

Implementing an Authorization model in a SIP User Agent to  
secure SIP sessions

by

Mudassir Fajandar

B.E., Bombay University,  
2000

A thesis submitted to the  
Faculty of the Graduate School of the  
University of Colorado in partial fulfillment  
of the requirements for the degree of  
Masters in Telecommunications  
Interdisciplinary Telecommunications Program  
2003



Scriptor, Mudassir Fajandar (M.S. Telecommunications)

Implementing an Authorization model in a SIP User Agent to secure SIP sessions

Thesis directed by Prof. Douglas Sicker

Authorization is an essential component that can increase the security of SIP sessions. There has been prior work in security in SIP that has integrated and defined mechanisms for authentication, integrity and confidentiality. Other work defines mechanisms for exchange of authentication details. There is however no work done in providing authorization in the SIP messages. The key question to answer is whether modifications to the Session Initial Protocol (SIP) User Agent (UA) to carry additional authorization parameters in SIP messages can be implemented. This work demonstrates an implementation with newly defined SIP behavior and a mechanism for a SIP UA to process messages containing information as assertions. This is achieved by incorporating an authorization model in SIP sessions. The changes to the UA were made to the open source Linphone application already developed by Simon Morlat. The new functions of the SIP UA are

- Processing the new 428 response messages sent by the proxy server
- Copying the headers fields from the response to a new request
- Attaching the information sent in the four new headers in the response message to a new request
- Sending the new request to the same proxy server

This thesis shows that the changes to the SIP messages and SIP UA can be implemented on a test bed. We also show minimal effect on the UA because of the new headers to be processed.

# **DEDICATION**

To my mother

## ACKNOWLEDGEMENTS

This thesis would have been even tougher to complete if my life had not intersected with any one of those that the next paragraph mentions. Firstly, I would thank my mom, dad and sis for all their encouragement and support that made cloudy days seem like full of sunshine and for understanding problems which they could not remotely relate to but would show immense interest in, e.g. segmentation errors in C programming. Then of course I would express deep gratitude to Prof. Douglas Sicker, who initially took me under his wings thinking I would be a good baby-sitter. Special thanks to Prof. Ray Nettleton for proof reading the thesis draft not once but twice, in the process enduring the most tortuous reading experience of his life. Also, Prof. Tom Lookabaugh for agreeing on a very short notice to participate on the committee.

Amongst friends, I would like to thank Shichao for all the help when stuck in coding. Shankar and Jayant for bearing with a grumpy 20-hours-a-day working roommate. Indu for the best home-cooked food in Boulder that was always bereft of salt; research mates Ameet and Anand, for pretending to understand my garbled contributions at the early morning group-conferences. I would also like to thank Shweta for her hardware assistance at the last moment and last but certainly not the least, Pita Pit, the place that feeds the hungry master's thesis student at 3 in the morning.

## CONTENTS

CHAPTER	
1	INTRODUCTION 1
1.1	Statement of Thesis..... 1
1.2	Overview..... 1
1.3	Motivation for the research..... 2
1.4	Signaling Protocols – a brief introduction..... 3
1.5	History of Signaling Protocols..... 3
1.6	SIP – the protocol for the future..... 4
1.7	Applications of the research..... 5
1.7.1	Research and Educational Institutes..... 5
1.7.2	Commercial use..... 5
1.7.3	Information Sharing..... 6
1.8	Research Overview..... 6
2	LITERATURE REVIEW 8
2.1	Session Initiation Protocol..... 8
2.1.1	SIP Network Elements..... 9
2.1.1.1	SIP User Agent..... 9
2.1.1.2	SIP Proxy Server..... 10
2.1.1.3	Registrar..... 11
2.1.1.4	Gateways..... 12
2.1.1.5	Interaction between SIP elements..... 12
2.2	SIP sessions..... 13
2.2.1	Resource Discovery..... 13

2.2.2 Session Initiation.....	14
2.2.3 Session Establishment.....	14
2.2.4 Session Termination.....	15
2.3 SIP messages.....	16
2.3.1 SIP requests.....	17
2.3.1.1 SIP method.....	17
2.3.1.2 Request-URI field.....	18
2.3.2 SIP responses.....	18
2.3.2.1 Response codes.....	19
2.3.3 SIP headers.....	19
2.4 Example of a SIP request message and its response.....	20
2.5 Directory Services.....	21
2.6 Federation.....	22
2.6.1 Trust model vs. Non-trust model.....	22
2.6.2 Example of a Federated model.....	23
2.6.3 A Non-trust model.....	24
2.6.4 A Trust model.....	25
2.7 Security Assertion Markup Language (SAML).....	25
2.8 Authorization.....	26
<b>3 LACK OF AUTHORIZATION FEATURES IN A SIP SESSION</b>	
3.1 Why is there need for additional security in a SIP session?.....	27
3.2 Call flows.....	28
3.2.1 Call flow diagram for a simple unsecured SIP call.....	29
3.3 Headers and Response codes in SIP catering to security.....	30
3.3.1 Responses catering to security.....	31

3.3.1.1	401 and 407 response codes.....	31
3.3.2	Proxy Authorization header.....	33
3.4	User Agent client behavior.....	34
3.5	Lack of authorization details in the proxy-authorization header...	35
3.6	Currently proposed or implemented mechanisms for securing SIP...	35
3.6.1	The Security Mechanism Agreement for SIP protocol.....	36
3.6.2	Content Indirection.....	37
3.7	Choice of mechanism for the solution.....	38
4	APPROACH TO IMPLEMENTATION	
4.1	The Solution – the authorization model in SIP sessions.....	40
4.1.1	Models for retrieving the authorization information.....	41
4.1.1.1	Origin proxy initiated authorization model.....	42
4.1.1.2	Destination proxy initiated authorization model.....	44
4.2	The new headers in SIP requests.....	45
4.3	New response code.....	46
4.4	Header field names and Response codes definitions.....	46
4.5	Syntax for the additions.....	47
4.5.1	Additional headers syntax.....	47
4.5.2	Response header syntax.....	48
4.6	Summary of header use.....	48
5	TEST AND RESULTS	
5.1	Experimental setup.....	50
5.2	Successful generation of a response.....	51
5.3	UA client processing.....	54
5.4	Processing time of the UA for the new headers.....	58

6	CONCLUSION	
6.1	Summary of work.....	59
6.2	Future Work.....	60
	BIBLIOGRAPHY	62
	APPENDIX	
A.	Cookbook for Linphone	63
A.1	Implementation steps for linphone.....	63
B.	Methods and Materials	65
B.1	Choice of the Operating System.....	65
B.2	Choice of the Software Libraries.....	65
B.2.1	oSIP - Open SIP.....	66
B.3	Architecture of oSIP.....	67
B.3.1	Finite state machines.....	67
B.3.2	The Parser.....	68
B.3.3	The Transaction Manager.....	68
B.4	Call flow using the oSIP components.....	69
B.5	Advantages of oSIP.....	70
B.6	Disadvantages of oSIP.....	71
B.7	Linphone.....	71
B.7.1	Requirements for Linphone.....	72
B.7.2	Modes for Linphone.....	72

**TABLES**

Table		
	2.1 SIP Request headers.....	20
	2.2 SIP Response headers.....	21
	4.1 Summary of header use.....	48
	5.1 Calculating average delay in processing information.....	58

## FIGURES

### Figure

2.1 UAC and UAS in a single network element.....	10
2.2 The Registration process.....	11
2.3 Interaction between SIP network elements.....	12
2.4 A SIP call flow.....	15
2.5 A SIP INVITE request.....	16
2.6 A 401 SIP response.....	18
2.7 A non-trust model.....	24
2.8 A trust model.....	25
3.1 An unsecured SIP flow.....	29
3.2 WWW-Authenticate header.....	32
3.3 A Proxy-Authorization header.....	34
3.4 Message flow for the agreement.....	36
4.1 Origin proxy initiated authorization model.....	42
4.2 Destination proxy initiated authorization model.....	44
5.1 The initial request and response captures.....	53
5.2 Captures of the response, ACK and second INVITE.....	57

# Chapter 1

## INTRODUCTION

### 1.1 Statement of Thesis

This research question that the thesis tries to answer is, “Can modifications to the Session Initial Protocol (SIP) User Agent (UA) to carry additional authorization parameters in SIP messages be implemented?”

### 1.2 Overview

SIP is a signaling protocol that performs the preliminary functions necessary to make communication between the two end-points smooth and easy. A SIP session is established between two end-points that seek to communicate with each other using any of the media- text, audio or video. SIP does the end-point discovery, establishes the session after the endpoints have exchanged the parameters required for communication, then maintains the session during the data communication between the end points and finally terminates the session.

There are several headers defined in the SIP protocol that carry security information, which are present in SIP requests and responses. Examples of security-specific headers are the Proxy-Authenticate, Proxy-Authorization, and WWW-Authenticate and Authorization headers <sup>[1]</sup>. Each of these headers has several parameters, which contain the value(s) needed to establish a secure session. However, none of these headers consider using authorization as a mechanism to achieve better security. The authorization and the proxy-authorization headers in SIP are misnomers since they carry authentication credentials and negligible information in the form of authorization information.

### 1.3 Motivation for the research

Apart from the built-in security headers in the protocol, there are security mechanisms devised in SIP to provide authentication, integrity and confidentiality as a mean of securing the SIP session. Though these mechanisms secure a SIP session from several attacks from within and outside a network, they leave the recipient of a SIP request with no means of differentiating users trying to access resources. This research is unique as it aims to make authorization an important mechanism for SIP network elements to implement access control in a more flexible manner than is now possible. In order to demonstrate the concept, changes have been made to the SIP messages and the behavior of a SIP UA, which are then implemented and tested.

The long-term goal of the research is to provide end-to-end security across domains by implementing an authorization model based on Security Assertion Markup Language assertions [9], which takes care of the policy aspects of SIP sessions, like privacy and confidentiality. Three members of the Internet Applications Laboratory (IAL) [36] group are working upon the various components of the authorization model. Ameet Kulkarni is working on the assertion retrieval mechanism at the proxy server. Anand Chavali is working on the authorization server and myself, Mudassir Fajandar, on the SIP User Agent client.

## **1.4 Signaling Protocols – a brief introduction**

Signaling protocols, as the term suggests, are the protocols used for the signaling between the end points that want to establish communication and pass information between each other. Their presence dates back to the days when Common Channel Signaling System #7 [2](also known as SS7) was introduced for signaling in Public Switched Telephone Networks (PSTN). With the advent of the Internet, signaling protocols used in legacy systems were supplemented with signaling protocols like SIP because of their ability to cater to the need for convergence of voice, video and text on the IP networks.

Signaling protocols exist at the application layer in the TCP/IP protocol model and the session layer in the OSI model. Signaling does not run between normal network elements like routers and switches, since it would require routers to look at the session layer before forwarding a packet. However, signaling protocols like SIP use IP addresses to route their messages through to the destination. However there are devices employed specifically for performing the session layer functionalities. These are called the SIP network elements and the network constituting these elements is called the signaling network.

## **1.5 History of Signaling Protocols**

There has been a dramatic increase in telecommunications spawning a similar rise in the line capacity for telecommunication networks. Signaling for circuit-switched networks was in-band until the digitalization of the signaling. In-band signaling involved

sending signaling information over the voice connection, hopping from switch to switch. This was disadvantageous since the entire connection was being utilized even in call failure situations and the voice connection took up the entire bandwidth. The introduction of digital connections ushered out-of-band signaling into networks.

The inability of switches to take the processing load and of the trunks to carry the usage load led to the development of signaling standards like the Common Channel Interoffice Signaling Systems #6 (CCIOS6). Digital networks brought in a lot more intelligence into the circuit-switched telecommunications world and separated the call management and signaling portions of a call from the voice channel. This was followed by further development of standards for signaling purposes like SS7, which was the precursor to the Intelligent Networks which are used worldwide now for billing, voice interaction and other applications.

With the pervasiveness of the IP world, SS7 reached out from being a pure voice network signaling protocol, to the data world. It achieved this by using the IETF Sigtran [35] protocols, an IETF standard. Protocols like Stream Control Transmission Protocol (SCTP) [3], developed within the standard ensured the call reliability and supported SS7 protocols to communicate over the IP networks.

## **1.6 SIP – the protocol for the future**

H.323 [5] has been the de-facto standard in signaling over IP networks until the recent introduction of Session Initiation Protocol (SIP). A shift towards SIP has been noted, partly due to the fact that SIP is a light and web-friendly protocol that is much easier to integrate due to simpler binding extension requirements with Internet related

protocols. It is heavily derived from Hyper Text Transport Protocol (HTTP) [6] and Simple Mail Transfer Protocol (SMTP) [7], which explain its web-friendliness. SIP provided ample scope to contribute to the work being done in the sphere of security, the availability of free SIP stacks for development and the temptation of being the protocol for the future.

