

Virtual Communities in the IMS Network

ing. C.A. Molanus
Technical University Eindhoven SAN group
Version 1, November 2008



Supervisors: prof.dr. J.J. Lukkien
dr. S. Chen

Abstract

This thesis concerns the introduction of virtual communities into the IP Multimedia Sub-system (IMS) framework. IMS allows (mobile) telecommunications providers to offer their users services based on and built upon Internet applications, services and protocols. Virtual communities have the ability to offer an access control and organizational mechanism for the distributed IMS networks services. IMS services are extra services that telecom operators can offer their clients such as voicemail, mobile child locator or call waiting service. This thesis investigates how a virtual community architecture can be used in the IMS network. Along with this it also considers the pros and cons of deploying the Virtual Communities components on either the mobile device or in the IMS infrastructure. As a proof of concept, an example application was built based on an existing Virtual Community framework named VICSDA. The proof of concept showed that implementing Virtual Communities in an IMS network is possible with very few changes to the D3.11 VICSDA framework. The changes are primarily due to using Session Initiation Protocol (SIP) as the transport protocol. Using SIP was necessary for communicating over the IMS network. The paper for which this proof of concept was built was subsequently accepted and shall be presented at the second conference on internet multimedia architecture and application (IMSAA) in 2008.

Acknowledgments

I would like to acknowledge those who helped me in research write this document. Whithou their help it would might not have been able to complete this thesis, or at least within the time that I did.

dr. S. Chen: For her irreplaceable help with, researching VICSDA, and the writing style of this document.

prof.dr. J.J. Lukkien: For his guidance and support during the course of my project.

dr. J.I. den Hartog : For his help in assessing the validity of the proposed security protocols.

MSc C. Verzijl: For his help during the proofreading phase of this thesis.

Contents

1	INTRODUCTION.....	5
2	IMS.....	6
2.1	SIP	6
2.2	INNER WORKINGS	7
2.3	IMS SERVICES.....	9
3	VIRTUAL COMMUNITY FRAMEWORK	11
3.1	CORE VC SERVICES.....	11
3.2	ACCESS CONTROL	11
3.3	SERVICE DISCOVERY	12
3.4	ORCHESTRATION.....	13
3.5	VC FORMATION	15
3.6	SIP AS TRANSPORT PROTOCOL	17
3.7	EXPERIMENT	18
3.7.1	<i>Tools</i>	18
3.7.2	<i>Implementation</i>	19
3.7.3	<i>Scenario</i>	21
4	POST IMPLEMENTATION ANALYSIS	24
4.1	VIRTUAL COMMUNITY SECURITY	24
4.1.1	<i>Evaluation</i>	24
4.1.2	<i>Motivation for alteration</i>	28
4.2	DEPLOYMENT SCENARIOS	33
4.2.1	<i>Complete user deployment</i>	34
4.2.2	<i>Complete operator deployment</i>	35
4.2.3	<i>Hybrid deployment</i>	35
5	CONCLUSION	37
6	FURTHER WORK.....	37
7	GLOSSARY.....	39
8	REFERENCES.....	42
8.1	SOAP MESSAGES.....	44
8.1.1	<i>Server result response</i>	44
8.1.2	<i>Server Message</i>	44
8.2	SEQUENCE DIAGRAM OF VC FORMATION	45

1 Introduction

This thesis is the result of the master project of C. Molanus for the System Architecture and Networking (SAN) group at the Technical University of Eindhoven. The SAN group studies parallel and distributed systems with an emphasis on the architecture of networked embedded systems. The group considers both the hardware and software aspects of such systems from an academic point of view.

This thesis investigates the concept of using virtual communities in the Internet Multimedia Sub-system (IMS) framework. Virtual Communities have been studied and proposed in previous projects as an access control and organizational mechanism for distributed services[1]. One of the purposes of this thesis is to ascertain if users can share services with confidentiality and integrity over IMS networks.

At the time this thesis was written the Internet Multimedia Sub-system (IMS) had just emerged out of its infancy. Reports have been released that KPN is testing one of the first IMS networks in Europe and Ericsson announced an agreement with Beijing Netcom to provide them with a Command Supporting System based on Ericsson's IMS solution for the 2008 Beijing Olympic Games. IMS, which was originally viewed by telecom operators as an extra service to their clients, is now being viewed as a platform that can be used to offer services. With IMS networks expected to provide mobile telephone operators with a forecasted \$300 billion US Dollars in extra revenue by 2013[5], operators are willing to make the needed investments. The percentage of this revenue earned by an operator seems to hinge on the services they choose to offer. IMS services are extra services that telecom operators can offer their clients such as voicemail, mobile child locator or call waiting. With the popularity of peer-to peer (P2P) services, such as Bit-Torrent, it is possible that there will be a market for services which use P2P communication in IMS. A Virtual Community framework for IMS would allow operators to develop Virtual Community services which they can use as a new source of revenue or to attract new clients. The framework enables a service developer to abstract from the access control and organizational mechanism intricacies which would enable them to rapidly create new services. This allows the operator to react quickly to market demands for services that require this type of communication. Some operators such as BT and Verizon plan to offer an API to allow third party developers easy access to the system. For third party developers this framework would then also lower the cost of creating mobile applications which use this type of communication.

This thesis can be split into three sections. Firstly the relevant aspects of the IMS framework will be discussed. The second section will provide the description of a VC framework named VCMobile. The framework is based on the VC framework VICSDA described in [1]. VICSDA is a Virtual Community framework developed by SAN to provide network scoping for secure service sharing. The VICSDA framework is a sharing technology that also allows for composing applications by composing services. VICSDA has already been used in the iShare project [1] to provide access control and secure service discovery which showed that the concept is sound. VICSDA was designed to work on an Internet-like IP network therefore a few modifications had to be made for it to function properly on the IMS network. This thesis then investigates whether this virtual community architecture, formed around P2P communication, can be used in the IMS network. A description of the tools used to create a proof of concept is then given along with an explanation of where this framework fits into the development process. The proof of concept application which uses the framework is then described along with the deployment scenario. The third section analyzes the security mechanism used and possible deployment scenarios. The security mechanism of VICSDA was reassessed for this network type which revealed a few necessary protocol modifications. The pros and cons of deploying the Virtual Communities components on either the mobile device or in the IMS infrastructure were considered. The criterion used considers both, the needed characteristics of VICSDA (Secure service discovery and access control), and practical concerns associated with the manner in which the VC will be used. The deployment assessment is an extension of the same assessment made in the "Virtual community management for enabling P2P services in the IMS network"[21] paper which was accepted to the IMSAA 2008 conference.

2 IMS

This chapter provides some background information on IP Multimedia Subsystem (IMS) which will be needed in later chapters. IMS is a telecommunications technology originally designed by the 3rd Generation Partnership Project (3GPP). It was expanded to support other networks such as wireless LAN, CDMA and fixed land line and has become means to deploy services across multiple type of networks. IMS allows providers to offer their users services based on and built upon Internet applications, services and protocols. Services such as voicemail, mobile child locator or call waiting can be designed and implemented much quicker in IMS because the implementation would be less equipment manufacturer specific. The framework has done this by attempting to conform as much as possible to IETF "Internet standards". This allows Public Land Mobile Network (PLMN) operators to have access independence (independent control of there access network) and smooth interoperation with wire-line terminals across the Internet [4]. Session Initiation Protocol (SIP) is the IETF "Internet standard" protocol chosen for use as the primary communications protocol in IMS.

2.1 SIP

Session Initiation Protocol (SIP) is an application-layer control protocol for creating, modifying, and terminating sessions with one or more participants [3]. SIP sessions are used in internet telephone calls, multimedia streaming and multimedia conferencing solutions. This protocol can run on top of several transport protocols such as TCP and UDP. SIP allows end-point devices/applications to find one another and to agree upon session characteristics such as media-types or encoding.

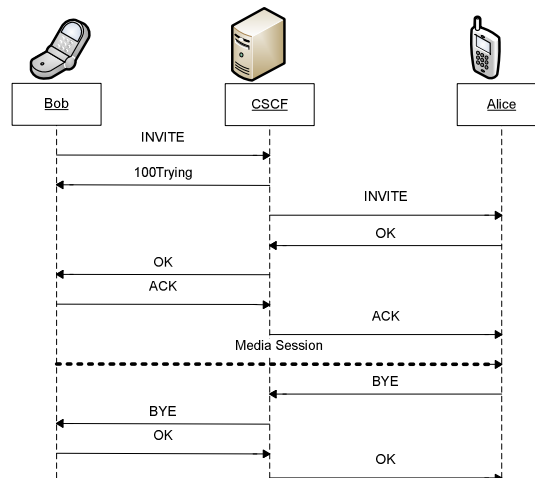


Figure 1 *SIP session in IMS*

Figure 1 shows a typical SIP session as it would occur in an IMS network where both end-users were registered on the same provider network. A provider network is considered to be the network infrastructure including the user devices connected to the network. As you can see all SIP communication needed for control is routed through the CSCF [4]. The CSCF, which will be explained in more detail in section 2.2, is somewhat of a central component of the IMS network. Having all SIP communication, or signaling as it is sometimes called, routed through the CSCF allows it to trigger a service before the signaling message arrives at the other end-device. A SIP address is typically displayed as sip:user@domain (i.e. sip:chris@tue.nl). The domain portion of the address (i.e. tue.nl) allows for the routing between networks. SIP was not intended to carry the data or media-content over a network. Instead it is intended to facilitate the setting up of a session so that another protocol can be used to carry the data or media-content over a network. Real-time Transport Protocol (RTP)[19] is typically used in IMS as the voice channel protocol. RTP is a protocol which was specifically designed by the IETF to deliver audio and video content over networks. SIP signaling is done using SIP Methods and Responses.

SIP Methods	Description
INVITE	Invite a user to start a session and to set up the connection
ACK	Used to allow for reliable message exchange during INVITE
BYE	Terminate a session or decline an INVITE
CANCEL	Terminate a request, or search for a user
OPTIONS	Request information about a services capabilities
REGISTER	Request a user in the current domain
INFO	Used in mid-session signaling
MESSAGE	Used in IM applications. RFC 3428
SUBSCRIBE	Request to be notified of an event RFC 3265
NOTIFY	Notification of an event RFC 3265

Table 1 SIP methods

Table 1 lists the SIP methods currently supported by the protocol. The methods MESSAGE, SUBSCRIBE, NOTIFY were added to the SIP standard after its Request for Comments (RFC) 3261 was posted. RFC 3261 specifies an internet standards track protocol for SIP and allows others to discuss and suggest improvements. Along with SIP methods there are also SIP Responses that were specified.

SIP Responses	Description
1xx : Provisional	Request received, continuing to process the request.
2xx : Success	The action was successfully received, understood, and accepted.
3xx : Redirection	Further action needs to be taken in order to complete the request.
4xx : Client Error	Bad syntax in request or it cannot be fulfilled at this server
5xx : Server Error	The server failed to fulfill an apparently valid request.
6xx : Global Failure	The request cannot be fulfilled at any server.

Table 2 SIP responses

The SIP responses currently supported by the SIP standard are listed in Table 2. The first digit of the Status-Code (i.e. 1 in 1xx) classifies the response. Class 1xx responses, for example, are responses with a Status-Code between 100 and 199.

2.2 Inner workings

For the purpose of this thesis an in-depth explanation of the inner working of IMS networks is not needed. Instead I will only cover the elements of the network pertinent to this thesis and describe the implementation of an example IMS service. For an in-depth explanation of IMS and all its components please refer to [4]. Figure 2 shows a layered model of a single domain IMS network displaying components in the transport, control, and service layer.

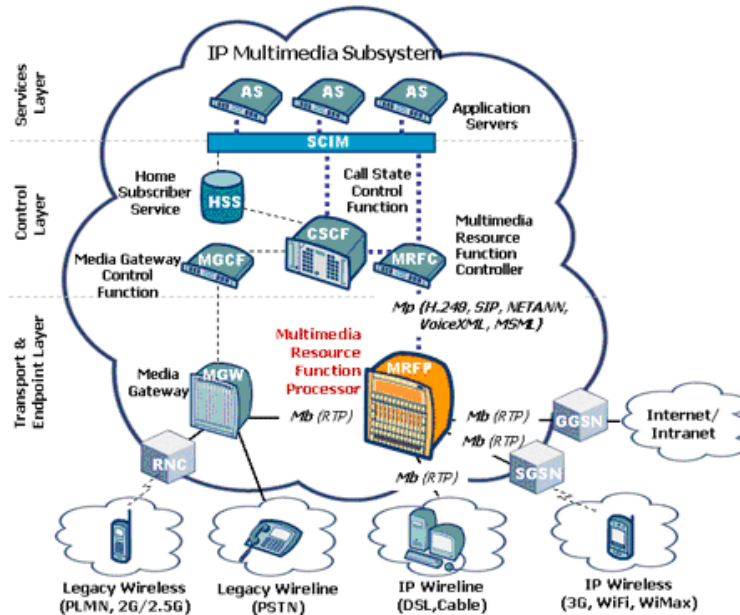


Figure 2 Layered IMS network diagram

Figure 2 shows the major components of the IMS network and how they are connected. The figure is layered to give an idea on which level the components function on. The service layer is concerned with the operations of the services / applications running on the application servers (AS). The control layer mediates calls and acts as a bridge between the transport and service layer. The transport layer is concerned with transition of data and providing end-points for the various networks.

HSS

The Home Subscriber Server (HSS) is a subscriber database which supports the server that processes SIP calls. The database holds all user-profiles (subscription related information) and performs authentication and authorization of users. The user profiles contain information such as the user's phone numbers, SIP address and which service that he or she is subscribed to such as voicemail.

CSCF

The Call Session Control Function (CSCF) is comprised out of several SIP servers. The CSCF handles all SIP signaling on the network and has the capability to trigger applications located on the application server in its domain. The list of services, which a particular user has subscribed to, is retrieved from the HSS when needed.

AS

The Application server (AS) can host and execute services which, through the CSCF, can be triggered by remote users. The services can be accessed by user both within and outside the provider's domain. The AS interfaces with the CSCF using SIP and typically executes services implemented as SIP servlets.

MRFC / MRFP

The Media Resource Function (MRF) is split up between two servers, namely the Media Resource Function Controller (MRFC) and the Media Resource Function Processor (MRFP). The MRFC acts as a SIP User agent to the CSCF and communicates over a channel referred to as the Mr reference point in [4]. The word channel is used here because the medium has not yet been defined in the IMS specification. The IMS specification describes nothing more than the protocol used for the Mr reference point, which is SIP (as defined by RFC 3261). The MRFC controls the MRFP via an Mp channel. The MRFP is a media node which implements all media related functions such as voice stream mixing and playing of tones. The MRFC and the MRFP communicate using a multitude of languages and protocols including:

- Session Initiation Protocol (SIP)

- Basic Network Media Services with SIP (NETANN)
- Media Gateway Control Protocol (H.248)
- Media Server Markup Language (MSML)
- VoiceXML

MGW

The MGW component is intended to provide support for network interconnections. One of the functions of the MGW may be to support transcoding between a codec used by the users' devices in the IMS network and the codec being used in the network of the other party.

