the message sent when Alice accepts a call. The Figure 5-5 shows the SIP response of Alice to the `“INVITE”` message. She forwards the message to her P-CSCF. This proxy forwards the message to Alice’s S-CSCF.



```
  122 5.143960         10.1.22.102        10.1.22.101        SIP      Status: 200 OK
  123 8.812992         10.1.22.101        10.1.22.102        SIP      Status: 200 OK
  124 8.813684         10.1.22.102        10.1.22.104        SIP      Status: 200 OK
  125 8.814089         10.1.22.104        10.1.22.109        SIP      Status: 200 OK
  126 8.851020         10.1.22.109        10.1.22.104        SIP      Status: 200 OK
Frame 123 (1537 bytes on wire, 1537 bytes captured)
Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)
Internet Protocol, Src: 10.1.22.101 (10.1.22.101), Dst: 10.1.22.102 (10.1.22.102)
User Datagram Protocol, Src Port: 5080 (5080), Dst Port: 4060 (4060)
Session Initiation Protocol
▷ Status-Line: SIP/2.0 200 OK
▽ Message Header
  ▷ To: <sip:alice@open-ims.test>;tag=19546474
  ▷ CSeq: 1 INVITE
  ▷ [truncated] Via: SIP/2.0/UDP 10.1.22.102:4060;branch=z9hG4bK0df5.2edd4d32.0,SIP/2.0/UDP 10.1.22.104:6060;received=10.1.22.104;rport=6060;branch=z9hG
  ▷ From: "bob" <sip:bob@open-ims.test>;tag=23817301
    Call-ID: 9fce4dd0eb988bae5ccd7143132298c1@10.1.22.101
    User-Agent: IMS-Communicator 081209
    Record-Route: <sip:mt@pcscf.open-ims.test:4060;lr>,<sip:mt@scscf.open-ims.test:6060;lr>,<sip:mt@scscf.open-ims.test:6060;lr>,<sip:mo@scscf.open-ims.
    Allow: INVITE,ACK,CANCEL,BYE,MESSAGE,PRACK,UPDATE
  ▷ Contact: <sip:alice@10.1.22.101:5080>
  ▷ Date_sent: Sun Aug 15 12:20:59 CEST 2010
  ▷ P-End-Pub-Identity: sip:alice@open-ims.test
  ▷ P-End-Identity: 302d02141f52467acd55a3add074def0fe47c4332cd329c902150091b5253636326d268fbd0d984ef2bc74248594c1
  ▷ P-End-Pub-Identity-Info: <urn:hit:719a95029058615c2b07e5b70f137ac11e95e163>;alg=SHA-1
    Content-Length: 0
```

***Figure 5-5.*** *Response 200OK: Alice -> PCSCF of Alice*

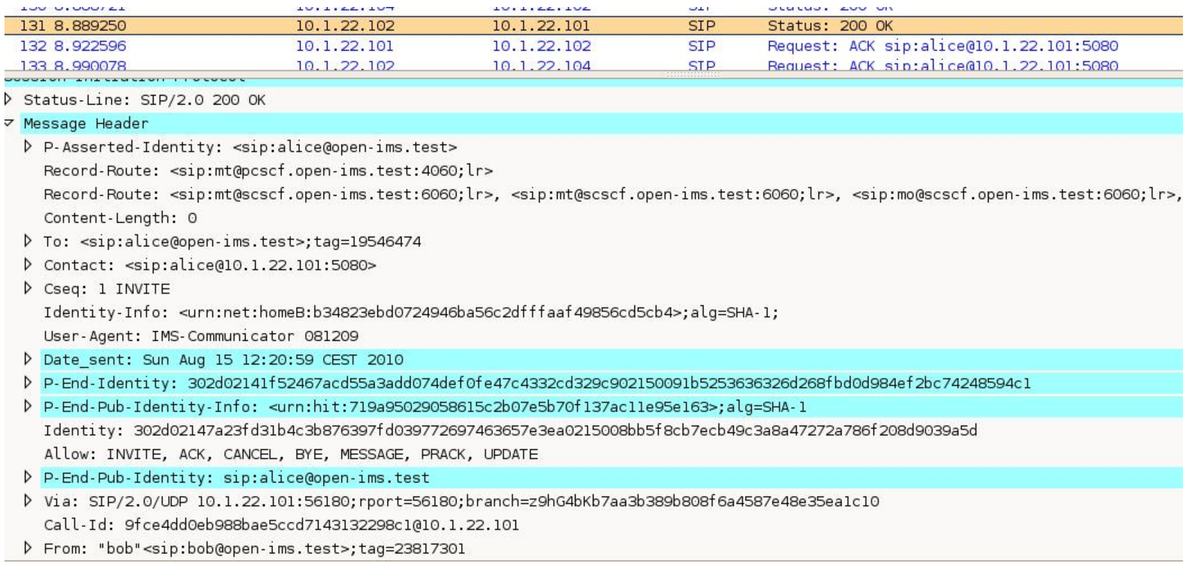


```
  126 8.851020         10.1.22.109        10.1.22.104        SIP      Status: 200 OK
  127 8.851349         10.1.22.104        10.1.22.104        SIP      Status: 200 OK
  128 8.851575         10.1.22.104        10.1.22.109        SIP      Status: 200 OK
  129 8.888460         10.1.22.109        10.1.22.104        SIP      Status: 200 OK
  130 8.888721         10.1.22.104        10.1.22.102        SIP      Status: 200 OK
▷ Status-Line: SIP/2.0 200 OK
▽ Message Header
  ▷ P-Asserted-Identity: <sip:alice@open-ims.test>
    Record-Route: <sip:mt@pcscf.open-ims.test:4060;lr>
    Record-Route: <sip:mt@scscf.open-ims.test:6060;lr>, <sip:mt@scscf.open-ims.test:6060;lr>, <sip:mo@scscf.open-ims.test:6060;lr>, <sip:mo@scscf.open-i
    Content-Length: 0
  ▷ To: <sip:alice@open-ims.test>;tag=19546474
  ▷ Contact: <sip:alice@10.1.22.101:5080>
  ▷ Cseq: 1 INVITE
    Identity-Info: <urn:net:homeA:b2d7083e2f64142c450616d4f72f4a3a35cca8bc>;alg=SHA-1;
    User-Agent: IMS-Communicator 081209
  ▷ Date_sent: Sun Aug 15 12:20:59 CEST 2010
  ▷ P-End-Identity: 302d02141f52467acd55a3add074def0fe47c4332cd329c902150091b5253636326d268fbd0d984ef2bc74248594c1
  ▷ P-End-Pub-Identity-Info: <urn:hit:719a95029058615c2b07e5b70f137ac11e95e163>;alg=SHA-1
    Identity: 302d02150094037fc528bc0e791f049eb2ffb9f9baf19422cb02146f3e63f7894f47b8e0bac77cf5432c85e01f52f7
    Allow: INVITE, ACK, CANCEL, BYE, MESSAGE, PRACK, UPDATE
  ▷ P-End-Pub-Identity: sip:alice@open-ims.test
  ▷ [truncated] Via: SIP/2.0/UDP 10.1.22.104:6060;branch=z9hG4bK0df5.5a481da1.0, SIP/2.0/UDP 10.1.22.109:41981;branch=z9hG4bKdaacccec11784a034194bb65587
    Call-Id: 9fce4dd0eb988bae5ccd7143132298c1@10.1.22.101
  ▷ From: "bob"<sip:bob@open-ims.test>;tag=23817301
```

***Figure 5-6.*** *Response 200OK:  S-CSCF_Alice -> S-CSCF_Bob*

In the Figure 5-6, we can see that the process is realized in the opposite direction: still through the application server associated, the server of Alice checks Alice identity, and then adds its own identity. It forwards the message to the server of Bob, which checks the identity of Alice's server. Then, if this identity is verified, it replaces Alice server specific headers with its own information and forwards it to the Bob's proxy.

In the Figure 5-7, Bob's P-CSCF forwards to him the Response "`200 OK`" coming from Alice. It contains Bob's server and Alice's identity information. Bob first checks the server's identity fields. If they are verified, it means that it can trust this message, and Alice identity which is associated. It saves Alice's cryptographic identifier (HIT).