## **1.7 Applications of the research**

There is a far-reaching impact of this research in the educational and commercial fields. Using authorization as a mechanism to implement access control without storing the attributes of users requesting resources within one's domain, will solve several privacy and confidentiality issues.

### **1.7.1 Research and Educational Institutes**

Implementing an authorization model, along with federation [8] between educational domains, would dramatically improve the ability of a domain to perform flexible access control. An ingress point to a SIP domain belonging to an educational institute would be able to allow or block users based on their college, department, position within the department, alliance to a research group as well as upon time determinants like the permissible time of access to particular resources.

### **1.7.2 Commercial use**

In the instant messaging as well as the videoconferencing world, there is a pressing need for a more stringent authorization mechanism to be in place in order to avoid spamming of user accounts. Though a more common occurrence in e-mails, there

is possibility of spam arriving in instant messaging and videoconferencing sessions as well. Through the more granular approach that the authorization model provides at policy decision points and policy enforcement points, spamming can be better controlled.

### **1.7.3 Information Sharing**

Another issue that we seek to resolve is to alleviate the problems that a network administrator faces regarding sharing of confidential information. Administrators prefer that information about the users remain within the domain. Using the authorization model with federation will ensure that only trusted members of the federation are able to obtain information about their users.

## **1.8 Research overview**

This research endeavor is an attempt to address lack of authorization capability in SIP by providing a solution for SIP sessions based on an authorization model. In this thesis, no attempt is made to address the quality of service or picture quality issues in videoconferencing or the latency issues in voice conferencing. Instead it focuses on the problem space, the solution, the implementation and tests conducted in order to integrate an authorization model into SIP. The solution consists of network diagrams and call-flows, the proposition for the addition of a new header in SIP responses and the addition of headers in a SIP request. The implementation deals with making changes to the protocol behavior in order to demonstrate that the new headers can be integrated into the protocol. The implementation also includes the modifications to the UA client software.

Chapter wise, Chapter 2 provides a more in-depth background of SIP along with a comparison of the trust and non-trust models in federations and a brief introduction to directory services and SAML. Chapter 3 discusses the problem space by asking for the

need for an additional security mechanism in SIP. The chapter touches upon the security headers in SIP and some newly introduced security mechanisms and concludes that portion with the shortcomings of the existing mechanisms. The chapter concludes with the choice of mechanism for the authorization model. Chapter 4 introduces the authorization model and the call flows that adapt the model to a SIP session. The new SIP message response and the new headers to be attached to the request are introduced. A few sections deal with the definition and syntax of the headers and the manner in which the proxy servers must handle them. Chapter 5 contains the tests conducted and results obtained, which demonstrate the fact that the modifications to SIP requests and responses as well as to the SIP UA can be implemented. We conclude with Chapter 6 where we summarize the research conducted and the future work to be conducted. There are 2 appendices - Appendix A that is a cookbook for implementing the SIP client we use and Appendix B, which is based upon the architecture and functioning of the software libraries.

## Chapter 2

### LITERATURE REVIEW

This chapter introduces the concepts of SIP as a protocol, SIP network elements, the role of the network elements, SIP messages and simple call flows, authorization servers, federation and SAML.

#### 2.1 Session Initiation Protocol

The Session Initiation Protocol (SIP) is the signaling protocol derived from HTTP and SMTP. One of the primary attractions of SIP is that it is a lightweight protocol with very low Round Trip Time (RTT) compared to backwards-compatible H.323 versions that have call setup time of 7 RTT as compared to 1.5 RTT for a SIP session [24]. SIP also draws its strength from the fact that other than the routing, most of the intelligence lies at the SIP end points. This concept is antithetical to the traditional legacy networks where the maximum intelligence is present at the central nodes. This gives the protocol greater flexibility, scalability and opens the network to implementation of new services [20]. SIP does not perform the communication of the media data, instead using protocols like the Session Description Protocol (SDP) [21] and Real Time Transport Protocol (RTP) [18] for that purpose.

SIP is a developing protocol that provides tremendous scope for research. The functionality of SIP is similar to that of any other basic signaling protocol and can be listed as below:

- Locating end points
- Initiating a session between the end points.

- Maintaining the session, which includes making any changes in the parameters of the end points if need be during the session.
- Terminating the session between those end points.

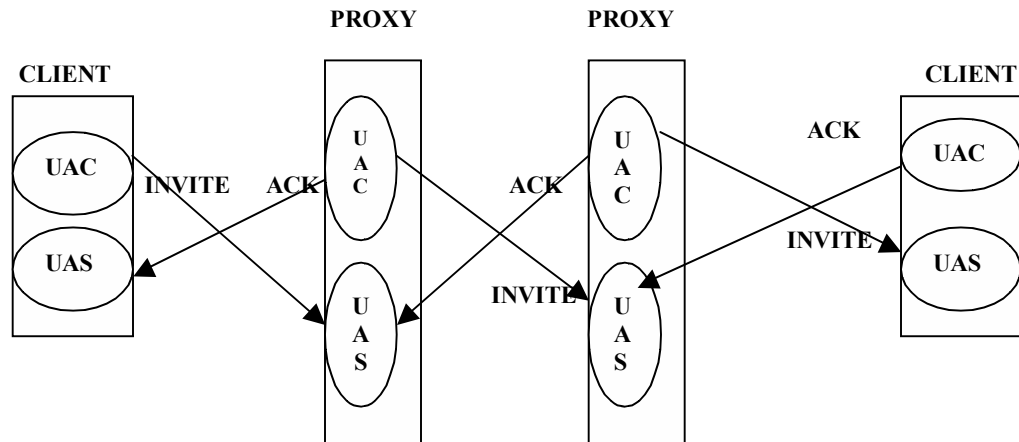
A SIP session is established using messages between the end points. Methods are used to define the different messages for each stage of the session. Examples of methods are INVITE, BYE and ACK. The messages are categorized into requests if they are initiated by a client or as responses if they are responding as a server. The next section describes the network elements, which constitute the SIP network.

### **2.1.1 SIP Network Elements**

A SIP network is primarily composed of SIP client end points called clients, proxy servers, registrars and signaling gateways for interoperating with other signaling protocols.

#### **2.1.1.1 SIP User Agent**

SIP User Agents are normally applications running on a device. They can assume the mantle of either a User Agent Client (UAC) or User Agent Server (UAS) depending on their function during a SIP session. If a User Agent generates a request, which calls for a particular action to be taken by the server to which it is directed and receives the response to the request it is called the UAC. Similarly, the recipient of the request is a UAS. Hence, a single physical entity in a SIP network can assume the properties of both, a UAC as well as a UAS. For example, a forwarding proxy acts as a UAS when it receives a request from the initiating UAC and as a UAC itself when it forwards the SIP requests towards the destination proxies.



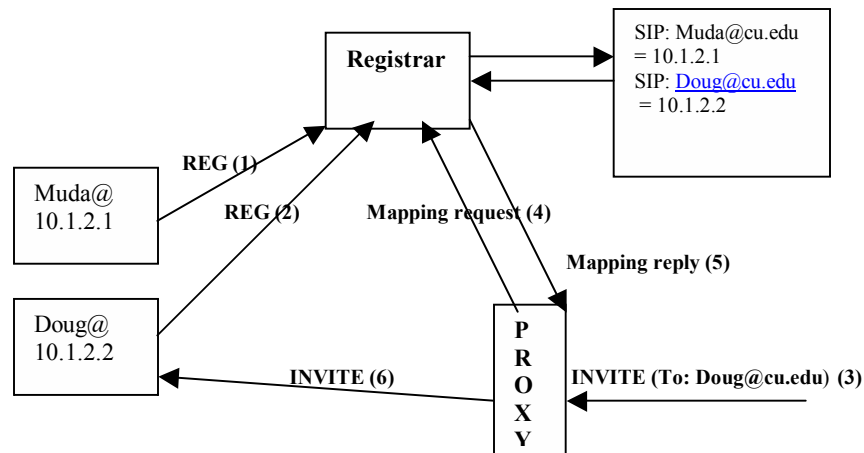
**Figure 2.1: UAC and UAS in a single network element**

### 2.1.1.2 SIP Proxy Server

SIP proxy servers are network elements representative of a particular domain, which engage in the routing of requests between different domains. They are generally considered as the ingress and egress points for a domain when INVITE messages are received or sent respectively. INVITE is the method in SIP representative of a request. The request is forwarded by the proxy server using the IP address to SIP Uniform Resource Identifier (URI) tables that it stores, and which are constantly updated and exchanged between proxy servers across domains. The storage of the forwarding table and the mapping is what differentiates a stateful proxy from a stateless. Stateful proxies cache a session for a short period of time while stateless proxies do not preserve any record of any sort. Hence stateless proxies are unable to perform functions, which a stateful proxy is capable of, like avoiding retransmission of requests, forking of requests and Network Address Translation features.

### 2.1.1.3 Registrar

The Registrar is the network element that serves as the Reverse Service Domain Name Server of the domain. All new incoming requests from outside domains are directed to the Registrar, which maps the SIP URI in the “To” field of the SIP header to the IP address of the device which assumes that SIP identity on the local domain. For successful communication, it is necessary that by default all SIP clients be configured to send a REGISTER [1] request to the Registrar in order for their SIP to IP mapping to be stored. This request might be done at regular intervals, since the Registrar refreshes its mapping tables. The registrar provides limited password security at the initiation of the session.



**Fig 2.2: The Registration process**

The above diagram illustrates the initial population of the registrar tables by the UAs, [Doug@10.1.2.2](mailto:Doug@10.1.2.2) and [Muda@10.1.2.1](mailto:Muda@10.1.2.1). When the proxy server intercepts an incoming INVITE, the SIP URI is mapped to the IP address via a query sent to the tables maintained by the registrar. The proxy then forwards the request to the SIP UA having the IP address.

### 2.1.1.4 Gateways

Gateways stand at the boundary of the signaling protocol networks and are necessary for interoperability across different domains. They act as an interface between different signaling networks using mapping protocols. An example of a protocol that runs between a SIP and a PSTN network gateway is Media Access Gateway Controller (MEGACO) [19].

### 2.1.1.5 Interaction between SIP network elements

The diagram below illustrates a session flow between two domains that have not communicated before.

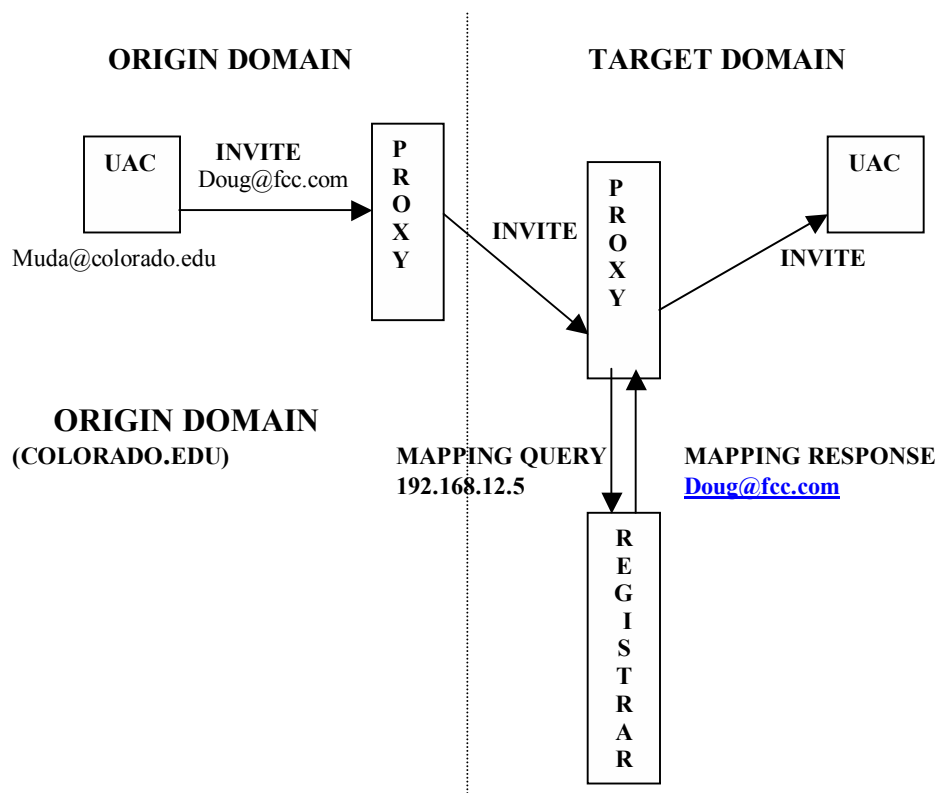


Figure 2.3: Interaction between SIP network elements

Figure 2.3 shows the UAC at the origin sending an INVITE to the origin proxy server and then to the destination proxy, which maps the destination SIP URI to an IP address using a mapping query. The destination proxy server then forwards the INVITE to the final destination device hosting the UAC.