MGCF

The MGCF is a SIP endpoint that initiates requests on behalf of the PSTN and Media Gateway. The subsequent nodes consider the signaling as if it came from a CSCF. The MGCF incorporates the network security functionality of the CSCF. This MGCF does not invoke Service Control, as this may be carried out in the General Switched Telephone Network GSTN or at the terminating CSCF.

In an effort to explain IMS to persons with a background in IP networks I offer the following simile: The core IMS network, shown in figure 2, could be best represented by an IP network gateway server. As in a gateway server its internal components (i.e. NAT, buffers) are not directly addressable, these components are transparent to the network user. This is also true for the IMS network; the components such as the MRFC and CSCF are also not directly addressable. The components simply react to the information passing through the system. The reaction is based on the information's characteristics and in some cases its content. However, IMS networks are likely to have a management system which, much like a gateway's management system, could address the internal components directly. This management system would likely only be accessible to the operator and is therefore outside the scope of this project.

An IMS service could be for example a voicemail service running on the Application server. The service could be triggered by a timeout started when the CSCF notified the voicemail service of a call being initiated to a user. The CSCF knows that the user has subscribed to a voicemail service because it retrieved the user's profile from the HSS when the call was initiated. When the voicemail service has not received a "OK" response from the user it assumes the user has not answered his phone. The call is then redirected to the MRFP where the voicemail service can then command the MRFC, using SIP commands, to make the MRFP open a voice channel within the session. The MRFP can then play the users pre-recorded voicemail greeting and record the caller's voice message. The MRFP could then encode the voice message left by the caller and transmit it back to the application server so that it may be stored. The user can then listen to his or her voicemail by placing a call to a predefined voicemail number. The CSCF would be programmed to trigger the voicemail playback application when this number is dialed. The application can then interact with the user in much the same way it interacted with the caller earlier to allow him or her to listen to the message left by the caller.

2.3 IMS Services

There has been a lot of hype around IMS services and how they might be used to revolutionize the mobile telecommunications industry. Unfortunately there have been very few practical business models put forth at this time. To give an idea of what IMS services could be used for, here are a few examples:

- Ring-back functions
- Mobile live, or on-demand, video.
- Voicemail
- Call forwarding
- Mobile child locator

Meridian Webster dictionary defines a service as the work preformed by one that serves. Within he context of this thesis applications would typically perform the work and which is used to ultimately serve a human client. IMS can basically be used to bring any type of electronic service to the client. A few of the services listed above are already widely available. IMS's contribution is to allow providers the ability to roll out

new services much quicker and cheaper than previously possible. IMS also hopes to give providers the ability to allow their users access to the services of their home operator even when they are located in another operator's network. A provider's service would be hosted on the application server within the network. Users located within and out side of the network can trigger a service by initiating a call session with a device. A call session is initiated by sending a SIP INVITE method tagged with the device's URI. Because this URI contains the domain where the service is located the session call can be routed to the correct domain over private networks or even the Internet. Once this call has arrived at the correct network it is captured by the CSCF which checks the users profile to determine if any action has to be taken. If the user is subscribed to a service the CSCF then triggers this service to process the session call.

SIP servlets

A current trend in the IMS service delivery realm is the use of SIP servlets as the bases for IMS services. A SIP servlet is typically in the form of a library or extendable class which can be used to create services which communicate using SIP. Vendors such as Oracle and IBM have their own implementation as do most of the J2EE Vendors. SIP servlets provide an execution platform in a full SIP network. Its limitation is that it can only provide the service via SIP (and in some instances HTTP), this is of concern to operators since their current telecommunications systems are typical SS7[9] based with a variety of value added service (VAS) protocols. SIP servlets provide an API with high level functions to allow developers to create services without worrying about the intricacies of the transport protocol. A SIP servlet used in IMS would run on the application server in the IMS network. However without defined interfaces for communicating with other components of the IMS network (i.e. MRFC) developed services would likely be equipment manufacture specific. The latest IMS specification document [4] does not do more than loosely define the interfaces available to the application server, with the exception of the CSCF/ Application server interfaces.

3 Virtual Community framework

The concept of virtual communities (VC) has been around for some time now. Some VC frameworks such as Napster, which is debatably the first widely used service orientated VC, made it into main stream news. The virtual community framework used in this thesis follows the ideology of VICSDA. VICSDA is a VC framework developed by SAN to provide network scoping for secure service sharing. The framework is a sharing technology that also allows for composing applications by composing services using orchestration. VICSDA is primarily the concept of using VCs for both access control and secure service discovery. It describes the components which it needs to manage aspects such as orchestration, resource management and fault tolerance. One of the differences between the VC framework used in this thesis and VICSDA is that this framework is closer to a Service Oriented Architecture (SOA) in that its components are loosely coupled. VICSDA does not go as far as to define a low level protocol, rather it abstracts from this by defining the information a component would need during the operation of the VC. As a starting point for this thesis the VICSDA D3.11 example implementation from [1] is used. This chapter describes the VCMobile framework, which is a VICSDA implementation for IMS. The VCMobile re-uses all components used in the VICSDA D3.11 example implementation. The only modifications made to the components where that which were necessary to allow them to function, as described in [1], on the IMS network. The components of VCMobile have to be lightweight and configurable and controllable from mobile devices such as mobile phones. This meant that parts of the protocol used in VICSDA D3.11 had to be modified to be able to provide that same functionality of the D3.11 implementation. Throughout the course of this chapter when a component design deviates from that described in the D3.11 framework an explanation will be given as to why this was done.

Both VICSDA D3.11 and the VCMobile framework are designed to allow users to share P2P services with service discovery and access control. VICSDA D3.11, and the VCMobile proof of concept associated with this thesis, use the SOAP protocol [8] for exchanging information between the framework's virtual community components. SOAP was chosen because it is an open standard and is widely used in Web Services to exchange data[8]. VCMobile uses SIP as its transport protocol as opposed to IP which is used by the D3.11 implementation. Using SIP as the transport protocol is primarily due to it being the only protocol that can be used to make a connection to another user based on the user's phone number or other permanent address. VICSDA, and therefore the VCMobile proof of concept, use secure network channels created by using pre-shared asymmetric keys, for all network communication. The distribution of these keys was left out of the scope of the VICSDA framework. VICSDA assumes that all services already possess the public key of any service with which it will communicate.

3.1 Core VC services

The services that are going to be described in this chapter where taken from the D3.11 implimentation of VICSDA, namely the *Certificate Manager*, *VCEntry*, *Repository*, and the *Orchestrator*. These services are considered to be the Core VC services. The four services must be present in the VC to have it function properly. These core VC services are responsible for creating managing and dissolving the virtual communities. They are what is also used to enforce access control and provide secure service discovery. All these services, including the Orchestrator, can be hosted on mobile devices (user's domain) or on the application server (the operator's domain).

3.2 Access control

VCMobile's access control allows the VC Entry point (*VCEntry*) managers to determine which users are allowed entry into the VC. By denying a user access, one can prevent a malicious user from querying the existence of services, within the VC. With no proof of membership to the VC, a service provider can deny the non-VC members service.

A *Certificate Manager* is a service that is authorized to issue certificates which prove membership to a specific VC. The *Certificate Manager* is the only service which issues VC member certificates. A VC member certificate is basically an X.509 certificate [14] and is signed using the *Certificate Manager's* Private Key. The *Certificate Manager's* Private Key along with the corresponding Public Key are

generated by the VC administrator during the VC formation phase. VCMobile assumes that all VC members can obtain the *Certificate Manager*'s public key in a manner which maintains its integrity. This allows a service provider to confirm that a service requester is a member of the VC and therefore is authorized to use the service. It can do this because the X.509 certificate can be verified using the public key of the certificate authority which in this case would be the *Certificate Manager*.

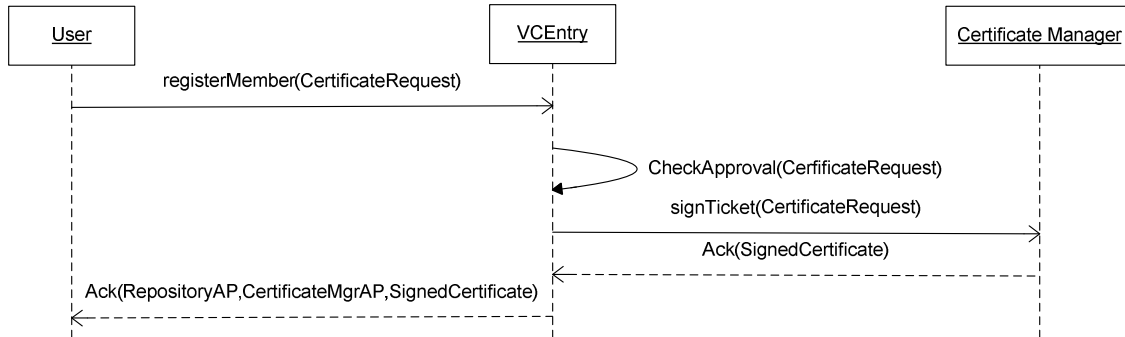


Figure 3 Sequence diagram for successful member registration.

The VC entry point, referred to as *VCEntry*, provides every new member with a VC Certificate, issued by a *Certificate Manager*, certifying acceptance into the VC. The *VCEntry* grants a membership request based on its join policy (i.e. blacklisted) in the manner shown in figure 3. A service provider can then be assured that if a member presents a valid member certificate he or she has been found, by the *VCEntry*, to adhere to the join policy. Once the user has been granted membership he can renew his certificate by sending a certificate request directly to the *Certificate Manager*.

3.3 Service discovery

Both VCMobile and VICSDA D3.11 use a *Repository* service for service discovery. This *Repository* service allows service providers to publish their services to a central point to allow other users to find them. The access point of the *Repository* is given to every VC member when they are granted admittance into the VC.

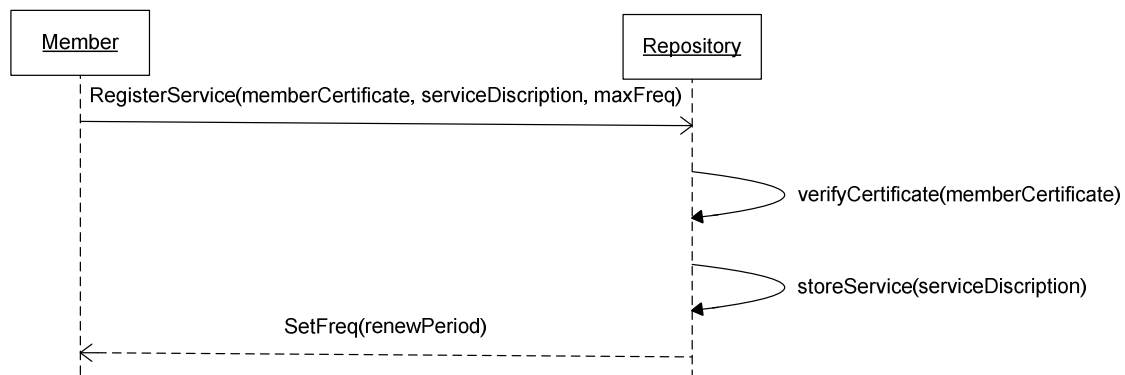


Figure 4 Service registration sequence diagram

A member that wishes to provide a service can then publish its service to the *Repository* in the manner shown in figure 4. To do this the member presents the VC member certificate, a service description and the maximum renewal period. The service description describes the service's interface and potentially a normal

language description. The maximum renewal frequency is the shortest time between registration renewals the service is willing to accept. After the *Repository* processes and stores the service's information it calls the service's SetFreq function which tells the service it has been registered and informs him of the renewal period the *Repository* has decided on. The renewal period is the maximum time a service can go without renewing its registration at the *Repository*.

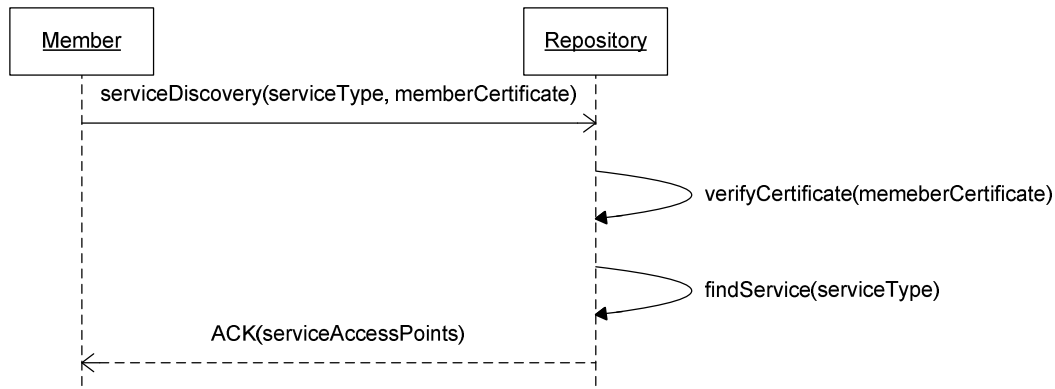


Figure 5 Service discovery sequence diagram

Once a service has been published to the *Repository* other users can obtain its access point by querying the *Repository* in the manner shown in figure 5. The *Repository* can potentially be very intelligent, in the sense that it could simply require basic a service description to search its internal database for a service which may fit the description. The main problem is the semantics for the service description. Natural language would be too difficult to parse due to its complex semantics and syntax. WSDL on the other hand would be suited for the task[15] but would require an extensive object/element type ontology to be truly dynamic. The ontology could be defined in, for example, OWL [17] which, like WSDL, is a XML based language. A service seeker could submit a sample service description in WSDL to the *Repository*. The *Repository* could then analyze the input and output parameter-types to find a service which has the same or compatible input and output parameter-types. The parameter-types ontology is needed to map between different element names for the same type of element.