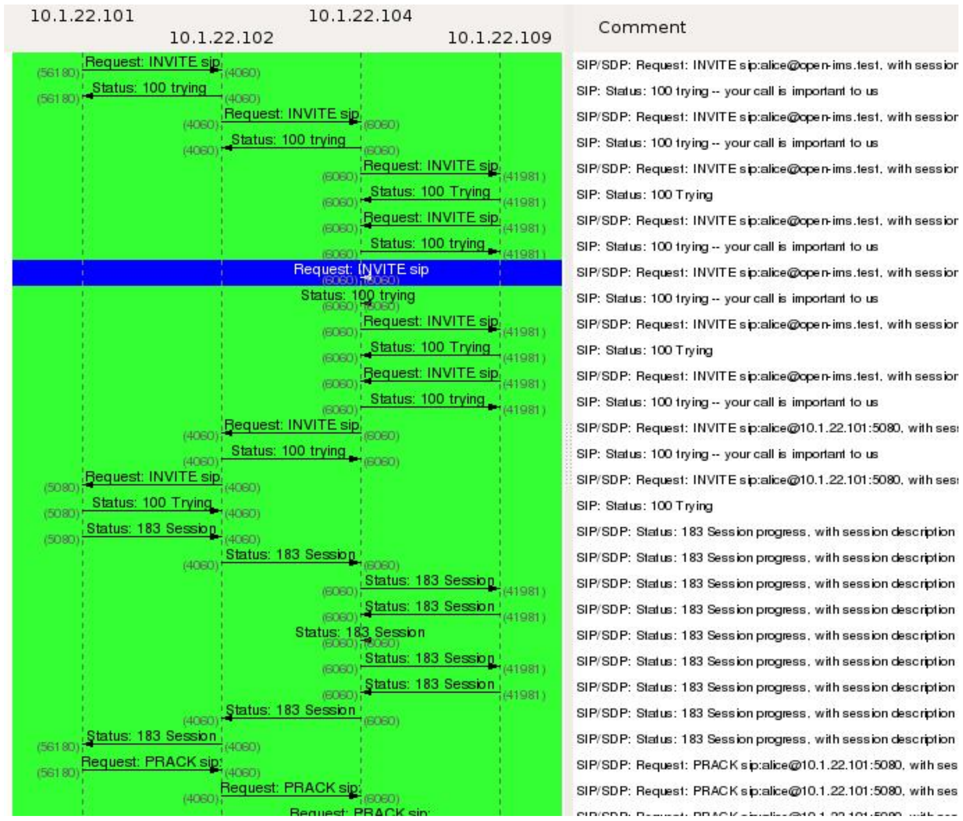
```
131 8.889250        10.1.22.102         10.1.22.101         SIP      Status: 200 OK
132 8.922596        10.1.22.101         10.1.22.102         SIP      Request: ACK sip:alice@10.1.22.101:5080
133 8.990078        10.1.22.102         10.1.22.104         SIP      Request: ACK sip:alice@10.1.22.101:5080
▷ Status-Line: SIP/2.0 200 OK
▽ Message Header
  ▷ P-Asserted-Identity: <sip:alice@open-ims.test>
    Record-Route: <sip:mt@pcscf.open-ims.test:4060;lr>
    Record-Route: <sip:mt@scscf.open-ims.test:6060;lr>, <sip:mt@scscf.open-ims.test:6060;lr>, <sip:mo@scscf.open-ims.test:6060;lr>,
    Content-Length: 0
  ▷ To: <sip:alice@open-ims.test>;tag=19546474
  ▷ Contact: <sip:alice@10.1.22.101:5080>
  ▷ Cseq: 1 INVITE
    Identity-Info: <urn:net:homeB:b34823ebd0724946ba56c2dfffaaf49856cd5cb4>;alg=SHA-1;
    User-Agent: IMS-Communicator 081209
  ▷ Date_sent: Sun Aug 15 12:20:59 CEST 2010
  ▷ P-End-Identity: 302d02141f52467acd55a3add074def0fe47c4332cd329c902150091b5253636326d268fbd0d984ef2bc74248594c1
  ▷ P-End-Pub-Identity-Info: <urn:hit:719a95029058615c2b07e5b70f137ac11e95e163>;alg=SHA-1
    Identity: 302d02147a23fd31b4c3b876397fd039772697463657e3ea0215008bb5f8cb7ecb49c3a8a47272a786f208d9039a5d
    Allow: INVITE, ACK, CANCEL, BYE, MESSAGE, PRACK, UPDATE
  ▷ P-End-Pub-Identity: sip:alice@open-ims.test
  ▷ Via: SIP/2.0/UDP 10.1.22.101:56180;rport=56180;branch=z9hG4bKb7aa3b389b808f6a4587e48e35ea1c10
    Call-Id: 9fce4dd0eb988bae5ccd7143132298c1@10.1.22.101
  ▷ From: "bob"<sip:bob@open-ims.test>;tag=23817301
```

**Figure 5-7.** *Response 200OK:  S-CSCF_Bob -> Bob*

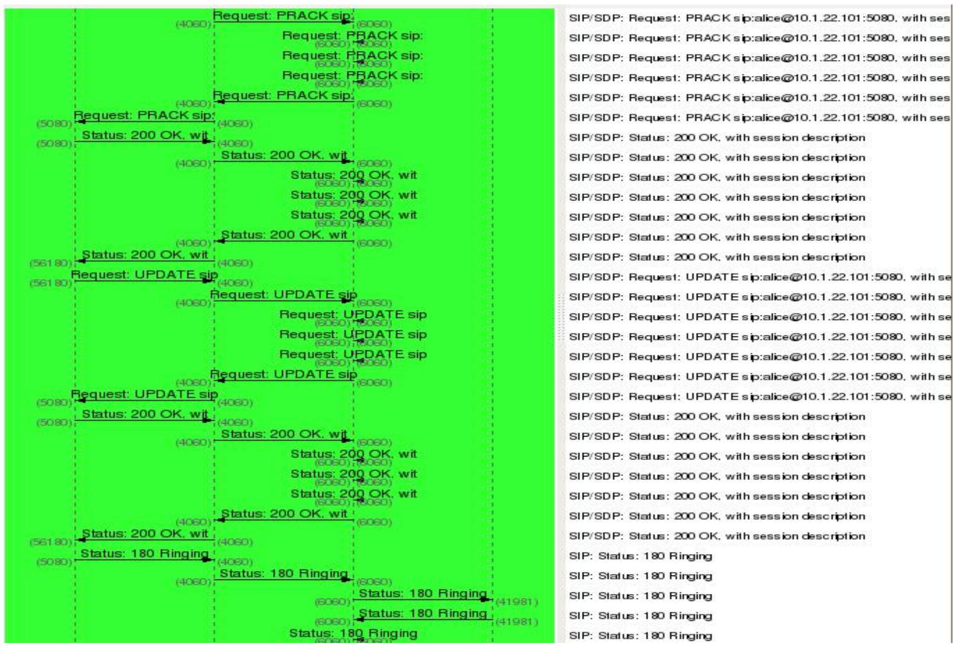
Once these different exchanges are performed, Bob and Alice are in possession of their respective secured identities, which have been asserted by their own trusted home operators. As a consequence, they can start to directly talk together. Their identities have been exchanged in a secure way, so they can be sure of their correspondent's identity.