## **2.2 SIP sessions**

There are two kinds of sessions that SIP supports, client-server and peer-to-peer sessions. Before initiating a session the availability of the end points is ascertained and capabilities are exchanged. In a peer-to-peer session, the destination IP address of the destination if known can be entered in the “To” field. This eliminates the process of locating the Registrar server that reads the SIP URI to IP address binding stored in the location server. Hence, in a peer-to-peer session, an INVITE can be sent directly to the destination end point. We now discuss at length how a normal client-server session takes place in SIP.

### **2.2.1 Resource Discovery**

In a client-server model the UAC initiating a session sends an INVITE message with the destination end point’s SIP URI in the “To” address field. The origin proxy server sends out forked INVITE messages in order to locate the route to the destination proxy server. Once the route to the destination end point is obtained, further routing within the destination domain is achieved by querying the registrar at the destination domain that holds the bindings. The registrar returns the IP address within its domain that corresponds to the SIP URI and the SIP INVITE is completed.

### **2.2.2 Session Initiation**

The first line of the message indicates the method being used, in this case the INVITE and the following fields carry information about the session. Examples of header fields are the “From” field carrying the SIP URI of the originator of the call, the “To” fields carrying the destination URI and the “Via” field carrying the path chosen. When the origin proxy server receives the INVITE, the domain of the destination URI in the “To” field is looked up in a forwarding table and if successful the SIP request is directly forwarded. On receiving the INVITE, the destination proxy server sends back responses, which indicate the state of the session.

### **2.2.3 Session Establishment**

The session is established between the end points when the ACK (acknowledge) message is sent by the initiator of the call in response to an OK received from the destination end. Hence the INVITE, OK and ACK 3-way handshake complete the session establishment. The ACK bypasses the intervening proxy servers end-to-end. Once the session is established, SIP withdraws from the communication and the media communication protocols like Real Time Transport Protocol (RTP) [18] take over. A point to note is that the media exchange does not take place via the same path as the signaling protocol. The signaling protocol is asked to intervene again only if the end points want to renegotiate the new or current parameters.

### 2.2.4 Session Termination

The call is terminated when one endpoint sends a BYE message to the other, circumventing the proxy servers. This is followed by an OK message from the recipient of the BYE, which terminates the session. There is no ACK message for this OK since ACK's are sent as responses to the INVITE messages only. The following diagram shows all the above steps for a SIP session between two domains. The provisional responses are ignored for brevity.

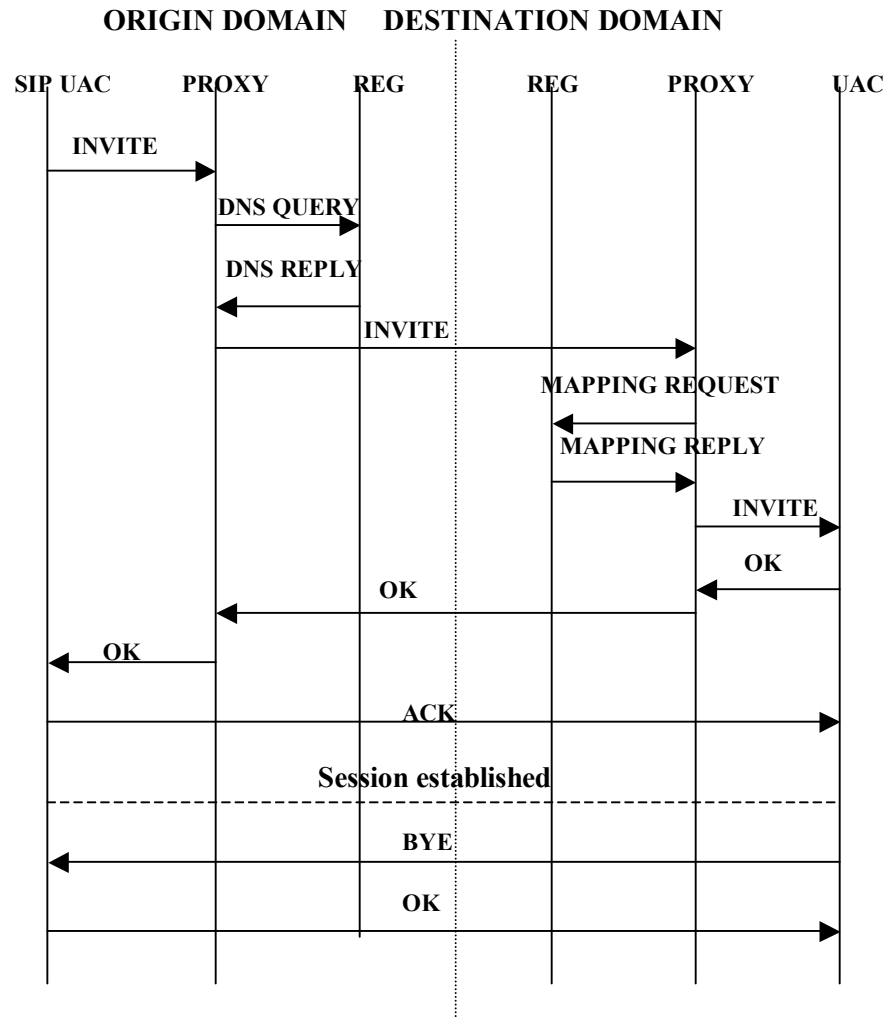


Figure 2.4: A SIP call flow

## 2.3 SIP messages

SIP messages are sent between SIP entities in order for the protocol to perform its functions. They can be classified into requests, if initiated by the client to the server or as a response if sent from the server to the client. Messages follow the Augmented Backus-Naur Format (BNF) [23] for syntax. The message begins with the start-line followed by the message header, a blank line indicating the end of the header and then the message bodies. An example of a SIP message is as follows [20]:

```

INVITE sip:7170@iptel.org SIP/2.0
Via: SIP/2.0/UDP 195.37.77.100:5040;rport
Max-Forwards: 10
From: "jiri" <sip:jiri@iptel.org>;tag=76ff7a07-c091-4192-84a0-
d56e91fe104f
To: <sip:jiri@bat.iptel.org>
Call-ID: d10815e0-bf17-4afa-8412-d9130a793d96@213.20.128.35
CSeq: 2 INVITE
Contact: <sip:213.20.128.35:9315>
User-Agent: Windows RTC/1.0
Proxy-Authorization: Digest username="jiri", realm="iptel.org",
algorithm="MD5", uri="sip:jiri@bat.iptel.org",
nonce="3cef75390000001771328f5aelb8b7f0d742da1feb5753c",
response="53fe98db10e1074
b03b3e06438bda70f"
Content-Type: application/sdp
Content-Length: 451

v=0
o=jku2 0 0 IN IP4 213.20.128.35
s=session
c=IN IP4 213.20.128.35
b=CT:1000
t=0 0
m=audio 54742 RTP/AVP 97 111 112 6 0 8 4 5 3 101
a=rtpmap:97 red/8000

```

**Figure 2.5: A SIP INVITE request [20]**

The first line indicates that it is an INVITE request, with the Request URI as the next-hop on the local network. The order of all fields except that of “Via” is inconsequential. The “Via” is the only header where the values are in a sequential order, since the path chosen by the request is inversed and used for the response. The Call-ID is similar to tags and is unique to the dialog between the end points. Cseq is the call

sequence, which keeps a count on the number of requests made, thus maintaining flow control and controlling retransmission. Assuming the cseq = 1 for the initial INVITE, the Proxy-Authorization and the cseq value = 2 indicate that this is possibly a retransmission of the request with the credentials of the SIP UAC. As mentioned before, the blank line separates the header from the body of the message.

### **2.3.1 SIP requests**

Requests are sent in a SIP session from the client to the server in order to invoke a particular operation [1]. They carry

- A request line in the first line, which contains the Method name.
- A Request-URI.
- The version of the signaling protocol being used.
- Terminating with the CRLF.

#### **2.3.1.1 SIP method**

The method name reflects the state of the SIP transactions taking place during a session. They are six in number -

- REGISTER needed for the registration of the end point.
- INVITE, ACK and CANCEL playing a part during the setup of the session.
- BYE for the termination of the session and
- OPTIONS for maintaining the session.

#### **2.3.1.2 Request-URI field**

The Request-URI field is populated with the user or the service to which it is addressed whereas the version of the protocol is present in both requests and response messages. Due to SIP being so closely related to HTTP the versions correspond closely to the HTTP standard [10] with the HTTP being substituted by SIP.

### 2.3.2 SIP responses

A SIP response is the action taken by the server that is conveyed to the client. The response message has a status line, which contains the protocol version, the response code and the response phrase with each having a single white space in between [1]. The protocol version is the same as described above whereas the response code and reason phrase have their own importance. An example of a SIP response is as below:

```
SIP/2.0 401 Proxy-Authenticate
Via: SIP/2.0/UDP 192.168.15.230:5060
From: sip:muda@colorado.edu
To: sip:doug@fcc.org;tag=124jahs09123h24h12j4h1j2
Call-ID: 12488221@192.168.1.30
CSeq: 22 INVITE
Contact: <sip:doug@236.87.48.68:5060;transport=udp>;q=0.00
Server: Winip (proxy)
Content-Length: 0
Proxy-Authenticate: Digest realm="att.com",
    domain="sip:fcc.gov", qop="auth",
    nonce="f84f1cec41e6cbe5aea9c8e88d359",
    opaque="", stale=FALSE, algorithm=MD5
```

**Figure 2.6: A 401 SIP response**

The header illustrates the difference between the request and the response. The first line shows the response code in numerical and reason phrase format. This is followed by the remainder information, which in this case is the challenge from the proxy server to the UAC asking for authentication credentials.

#### 2.3.2.1 Response codes

Response codes are 3-digit integer codes that convey the state of the SIP transaction during a response. Response codes range from 1xx to 6xx according to the state and the location from which these responses are being sourced. The first digit of the response code indicates the class of the response. 1xx codes are termed as provisional responses since they indicate the progress of a SIP transaction and not the culmination of one. Examples of provisional codes are Trying (100) and Ringing (180). Final response codes range from 2xx to 6xx with each having a different message. The numerical code in the response is for machine interpretation while the reason phrase is for better human understanding.

- 2xx class indicates success of the SIP transaction.
- 3xx indicates redirection to another proxy server that can accept the request or it may also provide the change in the location of the destination end point.
- 4xx indicates client error, which without any modification will not be accepted at the same error-generating proxy server.
- 5xx indicates server error where the blame lies with the server's incapability to complete a request.
- 6xx indicates a global failure, which means that the request cannot be completed at any server.

### **2.3.3 SIP headers**

The headers are an integral part of the message and carry information about the network as well as the end users between SIP end points. They have a general format but some of them are specified only for requests or responses. They can be sent in a compact form in order to adjust to the Maximum Transmission Unit (MTU) size of some

networks. Thus it becomes necessary for a SIP end point to be able to read both the verbose as well as the compact form of the headers.

## 2.4 Example of a SIP request message and its response

The actions of a UAS can be summed up by the following interesting example [32], which illustrates the response of the UAS to a request. Table 2.1 and 2.2 list each of the headers in the request and response messages followed by a description of the header fields:

Header name	Meaning of header
INVITE sip:bob@acme.com SIP/2.0	Request line: Method type, request, URI (SIP address of called party), SIP version.
Via: SIP/2.0/UDP alice_ws.radvision.com	Address of previous hop.
From: Alice A. <sip:alice@radvision.com>	User originating this request.
To: Bob B. <sip:bob@acme.com>	User being invited, as specified originally.
Call-ID: <a href="mailto:2388990012@alice_ws.radvision.com">2388990012@alice_ws.radvision.com</a>	Globally unique ID of this call.
CSeq: 1 INVITE Command sequence.	Identifies transaction.
Subject: Lunch today.	Call subject and/or nature.
Content-Type: application/SDP	Type of body—in this case SDP.
Content-Length: 182	Number of bytes in the body.
<b>Blank line marks end of SIP</b>	
<b>Headers and beginning of the body</b>	

**Table 2.1: SIP Request headers**

Header name	Meaning of header
SIP/2.0 200 OK	Status line: SIP version, response code, and reason phrase.
Via: SIP/2.0/UDP alice_ws.radvision.com	Copied from request.
From: Alice A. <sip:alice@radvision.com>	Copied from request.
To: Bob B. <sip:bob@acme.com>;tag=17462311	User being invited, as specified originally.
Call-ID: <a href="mailto:2388990012@alice_ws.radvision.com">2388990012@alice_ws.radvision.com</a>	Copied from request
CSeq: 1 INVITE	Copied from request.
Subject: Lunch today.	Call subject and/or nature.
Content-Type: application/SDP	Type of body—in this case SDP.
Content-Length: 200	Number of bytes in the body.
<b>Blank line marks end of SIP</b>	
<b>Headers and beginning of the body</b>	

**Table 2.2: SIP Response headers**

## 2.5 Directory services

Directory Services are similar to database systems, and are used for SIP to IP mappings as well as resource location by SIP end points. These services are located on authorization servers. Directory services provide enormous facilities to store user profiles of the end users on a local domain. A directory consists of several data elements defined as objects, which can be any of the following: the user, a resource or any other method of defining end points. Each object is then given different rights to read or write, thus offering a large degree of granularity.