For example, a service seeker may need a service which rotates an image to a specific orientation. The seeker can assume that this service would require an image and a degree of rotation and provide an image as output. A service provider who is offering such a service has already registered the service to the *Repository*. Unfortunately the element name for the image the provider has used was not “image”. The service provider may have used the name “file” or “input Image”. However since the service provider would have used a common object/element type ontology, his service description (in WSDL) would contain mappings from his terms to terms in the ontology. The input or output element which he refers to as “file” or “input Image” would therefore be linked to the term “image” in the ontology. The seeker could also use his own terms in the request to the *Repository* once he also has mapped his terms to the common object/element type ontology. Using this method the repository would be able to lookup and provide access points to services with interfaces which are similar to the requested interface.

Unfortunately this intelligent *Repository* remains a subject for future research. Creating the ontology and the mechanism which links the terms is tedious and time consuming work and therefore could not be incorporated into VCMobile. Similar work has already been done and can be found in [23].

3.4 Orchestration

The Orchestrator is a unique service in that its only purpose is to coordinate multiple services in such a manner as to provide a user or multiple users with a composed service. During the startup procedure of the *Orchestrator* it subscribes to the *Repository*'s “Service added” and “Service Removed” events. This allows

it to have knowledge of which services are registered at the *Repository* and their access points. The Orchestrator can then provide a member with a list of services which can be used to fill the roles needed for a composed service. Potentially the orchestrator could determine which services could be used to fill the roles needed for a composed service based on the services interface and description. However this would require a mechanism similar to the intelligent Repository described in section 3.3, and like the intelligent Repository it is a subject for future research. In VCMobile and VICSDA D3.11 the Orchestrator is hard coded with the interface description which identifies the services needed to fill the roles of a pre-defined composed service.

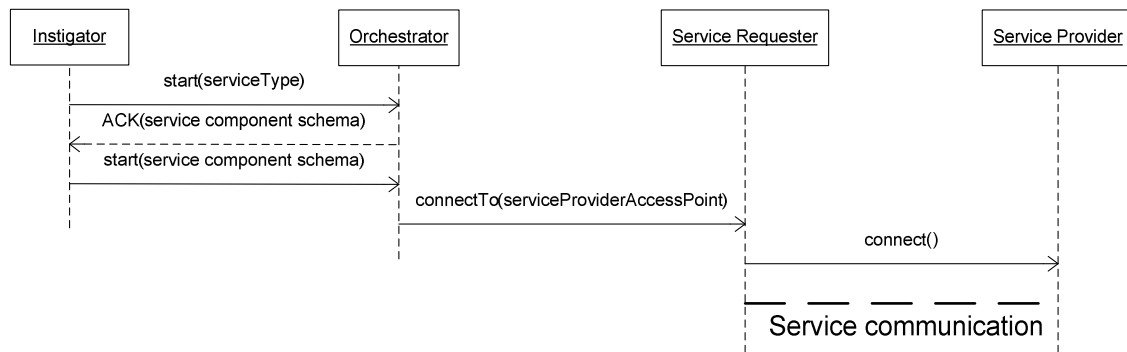


Figure 6 Orchestration of services by VC Orchestrator

The instigator shown in figure 6 is the actor which wishes to start such a composed service. A composed service is typically made up of one or more “Service Requester”/ “Service Provider” pairs. A “Service Requester” is a component that requires the help of another service to perform a task for which it is responsible for. A “Service Provider” is a service which can provide that help. It is also possible that the “Service Provider” itself may need to take on the role of a “Service Requester” in another pair to provide the help.

If the instigator in this scenario is not aware of the service’s interface it can query the *Orchestrator* by calling the start function with a blank service call. The instigator, which can be any member, may also wish to do this if she wishes to know the available sub-services for the composed service. The *Orchestrator* then responds with a service component schema which shows all service parameters which could be filled in. This response also contains all available services (which are registered at the *Repository*) which can be used as parameters for the start service call. Once an instigator calls the *Orchestrator*’s start function with a filled service component schema (parameters) the *Orchestrator* goes into action. The *Orchestrator* communicates with the sub-services and commands them to connect with one another to form a new composed service. Once the sub-services have been linked so that this composed service has been fully assembled and started, it removes itself from the communication dialog. In VICSDA D3.11 the orchestrator is considered a logic, in the sense that it has no local or external interface to allow users to configure it. This *Orchestrator* service uses almost the same logic but wraps it in an interface to allow it to be queried and commanded by a remote. To clarify, the *Orchestrator*’s only purpose is to combine services as is done in a typical Service Oriented Architecture but ceases to play a role in the operation of the newly created composed service once it has been started. The Orchestrator could play a monitoring role which could assist in a recovery procedure if a service were to become unavailable or non responsive. However this feature does not exist in either the VICSDA D3.11 or VCMobile it remains a topic for future research.

VICSDA envisions the *Orchestrator* receiving a composed service description which outlines the possible components of a composed service and how they could be combined. The semantics and syntax for this description is not finished and therefore could not be used in this experiment. In this experiment the *Orchestrator* is hard coded with a description of a specific composed service, the components needed for the service, and how to connect them.

Once an instigator requests to start a service the orchestrator responds with a service component schema.

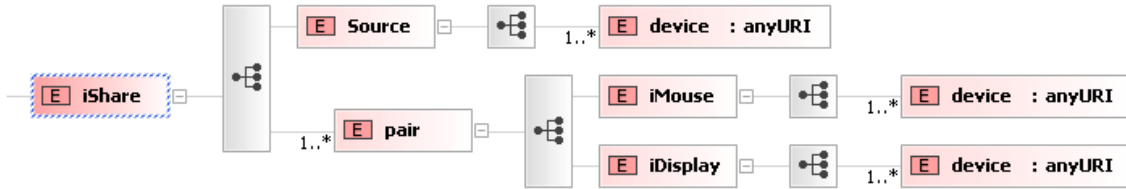


Figure 7 XML service component schema [18]

The graphical representations of an example XML schema shown in figure 7 would be used in a response to start a composed service name “iShare”. The items tagged with the **E** symbol are what are referred to as elements in XML schema [18]. This composed service requires one video stream source, indicated by the “Source” element, and one or more mouse and display pairs, indicated by the “pair” element in the schema. Both these items are defined as elements which are contained in the iShare element as a XML schema sequence. For every sub-service the Orchestrator provides a sequence of the URIs of all the devices which host the service, represented by the “device” element. This allows the instigator to see all the devices in the virtual community which can fill the necessary roles (sub-services) for the composed service. The instigator can then respond with a selection of devices to use for this particular composed service using the schema shown in figure 8. A selection is given by choosing one URI (device) which will fill the necessary role. The roles placed in a pair are done so to allow the instigator to indicate that the particular mouse role should be bonded to the particular display. The resulting choice should then be represented in a XML element which takes on the form shown below.

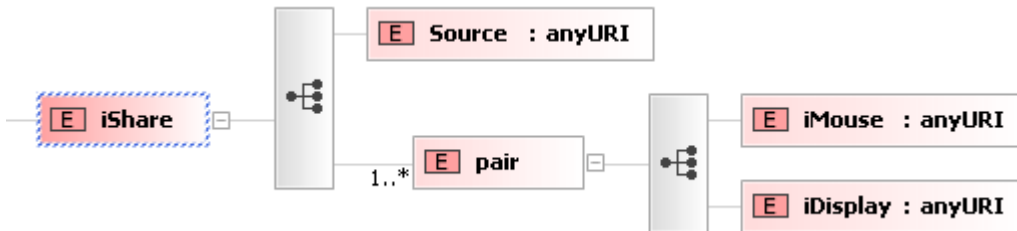


Figure 8 XML service response schema

This XML element (i.e. the iShare element in figure 8) informs the *Orchestrator* which devices to use to bring about the composed service. The intention was to allow the instigator to indicate which devices to use and which roles can be filled by any device available. If the instigator does not care which device fulfills a role, he or she simply leaves that element or pair out of the response. The *Orchestrator* will then select a device automatically. But this was not necessary for the proof of concepts and therefore was not implemented.

3.5 VC Formation

VCMobile was designed with the premise that users will be the ones who create the VCs. The formation of a VC is started by a user which appoints her-self the administrator role (Admin). The Admin was informed of a deployer service when he or she subscribed to the operator’s network. A *Deployer* is a service which the operator provides that has the capability to deploy *Certificate managers* and *Repositories*. VICSDA D3.11 does not use a *Deployer* service; instead it starts the services on the same device using system commands. The VC formation used in VCMobile is shown below in figure 9 .

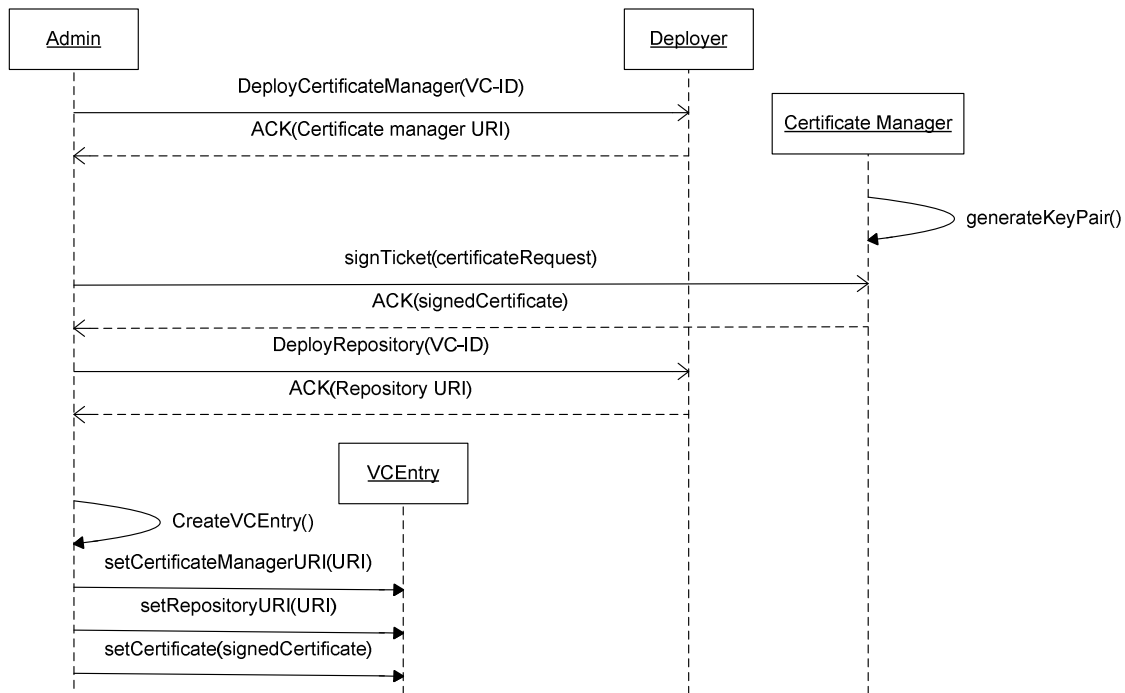


Figure 9 VC formation sequence diagram.

The Admin first contacts the *Deployer* and commands it to deploy a *Certificate Manager* who responds with the URI of the newly created *Certificate Manager*. The Admin then places a request to sign a certificate giving the Admin the administrator-role at the *Certificate Manager*. The *Certificate Manager* then signs the certificate using the private key, generated earlier, and sends the signed certificate to the Admin. The Admin then moves on to deploy the *Repository* by again contacting the deployer who responds with the URI of the newly created *Repository*. The Admin can now proceed to create the *VCEntree* service. The *VCEntree* service is always created on the device that hosts the Admin in both the D3.11 VICDSA implementation and in VCMobile. Once the *VCEntree* has been created it is given the signed certificate, received earlier, and the URIs of the *Certificate Manager* and *Repository*. The orchestrator service would be started on a device independently from the Admin. At that point, in VICSDA D3.11 implementation, the orchestrator would wait for all services, which are required for its composed service, to register at the *Repository*. The members would register at the *Repository* in the manner shown in figure 4. Once all the services were online it would start to construct the composed service. In this experiment the orchestrator is triggered by sending it a request to start a service as shown in figure 6. This was to allow the VC member to decide how and when the composed service would be started.

Thus far all the components of the VCMobile framework and the protocol they use to communicate have been explained. The VCMobile implementation is the result of changes made to the D3.11 implementation to allow it to function on a mobile telecommunications network. The components of VCMobile had to be lightweight and configurable and controllable from mobile devices such as mobile phones. The topic of this thesis was to investigate how a VC architecture could be used in specifically an IMS network. Therefore the framework had to be further altered to work with the addressing system of IMS, namely SIP. The most obvious way to allow the VCMobile framework to use the SIP addressing system could be to use the original protocol, which used IP, but wrapped in SIP sessions. This would require all services to exchange their IP addresses before communication could start. The device's IP address could however not be a public address. This meant that communication between users located in different provider networks may become problematic. For this reason using SIP as a transport protocol was considered.

3.6 SIP as transport protocol

The IMS CSCF server is the only server that can trigger a service running on the application server. The CSCF is designed to trigger services upon receiving SIP methods. This means that to use applications running on the application server, communication within the virtual community would have to be at least partially SIP based. SIP is also the only protocol that can be used when a client wishes to make a connection to another user based on the user's phone number or other permanent address. This is the reason why all VCMobile's communication in IMS is SIP based. Session Initiation Protocol SIP as the name suggests was not designed as a transport protocol. SIP is commonly used to set up and manage transport protocols such as Real-time Transport Protocol RTP. RTP is the protocol used in IMS to create voice channels, it could also be used to transport the data we will need to create and maintain a virtual community. This is however an excessive solution when you consider that what will be needed is to allow the virtual community components to pass a few messages that are each at the most a few hundred kilobytes.

SIP is the suggested protocol to use for communication between the user's device and a service on the IMS application server [4]. The currently advised way to create an IMS service is by designing it as a SIP servlet. The IMS service can then be created in a manner similar to that of creating a web service. This still does not completely eliminate the problem that SIP was not designed as a transport protocol. The term transport protocol is used here because VCMobile uses SIP to transport the commands and data used its protocol. The commands, for example those shown in figures 3 through 5, must be carried as data by SIP methods. For SIP to be used as a transport protocol it must be able to therefore carry data to the remote device. Instead of packages/data-streams as in TCP or UDP, SIP uses methods. Most of the methods can carry arbitrary data in one way or another. As not to depart too far from the original intentions of the methods, the INFO, MESSAGE and NOTIFY methods would be the most suited to the task of transporting other content.

- The INFO method [10] is intended to exchange session control information during a session. If a session were to be made between two components of a virtual community, the instructions and the associated data could be termed as control information.
- The MESSAGE method [11] was intended to provide instant messaging capabilities to SIP. The data sent between two VC components could be termed as messages. This means that sending information using this method could be seen as using it for its original purpose.
- The NOTIFY method [12] was intended to notify a user of an event. The method was intended to be used in combination with the SUBSCRIBE method. A user could request a subscription to be notified of the occurrence of an event. This mechanism could be used for the communication of some of the virtual community components, particularly the event based services.