## 5.3 Call Flow

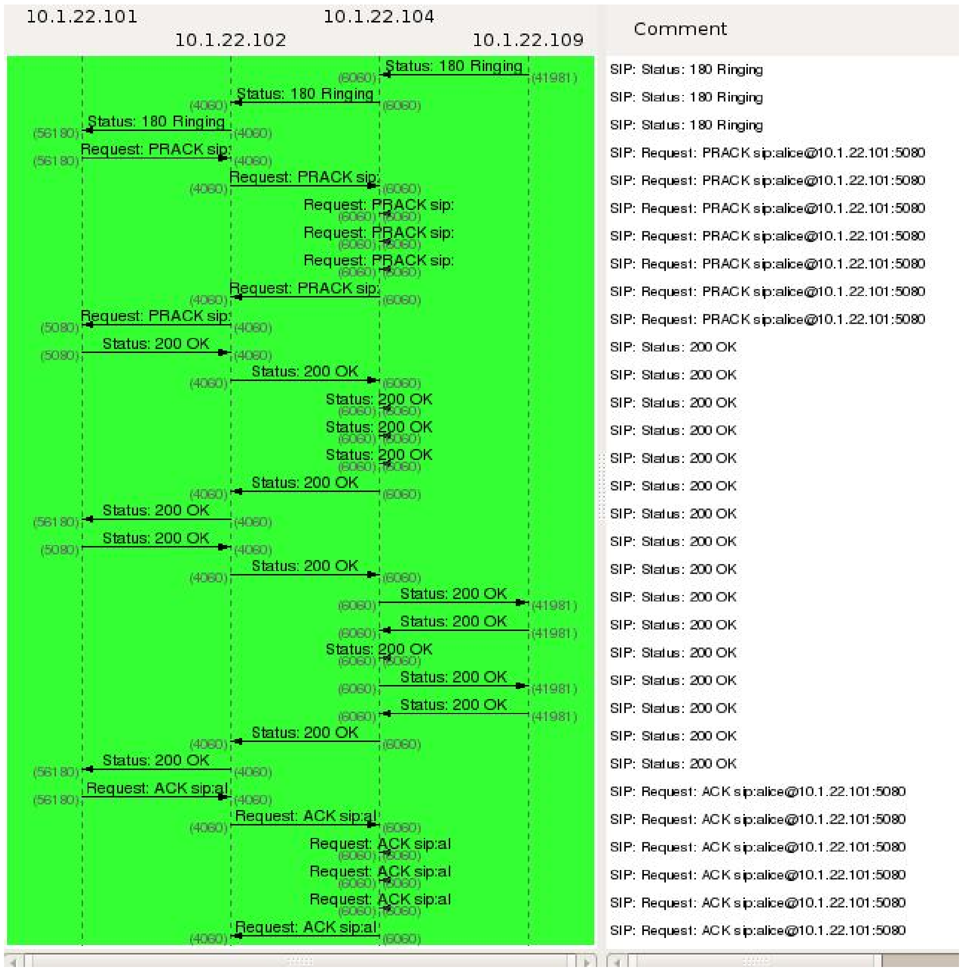
In this section, we present the message call flow. The Figure 5-8 enables to see an overall view of the messages exchanges, the involved entities, and to notice the differences with a basic call setup: we can see that there are some messages forwarded to the entity "`10.1.22.109`", which is the Application Server.



```
SIP/SDP: Request: INVITE sip:alice@open-ims.test, with session
SIP: Status: 100 trying -- your call is important to us
SIP/SDP: Request: INVITE sip:alice@open-ims.test, with session
SIP: Status: 100 trying -- your call is important to us
SIP/SDP: Request: INVITE sip:alice@open-ims.test, with session
SIP: Status: 100 Trying
SIP/SDP: Request: INVITE sip:alice@open-ims.test, with session
SIP: Status: 100 trying -- your call is important to us
SIP/SDP: Request: INVITE sip:alice@open-ims.test, with session
SIP: Status: 100 trying -- your call is important to us
SIP/SDP: Request: INVITE sip:alice@open-ims.test, with session
SIP: Status: 100 Trying
SIP/SDP: Request: INVITE sip:alice@open-ims.test, with session
SIP: Status: 100 trying -- your call is important to us
SIP/SDP: Request: INVITE sip:alice@10.1.22.101:5080, with ses
SIP: Status: 100 trying -- your call is important to us
SIP/SDP: Request: INVITE sip:alice@10.1.22.101:5080, with ses
SIP: Status: 100 Trying
SIP/SDP: Status: 183 Session progress, with session description
SIP/SDP: Status: 183 Session progress, with session description
SIP/SDP: Status: 183 Session progress, with session description
SIP/SDP: Status: 183 Session progress, with session description
SIP/SDP: Status: 183 Session progress, with session description
SIP/SDP: Status: 183 Session progress, with session description
SIP/SDP: Status: 183 Session progress, with session description
SIP/SDP: Status: 183 Session progress, with session description
SIP/SDP: Status: 183 Session progress, with session description
SIP/SDP: Request: PRACK sip:alice@10.1.22.101:5080, with ses
SIP/SDP: Request: PRACK sip:alice@10.1.22.101:5080, with ses
```



```
SIP/SDP: Request: PRACK sip:alice@10.1.22.101:5080, with ses
SIP/SDP: Request: PRACK sip:alice@10.1.22.101:5080, with ses
SIP/SDP: Request: PRACK sip:alice@10.1.22.101:5080, with ses
SIP/SDP: Request: PRACK sip:alice@10.1.22.101:5080, with ses
SIP/SDP: Request: PRACK sip:alice@10.1.22.101:5080, with ses
SIP/SDP: Request: PRACK sip:alice@10.1.22.101:5080, with ses
SIP/SDP: Status: 200 OK, with session description
SIP/SDP: Status: 200 OK, with session description
SIP/SDP: Status: 200 OK, with session description
SIP/SDP: Status: 200 OK, with session description
SIP/SDP: Status: 200 OK, with session description
SIP/SDP: Status: 200 OK, with session description
SIP/SDP: Status: 200 OK, with session description
SIP/SDP: Request: UPDATE sip:alice@10.1.22.101:5080, with se
SIP/SDP: Request: UPDATE sip:alice@10.1.22.101:5080, with se
SIP/SDP: Request: UPDATE sip:alice@10.1.22.101:5080, with se
SIP/SDP: Request: UPDATE sip:alice@10.1.22.101:5080, with se
SIP/SDP: Request: UPDATE sip:alice@10.1.22.101:5080, with se
SIP/SDP: Request: UPDATE sip:alice@10.1.22.101:5080, with se
SIP/SDP: Request: UPDATE sip:alice@10.1.22.101:5080, with se
SIP/SDP: Status: 200 OK, with session description
SIP/SDP: Status: 200 OK, with session description
SIP/SDP: Status: 200 OK, with session description
SIP/SDP: Status: 200 OK, with session description
SIP/SDP: Status: 200 OK, with session description
SIP/SDP: Status: 200 OK, with session description
SIP/SDP: Status: 200 OK, with session description
SIP: Status: 180 Ringing
SIP: Status: 180 Ringing
SIP: Status: 180 Ringing
SIP: Status: 180 Ringing
SIP: Status: 180 Ringing
```



```
SIP: Status: 180 Ringing
SIP: Status: 180 Ringing
SIP: Status: 180 Ringing
SIP: Request: PRACK sip:alice@10.1.22.101:5080
SIP: Request: PRACK sip:alice@10.1.22.101:5080
SIP: Request: PRACK sip:alice@10.1.22.101:5080
SIP: Request: PRACK sip:alice@10.1.22.101:5080
SIP: Request: PRACK sip:alice@10.1.22.101:5080
SIP: Request: PRACK sip:alice@10.1.22.101:5080
SIP: Request: PRACK sip:alice@10.1.22.101:5080
SIP: Status: 200 OK
SIP: Status: 200 OK
SIP: Status: 200 OK
SIP: Status: 200 OK
SIP: Status: 200 OK
SIP: Status: 200 OK
SIP: Status: 200 OK
SIP: Status: 200 OK
SIP: Status: 200 OK
SIP: Status: 200 OK
SIP: Status: 200 OK
SIP: Status: 200 OK
SIP: Status: 200 OK
SIP: Status: 200 OK
SIP: Status: 200 OK
SIP: Status: 200 OK
SIP: Request: ACK sip:alice@10.1.22.101:5080
SIP: Request: ACK sip:alice@10.1.22.101:5080
SIP: Request: ACK sip:alice@10.1.22.101:5080
SIP: Request: ACK sip:alice@10.1.22.101:5080
SIP: Request: ACK sip:alice@10.1.22.101:5080
SIP: Request: ACK sip:alice@10.1.22.101:5080
```

***Figure 5-8.*** *INVITE Call Flow between Bob and Alice*

## 5.4 Improvements

Now that the basic functioning is working, we could implement some enhancements to improve this work. First, we could configure this secured exchange identity mechanism as an option when launching the Client *IMS Communicator*. At present, the mechanism is automatically activated; the headers are directly modified when starting the Client IMS. Moreover, we could think about the reaction of the different entities when one's identity is not verified: should the entity send an error message or not, in case someone was trying to launch a Denial of Service attack for example. Maybe an alternative solution could be to limit the amount of error messages per identity.

We also could make the sources' use more dynamic: for the configuration of the Client (like to indicate the localization of the Public and Private Keys, the algorithm used to calculate the HIT, the name of the domain operator), it would be nicer to add them in the file Configurator.xml which defines all the configuration of the IMS client rather than letting them defined in java classes.

We could also think about the way we generated and saved the cryptographic keys in the IMS Clients and in the S-CSCFs. For the moment, there are saved in files that the programs can read. A better solution for the application server would be to save this information in the HSS database. In this case it would enable to save many identities.