Johnson [16] defines objects in directory services not only for H.323 but also for SIP, cellular phone services and pager services. The directory services play an important part in our design since the proxy server or the client, depending upon the model chosen, accesses it through the authorization server for gaining attributes or characteristics of the SIP UAC.

## **2.6 Federation**

The federated model is one of the primary drivers of the research done towards this thesis. A federation is an aggregation of devices that have a trust model between them. The trust model allows a network element in a domain to trust another network element, which is not part of its domain. In a role-based authorization model, implementing a federation reduces the overhead in SIP messages and reduces the burden of maintaining information about all user agent clients trying to access the network. We will explain how this is achieved with the help of an example.

### **2.6.1 Trust model vs. Non-trust model**

Consider two domains that do not have a trust model between each other. The origin proxy server forwards the request to the destination proxy server where the request is checked for information necessary to access any resource in its domain. If the information is found to be missing, a response is sent back to the originating client asking it to send another request with the necessary information. Compare this to when a trust model is present between the two domains. When the proxy server at the origin domain stops a request from a UAC on its domain, it checks to verify if there are any additional header fields required for the destination domain for which the request is heading. The parameters required for each domain will be exchanged between the proxy servers as part of the trust model. If additional information is required and is not present in the current

request, then a response can be sent back asking the client to add the appropriate headers and send another request. This reduces the round trip time as compared to the case when the destination proxy has to check for credentials and return a response asking the origin UAC to attach certain headers in order to gain access. Since the destination proxy server trusts the origin proxy server because of the trust model, most of the time the request will have the parameters required for entry. That becomes a time saving feature as well since the process time when passing the request through the access-list is reduced.

### **2.6.2 Example of a federated model**

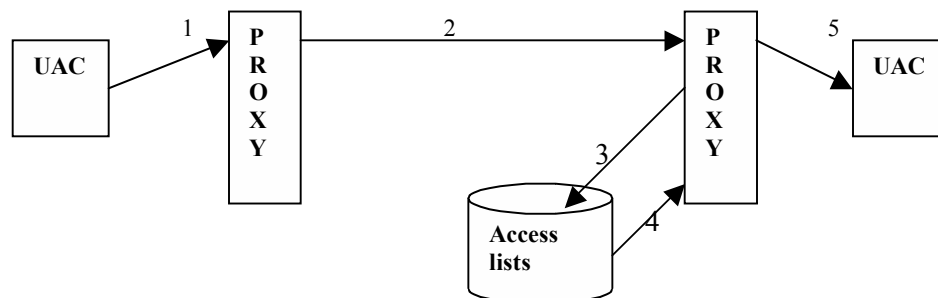
Consider a common initiative being carried out by research groups in two different universities, recipients of a shared grant. Such an arrangement may require shared access to resources distributed between the two university sites. However, to protect the confidentiality and integrity of the resources, it cannot simply be made publicly available. Further, there may need to be a granular method of access to segregate access privileges among the people within the research groups. For example, there will be different access rules for professors in a research group compared to the students in that group when accessing a remote resource. This means that a remote authority must be able to associate a user from another realm with a set of access rules. As mentioned before, this can be quite a burden. Furthermore, the user must trust the sharing of identifiable user information to access the remote resource.

Inter-domain sessions give rise to several opportunities to exploit that user's privacy. An alternative would be to assign a role (such as professor) and have the attributes for the subject examined by the authority of the remote resource. The remote authority may examine the authenticity of this assertion and make a decision regarding

access only if the authorization server in the origin domain grants it permission. The permission for different authorities demanding information on the originating client is granted according to the rights given to that domain by the network administrators in the origin domain. The extra round trip time consumed by this request can be avoided if the UAC or the proxy server in the origin domain adds the necessary information according to the destination domain. For the success of such a network, the network elements in different domains like the proxy servers must have an agreement according to which the authorization role of the domains is present in the authorization servers in the domains. This form of agreement gives rise to the federated model. It also eliminates the need for the remote authority to maintain a separate access control list for each remote user as well as protects the remote user, since personal information is protected and only the necessary attributes are exchanged across a network.

### 2.6.3 A Non-trust model

Fig 2.7 shows the UAC sending a request to the Proxy server, which then forwards it routinely to the destination proxy server. The destination proxy server as an authorization mechanism can then send the request through the access-lists and if the conditions are met the request will be forwarded to the destination UAC.



**Figure 2.7: A non-trust model**

### 2.6.4 A Trust model

The diagram shows the origin proxy server sending the requests to the authorization server that then replies with the certain attributes required to get access into the destination domain. These attributes are then sent as a new request from the origin to the destination where they are parsed at the destination before being let through to the resource (in this case the UAC).

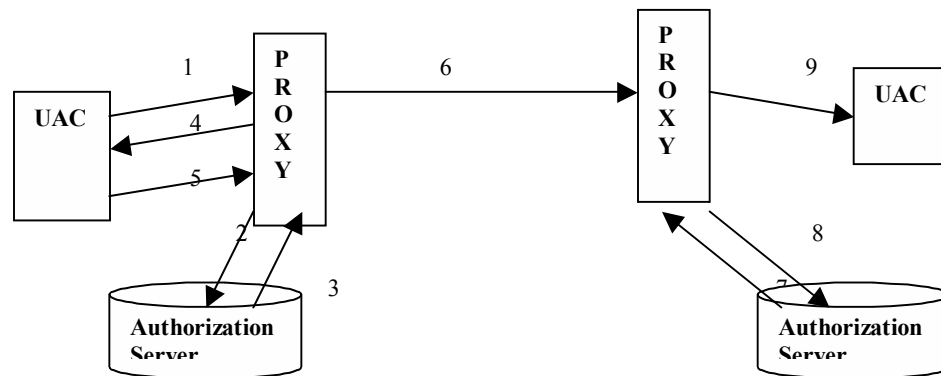


Figure 2.8: A trust model

## 2.7 Security Assertion Markup Language (SAML)

SAML is a security standard derived from XML (eXtensible Markup Language) [22]. It defines the protocol for exchange and format of authentication and authorization information. Assertions are used to carry information about the entity, which could either be a machine, user identity or a human. The information in an assertion is present in the element fields, present in a hierarchical manner. SAML uses an XML-based request and response scheme for exchanging information to and fro and can run on any underlying protocol. Currently, the only bindings available for SAML are with HTTP and are defined in the Simple Object Access Protocol [9]. Until the development of SAML bindings with SIP, a SIP session can use XML assertions by reference in the header, which is added to the INVITE.

The issues faced during the development of the network and flow diagrams, in accordance with the federated SAML model are documented in the next chapter.

## **2.8 Authorization**

Authorization is the permission to access a resource. An authorization authority grants the permission for access to the presenter of a particular credential [34] using policies for reference. Policies are the set of rules that state the information needed for successfully accessing a resource. There is no restriction on the form of the information e.g. assertions in SAML, IP header information, that an authorization authority examines. Thus, the information can be in a very granular form and this is utilized by protocols and technologies using authorization for increased security.

## Chapter 3

### LACK OF AUTHORIZATION FEATURES IN A SIP SESSION

In this chapter, we discuss the normal call flows for SIP sessions, the existing response messages and headers in SIP that cater to security and the lack of authorization header fields in them. We then look at the mechanisms presently used or proposed for use for securing a SIP session. The chapter concludes with the shortcomings of these mechanisms and the choice of our mechanism.

#### 3.1 Why is there need for additional security in a SIP session?

The Session Initiation Protocol (SIP) session is used by network elements for resource discovery as well as initiating, modifying and terminating a session. Just like any other protocol, security is an essential element of SIP. Security can be broadly defined into authentication, integrity, confidentiality and authorization of network elements. Some of the security threats to the confidentiality of a SIP session are the Denial of Service, Man-in-the-Middle and ping attack. Similarly, the integrity of the sessions can be compromised by the inspection of messages by intervening devices. There are mechanisms built into SIP for protecting it from these kinds of attacks. SIP relies on protocols like IPSec, Transport Layer Security (TLS) [13] to offer confidentiality and Secure Mime (S/MIME) [12] to offer integrity in a session.

There is still no firm mechanism of offering security to a SIP session using authorization. Using the attributes and characteristics of an end user for the differentiating we devise in the later chapters an authorization based model for SIP with the necessary modifications done to the SIP client.

## **3.2 Call flows**

A call flow is a diagrammatic representation of the flow of a SIP session from the initiation to the termination of the session. The call flows in SIP are used to depict each stage of the session and are the fundamental and initial basics for further work like programming and network architecture determination. The call flows in Fig 3.1 does not contain the session termination stage since our solution is pertinent in the session initiation and establishment stages only.

### 3.2 Call flow diagram for a simple unsecured SIP call flow

The first call diagram depicts a normal unsecured SIP call flow diagram across domains and consists of SIP network elements like SIP clients, SIP proxy servers and the registrar. They are split across two domains, origin and destination, according to the presence of the initiator of the call. Each leg of the call flow diagram is documented in text below the diagram.

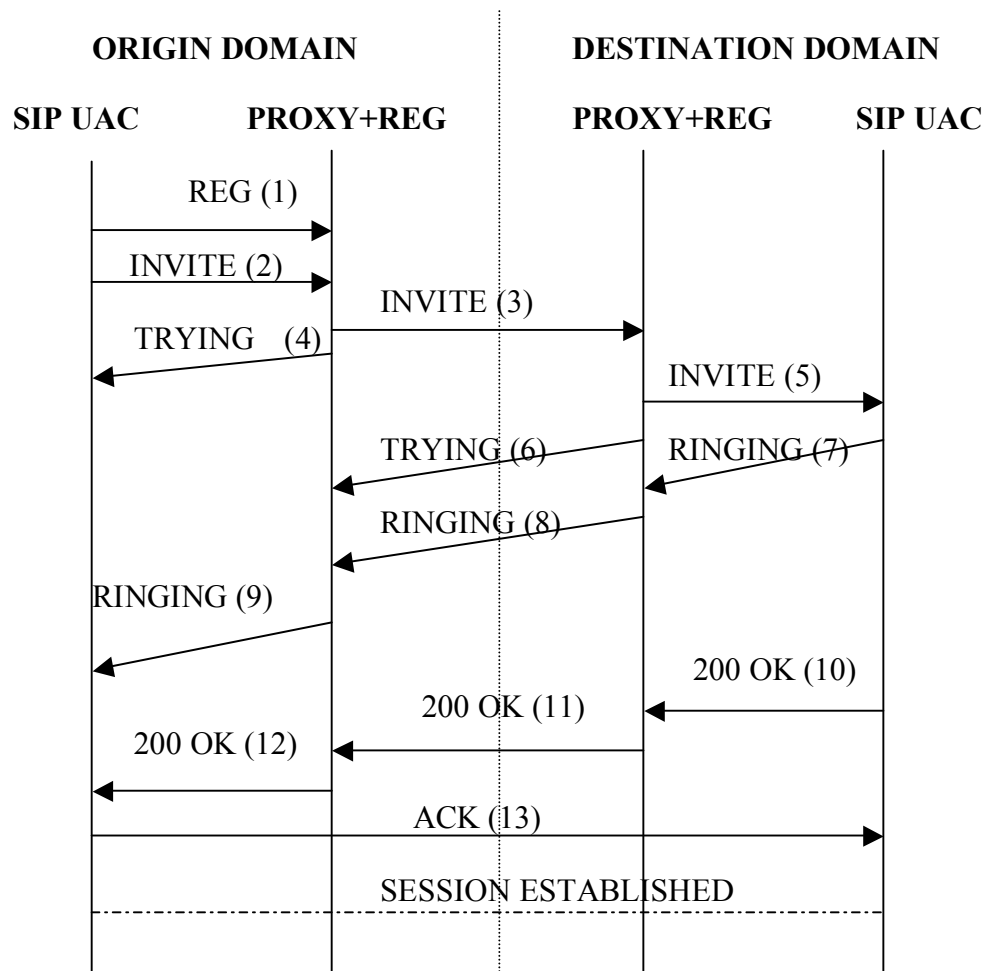


Figure 3.1: An unsecured SIP flow

**Steps:**

- 1) Registration - instructs the REGISTRAR to map the SIP URI of the username to its IP address and forward all incoming requests.
- 2) INVITE –the request sent by the initiator to the proxy server.
- 3) INVITE –forwarded by the origin proxy to the destination proxy server.
- 4) Trying - provisional message (1xx) sent back indicating that the proxy server is trying getting in touch with the destination. It prevents the normal timer at the client from timing out.
- 5) INVITE – The request is forwarded to the destination proxy server
- 6) Trying – provisional message sent back indicating destination proxy server is trying getting in touch with the destination end point.
- 7) Ringing – provisional message to the proxy server from the destination UAC indicating presence. This prevents another set of timers from expiring.
- 8) Ringing – message sent back from the destination proxy to the origin proxy.
- 9) Ringing – message sent from the proxy to the UAC to avoid expiration of timers on UAC.
- 10) (11) (12) –OK sent back from the destination UAC to the origin UAC.
- 13) ACK –sent back as answer to the OK telling the destination proxy that the origin is ready for communication.
- 14) Session is established between the end points.