In this project most of the communication will be in SOAP, thus plain text, which would be encoded using public key encryption to preserve privacy. The data which will be transported between the components in the virtual communities will therefore be in the form of byte arrays. All three of the above methods can carry a byte array payload essentially making SIP a transport protocol. The MESSAGE method was chosen for the implementation because out of the three, it was the most suited for the task in a terminology sense.

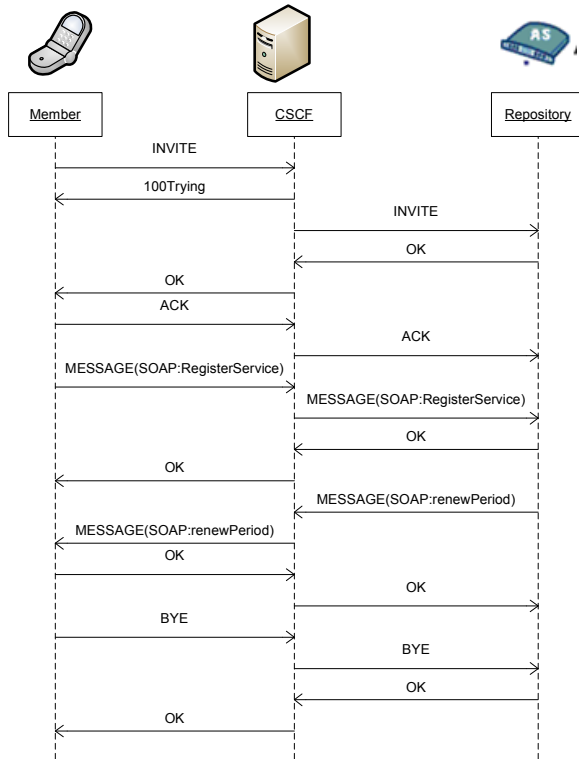


Figure 10 Sequence diagram of Service registration on SIP level

Figure 10 shows how the protocol described in figure 4, Register service, would look on the SIP level. The RegisterService RPC is encoded as a SOAP message which in turn is carried as the payload of a SIP MESSAGE method. The same goes for the renewPeriod() response of the *Repository*. Both these RPCs are enclosed in a SIP session between the Member and *Repository*. As figure 10 shows all communication is forwarded through the CSCF. The sequence diagram would look exactly the same if both services were hosted on mobile phones. In the case that the service is hosted on the Application server, such as in the case of the *Repository* and *Certificate Manager*, the CSCF routes the method to the application server instead of simply forwarding it back into the network.

3.7 Experiment

This experiment shows that VCMobile can form VCs which allow for secure access and service discovery on an IMS network. Because we did not have access to an operational IMS network the experiment was only carried out in a simulated IMS network.

3.7.1 Tools

The tools chosen to create this experiment were mainly chosen on the basis of availability. Only a few vendors and open source groups were active in this area at the time this thesis was written. The tools used to create this experiment were almost all still in their development stage. This meant that they were poorly documented and did not offer all the capabilities one might come to expect from such a tool.

3.7.1.1 SDS

Ericsson Service Development Studio (SDS) was chosen as the development environment for this experiment primarily because it was fully integrated with a simulated IMS network. SDS was also a freely available software package and at the time it was the only fully comprehensive tool for developing and testing IMS services. SDS has features that allow for end-to-end testing (which is testing of both the client and server side) of a new IMS service. This environment is based on the Eclipse's Integrated Development

Environment (IDE), and uses high level Application Programmer Interfaces (APIs) to hide the network complexity from developers. SDS uses an IMS network simulator with communication service emulators which are based on existing standards.

The SDS package includes the IMS Client Platform (ICP) which contains a high level IMS Core (pre-JSR281[6]), Communication Services (CoSe), and client APIs (pre-JSR325). Java Specification Requests (JSR) 325' ballot was submitted by Ericsson AB, and at the time of this thesis had just been approved. This specification will define a high level, IMS Communications Enabler framework API that hopes to provide Java ME based devices access to a set of IMS Communication Enablers[7]. Java specification requests JSR 281 is explained below. The package integrates easily with JavaEE/SIP Servers such as Glassfish/Sailfin which is also explained in more detail later in section 3.7.1.4.

3.7.1.2 JSR 281

Java specification requests JSR 281 provides a high-level API to access IP Multimedia Subsystem (IMS) services. This API hides IMS technology details and exposes service-level support to enable easy development of IMS applications [6]. To allow non SIP devices the capability to use the IMS network, Ericsson, who lead the standardization of JSR 281, also developed an IMS Client Platform (ICP). Ericsson's IMS Client Platform (ICP) pre-dates the JSR 281 specification but provides most of the same functions. It allows developers to design and test applications that use IMS networks and services on non SIP devices. ICP was designed to be used on commercial devices such as Symbian and Windows OS based terminals. In this experiment a UIQ terminal emulator was used to simulate actual mobile devices.

3.7.1.3 UIQ


 UIQ (formerly known as User Interface Quartz) is a mobile phone platform based on Symbian OS. The emulator was chosen because it was capable of running the ICP that comes with SDS. The emulator tries to reproduce the resource constraints that the actual phone would have. This allows the developer to see what the reaction time would be on such a device. Unfortunately the standard software does not output any quantifiable data but for the purpose of this experiment it will suffice. Applications can be created in SDS and then preprocessed to run on the emulator.



Figure 11 UIQ P1 emulator

3.7.1.4 Glassfish/Sailfin

Glassfish was used as the application server for this experiment. Glassfish support SIP servlets and was recommended by the SDS development environment. It also integrates very well with the SDS development environment which made it the most logical choice for an application server. The servlets must be written in JAVA which allowed the entire experiment to be consistent in the sense that it is all written in the same programming language.

3.7.2 Implementation

The framework is intended to be used by software developers who wish to use secure network scoping. The framework allows for scoping by using the virtual community concept. Developers can design applications without having to worry about forming and managing the virtual community by simply building their application on top of this framework. The framework then in turn uses SIP as its transport protocol to allow for communication over the IMS network.

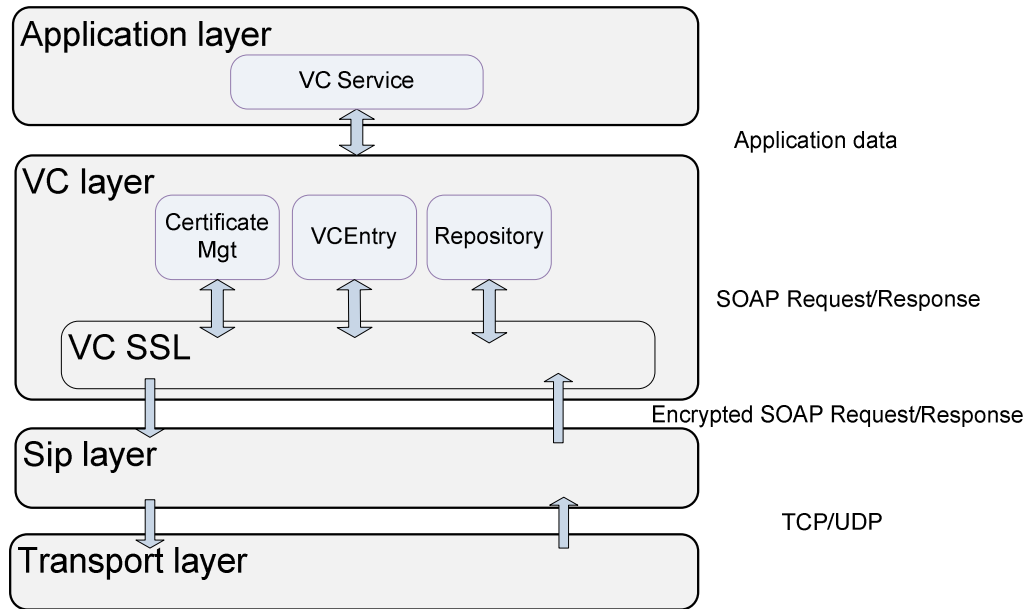


Figure 12 Virtual Community stack.

Instead of designing an application which interfaces directly with the SIP layer, developers would only need to call simple functions (the VC layer's API) which would handle the necessary communications to form and manage the VC. The framework would provide a library or extendable class which contained the functions which:

- Create a virtual community
- Publish services to the virtual community.
- Discover services in the virtual community
- Remove services registration from *Repository*.
- Accepting or denying membership requests.
- Defining a join policy.
- Call functions on other services over a secure channel.
- Destroy the virtual community.

The software developer could then focus on implementing his or her service without worrying about forming or managing the virtual community. The core VC components (i.e. *Certificate Manager*, *Repository*) are fully implemented in the libraries which make up the VC layer. A developer can create a Graphical User Interface (GUI) to allow the user to configure these components in a manner which is consistent with his or her application. The VC SSL component is suppose to represent a layer which creates and maintains secure channels similar to how standard Secure Socket Layer (SSL) communication channels are formed and managed. However work on this layer could not be incorporated in to the VCMobile proof of concept due to lake of time. This layer is contained within the VC layer because it is implemented specifically for this framework. In this context the transport layer is the IMS network hence the need for the SIP layer. The transport layer in the experiment was simulated by an IMS network emulator. VCMobile is designed as described here; however some of the functions mentions before were not implemented due to lake of time. Functions such as "Remove services registration from *Repository*" and "Accepting or denying membership requests." Where not implemented since it was not pertinent to prove that the concept could work.

To illustrate how this framework could be used, take for example a picture sharing application. A software developer could design an application which can host multiple services at the same time. The first would be a source service which could supply a picture stored on the user's mobile phone, upon request, to a VC member. The developer would then need to design a client (or sink) service which could make such a

request and display the received picture on the member's phone. The application would only allow authorized users access to the pictures by having the device hosting the source service also host the *VCEntry* service for the VC. The application could then specify which mobile phones could have access to the VC and therefore the pictures. This entire application could be implemented without the developer having a full understanding of the virtual community's inner workings.

3.7.3 Scenario

In order to test the feasibility of the VCMobile framework in IMS, a file sharing example scenario was designed.

In this scenario, an end user wants to share a picture stored on her mobile phone with her two friends. Considering the privacy of the content, she first creates a virtual community and then invites her friends to join. She registers a *file server service* into the *Repository* of this VC. And her two friends, who are VC members, now publish their *file sink services and mouse services* into the *Repository*. The file sink service will allow them to receive, and view, the file when the mouse service recognizes, and sends, a double click event to the file server. She then starts an *Orchestrator* service which subscribes to the *Repository* and collects all service currently registered. Because she already knows the URIs of her friends she can issue the *Orchestrator* the start command with a completed service selection schema. The *Orchestrator* then binds the *file server service* with both of the *file sink services* by commanding them to connect to the *file server service*. The orchestrator then commands the *file server service* to subscribe to the two *mouse services* for their double click mouse events. Once this is done, the picture will be transmitted from her mobile phone to one of her friends' mobile phone. The picture destination can be switched to her other friend through a double click mouse event generated by another mobile phone.

3.7.3.1 Deployment

To gain experience in deploying services in both the operator and user domain some services were implemented in each domain. The experiment does not cover all deployment scenarios, but rather aims to show that all are possible. Figure 13 shows the chosen deployment scenario for this experiment.

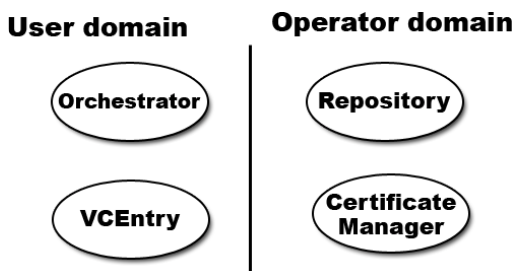


Figure 13 Deployment scenario of experiment

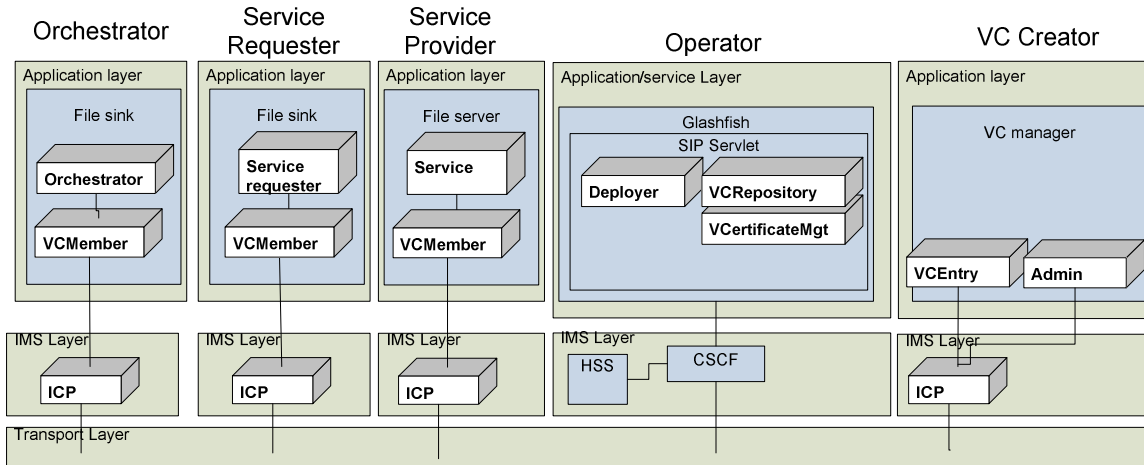


Figure 14 Layered deployment diagram.

Figure 14 shows the layers deployment diagram for deployment scenario of the experiment. The ICP components in the IMS layers of the mobiles are pre-JSR 281 implementation of a IMS communication API(see 3.7.1.2). The services which needed a lot of storage capacity or processing power were implemented on the application server and as many as possible were implemented in the user domain. Deployment scenarios are discussed further in section 4.2 later in this thesis.

3.7.3.2 Mobile application

The mobile application is limited to two basic roles. The application can either be triggered to create a virtual community or to host one or more VC services.