Moreover, the distribution of these keys can also be studied. In the solution we presented in this work, they are pre-saved at the user and server code source directory. This scenario can be realistic as it would look like the one used for the shared secret between a phone user and his operator which is already saved in the SIM card of the user before he buys the phone. But we could also imagine a scenario where the cryptographic Keys distribution would be dynamically realized every time a SIP user wants to call another SIP user.

Simulate a real roaming of a user in a visited network could be envisaged, but seems to be a little bit complicated to implement. Indeed, this implicates a lot of programming to realize the roaming, and all the carried signaling. Nevertheless, implementing this scenario could be really interesting for the future of this project. It would enable us to have a better vision of its realistic characteristics.

## 5.5 Future work recommendations

Even though in this work we haven't exchanged any HIP specific information, there are some similarities to its use. The work can be improved in the future by implementing a real HIP identity association through a four way handshake mechanism as presented in the Figure 5-9.

The simple difference might be that in this case there will be no I1 message and the user A will start the handshake by sending in a HIP packet the message R1 which is meant to be a SIP "`INVITE`" message containing the identity and keying information of that user. Then the exchange will follow as a typical HIP handshake. No puzzle scheme is needed because it is supposed that the sender is known by the signaling network by being registered with the home operator using Authentication and Key Agreement (AKA) procedure.

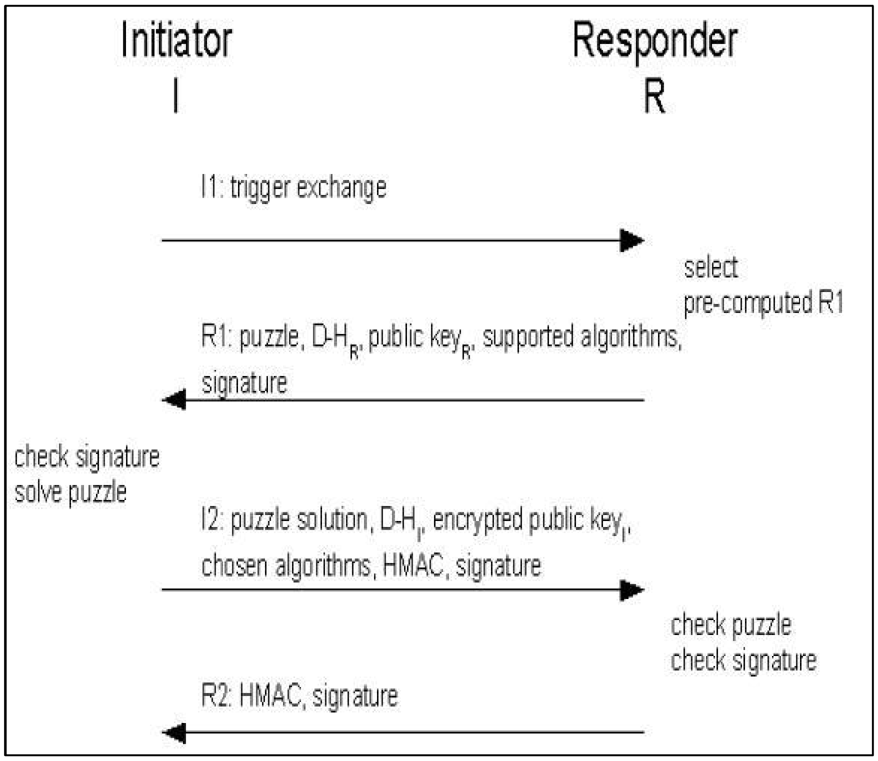

Initiator
I
Responder
R
I1: trigger exchange
select
pre-computed R1
R1: puzzle, D-H$_R$, public key$_R$, supported algorithms, signature
check signature
solve puzzle
I2: puzzle solution, D-H$_I$, encrypted public key$_I$, chosen algorithms, HMAC, signature
check puzzle
check signature
R2: HMAC, signature

***Figure 5-9.*** *HIP handshake [9]*

Another direction for future work can be the usage of a certificate. With the presented mechanisms, the identity and the other keying information are in the body section of the message. There can be another way of including them in a kind of certificate. This solution authorizes another network element to take care of the HIP negotiation as presented in [39] and then explained more in detail in [9].

# 6 Conclusions

During this master thesis we have accomplished several tasks that were set as objectives at the very beginning of the work. We succeed in installing and configuring two IMS platforms and integrate IMS clients on them. At the same time we implemented an AS on which we deployed the SIP Servlet application. The main purpose of the work was to implement a secure identity exchange based on the IMS architecture. For this, we modified the IMS client code by developing new classes and methods in order to calculate and then add the identity and the signature of the client and also to verify those of the SIP server. The same functionalities have been developed for the S-CSCF to assert its identity to its own users using the trust established by the roaming agreement.

The identity that we added as a new header (hash of the SIP entity public key) on the SIP messages is similar to HIT. Even though this identity is based on the HIT representation it is not exactly the same as HIT in a typical HIP exchange. In this work we haven't implemented HIP specific exchange, but just the assured identity exchange.

All this work relies on the fact that the trust between two end SIP entities is based on the existing trust relationship between the operators. In addition, every entity is in possession of a secure cryptographic identifier. One of the prerequisites of this work is that the IMS user has already registered previously with the home operator and has communicated his identity to its operator. At the same time the operators have communicated their identity to the users. In this case, the operator can check the correspondence of the identities and once those identities verified it can add its own identity. As the IMS users trust their own home operators and as the operators trust each other based on the pre-established agreement, the end users can be sure about the identity of their counterparts.

The mechanisms presented on this work have been evaluated according to their scalability aspects. We have analyzed the SIP messages containing the new headers. This has shown that the total size of the SIP message does not overpass the maximum size of the UDP datagram. The result of this work is that the high level architecture presented in order to enhance the security aspects of multimedia communications based on IMS architecture and SIP exchange is implementable. Nevertheless some scalability aspects such the size of the new SIP packets and the time delay has to be taken into account.

# References

The validity of the online references has been checked on September 11, 2010.

[1] ITU. The World in 2009: ICT FACTS AND FIGURES. A decade of ICT growth by mobile technologies. (2009) [Online]. http://www.itu.int/ITU-D/ict/material/Telecom09_flyer.pdf

[2] Dharwadkar S. N., Masood N.: Next Generation Network. Consumer Electronics, 2007. ISCE 2007. IEEE International Symposium on, ISBN: 978-1-4244-1109-2.

[3] Jarry-Lacombe B.: Network Global Strategy; Fixe/mobile convergence and IMS. Telecommunications Network Strategy and Planning Symposium. Networks 2008. The 13th International.

[4] ETSI Mobile Competence Centre (MCC). Overview of 3GPP Release 5. (Jun. 2003) [Online]. http://www.3gpp.org/article/ims

[5] FOKUS, OSIMS (Open Source IMs Core). [Online]. http://www.fokus.fraunhofer.de/en/fokus_testbeds/open_ims_playground/components/osims/index.html

[6] Handa A.: System Engineering for IMS Networks. Chapter 1 Introduction to IMS. ISBN-10: 0750683880. 2008.

[7] Camarillo G., García-Martín M. A.: The 3G IP multimedia subsystem (IMS): merging the internet and the cellular worlds. John Wiley & Sons pp. 1-268. May 2006.

[8] Poikselka M., Mayer G., The IMS: IP Multimedia Concepts and Services, 3rd edition. Wiley. 2009.

[9] Heikkinen S.: Establishing a Secure Peer Identity Association Using IMS Architecture. The Third International Conference on Internet Monitoring and Protection, pp.145-151. 2008.

[10] Open IMS Core, FAQ. [Online]. http://www.openimscore.org/faq/

[11] Perea R. M., Internet Multimedia Communications Using SIP: A Modern Approach Including Java Practice - 4th Edition. Morgan Kaufmann. 2008.

[12] Rosenberg et. al.: Session Initiation Protocol (SIP). RFC 3261. Jun. 2002.

[13] Handley M. et al.: Session Description Protocol (SDP). RFC 4566. Jul. 2006.

[14] Berners-Lee T. et al.: Uniform Resource Identifier (URI): Generic Syntax, RFC 3986. Jan. 2005.