**3.3 Headers and Response codes in SIP catering to security**

A header is used in a SIP message to convey more information about the message. These headers are present in the requests like INVITE sent by the UAC as well as in the responses sent by the server to the UAC. This section explains the headers used in requests and responses in SIP that carry authentication or authorization information.

This section is in two parts- one dealing with the headers in requests and the other with the responses which are used to inform the client of the failure of some parameter required for secure SIP sessions.

### **3.3.1 Responses catering to security**

Responses are the answers sent by the recipient of the requests sent by the UAC. The responses inform the initiator of the request of the state of the session after it has been sent. They are classified into 6 types, ranging from 1xx to 6xx, according to the answer that needs to be provided to the UAC. The 1xx response codes signify that the session is still in progress and are hence termed as provisional response codes. The 2xx class of response signifies success, but from 3xx to 6xx, the response codes depict some sort of failure in the initial request sent by the UAC.

There are two known response codes, which depict the lack of a security parameters in the request sent by the UAC. They are the WWW-Authenticate and Proxy-Authenticate responses. The numerical codes assigned to them are 401 and 407 respectively. On receiving these responses the UAC responds with another request or re-INVITE which contains the parameters required for the request to be accepted. The UAC can also opt to send the original request to another proxy server if it does not want to change the SIP message header.

#### **3.3.1.1 401 and 407 response codes**

The 401 response code tells the UAC that it is unauthorized to forward its requests. The WWW-Authenticate header must be present in a 401 response sent back to the UAC by the UAS, which can be a client that received the initial INVITE for this transaction [1]. This response signifies that there is no information in the Proxy-

Authorization header sent in the original INVITE by the UAC and is a challenge to the UAC. The UAS sends back at the least the authentication schemes and the parameters associated with the realm in the challenge. These constitute the mandatory fields that need to be reassigned to the Proxy-Authorization header when the UAC sends a follow-up INVITE. An example of the WWW-Authenticate header is depicted below.

```
WWW-Authenticate: Digest
    realm="colorado.edu",
    qop="auth,auth-int",
    nonce="assa234243e2nm234joobvu85332",
    opaque="aa32kj23k4j2k3j42k34j000"
```

**Figure 3.2: WWW-Authenticate header**

The following points can be gleaned from such a header:

- The Digest in the header indicates that the HTTP Digest Authentication scheme [11] is being used.
- The realm is an indicator to the end user about the username and password he/she needs to use.
- The qop field value is the quality of protection, which determines the level of security that needs to be implemented when there is an exchange of messages between the UAC and UAS. The default value for qop is auth with auth-int as another option. Setting the qop field to auth-int leads to integrity checks along with the authentication [25].
- The nonce is a unique value generated every time a challenge is made to the UAC and is used to detect security breaches.
- The opaque is a string, which is returned to the UAS by the UAC without any modifications.

On similar lines, the proxy-authenticate header is sent back in the 407 response code by a proxy server only, which has decided to authenticate the UAC. The challenge sent by the proxy to the UAC is treated in the same manner by the UAC, which incorporates the header fields into the proxy-authorization header that it sends back in the second INVITE. The header values of both the 401 and 407 response codes are the same with the header name differing. The former can be classified as a user-to-user authentication whereas the latter can be classified as a Proxy-to-user authentication. <sup>[1]</sup>

### 3.3.2 Proxy-Authorization header

The UAC reacts to the responses, 401 and 407 that it receives from the UAS or the proxy, by creating a proxy-authorization header or populating the incomplete proxy-authorization header sent earlier in the INVITE [1]. It can be summed up as a response of the client to identify itself to proxy servers in user-to-proxy server environments, vis-à-vis the authorization header that cannot be modified by the proxy server. The proxy server cannot read, delete or modify the authorization header, which though it carries the same parameters as the proxy-authorization header, is meant for user-to-user authentication. The repopulated or newly created header is the proxy-authorization header, which is then sent in an INVITE to the client or the proxy server that is requesting authentication. The header fields of the proxy-authorization header are illustrated below.

The proxy-authorization header is made up of some required and some optional fields.