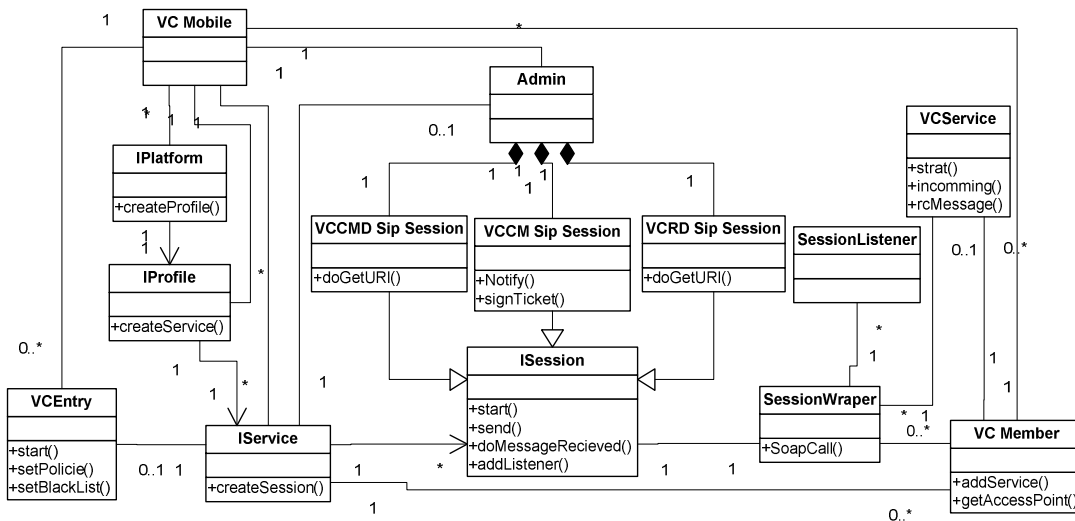


Figure 15 Mobile application class diagram.

Figure 15 shows the class diagram of the mobile application. If the application is to function as the creator of the VC the Admin class is instantiated. This class starts a remote instance of each of the core VC services (i.e. *Repository*, *Certificate Manager*) located on the application server in the manner described in section 3.5. The VC Certificate Manager Deployer (VCCMD) session class is specifically designed to interact with the deployer to bring about an instances of the *Certificate Manager* located on the application server. The *doGetURI()* function in this class generates the *DeployCertificateManager(VC-ID)* remote procedure call (RPC) shown in figure 9. This is the call which asks the deployer service on the application

server for a URI of a *Certificate Manager*. The VC Certificate Manager (VCCM) session class handles the initiation sequence of the *Certificate Manager*, this includes the `signTicket(certificateRequest)` RPC shown in figure 9. The VC Repository Deployer VCRD session class is specifically designed to interact with the deployer to bring about an instances of the *Repository* located on the application server. The `doGetURI()` function in this class generates the `DeployRepository(VC-ID)` RPC shown in figure 9. Once all of the core VC components are started the Admin class creates a *VCEnter* service. From that point on the *VCEnter* will handle all incoming connections.

If the application was to start hosting one or more VC services the *VCMember* class would be instantiated. The *VCMember* class provides all basic functions including queries to the core VC services (i.e. `getAccessPoint()`). By calling the `addService` function the *VCMember* class registers the service to the Registry and instantiates the class. The *VCMember* class can host any service which extends the *VCService* class. Once an incoming session is detected the session name is inspected to determine which service to bind the session to. The *VCService* class maintains a collection of open sessions which allows it to have multiple simultaneous connections.

3.7.3.3 Server application

The server application was not split up into different classes to allow for it to quickly be written. However it was implemented in a manner that would allow each service to be easily isolated. No information is exchanged internally between services. All communication is carried out using sessions. The application dispatches the request to the appropriate code based on the “To” address of the received request.

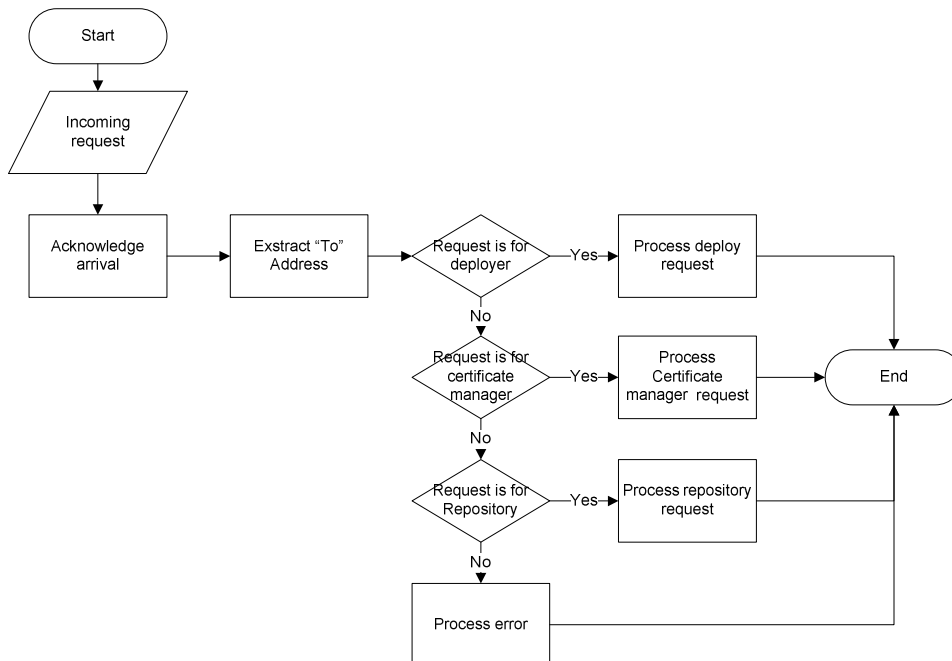


Figure 16 Flow chart of server application dispatcher

The application first functions as a deployer and upon request can deploy a *Certificate Manager* and a *Repository*. The application determines which role it should function as by assigning a URI to every role. If a session request is received which is addressed with the URI of the *Repository* the session is processed accordingly. In the case of the orchestrator subscription request, the application retains the session response variable so that it may notify the orchestrator if someone has registered a new service.

4 Post implementation analysis

The goal of this thesis was to investigate how a VC architecture could be used in an IMS network to provide secure service discovery and access control. A few modifications had to be made to the existing D3.11 VC framework [1] to allow it to be practical for use in a mobile telecommunications network. The framework provides network scoping for sharing distributed services on an IMS network. This chapter analyzes the changes and choices which led to the VCMobile framework described in the previous chapter and proposes further modifications where needed.

The VICSDA D3.11 security mechanism, which has been reused by VCMobile, is what was essentially intended to provide access control and secure service discovery. The security mechanism of VICSDA D3.11 had to be re-assessed for this network type because it was not designed with the characteristics of the IMS network in mind.

4.1 Virtual Community Security

The VICSDA D3.11 security mechanism, which is reused by VCMobile, is what was essentially intended to provide access control and secure service discovery. The mechanism had to be re-assessed to determine if it could still provide the characteristics by which VICSDA is defined. Therefore the framework was re-assessed on the basis of its vulnerability to attacks and if it could maintain:

- Secure service discovery
- Sharing of services
- Access control

VICSDA is defined as using VCs to provide these three characteristics. It will be shown that due to the modifications made to the D3.11 framework VCMobile's ability to maintain these characteristics was severely hampered.

4.1.1 Evaluation

This assessment was restricted to attacks on these three points because they are the core characteristics of VICSDA. Attacks on other characteristics, such as fault tolerance and recovery, were not assessed because VICSDA does not claim to maintain them. To facilitate the assessment, attack trees [20] were generated for the above listed requirements of VICSDA. The attack trees helped in examining possible strategies of potential attackers. The tree itself does not provide any countermeasures but does help in the threat analysis. Based on the analysis a proposed alteration to the security protocol is then given which should ensure that the framework can provide access control and secure service discovery.

4.1.1.1 Espionage

Espionage involves a user obtaining information which he or she is not permitted to possess. If a user were to gain knowledge of which members were requesting the location of services it would be a breach of the security of the service discovery mechanism.

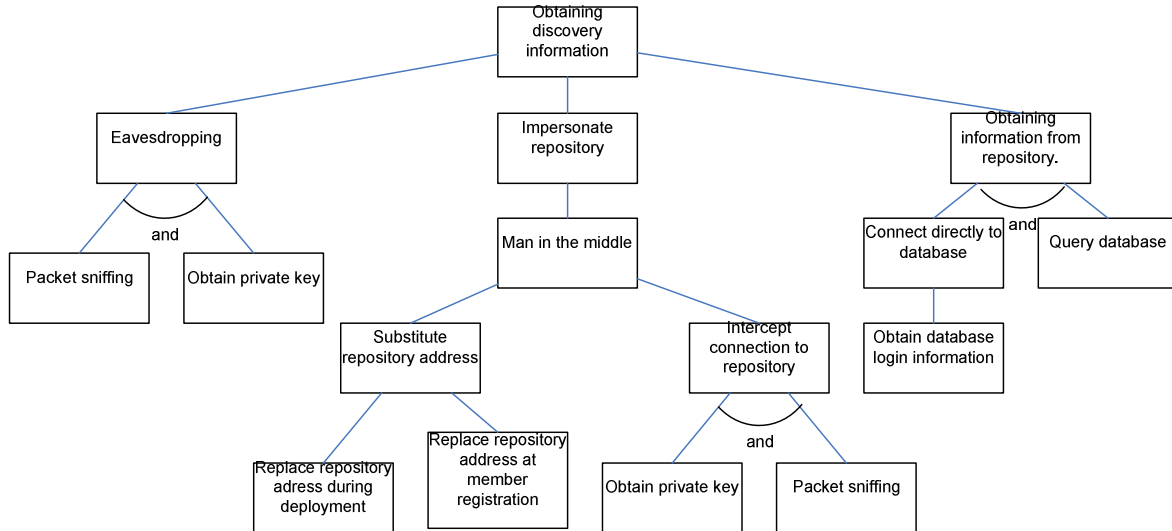


Figure 17 Espionage

When evaluating an attack tree the leaf nodes are considered to be the most important, they are the only ones covered in the following section.

Packet sniffing

Packet sniffing is a technique which is almost always possible on IP networks where the nodes are in the same subnet. However it is unlikely that an IMS implementation will allow the broadcasting of packets to users which they are not addressed to. Packet sniffing on the signal level could although be possible with enough insight into the operator's system. Nevertheless the VICSDA framework uses asymmetric keys to create secure communication channels

Obtaining private key

Obtaining the private key of the *Repository* would be a difficult task since the keys are generated at start-up and never sent over the network. It is still possible to obtain the keys but these attacks fall outside the scope of this assessment.

Replacing repository address during deployment

If the communication channel to the *Deployer* was not encrypted replacing the *Repository*'s address with an attackers address could be possible. However if the *Repository* were to be deployed in the user domain the address would likely be entered by hand. The deployment of services in the user or operator's domain will be covered later.

Replacing repository address at member registration

Replacing the *Repository*'s address at member registration is possible if the channel was not encrypted. A man in the middle attack is possible since it is not possible to require the user to have a certificate to verify his or hers identity at this time. The attack would involve an attacker executing a man-in-the-middle attack on the communication between the valid user and the *VCEnter*. But if this channel was encrypted it would not be possible to accomplish this.

Obtain database login information

If the *Repository* were to use a standard database application it is possible that the database could be accessed remotely. If so, once an attacker has obtained the login information he or she can have access to the service requesting information if it is stored in the database.

4.1.1.2 Denial of service

Preventing a user from accessing a service would hinder service sharing within the virtual community. This can be through hindering the discovery process, disturbing the service directly or disrupting the communication between the service and the user.

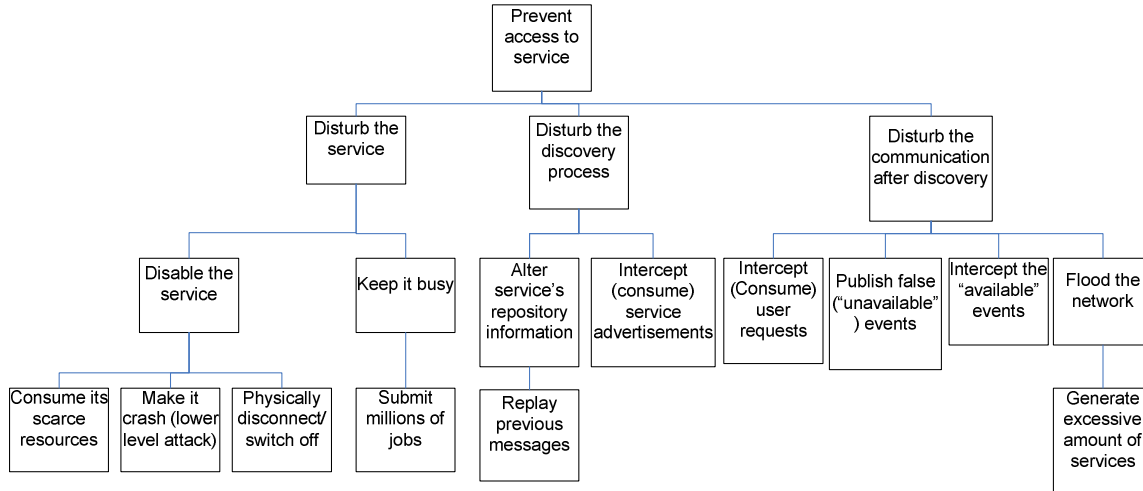


Figure 18 Denial of service attack tree

All leaf nodes of the attack tree, with the exception of flooding the network, present attacks which would require the attacker to know the location, or address, of the service. VICSDA assumes that once a member has gained access to the virtual community he or she is relatively trust worthy. Since service discovery can only be done once a user has gained access to the community these attacks depend on the attacker knowing the location of the service(s). It is conceivable for an attacker to obtain this information since locations in the IMS network are the user's SIP address or telephone number.

4.1.1.3 Unauthorized Access

A user has obtained "unauthorized access" if he or she gained access to the virtual community in a manner other than being authorized by the *VCEntry* in the way the Admin intended.