[15] Sun Microsystems. The SIP Servlet Tutorial. [Online]. http://docs.sun.com/app/docs/doc/820-3007?l=en

[16] Ranganathan M., O'Doherty P.: JAIN SIP Developer Tools. [Online]. https://jain-sip.dev.java.net/

[17] Package javax.servlet.sip. [Online]. http://www.wesip.com/mediawiki/API/javax/servlet/sip/package-summary.html

[18] Dauman A.: How to implement an open IP encryption flow. (2006) [Online]. http://www.eetimes.com/design/industrial-control/4014835/How-to-implement-an-open-IP-encryption-flow

[19] Software Technology Support Center. (Nov. 2009) [Online]. http://www.stsc.hill.af.mil/crosstalk/2009/11/0911ChandlerLoyless.html

[20] Idrissi K.: Coding techniques and modulations, Data encryption lecture. INSA Lyon, Telecommunications, Services and Uses. 2008.

[21] Hunter M. T., Clark R. J., Park F. S.: Security Issues with the IP Multimedia Subsystem (IMS): A White Paper. 2007.

[22] Network working group Franks J.: HTTP Authentication Basic and Digest Access Authentication. RFC 2617. Jun. 1999.

[23] Salsano S., Veltri L., Papalilo D.: SIP security issues: the SIP authentication procedure and its processing load, IEEE Network 16 (6) 38–44. 2002.

[24] Wang F., Zhang Y.: A new provably secure authentication and key agreement mechanism for SIP using certificateless public-key cryptography. 2008.

[25] 3GPP. Security architecture. 3rd Generation Partnership Project Technical Specifications, TS 33.102 V7.1.0. Dec. 2006.

[26] Peterson J.: Session Initiation Protocol (SIP) Authenticated Identity Body (AIB) Format. RFC 3893. Sep. 2004.

[27] The Open IMS Core Project. [Online]. http://openimscore.org/

[28] FOKUS OPEN IMS PLAYGROUND, Fraunhofer. Serving CSCF. [Online]. http://www.fokus.fraunhofer.de/en/fokus_testbeds/open_ims_playground/components/osims/cscfs/s_cscf/index.html

[29] 3GPP. IP Multimedia (IM) Subsystem Cx and Dx Interfaces; Signalling flows and message contents. Specification detail: 29.228. Release 5. [Online]. http://ftp.3gpp.org/specs/html-info/29228.htm

[30] OpenIMSCore, Installation Guide. [Online]. http://openimscore.org/installation_guide

[31] Communications Research Group at the University of Cape Town, South Africa. UCT IMS Client. (2008) [Online]. http://uctimsclient.berlios.de/

[32] FOKUS Fraunhofer, Monster the client. (2009) [Online]. http://www.monster-the-client.org/

[33] Inovação PT. IMS Communicator. [Online]. http://imscommunicator.berlios.de/

[34] Sailfin Project. [Online]. https://sailfin.dev.java.net/

[35] Download SailFin Builds. [Online]. https://sailfin.dev.java.net/downloads/download_linux.html

[36] Wireshark. Analyze Wireless Data And Get Results. [Online]. http://wireshark.com/

[37] Eclipse, Open source community. [Online]. http://www.eclipse.org/

[38] Dinsing T. et al.: Service composition in IMS using Java EE. Ericsson Review No. 3, 2007.

[39] Koponen T., Gurtov A., Nikander P.: Application mobility with Host Identity Protocols. Proceedings of Network and Distributed System Securty Workshop'05. Feb. 2005.

[40] NIST-SIP: The Reference Implementation for JAIN-SIP 1.2. [Online]. http://snad.ncsl.nist.gov/proj/iptel/jain-sip-1.2/javadoc/overview-summary.html

## Appendix A: Configuring virtual interfaces on “/etc/network/interfaces”

```
auto lo
iface lo inet loopback

auto lo:1
iface lo:1 inet static
    address 10.1.22.101
    netmask 255.255.255.0

auto lo:2
iface lo:2 inet static
    address 10.1.22.102
    netmask 255.255.255.0

auto lo:3
iface lo:3 inet static
    address 10.1.22.103
    netmask 255.255.255.0

auto lo:4
iface lo:4 inet static
    address 10.1.22.104
    netmask 255.255.255.0

auto lo:5
iface lo:5 inet static
    address 10.1.22.105
    netmask 255.255.255.0
auto lo:6
iface lo:6 inet static
    address 10.1.22.106
    netmask 255.255.255.0
auto lo:7
iface lo:7 inet static
    address 10.1.22.107
    netmask 255.255.255.0
auto lo:8
iface lo:8 inet static
    address 10.1.22.108
    netmask 255.255.255.0
auto lo:9
iface lo:9 inet static
    address 10.1.22.109
    netmask 255.255.255.0

iface eth0 inet dhcp
```

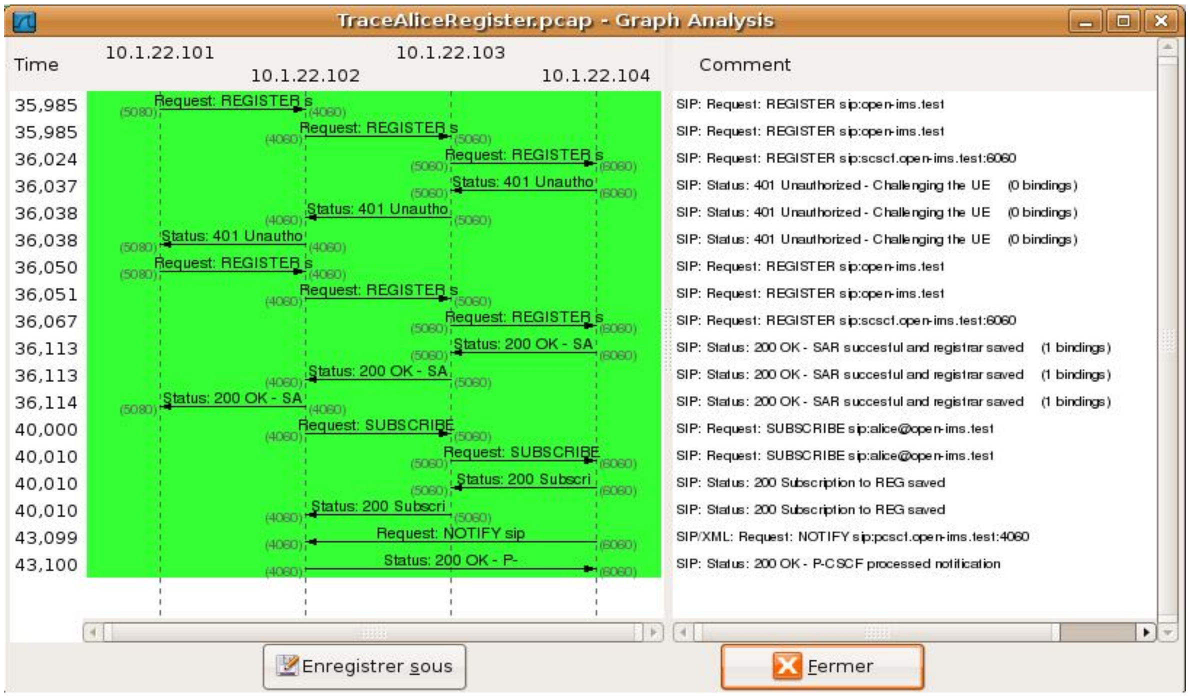
## Appendix B: DNS zone configuration file /etc/bind/open-ims.dnszone

```
$ORIGIN open-ims.test.
$TTL 1W
@                       1D IN SOA       localhost. root.localhost. (
                                        2006101001      ; serial
                                        3H              ; refresh
                                        15M             ; retry
                                        1W              ; expiry
                                        1D )            ; minimum

                        1D IN NS        ns
ns                      1D IN A         10.1.22.103

dns                     1D IN A         10.1.22.101

pcscf                   1D IN A         10.1.22.102
_sip.pcscf              1D SRV 0 0 4060 pcscf
_sip._udp.pcscf         1D SRV 0 0 4060 pcscf
_sip._tcp.pcscf         1D SRV 0 0 4060 pcscf

icscf                   1D IN A         10.1.22.103
_sip                    1D SRV 0 0 5060 icscf
_sip._udp               1D SRV 0 0 5060 icscf
_sip._tcp               1D SRV 0 0 5060 icscf

open-ims.test.          1D IN A         10.1.22.103
open-ims.test.          1D IN NAPTR 10 50 "s" "SIP+D2U" ""      _sip._udp
open-ims.test.          1D IN NAPTR 20 50 "s" "SIP+D2T" ""      _sip._tcp

scscf                   1D IN A         10.1.22.104
_sip.scscf              1D SRV 0 0 6060 scscf
_sip._udp.scscf         1D SRV 0 0 6060 scscf
_sip._tcp.scscf         1D SRV 0 0 6060 scscf

imsCommA                1D IN A         10.1.22.106
_sip.imsCommA           1D SRV 0 0 5070 imsCommA
_sip._udp.imsCommA      1D SRV 0 0 5070 imsCommA
_sip._tcp.imsCommA      1D SRV 0 0 5070 imsCommA

imsCommB                1D IN A         10.1.22.107
_sip.imsCommB           1D SRV 0 0 5080 imsCommB
_sip._udp.imsCommB      1D SRV 0 0 5080 imsCommB
_sip._tcp.imsCommB      1D SRV 0 0 5080 imsCommB

trcf                    1D IN A         127.0.0.1
_sip.trcf               1D SRV 0 0 3060 trcf
_sip._udp.trcf          1D SRV 0 0 3060 trcf
_sip._tcp.trcf          1D SRV 0 0 3060 trcf

bgcf                    1D IN A         127.0.0.1
_sip.bgcf               1D SRV 0 0 7060 bgcf
_sip._udp.bgcf          1D SRV 0 0 7060 bgcf
_sip._tcp.bgcf          1D SRV 0 0 7060 bgcf

mgcf                    1D IN A         127.0.0.1
_sip.mgcf               1D SRV 0 0 8060 mgcf
_sip._udp.mgcf          1D SRV 0 0 8060 mgcf
_sip._tcp.mgcf          1D SRV 0 0 8060 mgcf

hss                     1D IN A         10.1.22.105

sailfin                 1D IN A         10.1.22.109

pcrf                    1D IN A                 127.0.0.1
clf                     1D IN A                 127.0.0.1
```