An example of a proxy-authorization header is as documented below. [11][25]

```
Proxy-Authorization: Digest username="bob",  
realm="colorado.edu",  
nonce="ad7a24ks9f909ff3fas31af4qda4c093",
```

```
uri="sip:muda@Colorado.edu",  
qop=auth,  
nc=00000001,  
cnonce="a782kn9",  
response="asd8af87a9f85a72jh124h1k2hb412",  
opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

**Figure 3.3: A Proxy-Authentication header**

As can be seen, the parameters are similar to the response received from the proxy server or the UAS in the 401 and 407 challenges. Following the challenge, either the user is prompted for credentials, which then populate the header. We will now discuss where the Proxy-Authentication header is lacking or insufficient in terms of providing enough information to the recipient of a request.

### **3.4 User Agent Client behavior**

The UAC reacts to the response headers asking for more authentication information by inserting its credentials into the proxy-authorization header. The steps a UAC takes are as follows:

- 1) It checks for the mandatory values and clones them from the earlier INVITE message.
- 2) It checks for the optional values and clones them from the earlier INVITE message.
- 3) It sends across a second INVITE.

This is in accordance with the SIP RFC 3261, which states, “For all the known 4xx responses the request is retried by creating a new request with the appropriate modifications.” The only change to the existing parameters that needs to be changed is

the incrementing of the Cseq (call-sequence) value. Please refer to Section 2.5 for a similar server response in terms of messages exchanged.

### **3.5 Lack of authorization details in the proxy-authorization header**

The information passed in the proxy-authorization header helps authenticate the UAC to the UAS or the proxy server but apart from that provides very little information to those endpoints. Thus, there is very little ability to differentiate User Agents with the information present.

Consider a SIP application where there are several users in each realm. These users send only their username and realm information in the SIP headers, out of which the “From” header field is easily manipulated. For a proxy server monitoring all incoming requests, there is little choice but to accept the SIP INVITE from the initiator. There are several parameters, which an end user profile can carry, like the department affiliated to, position within that department, organization affiliated to, age, sex and several other attributes. The revelation of these attributes can be made at the permission of the user who can also determine whether his/her name should be revealed to the destination recipient proxy server or UAS.

### **3.6 Currently proposed or implemented mechanisms for securing SIP**

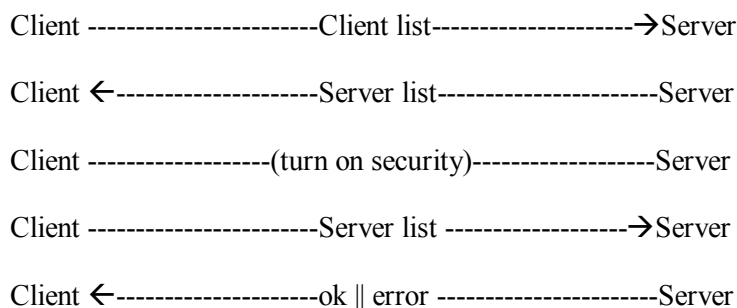
There has been less work done in the security aspect of SIP compared to other aspects. The cobbled solution for securing SIP sessions, as provided in the SIP Request for Comments 3261, using various protocols for local authentication and inter-domain authentication along with integrity and confidentiality is generally accepted for most SIP

implementations. There are security mechanisms that are already defined and in the proposal stage, which we will discuss briefly in this section.

### 3.6.1 The Security Mechanism Agreement for SIP protocol

This agreement [26] seeks to introduce uniformity in the manner in which the security options are chosen by a SIP UA and the first hop from it. Unless in peer-to-peer conditions, the first hop is always a proxy server local to the realm from which the initiator has launched the request. The mechanism proposes the use of three headers to resolve the security mechanisms that are to be used between the UA and the first hop. The exchanges leading up to the determination of the security protocols to be used, is also secured by multiple exchange of the features between the UA and the first hop proxy.

The three headers introduced by this mechanism are the “security client”, “security server” and the “security verify” [26]. These headers are involved in an exchange resembling something like the following



**Figure 3.4: Message flow for the agreement** <sup>[3]</sup>

The client list is present in the security client header, the server list sent by the server in the security server header and the server list sent by the client in the security verify header. These security client and the security verify headers are sent in Requests and can be added, deleted and read by the proxy server. The security server header is sent back in responses 421(Extension Required) and 498 (Security Agreement Required) with the list of security mechanisms the server supports [26]. These security mechanisms are then used for future exchanges between the UA and the proxy server.

### **3.6.2 Content indirection**

Content indirection [27] is a mechanism that allows for indirect reference through a Uniform Resource Identifier (URI) to the Multipurpose Internet Mail Extension (MIME) parts in a SIP message. This mechanism has several implementations apart from security in SIP. Indirection aims at giving the recipient of the request the right to determine whether it should obtain the content via a channel other than the signaling network. This is achieved by providing the recipient of the request with a URI where the content is actually hosted.

The indirection of a message is useful in low bandwidth networks or when the intermediate proxy servers between the origin and the domain realms are not adept at handling large sized messages. Security profiles of users will be stored at an authorization server, which is an independent entity. Access for the origin UAC or proxy or the destination proxy to the authorization server can be provided in the form of a URI. The retrieval of information by the recipient is done over any channel ranging from HTTP to LDAP [28] and is flexible in that respect.

The content-type header <sup>[1]</sup> used to transfer information about the content to the recipient has information pertinent to the URI like the expiration time, size of the content, purpose of the content and the type of the content. These header fields are a comprehensive information database to the recipient about the characteristics of the content.

There are several disadvantages related to content indirection. Primarily, there is no control over the security of the channel to be used for retrieval of information about the endpoint from the host on which it is stored. The URI can reveal the geographical location and the username of the host, which infringes upon the privacy of the end user. Another security loophole is that the URL could be a fake URL directed towards some element in the local realm of the recipient, thus opening an avenue for a port scan attack.

### **3.7 Choice of mechanism for the solution**

Either of the two mechanisms discussed above has features that are not currently present in SIP but neither can offer granularity while performing authorization. Each of them has its own fallacies, which we use to eventually arrive at our choice of methodology.

The security mechanism is devised for the SIP endpoints to successfully exchange information about the security protocols they support before the establishment of the session. The security headers advised- security client, security server and security verify have their parameters listed as the security protocols which each of the client and server support. This mechanism does nothing to alleviate the lack of authorization for a SIP session. The content indirection mechanism does away with the troublesome issue of carrying assertion as attachments. It is a well-known issue that there are issues on the

Internet when carrying XML [22] extensions and until XML accelerators become a norm in most networks, we will keep the attaching of a MIME body as an option but not as compulsory. Content indirection however has no underlying mechanism specified for retrieval of the information, and is a more generic method for retrieval of data in any form. Also, the passing of HTTP [10] and File Transfer Protocol (FTP) URLs is seen as a security loophole that can be exploited for Denial of Service and port scan attacks. Shibboleth [8] is a protocol that has been primarily devised for web services and at this point of time has only SOAP to HTTP bindings specified. Thus it cannot be completely integrated into the SIP architecture though it can be used for drawing up a similar architecture for the SIP sessions.

Due to the limited research done in securing SIP sessions using authorization we had to devise the use of new headers in requests and responses and call flows. What we aim at is similar to the content indirection ability, albeit in a completely different manner, and we do draw from the access control performed by Shibboleth [8] in our solution. Through this chapter we sought to explain where the current work is limited and how we could contribute to the security of a SIP session by devising another solution, which could be integrated with the above solutions to provide a complete secure SIP network.

## Chapter 4

### APPROACH TO IMPLEMENTATION

This chapter introduces the authorization model followed by the call flow diagrams devised for our unique solution. Details the addition of a new response and headers are provided, finally concluding with the demonstration of the syntax of the new header fields and a summary of their use.

#### 4.1 The Solution – Incorporating the authorization model in SIP sessions

As addressed and analyzed in chapter 3 there are protocols present that guarantee authentication, confidentiality and integrity as well as mechanisms where headers are exchanged to determine compatibility between end points using these protocols. But the potential of implementing authorization parameters as a tool for security has not been exploited by any of them. The solution presented by this thesis does not recommend any changes to those security mechanisms but instead suggests means of improving security for SIP sessions by addition of headers in SIP requests and responses in order to incorporate an authorization model [33].

The authorization model uses the increased granularity, in the form of attributes and characteristics of the end user agent, for improving the decision-making ability of a SIP proxy server that receives a SIP request. The user profiles, which contain the attributes, are present at the directory servers from where they are retrieved using the SAML authorization server. The SAML authorization server then transcodes these attributes into assertions, which can be either by reference or by value. An assertion by reference does not carry the value or the information but rather something that loosely

defines the information or the place where the information is stored. The transcoding into SAML assertions of user profiles and then subsequent change into another form is out of scope of this research and is a work in progress by Anand Chavali of the Internet Applications Lab Group of the ITP. To access the authorization information we devised call flows and changes to the existing SIP message requests and responses. The additional headers that have been constructed in the SIP message are at present only capable of carrying strings or integers in the header values and have been named after the elements carried in an XML assertion. This thesis shows that changes can be made and implemented at the SIP UA and to the SIP messages sent and received, in order to incorporate the authorization model.

#### **4.1.1 Models for retrieving the authorization information**

The authorization model distributes information about the originating UAC to the proxy servers or a UAS in the destination domain. The origin proxy server can check whether the necessary information for the destination domain is present when the initial INVITE arrives from the originating UA. Alternatively, the destination proxy server can ask for additional information when it receives the SIP INVITE. The federated trust model between the domains determines the information needed when a request is sent to a particular domain. Thus by having the proxy server in the origin domain check whether the necessary information for a destination is present we eliminate the extra processing at the destination proxy as well as avoid the extra round trips. Thus the steps in a SIP network implementing the authorization model are as follows:

#### 4.1.1.1 Origin proxy initiated authorization model

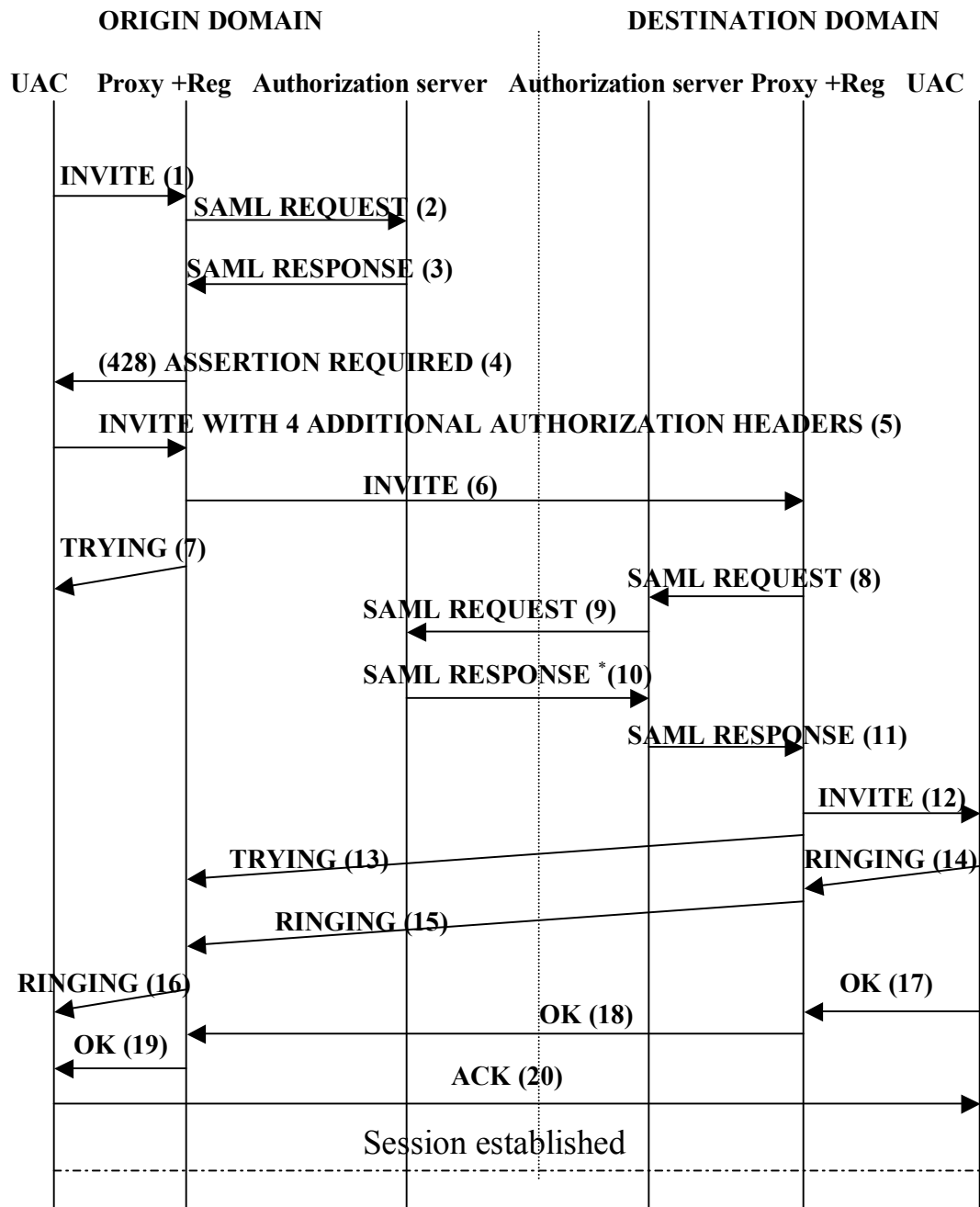


Fig 4.1: Origin proxy initiated authorization model

\* SAML response is either an accept or a reject. The proxy server then accordingly forwards an INVITE to the UAC

The steps for this model are similar to a normal unsecured SIP call flow but for the addition of steps 3 to 6. Hence only those steps will be explained below. In the case of missing authentication information when the 401 or 407 responses are sent back the only additional steps are 3 and 4 but with different headers in the re-INVITE. The aim is to keep the number of round trips as low as possible.

- Step 3 is the proxy finding that the initial INVITE does not have the authorization information required for the destination domain and hence the proxy server sends a request for assertions to the authorization server.
- Step 4 is the authorization server returning values to the proxy server.
- The proxy server sends the additional headers back to the UAC in Step 5 in the form of a response 428 [33] (Reqassertion).
- Step 6 involves the UAC taking the header values in the response and copying them into corresponding header values that are added to the second INVITE message being sent across to the proxy server.

As can be seen from the call flow diagram, the changes that would be required to be made to the existing SIP messages would be the addition of four headers to carry the authorization information needed and the addition of a new SIP response 428 [33] labeled “Required Assertion”. The header details and the response message are explained in more detail in section 4.2 and 4.4.

### 4.1.1.2 Destination proxy initiated authorization model

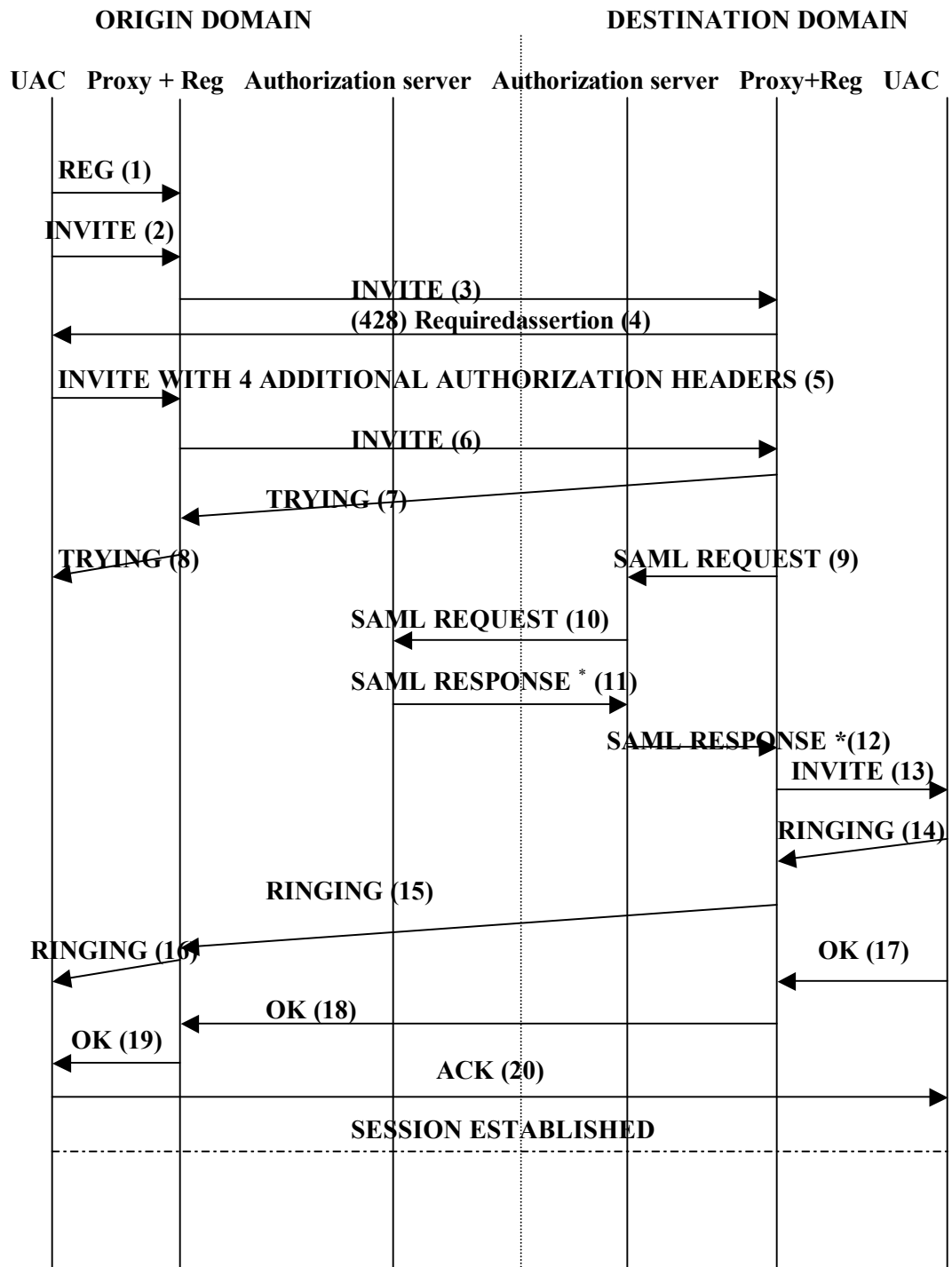


Fig 4.2: Destination proxy initiated authorization model

\* The SAML response is either an accept or a reject. The proxy server only forwards the INVITE to the UAC at the destination domain.

This model is implemented when the proxy server at the origin domain does not have a strict policy of access control for requests coming from users in its own domain. If the proxy server at the destination does see that the authorization parameters that are needed to access a user within are not present, then it will generate a response that will ask the UAC to respond with the four new header fields in the second INVITE sent out. This is covered in steps 4 and 5 of section 4.1.1.2. After receiving assertions, the destination proxy server will send SAML requests to the authorization server to gather those assertions (Steps 9-12). These assertions can be then stored for future transactions between the source and destination, though it may also be discouraged according to the security policies in place at the respective domains.

## **4.2 The new headers in SIP requests**

Though there are existing SIP headers labeled as authorization and proxy authorization headers, we do not use them since they are already specified to be generated for certain response messages received by a SIP UAC. Instead of modifying existing SIP headers we devised four new headers to carry the authorization information. These four headers are labeled Attribute Assertion, Authorization Assertion, Decision Assertion and Authentication Assertion. The four header names are tentative, since these decisions depend on other work in the IAL group [36]. For modifying the client software the header names and the value types have been assumed and their syntax defined accordingly. These headers can be easily changed in name or value type via the flexible code for the client.

These headers are named after the elements carried in the XML assertions for security purposes. The roles or information content of each of the headers has not been

defined as yet but in the current implementation they are capable of carrying character and integer information as their header field values.

- Attribute assertion: This carries values of attribute defined in the user profile of the UAC.
- Decision assertion: This will carry the whether a decision has been made about the UAC in question.
- Authorization assertion: This carries the authorization information in the user profiles about the UAC.
- Authentication assertion: This field carries information from the user profiles on the authentication of the UAC.

### **4.3 The new response code**

There is a new response code, 428 [33], which is of the 4xx class of response messages in SIP, which indicate client error. This header is sent from the proxy server to the UAC as a challenge to provide more authorization credentials. The “428” response type is labeled “Reqassertion” indicating required assertion and has the same four new header names and values which are then added to the second INVITE sent by the UA.

### **4.4 Header field names and Response codes definitions**

This thesis proposed four new headers and the response 428 to the sub-registry for SIP headers under <http://www.iana.org/assignments/sip-parameters> as:

Header Name: Authorizationassertion

Compact form: \*

Header Name: Attribute assertion

Compact form: \*

Header Name: Authentication assertion

Compact form: \*

Header Name: Decision assertion

Compact form:\*

Response Code Number: 428

Default Reason Phrase: Reqassertion

## 4.6 Syntax for the additions

We define a new response, Reqassertion, and new header fields. For encoding, the augmented Backus-Naur Form (BNF) [23] grammar is used, as in SIP. The syntax for the four new headers and new syntax for the response “Requiredassertion” will be more comprehensively specified once the complete model will be in implementation. The existing syntax is devised as follows:

### 4.6.1 Additional headers syntax

The following are the augmented BNF for the four additional headers.

attributeassertion = Attributeassertion EQUAL attribute assertion value

attribute assertion value = quoted-string

decisionassertion = Decisionassertion EQUAL decision-assertion-value

decision-assertion-value = quoted-string

authorizationassertion = Authorizationassertion EQUAL authorization-assertion-value

authorization-assertion-value = quoted-string

authenticationassertion = Authenticationassertion EQUAL authentication-assertion-value

---

\* Status is unknown as of now. To be decided as more protocol development is performed.

authentication-assertion-value = quoted-string

#### 4.6.2 Response header syntax

The syntax for the new response, Reqassertions, with status code 428 is defined below:

Required assertions = “Reqassertions” HCOLON assertions

assertions = attributeassertion / authorizationassertion / decisionassertion /

authenticationassertion

The assertion header field values will be the same as specified in section 5.6.1.

#### 4.7 Summary of header use

The following table summarizes the use of the headers in relation to the processing of the proxy server and SIP methods. There has been no work done in defining the response of different methods to the new response header, Required Assertion and all columns for that will be documented as Unknown.

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Attribute assertion	R	adr	o	o	-	o	o	o
Authorization assertion	R	adr	o	o	-	o	o	o
Decision assertion	R	adr	o	o	-	o	o	o
Authentication assertion	R	adr	o	o	-	o	o	o
Required Assertion	428		u	u	u	u	u	u

**Table 4.1: Summary of header use**

The “where” column specifies either the request or response in which the header is used. The “proxy” column specifies the actions the proxy server can take with the particular header type.

R: the header can only appear in requests

428: A numerical value indicating the response codes in which the header field can be present.

a: The proxy can add a header field value

d: The proxy server can delete any header field value

r: The proxy server can read the header field value

u: Current status is unknown.

## Chapter 5

### TESTS AND RESULTS

This chapter describes the tests that confirm the changes made to the linphone code. The tests also demonstrate the SIP messages that are needed in order to successfully implement the authorization model for SIP sessions. The tests are followed by the SIP message flow as captured from the command line of the SIP client and illustrations of the SIP message flow. There are also Ethereal network analyzer outputs for verifying the attachment of the extra authorization headers and the effect on the processing time of the SIP UA when it receives the headers in a “428” message response.

#### 5.1 Experimental setup

The set up is on a LAN with two computers connecting to a hub with a network analyzer, Ethereal running on the computers. One of the computers has a protocol development environment, with the “Linphone” client running on it, on Linux Red Hat 8.0. The Linphone package running in the current set up is the 0.10.2 version and the Libosip libraries, which it uses, are the 0.9.6 version. The other computer has a socket program that simulates a proxy server by listening on port 5060, processes the INVITE from the UA and sends back the desired response in the form of a 428 response message. The socket program generates random strings every time it receives the SIP INVITE message using a standard function. A SIP proxy that can perform this and more functions using SIP libraries is a work in progress by Ameet Kulkarni of the IAL research group.

## 5.2 Successful generation of a response

- **Objective**

To demonstrate the successful generation of the new “428” response message by the socket program

- **Procedure**

In this test will try to send an INVITE message from the client to the socket program acting as the proxy server. The socket program reacts to the original INVITE with a “428” response message carrying four additional header fields.

- **Result**

The socket program performs the function of listening to a message on the 5060 port, which is the SIP port. It copies the incoming message into a buffer and then copies the message into a new buffer from the old buffer. It then passes the message to the new buffer line by line by using the CRLF (Carriage Return Line Forward) present at the end of each SIP message line for the demarcation of fields. It attaches the four additional headers at the end of the SIP message, using the random string function as a string generator for the header values every time it sends back a response. It then attaches a new startline, “SIP2.0/UDP Requiredassertion 428”. This response is then sent back to the SIP client from the new buffer.

- **Messages exchanged are as follows:**

**Original SIP request:**

INVITE sip:doug@128.138.75.155 SIP/2.0  
 Via: SIP/2.0/UDP 127.0.0.1:5060;branch=z9hG4bK764867783  
 From: <sip:muda@128.138.75.188>;tag=3905387453  
 To: <sip:doug@128.138.75.155>  
 Call-ID: 3620384630@127.0.0.1  
 CSeq: 20 INVITE  
 Contact: <sip:muda@127.0.0.1>  
 max-forwards: 10  
 user-agent: oSIP/Linphone-0.10.2  
 Content-Type: application/sdp  
 Content-Length: 240

v=0  
 o=muda 123456 654321 IN IP4 127.0.0.1  
 s=A conversation  
 c=IN IP4 127.0.0.1  
 t=0 0  
 m=audio 7078 RTP/AVP 115 110 101  
 b=AS:110 8  
 a=rtpmap:115 1015/8000/1  
 a=rtpmap:110 speex/8000/1  
 a=rtpmap:101 telephone-event/8000  
 a=fmtp:101 0-11

**The 428 Reqassertion response**

SIP/2.0 428 Reqassertion  
 Via: SIP/2.0/UDP 127.0.0.1:5060;branch=z9hG4bK764867783  
 From: <sip:muda@128.138.75.188>;tag=3905387453  
 To: <sip:doug@128.138.75.155>  
 Attributeassertion: 762299093  
 Authorizationassertion: 1271565896  
 Decisionassertion: 1773134790  
 Authenticationassertion: 1196713515  
 Call-ID: 3620384630@127.0.0.1  
 CSeq: 20 INVITE  
 Contact: <sip:muda@127.0.0.1>  
 max-forwards: 10  
 user-agent: oSIP/Linphone-0.10.2  
 Content-Type: application/sdp  
 Content-Length: 240

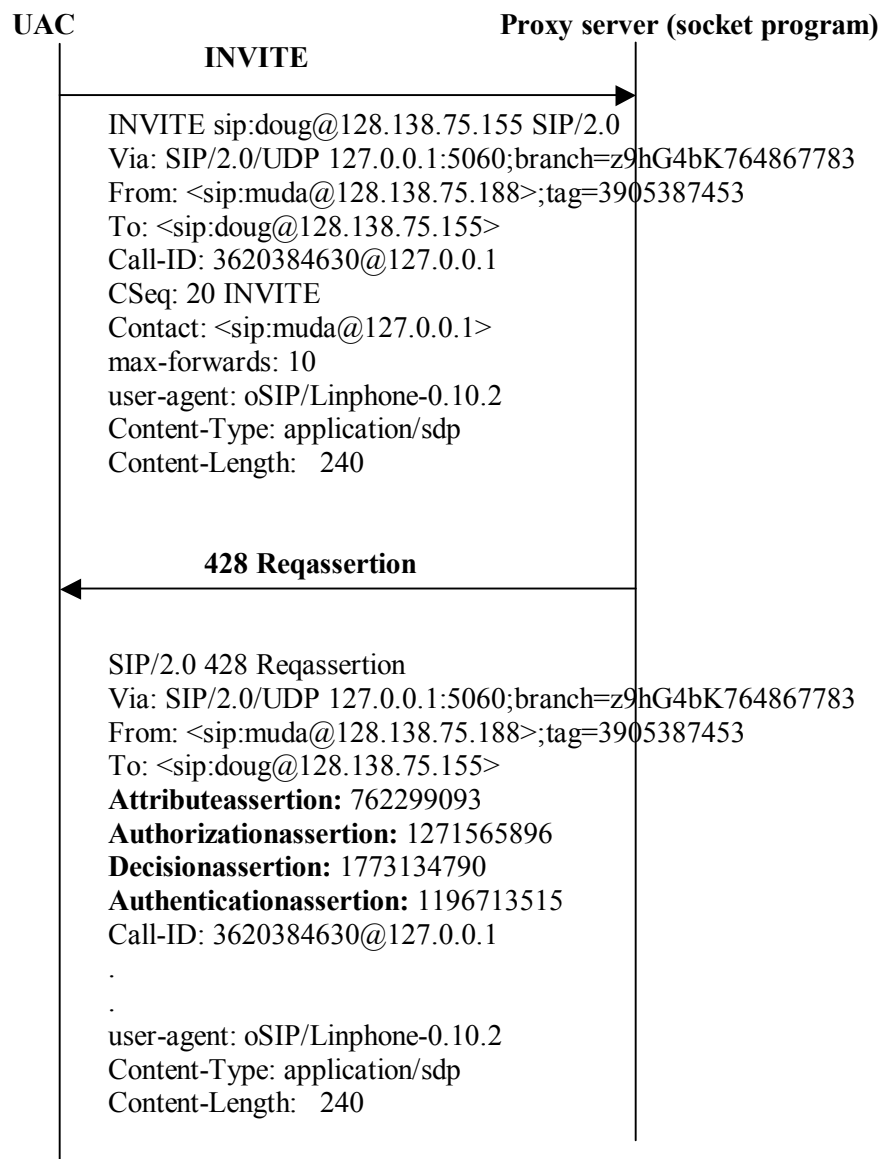
v=0