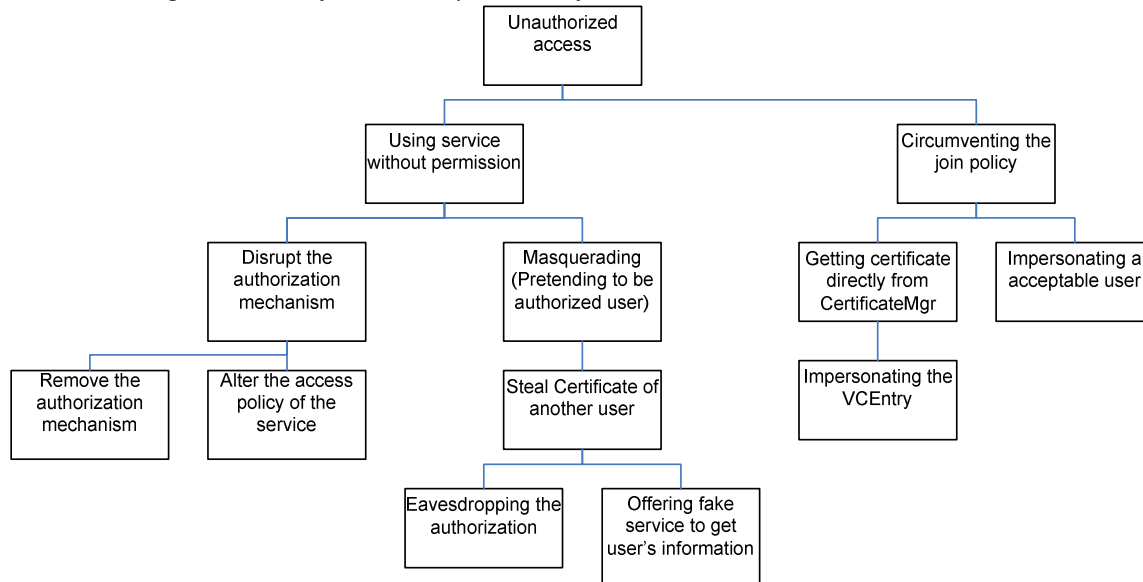


Figure 19 Unauthorized access attack tree

Again, in evaluating VCMobile's security mechanism we consider only the leaf nodes of the above graph. This is done because it is the most specific attack, while the nodes higher up the tree would be more general attacks.

Removing the authorization mechanism

Removing the authorization mechanism is a valid attack but would require physical access to the device which is running the *VCEntry* service. The *VCEntry*'s remote interface does not allow for the disabling or removal of the authentication mechanism. It is however possible for an attacker to gain access to the device on an OS level and thereby removing the authorization mechanism, but this falls outside the scope of this evaluation.

Altering the join policy

Altering the join policy of the *VCEntry* service would also require physical access to the device hosting the service. The *VCEntry*'s remote interface does not allow for the alteration or substitution of the join policy. The policy can only be configured once before the service is started. It is however possible for an attacker to gain access to the device on an OS level and thereby altering the join policy before the service is started, but this falls outside the scope of this evaluation.

Eavesdropping on the authentication

During the registration procedure the users have no certificate to authenticate themselves. The connection is therefore susceptible to a man in the middle attack. But if this communication were to be done over a secure channel the man in the middle would have to possess the private key of the user. If the attacker would not have the private key of the registering user he or she would not be able to prevent the registering user from becoming aware that something is wrong.

Offering fake service to get user information

Offering fake services to get user information can only deliver the user's certificates. Since the certificates are signed by the *Certificate manager* they are computationally unfeasible to successfully fake. A service could request the members private key as an input-value, but this request would be suspicious.

Impersonating the VCEntry

Impersonating the *VCEntry* is possible if the attacker was able to intercept the connection. Once the attacker has successfully intercepted the connection he or she can then substitute the necessary information they have gotten from the would-be member before sending it to the real *VCEntry*. This would essentially be a man in the middle attack. But this is not possible if secure channels with pre-shared keys are used

Impersonating an acceptable user

This strategy is similar as the one above but in this situation the attacker already knows the information of a valid member.

4.1.1.4 Vulnerabilities

While examining the security mechanism of VCMobile a vulnerability was found. The vulnerability became evident when the distribution of public keys was considered. The distribution of these keys was left out of the scope of the VICSDA framework. VICSDA assumes that all services already possess the public key of any service which it will communicate with. In the VICSDA D3.11 implementation the *VCEntry* services had to be host on the same device which hosted the certificate manager. Hosting these services on different devices in an attempt to make the framework more loosely coupled is what created this vulnerability. Having a truly loosely coupled framework means that there is no practical way to provide the operator with the public keys of the services other than over the network. This would mean that the public key of the *Certificate Manager* would need to be sent to the *VCEntry* service. Since no key manager is used in the VICSDA or IMS framework there is no way to create a secure channel to send this key. This would make the communication vulnerable to a man-in-the middle attack.

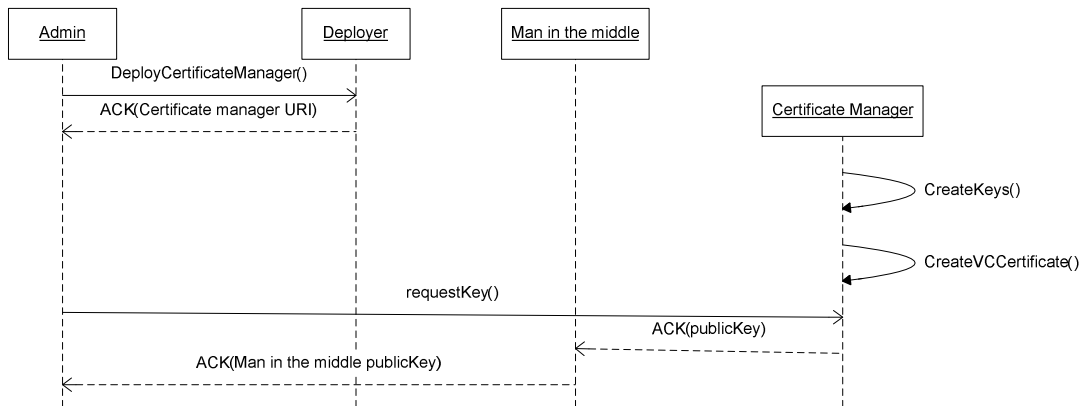


Figure 20 Sequence diagram of Man-in-the-middle attack

The message which contains the public key of the *Certificate Manager* could be intercepted in route by a third party. The third party could then imitate the *Certificate Manager* by sending his own public key to the *VCEntry*. The third party can then intercept all requests from the *VCEntry*, decrypt them, using his private key, and re-encrypt them, using the public key it intercepted, before sending them on to the certificate manager. By being in this position the third party can generate VC Certificate independent of the *VCEntry* without arousing suspicion.

4.1.2 Motivation for alteration

This security mechanism is only needed if it is possible to intercept a communication over the network. GSM has an encrypted channel between the mobile and the base station which would not allow an attacker to intercept a communication without internal knowledge of the operators system. However IMS was designed to communicate with other networks, including IP networks, where it is possible to intercept a communication. Therefore the framework must be secure against attacks that intercept communication between services. This thesis proposes a security protocol based on the SSL/TLS protocol which along with creating secure communication channels provides a mechanism for distributing keys. SSL/TLS was designed to be transport independent; therefore using this protocol over SIP should be as secure as using SSL/TLS over TCP/IP. This security framework also has some benefits when compared to the existing framework. The proposed security modifications allow for multiple *Certificate Managers* which can issue certificates for the same VC. In the current framework the *Certificate Manager* is a central component, there can only be one *Certificate Manager* which signs certificates and renews them when they expire. As the framework is now defined, if the *Certificate Manager* were to go down it would not be possible for users to enter the VC or renew their certificates. Using the proposed security framework would allow for redundancy; in that if a *Certificate Manager* goes down the user can simply use another in its place. This design actually fits better with the decentralized ideology of VICSDA.

4.1.2.1 Proposal

The proposed security framework resembles that of today's web services[24]. This would mean making use of the X.509 certificates and a certification path.

```

Certificate:
Data:
  Version: 1 (0x0)
  Serial Number: 7829 (0x1e95)
  Signature Algorithm: md5WithRSAEncryption
  Issuer: C=NB, ST=Eindhoven, L=Netherlands, O=TUE,
         OU=Certification Services Division,
         CN=TUE/emailAddress=server-info@tue.nl
  Validity
    Not Before: Jul  9 16:04:02 1998 GMT
    Not After : Jul  9 16:04:02 1999 GMT
  Subject: C=KPN, ST=?, L=?, O=?,
         OU=VC-ID, CN=SIP:bob@kpn.nl
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (1024 bit)
      Modulus (1024 bit):
        00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:33:35:19:d5:0c:64:b9:3d
        :41:b2:96:fc:f3:31:e1:66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:70
        :33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:16:94:6e:ee:f4:d5:6f:d5:ca:
        b3:47:5e:1b:0c:7b:c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:8fa0:
        21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:d2:75:6b:c1:ea:9e:5e:5c:ea:7d:
        c1:a1:10:bc:b8:e8:35:1c:9e:27:52:7e:41:8f
      Exponent: 65537 (0x10001)
    Signature Algorithm: md5WithRSAEncryption
    93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:92:2e:4a:1b:8b:ac:7d:99:17:5d
    :cd:19:f6:ad:ef:63:2f:92:ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:d0:a
    2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:0d:19:aa:ad:dd:9a:df:ab:97:50:6
    5:f5:5e:85:a6:ef:19:d1:5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:8f:0e:
    fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:68:9f
  
```

Figure 21 sample X.509 certificate issued by the certificate authority

A certificate authority (CA) would therefore be needed with its root certificate included in the framework package. The CA could then issue signed “Certificate Manager certificate” upon request. The request would be placed by the Administrator of the VC and contain the addresses of the certificate managers he is intending to use. The CA would then only be guaranteeing that it is not possible that a user other than those the Admin specified can obtain certificate manager certificates for a VC with the Admin’s chosen VC-ID. The certificates subsequently issued by the certificate managers will then contain a certification path which leads to the CA.

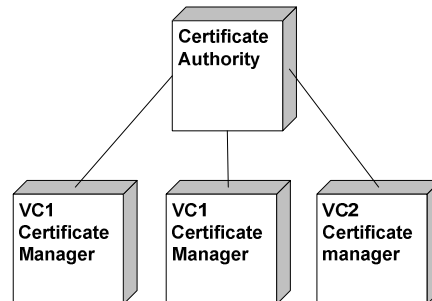


Figure 22 Certification path

Since any client would have to be built using the framework package, it would therefore contain the root certificate of the CA and can therefore verify the certificate manager’s certificate. The verification mechanism is explained in detail in [14]. A brief explanation of this mechanism would be as follows. The root certificate of the CA contains the CA’s public key. Since the certificate was delivered with the software it is assumed it has not been tampered with. Using the public key of the CA a user can decode the signature of the certificate manager’s certificate. The signature is essentially a MD5 hash of the rest of the certificate. The user then calculates its own MD5 hash of the certificate for comparison. If the decoded hash matches the calculated hash the certificate is assumed to be valid.

4.1.2.1.1 Server authentication handshake

The proposed handshake protocol is essentially a chopped down version of the SSL/TLS handshake. An example of the SSL/TLS handshake protocol is shown in the figure below.

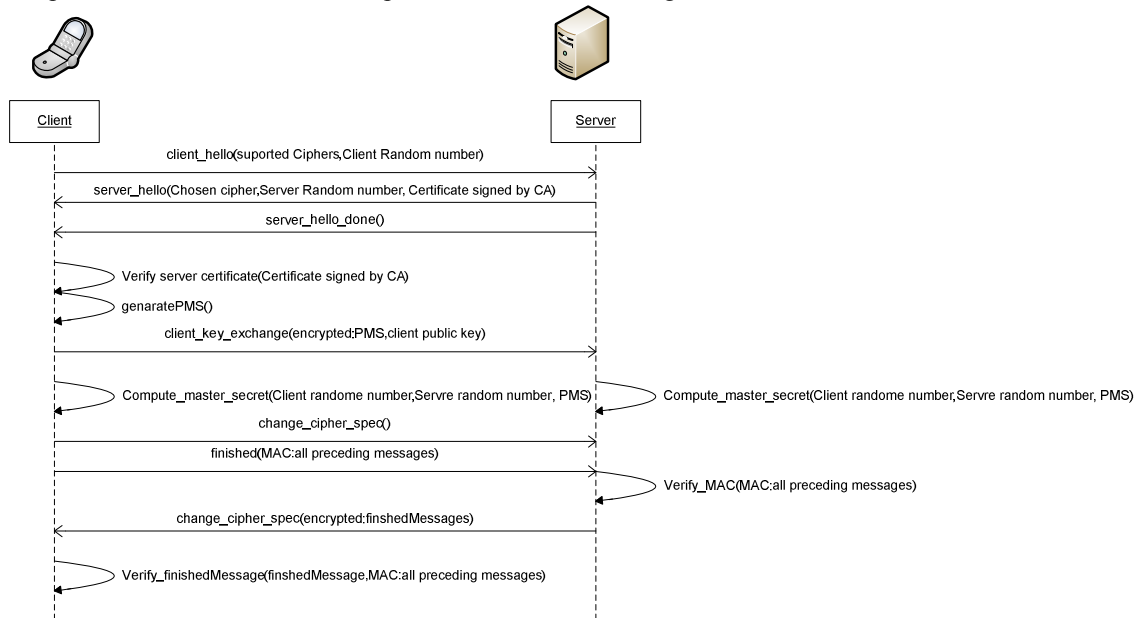


Figure 23 Simple SSL/TLS handshake

The following message description is taken from [22].

- A Client sends a ClientHello message specifying the highest TLS protocol version it supports, a random number, a list of suggested cipher suites and compression methods.
- The Server responds with a ServerHello message, containing the chosen protocol version, a random number, cipher suite, and compression method from the choices offered by the client. The server may also send a session id as part of the message to perform a resumed handshake.
- The Server sends its Certificate message (depending on the selected cipher suite, this may be omitted by the Server).
- The Server sends a ServerHelloDone message, indicating it is done with handshake negotiation.
- The Client responds with a ClientKeyExchange message, which may contain a PreMasterSecret, public key, or nothing. (Again, this depends on the selected cipher.)
- The Client and Server then use the random numbers and PreMasterSecret to compute a common secret, called the "master secret". All other key data for this connection is derived from this master secret (and the client- and server-generated random values), which is passed through a carefully designed "pseudorandom function".
- The Client now sends a ChangeCipherSpec record, essentially telling the Server, "Everything I tell you from now on will be encrypted."
- Finally, the Client sends an encrypted Finished message, containing a hash and MAC over the previous handshake messages.
- The Server will attempt to decrypt the Client's Finished message, and verify the hash and MAC. If the decryption or verification fails, the handshake is considered to have failed and the connection should be torn down.
- Finally, the Server sends a ChangeCipherSpec and its encrypted Finished message, and the Client performs the same decryption and verification.

At this point, the "handshake" is complete and the Application protocol is enabled. Application messages exchanged between Client and Server will be encrypted using the Master Secret (MS).

The proposed handshake protocol creates a fresh secure communication channel which is based on the SSL/TLS handshake. The only significant change was eliminating the choice of ciphers since VCMobile uses only one cipher algorithm. Below in figure 24 is the proposed message sequence for entering the virtual community.