## Appendix C: Functionning of IMS Communicator

Using *Wireshark*, we could launch a simulation and follow the different messages exchanged by the different entities. In this simulation, we have one IMS Client that is performing a registration to his home operator. The message flow is shown in the Figure 8-1. In this one, the addresses of the different entities are:

- IMS Client: `10.1.22.101`
- P-CSCF : `10.1.22.102`
- I-CSCF : `10.1.22.103`
- S-CSCF :`10.1.22.104`



| Time | Message | Comment |
|---|---|---|
| 35,985 | (5080) Request: REGISTER s (4060) | SIP: Request: REGISTER sip:open-ims.test |
| 35,985 | (4060) Request: REGISTER s (5060) | SIP: Request: REGISTER sip:open-ims.test |
| 36,024 | (5060) Request: REGISTER s (6060) | SIP: Request: REGISTER sip:scscf.open-ims.test:6060 |
| 36,037 | (5060) Status: 401 Unautho (6060) | SIP: Status: 401 Unauthorized - Challenging the UE (0 bindings) |
| 36,038 | (4060) Status: 401 Unautho (5060) | SIP: Status: 401 Unauthorized - Challenging the UE (0 bindings) |
| 36,038 | (5080) Status: 401 Unautho (4060) | SIP: Status: 401 Unauthorized - Challenging the UE (0 bindings) |
| 36,050 | (5080) Request: REGISTER s (4060) | SIP: Request: REGISTER sip:open-ims.test |
| 36,051 | (4060) Request: REGISTER s (5060) | SIP: Request: REGISTER sip:open-ims.test |
| 36,067 | (5060) Request: REGISTER s (6060) | SIP: Request: REGISTER sip:scscf.open-ims.test:6060 |
| 36,113 | (5060) Status: 200 OK - SA (6060) | SIP: Status: 200 OK - SAR succesful and registrar saved (1 bindings) |
| 36,113 | (4060) Status: 200 OK - SA (5060) | SIP: Status: 200 OK - SAR succesful and registrar saved (1 bindings) |
| 36,114 | (5080) Status: 200 OK - SA (4060) | SIP: Status: 200 OK - SAR succesful and registrar saved (1 bindings) |
| 40,000 | (4060) Request: SUBSCRIBE (5060) | SIP: Request: SUBSCRIBE sip:alice@open-ims.test |
| 40,010 | (5060) Request: SUBSCRIBE (6060) | SIP: Request: SUBSCRIBE sip:alice@open-ims.test |
| 40,010 | (5060) Status: 200 Subscri (6060) | SIP: Status: 200 Subscription to REG saved |
| 40,010 | (4060) Status: 200 Subscri (5060) | SIP: Status: 200 Subscription to REG saved |
| 43,099 | (4060) Request: NOTIFY sip (6060) | SIP/XML: Request: NOTIFY sip:pcscf.open-ims.test:4060 |
| 43,100 | (4060) Status: 200 OK - P- (6060) | SIP: Status: 200 OK - P-CSCF processed notification |

*Figure 8-1. REGISTER Call Flow*

This Figure can be understood by reading it step by step:

```
10.1.22.101 -> 10.1.22.102: SIP. Request "REGISTER" (1)
```

The request "REGISTER" created by the user is first sent to the P-CSCF (the proxy is always the first point of contact of the user). This message contains the IMS Client SIP identity.

```
10.1.22.102 -> 10.1.22.103: SIP. Request "REGISTER" (1)
```

The proxy just forwards it to the interrogating server to check the user's identity.

```
10.1.22.103 -> 10.1.22.104: SIP. Request "REGISTER" (1)
```

After interrogating the database HSS, the I-CSCF forwards the request to the S-CSCF associated to the user.

```
10.1.22.104 -> 10.1.22.103: SIP. Status "401 Unauthorized"
```

The S-CSCF sends a message "401 Unauthorized" containing a challenge to the I-CSCF. This challenge aims at authenticating the user, by providing a puzzle that he is the only one able to solve.

```
10.1.22.103-> 10.1.22.102-> 10.1.22.101:SIP. Status "401 Unauthorized"
```

The message is forwarded to the client IMS by the Interrogating and Proxy SIP Servers.

```
10.1.22.101 -> 10.1.22.102: SIP. Request "REGISTER" (2)
```

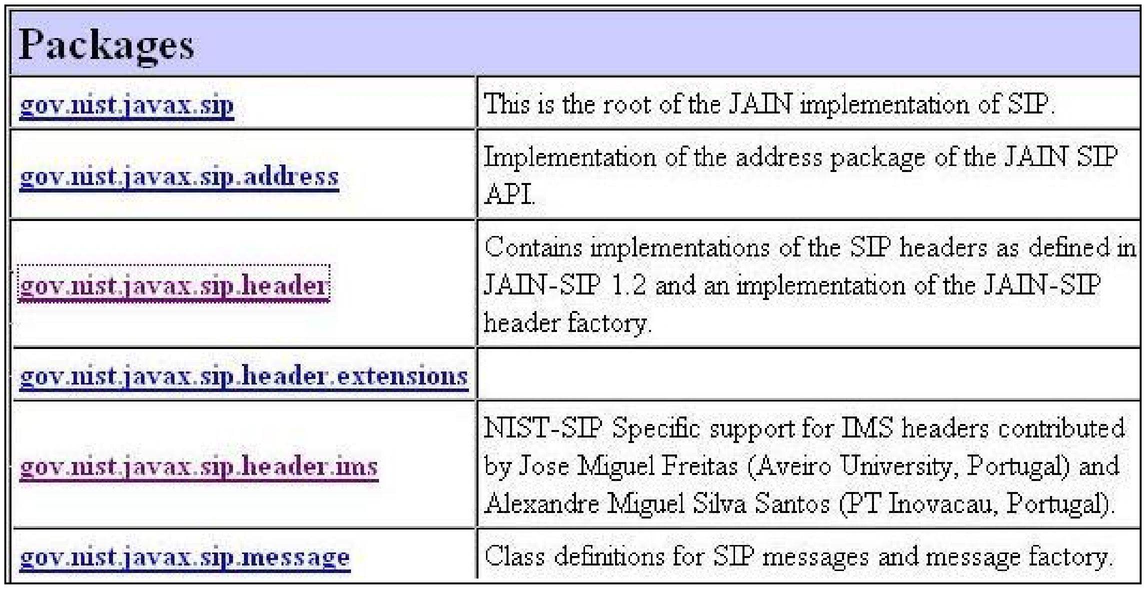
When receiving this message, the user resolves the challenge using his password (shared with the server), generates the answer (The second sequence of the request REGISTER) and sends it to the proxy.

```
10.1.22.102 -> 10.1.22.103 -> 10.1.22.104: SIP. Request "REGISTER" (2)
```

This message is then forwarded to the server by the SIP Servers which can check if the user answered the good response to the challenge, which would prove that he has the password, and so, that he is who he pretended to be. The S-CSCF generates so a "200 OK" message, which is forwarded to the user to warn him that he is well registered.