```

o=muda 123456 654321 IN IP4 127.0.0.1
s=A conversation
c=IN IP4 127.0.0.1
t=0 0
m=audio 7078 RTP/AVP 115 110 101
b=AS:110 8
a=rtpmap:115 1015/8000/1
a=rtpmap:110 speex/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-11

```

- **Illustration of the result**



**Figure 5.1: The initial request and response captures**

### 5.3 UA client processing

- **Objective**

Successful recognition of the new response message by the SIP UA and attachment of the four new header fields in the new INVITE message

- **Procedure**

The original INVITE is sent as usual by the SIP UA and the response is sent back by the socket program as defined in the first test. This leads to the response message being parsed by the SIP UA, which then attaches the new headers into the new INVITE after getting the header names and values from the response message.

- **Result**

On receiving the response 428, indicating lack of authorization information in the original request, the client attaches the fields carrying the assertion information and sends a follow-up INVITE containing the new header fields it has obtained from the response. The parsing ability of the client is confirmed by the attachment of new header values every time the socket program that generates the random strings sends back a response.

- **Messages exchanged**

```

root@dlc75-155-dhcp rootj# ./server 5060
server is working
The received message is INVITE sip:doug@128.138.75.155 SIP/2.0
Via: SIP/2.0/UDP 127.0.0.1:5060;branch=z9hG4bK764867783
From: <sip:muda@128.138.75.188>;tag=3905387453
To: <sip:doug@128.138.75.155>
Call-ID: 3620384630@127.0.0.1
CSeq: 20 INVITE
Contact: <sip:muda@127.0.0.1>
max-forwards: 10
user-agent: oSIP/Linphone-0.10.2
Content-Type: application/sdp
Content-Length: 240

```

```

v=0
o=muda 123456 654321 IN IP4 127.0.0.1
s=A conversation
c=IN IP4 127.0.0.1
t=0 0
m=audio 7078 RTP/AVP 115 110 101
b=AS:110 8
a=rtpmap:115 1015/8000/1
a=rtpmap:110 speex/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-11

```

```

The reply is SIP/2.0 428 Reassertion
Via: SIP/2.0/UDP 127.0.0.1:5060;branch=z9hG4bK764867783
From: <sip:muda@128.138.75.188>;tag=3905387453
To: <sip:doug@128.138.75.155>
Attributeassertion: 762299093
Authorizationassertion: 1271565896
Decisionassertion: 1773134790
Authenticationassertion: 1196713515
Call-ID: 3620384630@127.0.0.1
CSeq: 20 INVITE
Contact: <sip:muda@127.0.0.1>
max-forwards: 10
user-agent: oSIP/Linphone-0.10.2
Content-Type: application/sdp
Content-Length: 240

```