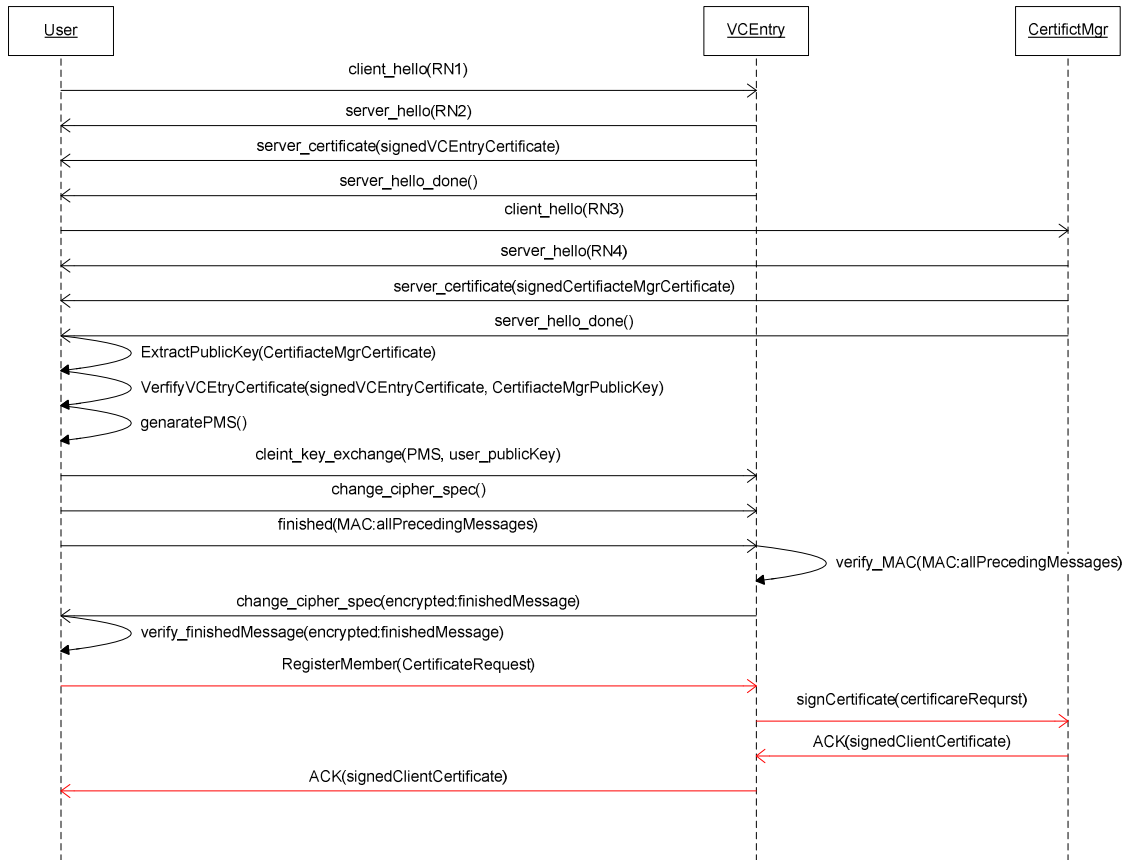


Figure 24 Membership request using *VCEntry*

Once the user has requested a secure channel the *VCEntry* submits its certificate which was signed by the VC’s certificate manager. The RNs are random numbers which assure each party that a fresh cipher will be generated later. For the user to verify the certificate he or she has to contact the certificate manager to obtain its public key. Once the user has requested a secure channel the *Certificate Manager* responds with its certificate. This certificate would have been issued by the CA for which the public key is already known to the user. The user can then use the CA’s public key to verify the certificate manager’s certificate. The user then extracts the public key from the certificate manager’s certificate to verify the *VCEntry*’s certificate. The certificate verification process proves that all the information contained in the certificate is that which is associated with the public key in the certificate. The user can then be assured that if a message is encrypted with this public key it can be only read by the person whose information is listed in the certificate. The user then generates a pre-master secret (PMS) which, along with a random number exchanged in the beginning of the handshake, will be used by both the user and the *VCEntry* to generate the Master Secret (MS). The master Secret will be used as the symmetric key during this communication. The Master secret is used because it is less computationally intensive to encrypt and decrypt messages when compared to doing so with a full asymmetric key. At this point the user can call the registerMember procedure to begin the registration process. At the end of this process, if the *VCEntry* permits, the user is granted a VC certificate.

Figure 24 shows that the *VCEntry* service has become somewhat superfluous because it can no longer hide the *Certificate Manager* from non-members. It can also no longer relieve the *Certificate Manager* of any significant load by enforcing the “Join Policy”. Therefore it is proposed that the capabilities of the *VCEntry* be provided by the certificate manager. This would mean that an Admin which wishes to open a VC would himself have to be one of the certificate managers (see Chapter 3.5 VC formation).

4.1.2.1.2 Dual authentication handshake

Below is the proposed handshake protocol in the instance when the client has already obtained a certificate from the *Certificate Manager* upon entry to the VC. The handshake protocol like the one shown in figure 24 is based on the standard SSL/TLS [24][22], the only major change was to eliminate the choice of cipher algorithm. This handshake could occur when a VC member wishes to connect to a service.

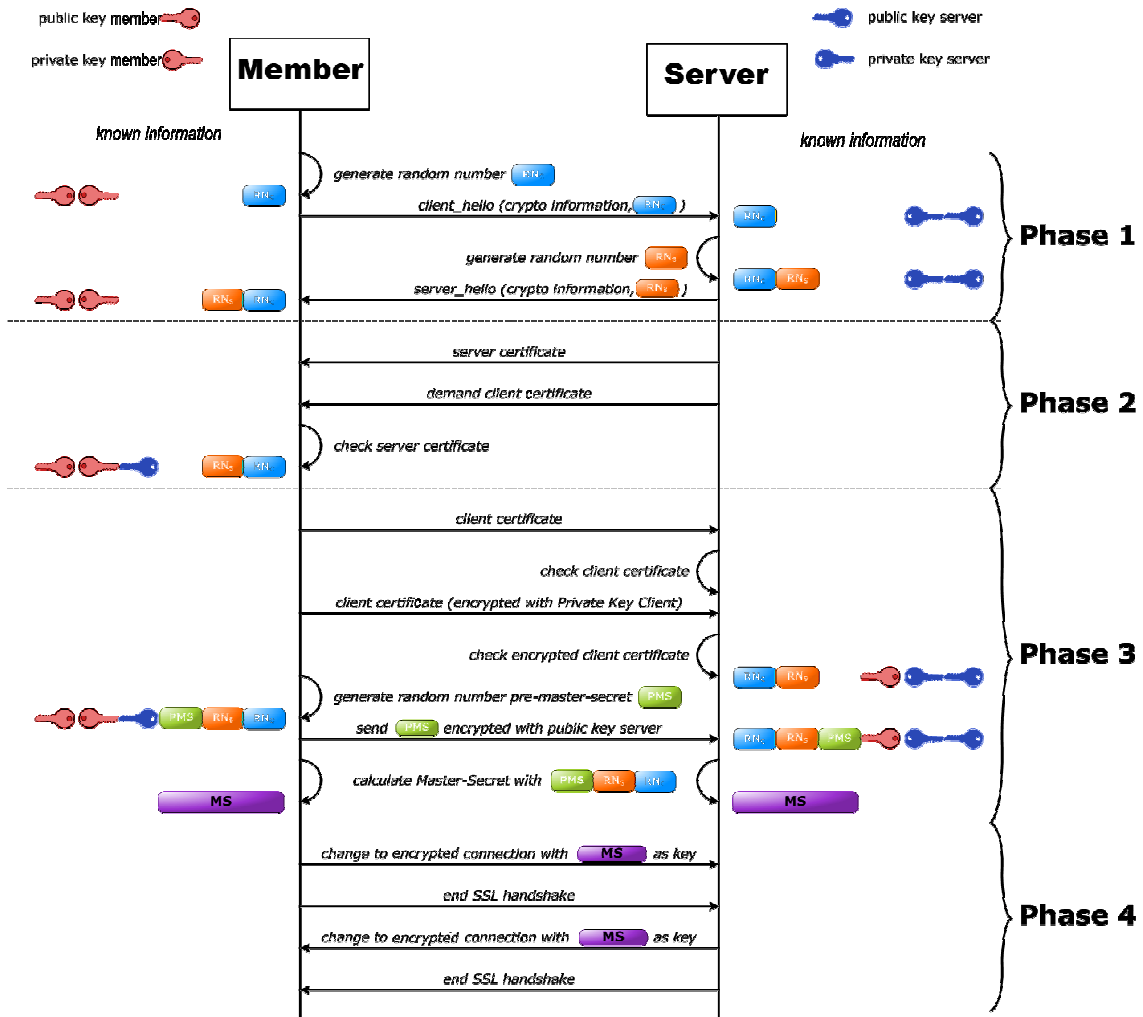


Figure 25 Two way authentication handshake

The following message description is also taken from [22].

- A Client sends a ClientHello message containing a random number which will guarantee to the client the generation of a fresh cipher later in the protocol.
- The Server responds with a ServerHello message containing its random number which will guarantee the server the generation of a fresh cipher later in the protocol. The server may also send a session id as part of the message to perform a resumed handshake.

- The Server sends its Certificate message.
- The Server requests a certificate from the client, so that the connection can be mutually authenticated, using a CertificateRequest message.
- The Server sends a ServerHelloDone message, indicating it is done with handshake negotiation.
- The Client responds with a Certificate message, which contains the client's certificate and therefore its public key.
- The Client resends the client certificate but now encrypted with the private key of the client.
- The Client then sends a ClientKeyExchange message, which contains a PreMasterSecret(PMS). This PMS is encrypted using the public key of the server which was in its certificate.
- The Client sends a CertificateVerify message, which is a *signature* over the previous handshake messages using the client's certificate's private key. This signature can be verified by using the client's certificate's public key. This lets the Server know that the Client has access to the private key of the certificate and thus owns the certificate.
- The Client and Server then use the random numbers (RNs) and PreMasterSecret (PMS) to compute a common secret, called the "master secret" (MS). All other key data is derived from this master secret (and the client- and server-generated random values), which is passed through a carefully designed "pseudorandom function".
- The Client now sends a ChangeCipherSpec record, essentially telling the Server, "Everything I tell you from now on will be encrypted."
- Finally, the Client sends an encrypted Finished message, containing a hash and a Message Authentication Code (MAC) over the previous handshake messages.
- The Server will attempt to decrypt the Client's Finished message, and verify the hash and MAC. If the decryption or verification fails, the handshake is considered to have failed and the connection should be torn down.
- Finally, the Server sends a ChangeCipherSpec and its encrypted Finished message, and the Client performs the same decryption and verification.

At this point, the "handshake" is complete and the Application protocol is enabled. Application messages exchanged between Client and Server will be encrypted using the MS as a symmetric key.

Of course neither one of these handshake protocols can guarantee that the person operating the would-be member device is the person she says she is. However at the very least the *VCEnter*, or now the *Certificate Manager*, can be authenticated to the would-be member. With this handshake protocols, and the certification path which is proposed here, the framework would have a mechanism for distributing encryption keys and creating secure connections. A mechanism for distributing encryption keys and creating secure connections was not part of VICSDA or its D3.11 implementation. With the inclusion of the proposed security modifications, the VCMobile framework can now offer access control and secure service discovery for distributed services. The choice to use SSL as the basis for the proposed protocol was simply based on availability. SSL, which is what this protocol was based on, has already been assessed by the security community and found to have no computationally feasible way to crack. SSL also provides a level of security which developers are familiar with.

4.2 Deployment scenarios

An issue which still needed to be addressed is the matter of where to host the VC components. The VCMobile implementation uses what is referred to as a hybrid deployment scenario (see figure 13 and 14). An implementation with this deployment scenario offered the most insight into the repercussions of implementing VC components in either domain. Therefore this is the deployment scenario which was chosen for the proof of concept described in the previous chapter. This section discusses the insight gained from implementing the proof of concept and the tradeoffs which have to be considered when implementing VC components in either domain. This discussion is also to the topic of the paper [21] which I have co-authored.

Mobile devices are of course resource constrained devices and therefore hosting core services on the device may have implications for the response time of the services. This can then have repercussions throughout the virtual community. The operator can provide a resource rich device to host the VC core services, although this would potentially be giving the operator access to secure communications. For the sake of evaluation the deployment scenarios shall be evaluated based on a criterion. The topic of this thesis was to investigate how a VC architecture could be used in an IMS network to provide secure service discovery and access control. For this reason the criterion considers both, the needed characteristics of VICSDA (Secure service discovery and access control), and practical concerns associated with the manner in which the VC will be used.

The deployment strategies will be evaluated on the basis of:

- Secure service discovery
- Access control
- Trust that has to be put in users and operators.
- Reliability of the system.
- Costs for end users and operators.
- Scalability of the architecture in terms of service response time.

This criterion addresses concerns that are important to the user as well as those important to the mobile operator. Both these groups are stakeholders in the wide scale acceptance of this framework.

4.2.1 Complete user deployment

Having all core VC services deployed in the user domain provides the most privacy to the user since all data is also stored in the user domain. Data such as which devices are hosting services and which users are using these services are not available to persons outside the virtual community. These characteristics of this deployment scenario would provide secure service discovery. By having the *VCEntry* hosted in the user's domain the user has full control over entry policies and blacklists. Had this service been hosted on an operators device the entry policies would have to be defined within the limitations of the operators' *VCEntry* implementation. Having full control of the implementation of the *VCEntry* provides the user full access control of the VC.

In this deployment scenario the user does not have to place any significant trust in the operator since all communications could be performed in such a way as to maintain confidentiality and integrity of the data. The operator would need the same amount of trust in the users as they currently have when they allow IP based communications over their network. One of the drawbacks of this deployment scenario is the reliability of the system.

Mobile networks are notorious for unpredictable communication state changes (i.e. bad coverage areas, and device battery depletion). This would mean that communication with core components would be somewhat more unreliable than if they were hosted on a device which had a fixed (line) connection to the network (i.e. server or home PC). It is conceivable that operators may allow their users to connect to the operator's network from their home PCs via the Internet using for instance a soft-phone application. A soft-phone is a application which runs on a PC which mimics the capabilities of a telephone. If the operators were to do this the core services could be deployed on devices with more resources and more reliable connections. Unfortunately mobile phone operators currently have a walled garden concept when it comes to services, which seems to indicate that it is unlikely to be possible in the near future. However it is possible to obtain a wireless connection to the operators network using a GPRS [13] PCMCIA card or a mobile phone as a modem. These two options have existed for some time now and are in some cases even supported directly by mobile phone operators. This may not solve the problem with the unstable connection but would provide a means to connect a device with more resources to the operator's network.