## Appendix D: NIST-SIP: The Reference Implementation for JAIN-SIP 1.2

The objective of this specification is to develop a standard interface to SIP that can be used independently or by higher level programming entities and environments. This specification was designed to provide a developer with a standardized interface for SIP services which are functionally compatible with the RFC 3261 [12]. This specification is a general purpose transaction based Java interface to the SIP protocol. It is rich both semantically and in definition to the SIP protocol. Some useful packages proposed by *Nist-Sip* are presented in the Figure 8-2.



| Packages | |
|---|---|
| gov.nist.javax.sip | This is the root of the JAIN implementation of SIP. |
| gov.nist.javax.sip.address | Implementation of the address package of the JAIN SIP API. |
| gov.nist.javax.sip.header | Contains implementations of the SIP headers as defined in JAIN-SIP 1.2 and an implementation of the JAIN-SIP header factory. |
| gov.nist.javax.sip.header.extensions | |
| gov.nist.javax.sip.header.ims | NIST-SIP Specific support for IMS headers contributed by Jose Miguel Freitas (Aveiro University, Portugal) and Alexandre Miguel Silva Santos (PT Inovacau, Portugal). |
| gov.nist.javax.sip.message | Class definitions for SIP messages and message factory. |

***Figure 8-2.*** *Some useful packages proposed by Nist-Sip [40]*

In the project, we used several objects and methods proposed by these packages. Actually, they were especially created and used for the IMS Client that we used (by *Jose Miguel Freitas*). More information can be found in [40].

## Appendix E: Testing the project

To test the project files, we first have to install the platform Open Source IMS Core and the tools (procedures described in the section **3**). Then, you need to configure the folders containing the IMS Clients (`/ims-communicator`) and the application server (`/sailfin`) in the place of your choice. In the **Appendix F**, an example of script that can be used is presented. You first need to adapt it to your configuration to indicate the location of all the elements, and save it as "`script.sh`" for example. It can be noted that if you want to test a configuration with two users (*Bob* and *Alice*), you need to create two folders containing the same code (`"/ims-communicator6"` and `"/ims-communicator7"` in the example), and then follow the procedure described in the **Appendix G** to configure the user as *Bob* or as *Alice*.

The script enables to easily launch the environment and the different elements. It's recommended to open several Consoles (or different instances in *Putty Connection Manager*) which are going to contain all the different elements. To do so, you first need to be "Root" in your Console, and then launch the script:

```
#>sudo bash
#>sh script.sh
```

Then, the script is launched and it writes all the possibilities that you have:

```
"*****Commands available : init, restart_as, p, i, s, hss, as, as_log,
as_log_vi bob, bob_run, bob_log, alice, alice_run, alice_log,
quit*****"
```

In a first window, you can start the environment (it restarts DNS, MySQL and Sailfin):

```
Init
```

Then, in each of the 6 consoles, you launch the different servers: p-cscf, i-cscf, s-cscf, hss. So first, you need to launch the script in each console:

```
sh script.sh
```

In the first console, you need to write :

```
P
```

Then, in the second one:

```
I
```

Then in the other consoles:

`s, hss`.

To be sure that the environment is working, you can read the numerous logs appearing in the SIP servers windows. The I-CSCF and S-CSCF must show [Open] connections, this means that they are dialoging with the HSS, waiting for users to register themselves. Before launching the users, you can run Wireshark, so that you will be able to analyze the traffic. More information about the configuration of Wireshark can be found in the section **3.4.1**.

Then, you can launch the two users *Bob* and *Alice*. For each of them, there are several options when you choose “bob” or “alice”. We recommend to first rebuild and then execute. For example, for Bob:

```
#>sh script.sh
Bob
Bob_rebuild
```

If the rebuild returns a “BUILD SUCCESSFUL” (or else, correct the mistakes, it is probably a background element missing):

```
#>sh script.sh
Bob
Bob_run
```

When this command is launched, a window opens representing the IMS Client. This one is shown in the Figure 8-3. The next step is to REGISTER the user, as shown in the Figure 8-4 and Figure 8-4 (Bob’s password is “bob”; Alice’s password is “alice”).

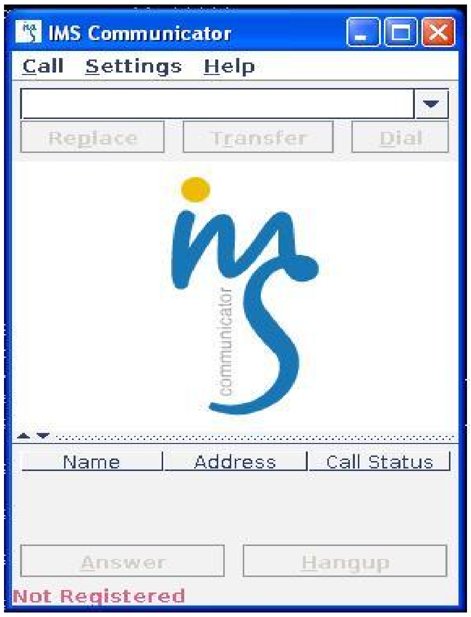


***Figure 8-3.*** *Client not registered*

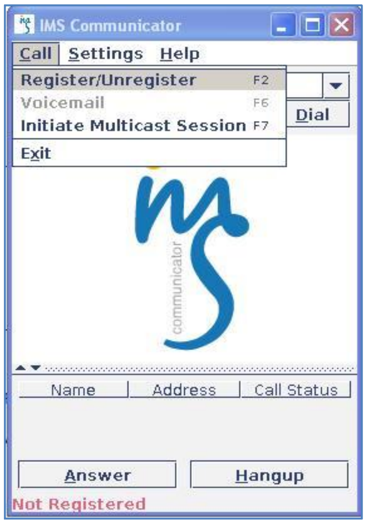
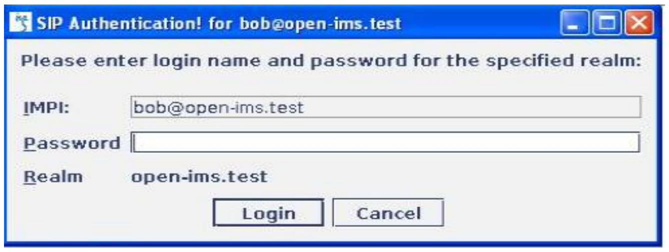


***Figure 8-4.*** *Procedure to register the user*

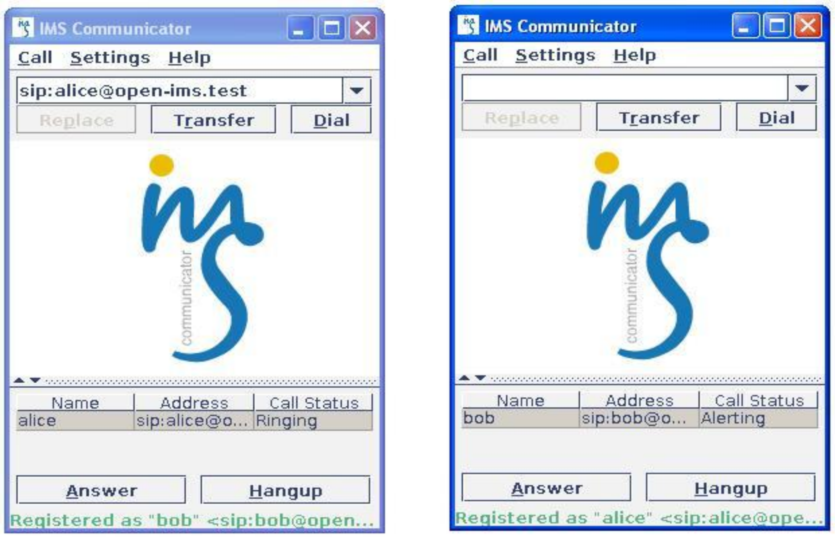


Please enter login name and password for the specified realm:

| | |
|---|---|
| IMPI: | bob@open-ims.test |
| Password | |
| Realm | open-ims.test |

***Figure 8-5.*** *Registration box*

Then, when another client "Alice" is registered, we can try to call her: as represented in the Figure 8-6, Bob has entered the Alice's sip address and has pushed the button "Dial".

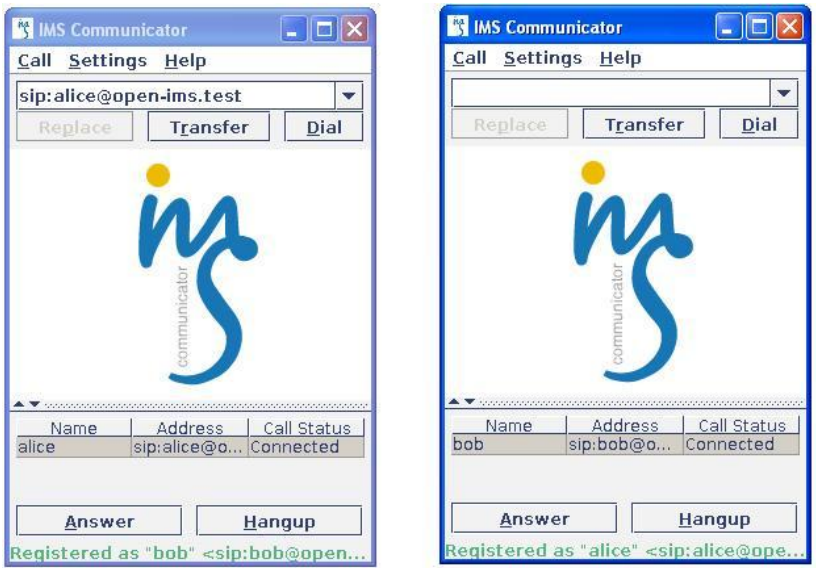


| Name | Address | Call Status |
|---|---|---|
| alice | sip:alice@o... | Ringing |

| Name | Address | Call Status |
|---|---|---|
| bob | sip:bob@o... | Alerting |

***Figure 8-6.*** *Bob calls Alice*

Alice can see the incoming call, and can decide to accept it by pushing “Answer”. Then when the connection is established, the status is turned to “Connected” as shown in the Figure 8-7.



sip:alice@open-ims.test

| Name | Address | Call Status |
|---|---|---|
| alice | sip:alice@o... | Connected |

Registered as "bob" <sip:bob@open...

| Name | Address | Call Status |
|---|---|---|
| bob | sip:bob@o... | Connected |

Registered as "alice" <sip:alice@ope...

***Figure 8-7.*** *Bob and Alice connected*

We can follow in Wireshark the different messages exchanged presented in the Section **5**.

## Appendix F: Script Shell: script.sh

```
#!/bin/bash
echo ""
echo ""
echo "*****Commands available : init, restart_as, p, i, s, hss, as, as_log, as_log_vi bob, bob_run, bob_log, alice, alice_run, alice_log, quit*****"
read a
case $a in
   init)
   /etc/init.d/bind9 restart;
   cd /etc/init.d/;
   /home/hay/sailfin/bin/asadmin start-domain domain1;
   /home/hay/sailfin/bin/asadmin start-database;
   mysql -p;
   echo "*****DNS, MySQL and Sailfin launched*****"
   ;;

   quit)
   /etc/init.d/bind9 stop;
   /home/hay/sailfin/bin/asadmin stop-domain domain1;
```

```
/home/hay/sailfin/bin/asadmin stop-database;
echo "*****DNS and AS servers stopped *****";;

restart_as)
/home/hay/sailfin/bin/asadmin stop-domain domain1;
/home/hay/sailfin/bin/asadmin stop-database;
/home/hay/sailfin/bin/asadmin start-domain domain1;
/home/hay/sailfin/bin/asadmin start-database
;;

p)
cd /opt/OpenIMSCore;
sh pcscf.sh;
echo "*****P-CSCF launched*****"
;;

i)
cd /opt/OpenIMSCore;
sh icscf.sh;
echo "*****I-CSCF launched*****"
;;

s)
cd /opt/OpenIMSCore;
sh scscf.sh;
echo "*****I-CSCF launched*****"
;;

hss)
cd /opt/OpenIMSCore/FHoSS/deploy;
sh startup.sh;
echo "*****HSS launched*****"
;;

as)
cd sailfin/samples/sipservlet/Identity;
ant all;
;;

as_log)
gedit        sailfin/domains/domain1/logs/server.log        trunk/ims-
communicator4/log/ims-communicator.app.log;;

as_log_vi)
vim sailfin/domains/domain1/logs/server.log;;

bob)
```

```
   cd /home/hay/trunk/ims-communicator6/;
   echo ""
   echo "***** Client IMS-Communicator 6 (bob)*****";
   echo "***** Choice : clean, make, rebuild, run-project, run, all
*****";
   read b ;
   case $b in
      clean)
      ant clean;;

      make)
      ant make;;

      rebuild)
      ant rebuild;;

      run-project)
      ant run-project;;

      run)
      ant run;;

      all)
      ant all;;
   esac;;

   bob_run)
   cd /home/hay/trunk/ims-communicator6/;
   ant run;;

   bob_log)
   cd /home/hay/trunk/ims-communicator6/log;
   gedit ims-communicator.app.log ims-communicator.debug.log ims-
   communicator.stack.log
   ;;

   alice)
   cd /home/hay/trunk/ims-communicator7;
   echo ""
   echo "***** Client IMS-Communicator 7 (alice)*****";
   echo "***** Choice : clean, make, rebuild, run-project, run, all
   *****";
   read b ;
   case $b in
      clean)
      ant clean;;
```

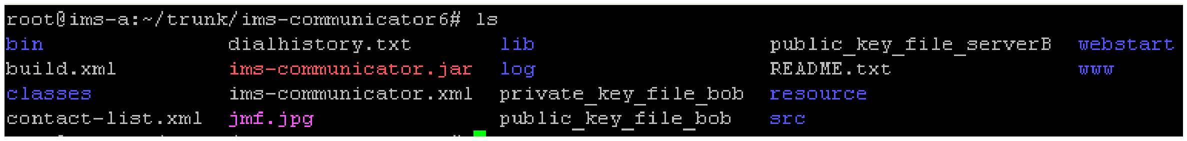
```
    make)
    ant make;;

    rebuild)
    ant rebuild;;

    run-project)
    ant run-project;;

    run)
    ant run;;

    all)
    ant all;;
esac;;

alice_run)
cd /home/hay/trunk/ims-communicator7;
ant run;;

alice_log)
cd /home/hay/trunk/ims-communicator7/log;
gedit    ims-communicator.app.log    ims-communicator.debug.log    ims-
communicator.stack.log

esac
```

## Appendix G: Configure the folder ims-communicator as “Bob” or as “Alice”

Some modifications need to be done to configure the project as a “bob” or “alice” profile. Bob and Alice differ from their sip addresses, the cryptographic keys that they have, and the server that they trust (if they belong to different domains). So, the next steps are going to describe how to change these values.

First, we need to open the file “`ims-communicator.xml`”. This file contains all the configuration of the user. Some fields need to be adapted to Bob or to Alice profile as described in the Section **3.2.3.**

Then, we need to put the correct keys in the folder “`/ims-communicator`”. In the Figure 8-8, we show the example of a user configured as “bob”. We can see that he has his private key and his public key, and also his home operator’s public key (“`public_key_file_serverB`”).



```
root@ims-a:~/trunk/ims-communicator6# ls
bin                 dialhistory.txt       lib                   public_key_file_serverB  webstart
build.xml           ims-communicator.jar  log                   README.txt               www
classes             ims-communicator.xml  private_key_file_bob  resource
contact-list.xml    jmf.jpg               public_key_file_bob   src
```

***Figure 8-8.*** *Ims-communicator folder*

Finally, some modifications need to be done in the file "`src/net/java/sip/communicator/sip/security/Security_ims_project_tools.java`". They represent the algorithm used by the user and his home operator's name:

```
private String algorithm ="SHA-1";
private String name_server = "homeB";
```

Some other modifications have also to be done in the file "`src/net/java/sip/communicator/sip/CallProcessing.java`". These different values represent the names of the files containing the keys presented in the Figure 8-8:

```
private String private_key_file_client="private_key_file_bob";
private String public_key_file_client="public_key_file_bob";
private String public_key_file_server="public_key_file_serverB";
```