A concern of the user would undoubtedly be the cost of using such a virtual community. Assuming that providers use subscription packages similar to those in current 3G networks (i.e. charge by connection or data traffic) the cost of running a VC hosted completely in the user domain may be greater than if some or all core VC components were hosted in the operator's domain. This would primarily be due to the fact that

the cost of service to service communications where both services are in the operator's domain are typically not transferred to the user that initiated the process.

Another problem with hosting all core VC services in the user domain has to do with the scalability of the virtual community. Because of the resource constraints of a mobile device, scaling a virtual community up to 10 or 20 members may pose a problem for services such as the *Certificate Manager* which have to make computationally intensive calculation for every request.

4.2.2 Complete operator deployment

Hosting all core VC services in the operator's domain would mean that the operator could have almost full access and control if they wished. This would make it extremely difficult to guarantee secure service discovery to VC members. A user or VC member would undoubtedly see this as a negative aspect of this deployment scenario.

Since the *VCEntry* would be running on a operator's device, the operator is likely to have his own implementation of the *VCEntry* service which is configurable for any VC. This would mean that the operator could circumvent the access policies and blacklists if they wished. If the operators were to allow users to upload their own *VCEntry* they would undoubtedly require the source to verify the program would not be going outside its mandate.

Both these reasons are examples that show that this deployment scenario would require the user to have a great deal of trust in their mobile telecommunications operator. If the operator would allow the users to upload precompiled services, the roles would be reversed. The operators would then have to trust the users not to misuse the service, although this would not be much different than a simple web hosting service. The core service's connection to the network is also more reliable compared to if it were hosted on a mobile device.

As stated before in "Complete user deployment", deploying all services in the operator's domain can potentially lower the cost for the user.

Having all core VC services deployed in the operator domain also releases the resource constrained mobile devices from the burden of the computationally intensive calculations. This would allow the VC to easily be able to scaled up to more then 20 users.

4.2.3 Hybrid deployment

A hybrid deployment scenario would entail having some core VC components deployed in the user domain and others in the operator's domain. The ideology behind this is that components for which it is beneficial to deploy them in the operators domain can be deployed on the application server. Beneficial, meaning when compared to a deployment in the user domain. Components for which it is beneficial to deploy them in the user domain can be deployed on a user's device. Beneficial is meant to express that from the perspective of the relevant stakeholder the choice would provide less risk and/or be monetarily beneficial. This is somewhat of a best of both worlds approach; however it falls short of an optimum solution. For example once the *Certificate Manager* is deployed in the operator's domain the operator can potentially gain access to the VC. However if the *Certificate Manager* were to be deployed on the user domain it would likely have to be running on a mobile device with limited resources. A *Certificate Manager* running on a resource constrained device would have a very sluggish response time considering it only performs computationally intensive operations. The choice of where to host the *Certificate Manager* would then be a tradeoff between security and performance with little to no options in between the two extremes. This means that the question of where would be best to host particular VC core service does not have a clear answer. A deployment scenario would have to be chosen based on the specific requirements of the application build on top of the VC framework. If the application demanded performance and is not concerned with the possibility of the provider gaining access to the VC, a full operator deployment scenario may be best suited. In all other cases a full user domain deployment would be the best choice. This is

because deploying even one of the core VC components in the operator's domain would potentially give the operator access to the VC. Using a non-core VC service which is deployed in the operator's domain would only be done if the application was not concerned with the possibility of the provider gaining access to the VC. A hybrid deployment scenario would then only be usefully if the application is not concerned with the possibility of the provider gaining access to the VC and it wishes to provide the user with a form of flexibility which is only possible if certain services were deployed on the user's device.

5 Conclusion

The VCMobile framework described in this thesis gives mobile operators and mobile application developers the means to quickly implement services or applications that can create virtual communities. In this thesis I explain how the virtual community framework could provide access control and secure service discovery in an IMS network.

The components which were re-used from the VICSDA D3.11 example implementation from [1] were not designed to function on a mobile telecommunications network. The alterations which are described in sections 3.1 through 3.6 needed to be made to allow the components to be more loosely coupled.

To allow the framework to function properly on the IMS network it needed to be adapted to make use of SIP which is the control protocol of IMS. SIP allowed the components to set up sessions to exchange data. Because very little data needed to be exchanged this data was also exchanged using SIP, essentially making it a transport protocol.

The changes made to the D3.11 implementation in [1] left the framework open to attacks which breach the access control and secure service discovery protocols. For this reason a new security mechanism was proposed in section 3.7.1 which allowed for public key distribution to create encrypted communication channels. The proposed mechanism would also allow for a virtual community to function properly with multiple certificate managers and some services to be authenticated.

The question of where to host the VC components was addressed in section 4.2 which showed that deploying any VC service in the operator's domain could potentially give the operator access to the VC. However deploying a service in the operator's domain would provide it with a host which has relatively more resources which can increase the response time of the service.

Like most academic software this framework, in its current state, likely does not have many applications outside of the academic world. However I do believe that once the security modifications have been applied and the framework has been conglomerated into a package it will have great potential as a development platform for IMS services.

6 Further work

I believe that all security holes in the current framework can be fixed by implementing the security mechanism proposed in section 4.1.2.1. To determine if this is truly the case the protocol would need to be validated by a security expert.

The *Repository* service has a lot of potential for expansion, particularly in the area of searching for a service that fits a generalized description. I have discussed, in theory, how this could be done is explained in section 3.3 but more research is needed to have a working concept.

The frame work is in need of proper WSDL schemas and namespaces. I have come to realize that this is no simple task, particularly if you strive for completeness. Before the framework could be deemed complete these elements must be defined.

Because mobile devices are typically resource constrained devices, management of device resources has great potential to improve the response time of the framework. Potentially a holistic scheduling ideology could be applied to the framework to increase performance. This could include acceptance testing of service request before the service is started along with performance evaluation of access point to decide which service to use.

Certain methods of SIP could be used by the framework for community management. Specifically the SIP redirect response could be used to redirect service requests to other services if the host device is unable to handle the request.

It would be possible to implement a VICSDA framework that can work on an IMS network which uses IP as the transport protocol by using only SIP to initiate the session. This concept was not explored in this thesis but remains a plausible approach.

7 Glossary

3GPP	The 3rd Generation Partnership Project (3GPP) is a collaboration agreement that brings together a number of telecommunications standards bodies which are known as "Organizational Partners". The original scope of 3GPP was to produce globally applicable Technical Specifications and Technical Reports for a 3rd Generation Mobile System based on evolved GSM core networks and the radio access technologies that they support
API	Application Programming Interface is a collection of classes, functions or procedures provided by an operating system, programming liberty or service to handle requests made by an application.
Access Point	The network address were a service or user can be reached.
CDMA	Code Division Multiple Access is used in both used in both 2G and 3G mobile telecommunications networks. It is a "spread spectrum" technology, allowing many users to occupy the same time and frequency allocations in a given band/space.
End-to end testing	In the context of this thesis, end-to-end testing is testing of an application on both the client an server side.
GPRS	General Packet Radio Service is a packet oriented Mobile Data Service used in Global System for Mobile Communications (GSM). It is used in Wireless Application Protocol(WAP), Short Message Service(SMS), Multimedia Messaging Service(MMS) and internet communications in mobile telecommunications networks.
GSTN	General Switched Telephone Network is the same as PSTN.
GUI	A Graphical User Interface an application or section of source code which provide a graphically rich interface for a user to interact with underlying software.
H.248	Media Gateway Control Protocol (RFC 3525) is a protocol used between elements of a physically decomposed multimedia gateway, i.e., a Media Gateway and a Media Gateway Controller.
IETF	The Internet Engineering Task Force (IETF) is a large open international community of network designers, operators, vendors, and researcher s concerned with the evolution of the Internet architecture and the smooth operation of the Internet.
IMS	Internet Multimedia Sub-system (IMS) in a framework designed to provide Internet protocol multimedia capabilities in telecommunications networks.
J2EE	Java 2 Platform, Enterprise Edition is a platform used for server programming in the Java programming language.
Join policy	A join policy is a collection of rules that define the circumstances under which a user may join a group
JSR	Java Specification Requests are the actual descriptions of proposed and final specifications for the Java platform.
LAN	Local Aria network is a computer network which covers a small geographical area.
NETANN	NETANN is a convention for addressing media resources using SIP , RFC 4240

OWL	OWL Web Ontology Language is compatible with XML and RDF and is designed for use by applications that need to process the content of information instead of just presenting information to humans.
P2P	Peer-to-Peer computer network is a type of network where the members connect directly with one another rather than through a centralized server.
PSTN	The Public Switched Telephone Network is the world wide circuit-switching network.
PLMN	Public land mobile network is an established telecommunications network operated by an authority of or a recognized group.
MAC	Message Authentication Code (MAC) is used to verify a message's integrity. A MAC is similar to a HASH function but has slightly different requirements. A function is a MAC if it protects against external forgery of specific plain text messages. It is somewhat of a confidential hash function.
MSML	Media Server Markup Language (MSML) is used to control and invoke many different types of services on IP Media Servers. MSML can be used, for example, to control Media Server conferencing features such as video layout and audio mixing, create sidebar conferences or personal mixes, and set the properties of media streams.
SAN	The System architecture and networks group studies parallel and distributed systems. The emphasis is on the architecture of networked embedded systems, including hardware and software aspects.
SDS	Service Development Studio is the only fully comprehensive tool for development and end-to-end testing of both the client and server side of new convergent all-IP (IMS) applications developed by Ericsson.
SOAP	SOAP is a XML based protocol used to exchange messages over a network. SOAP is typically used in combination with HTTP.
SIP	Session initiation protocol is an application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants.
SSL	Secure Socket Layer is a mechanism used to create secure communication channels over an unsecure network.
RTP	Real-time Transport Protocol is a standard for delivering audio and video over the Internet designed by the IETF.
XML	Extensible Markup Language is a simple, very flexible text format derived from SGML (ISO 8879). XML is playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.
VC	Virtual community is a group of people who primarily interact using electronic communications rather than face to face.
VICSDA	VICSDA is a framework which provides secure service discovery and access by using Virtual Communities developed by the SAN group.
VoiceXML	An XML based language used to design voice interfaces for applications.
Walled Garden	The walled garden concept refers to a closed or exclusive set of services provided to users as opposed to allowing the user full access to internet content or e-commerce.

WSDL	Web Services Description Language is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.
UIQ	UIQ is a software platform based on Symbian OS typically use in mobile phones.
URI	Uniform Resource Identifier is a character string used to identify a resource on the internet(i.e. sip:user@domina.com).

8 References

- [1] S. Chen, et al., “VICSDA: Using Virtual Communities to Secure Service Discovery and Access”, Department of Mathematics and Computer Science Eindhoven University of Technology, the Netherlands
- [2] S. Chen, J. J. Lukkien, R. Verhoeven, R. Bosman, et al., “Prototype of secure service discovery and access in virtual communities, Contribution Deliverable 3.11”, Department of Mathematics and Computer Science Eindhoven University of Technology, the Netherlands
- [3] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, et al. “SIP: Session Initiation Protocol”, RFC 3261, IETF Network Working Group, <http://www.ietf.org/rfc/rfc3261.txt>, June 2002.
- [4] 3rd Generation Partnership, “Digital Cellular Telecommunications System (Phase 2+), Universal Mobile Telecommunications System (UMTS), IP Multimedia Subsystem (IMS), Stage~2”, V7.6.0, TS~23.228, 3GPP, December 2006.
- [5] ABI research, “IMS Core Networks: A Dynamic Service-Based Architecture”, 2008 available at: http://www.abiresearch.com/products/market_research/IMS
- [6] P. Kessler, S. Svenberg, “JSR 281 IMS Services API, Final Release Version 1.0” Java Community Process. 2008, available at: <http://www.jcp.org/en/jsr/detail?id=281>
- [7] M. Johansson, N. Palm, “JSR 325 IMS Communication Enablers (ICE)” Java Community Process. 2008, available at: <http://www.jcp.org/en/jsr/detail?id=325>
- [8] E. Cerami, “Web Services Essentials. Sebastopol”, CA: O’Reilly, 2002.
- [9] R. Travis. “Signaling System #7, 4th Edition, New York: McGraw-Hill”, 2002 ISBN 978-0071387729
- [10] S. Donovan, “Request for Comments: 2976 :The SIP INFO Method”, Network Working Group, 2000
- [11] B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema, et al, “Request for Comments: 3428 Session Initiation Protocol (SIP) Extension for Instant Messaging”, Network Working Group, 2002
- [12] A. B. Roach, “Request for Comments: 3265 Session Initiation Protocol (SIP)-Specific Event Notification”, Network Working Group, 2002
- [13] 3rd Generation Partnership Project (3GPP) group, <http://www.3gpp.org>, August 2007
- [14] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, et al. ” Request for Comments: 5280, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile”, Network Working Group, May 2008
- [15] D. Booth, Canyang Kevin Liu “Web Services Description Language (WSDL) Version 2.0”, Part 0-2, W3C Recommendation 26 June 2007
- [17] S. Bechhofer, et al. OWL Web ontology language reference. W3C Recommendation. Available at <http://www.w3.org/TR/owl-ref/> (2004)
- [18] D. C. Fallside, P. Walmsley “XML Schema Part 0: Primer Second Edition. W3C Recommendation”. available at <http://www.w3.org/TR/xmlschema-0/> (2004)

- [19] H. Schulzrinne, "Request for Comments: 3551 RTP Profile for Audio and Video Conferences with Minimal Control", Network Working Group, 2003
- [20] S. Bruce ., "Attack Trees". Dr Dobb's Journal, v.24, n.12. December 1999, Retrieved on 2007-08-16.
- [21] I. Radovanović, J. Lukkien, S. Chen, C. Molanus, "Virtual community management for enabling P2P services in the IMS network", The second conference on Internet Multimedia Systems Architecture and Applications 2008
- [22] Wikipedia contributors, "Transport Layer Security," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title=Transport_Layer_Security&oldid=252670958 (accessed November 27, 2008).
- [23] M. Klusch, B. Fries, and K. Sycara "Automated semantic web service discovery with owls-mx." In Proceedings of 5th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS) (2006), ACM Press
- [24] J. Rosenberg, D. Remy, "Securing Web Services with WS-Security : Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption." Sams, 2004.
- [25] T. Dierks, E. Rescorla, "Request for Comments: 5246, The Transport Layer Security (TLS) Protocol Version 1.2" , Network Working Group, 2008

Appendix

8.1 SOAP Messages

8.1.1 Server result response

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=http://www.w3.org/2001/12/soap-envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <result>
      <message xsi:type="xsd:string">
        ...
      </message>
    </result>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

8.1.2 Server Message

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=http://www.w3.org/2001/12/soap-envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

8.2 Sequence diagram of VC formation