```

v=0
o=muda 123456 654321 IN IP4 127.0.0.1
s=A conversation
c=IN IP4 127.0.0.1
t=0 0

```

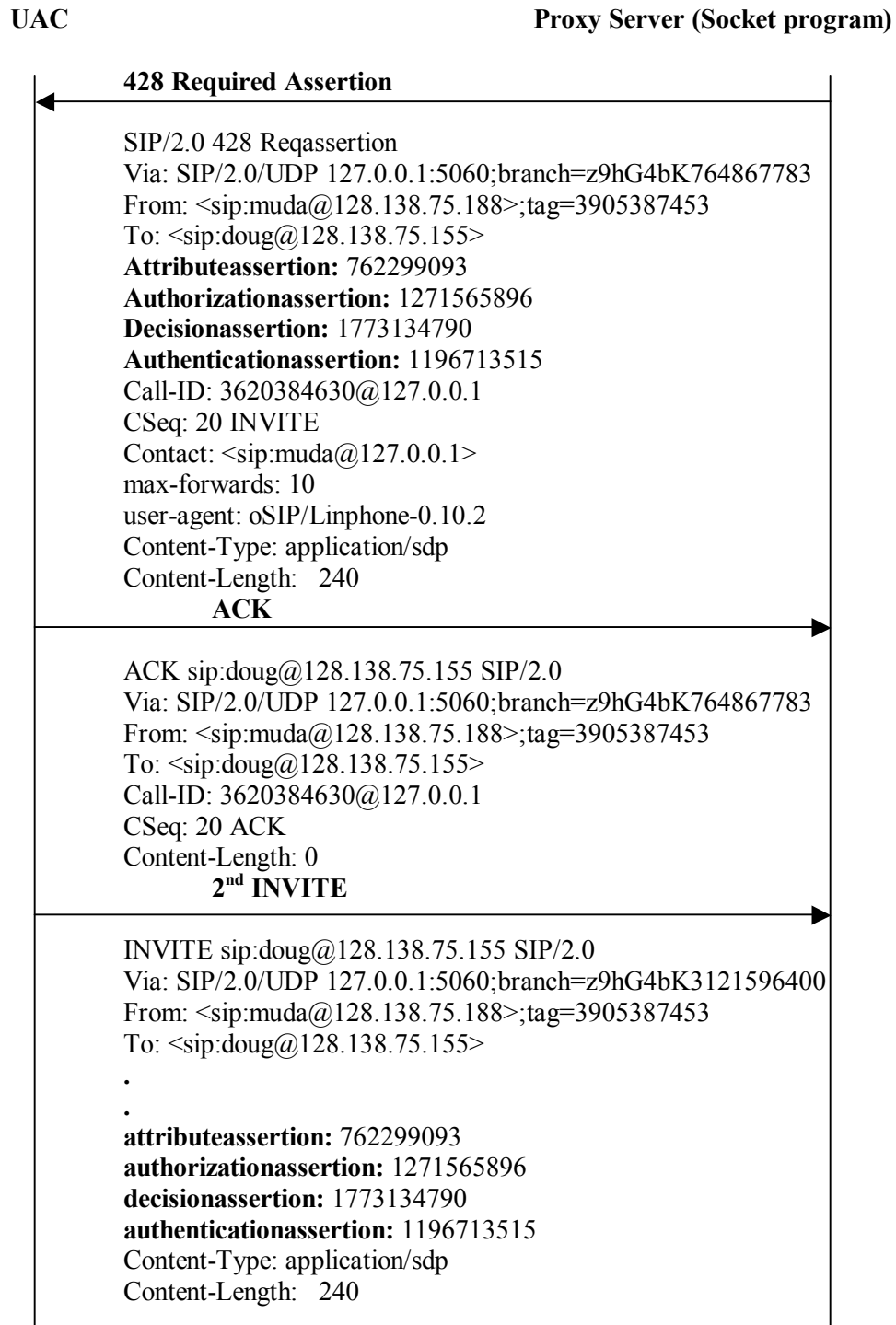
m=audio 7078 RTP/AVP 115 110 101  
 b=AS:110 8  
 a=rtpmap:115 1015/8000/1  
 a=rtpmap:110 speex/8000/1  
 a=rtpmap:101 telephone-event/8000  
 a=fmtp:101 0-11

The received message is ACK sip:doug@128.138.75.155 SIP/2.0  
 Via: SIP/2.0/UDP 127.0.0.1:5060;branch=z9hG4bK764867783  
 From: <sip:muda@128.138.75.188>;tag=3905387453  
 To: <sip:doug@128.138.75.155>  
 Call-ID: 3620384630@127.0.0.1  
 CSeq: 20 ACK  
 Content-Length: 0

The received message is INVITE sip:doug@128.138.75.155 SIP/2.0  
 Via: SIP/2.0/UDP 127.0.0.1:5060;branch=z9hG4bK3121596400  
 From: <sip:muda@128.138.75.188>;tag=3905387453  
 To: <sip:doug@128.138.75.155>  
 Call-ID: 3620384630@127.0.0.1  
 CSeq: 21 INVITE  
 Contact: <sip:muda@127.0.0.1>  
 max-forwards: 10  
 user-agent: oSIP/Linphone-0.10.2  
 attributeassertion: 762299093  
 authorizationassertion: 1271565896  
 decisionassertion: 1773134790  
 authenticationassertion: 1196713515  
 Content-Type: application/sdp  
 Content-Length: 240

v=0  
 o=muda 123456 654321 IN IP4 127.0.0.1  
 s=A conversation  
 c=IN IP4 127.0.0.1  
 t=0 0  
 m=audio 7078 RTP/AVP 115 110 101  
 b=AS:110 8  
 a=rtpmap:115 1015/8000/1  
 a=rtpmap:110 speex/8000/1  
 a=rtpmap:101 telephone-event/8000  
 a=fmtp:101 0-11

- **Illustration of the result**



**Figure 5.2: Captures of the response, ACK and second INVITE**

## 5.4 Processing time of the UA for the new headers

- **Objective**

The new headers cause minimal delay at the SIP UA processing level compared to a normal SIP message.

- **Procedure**

The Ethereal network analyzer output is used to analyze the time difference between the original INVITE and the second INVITE.

- **Result**

The time taken for the processing of the new headers is very small and is testament to the fact that the oSIP code is very light and fast. The table below, for which data was obtained from Ethereal output, corroborates the point.

	<b>Sample 1 (ms)</b>	<b>Sample 2 (ms)</b>	<b>Sample 3 (ms)</b>	<b>Sample 4 (ms)</b>	<b>Sample 5 (ms)</b>
<b>Req 1 (R1)</b>	5.806	3.976	3.062	3.663	4.203
<b>Response</b>	5.809	3.977	3.063	3.665	4.204
<b>ACK</b>	5.819	3.988	3.075	3.675	4.214
<b>Req 2 (R2)</b>	5.820	3.989	3.076	3.676	4.215
<b>Delay (R2-R1)</b>	0.014	0.023	0.014	0.013	0.012

**Table 5.1: Calculating average delay in processing information**

The average delay or time lag between the original request and the second request from the above 5 samples can be calculated as 0.015 seconds.

## Chapter 6

### CONCLUSION

This chapter discusses the answer to the research question in the thesis by summarizing the results and analyzing them. It also presents the future work that needs to be done to realize the complete authorization model.

#### 6.1 Summary of work

The research performed can be summarized into protocol development in the form of devising the architecture and call flows of an authorization model for SIP sessions. Since the authorization model required certain change in the behavior of the SIP client, the changes were implemented in the form of additional code and were tested by ensuring workability and adaptability of the SIP client to a newly defined response message.

The thesis answers the research question “Can modifications to the Session Initial Protocol (SIP) User Agent (UA) to carry additional authorization parameters in SIP messages be implemented?” It successfully implements a SIP client that can respond to a new response message devised, the 428 “Required Assertion”. This response indicates lack of authorization information in the initial request sent by the client. The client behavior to this response has been defined and implemented. This involves the attaching of four new headers sent in the response message to a new INVITE message. Each of the four headers and the response have been defined along with a summary of use indicating the behavior of the proxy servers when they receive the headers. These headers can be used to carry the values needed by the proxy servers for authorizing a UA to leave or enter a domain. They can also carry references to where the UA attributes are

stored. The experimental results confirm that the client performs the function of parsing the new incoming response message and attaching the resent request with the new headers it captures from the response. The results prove that there is minimal processing time required by the client for these new headers. They also demonstrate the increase in time for the processing of the extra headers is about 0.015 seconds.

The results above are obtained using a socket program that simulates a proxy server and will require interfacing with an actual proxy server that can generate the required response (428 Required assertion) when it receives an original INVITE without the four required headers. There is also the assumption based on the facts discussed in section 3.6.2 that the values returned to the proxy or processed by the proxy and returned to the client will be in the form of character strings. That has been the implementation in the SIP client. The fact that the Linux platform will provide the best opportunity to perform code changes led us to choose the Linphone as a SIP application to implement and test these changes. However, for a more accepted client in use we need to have Windows compatibility.

## **6.2 Future Work**

The work performed here is only the client side development required in the authorization model. The greatest challenge will be integrating all the components of the SIP network model to work successfully in the authorization model. Apart from the compatibility of the network elements future work could also include optimizing the call flows to improve the performance by optimizing the round-trip times for the SIP session.

## BIBLIOGRAPHY

- [1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, "SIP: Session Initiation Protocol", Network Working Group, RFC 3261, June 2002.
- [2] ITU-T Recommendations Q.700-775, Signaling System No. 7
- [3] Stewart, R., Xie, X., "Stream Control Transmission Protocol, A Reference Guide"
- [4] RFC 3015 - Megaco Protocol Version 1.0
- [5] ITU-T Recommendation H.323v.4 "Packet-based multimedia communications systems", November 2000.
- [6] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1", RFC 2616, June 1999
- [7] RFC 821, Jonathan B. Postel, August 1982, Simple Mail Transfer Protocol
- [8] Shibboleth Project, <http://shibboleth.internet2.edu>.
- [9] "Security Assertion Markup Language (SAML)", OASIS, <http://xml.coverpages.org/saml.html>
- [10] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1", RFC 2616, June 1999
- [11] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", Network Working Group, RFC 2069, June 1999.
- [12] B. Ramsdell, "S/MIME Version 3 Message Specification", RFC 2633, June 1999.
- [13] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", RFC 3268, June 2002.
- [14] <http://middleware.internet2.edu/video/draftdocs/draft-vandenberg-vidmid-dir-scenarios-00.html>
- [15] <http://www.videnet.gatech.edu/cookbook/appdx.html>
- [16] <http://middleware.internet2.edu/video/docs/johnson-dirs-for-multimedia-tel-00.pdf>
- [17] "Videoconferencing Security Vulnerabilities", Technical Services Discussion, [http://www.navastream.com/WhitePapers/VC\\_Vulnerabilities.pdf](http://www.navastream.com/WhitePapers/VC_Vulnerabilities.pdf)

- [18] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC1889, January 1996.
- [19] Cuervo, F., Greene, N., Rayhan, A., Huitema, C., Rosen, B. and J. Segers, "Megaco Protocol Version 1.0", [RFC 3015](#), November 2000.
- [20] "SIP Introduction", Jan Janak,  
[http://www.iptel.org/ser/doc/sip\\_intro/sip\\_introduction.pdf](http://www.iptel.org/ser/doc/sip_intro/sip_introduction.pdf)
- [21] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.
- [22] The 'application/xhtml+xml' Media Type, Request for Comments: 3236, M. Baker, P. Stark, January 2002
- [23] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997
- [24] <http://www.iptel.org/info/trends/sip.html>
- [25] <http://www.zvon.org/tmRFC/RFC2069/Output/chapter2.html>
- [26] J. Arkko, V. Torvinen, G. Camarillo, A. Niemi, T. Haukka, "Security Mechanism Agreement for the Session Initiation Protocol (SIP)", Request for Comments: 3329, January 2003
- [27] S. Olson, "A Mechanism for Content Indirection in Session Initiation Protocol", Internet-Draft, SIP WG, June 2, 2003
- [28] Wahl, M., Howes, T., and S. Kille, "Lightweight Directory Access Protocol (v3)", [RFC 2251](#), December 1997.
- [29] <http://www.gnu.org/software/osip/doc/osip-0.7.x.html>
- [30] "Linphone user manual", <http://www.linphone.org/doc/us/manual/index.html>
- [31] Vocal website, [www.vovida.org/applications/downloads/vocal](http://www.vovida.org/applications/downloads/vocal)
- [32] "SIP: Protocol Overview", <http://www.radvision.com/NR/rdonlyres/0F6C2E81-72B3-4C42-B33C-DDBC9115AC3D/234/SIPProtocolOverview.pdf>
- [33] Peterson, J., "Trait-based Authorization Requirements for the Session Initiation Protocol (SIP)", SIPING WG, Internet-Draft, March 2004.
- [34] <http://www.simc-inc.org/archive9899/Feb16-1999/dascom/Default.htm>
- [35] <http://www.ietf.org/html.charters/sigtran-charter.html>
- [36] <http://itd.colorado.edu/iapps/itd/>

## APPENDIX A

### Cookbook for Linphone

This appendix aims to make life easier when trying to download and install the Linphone client and making sure that the kernel packages in Linux are present for successful compilation of the Linphone packages.

#### A.1 Implementation steps for linphone

- 1) Install the Red Hat Linux to run the gnome environment and the gnome development packages for Linphone changes to the code.
- 2) Install and configure libosip libraries compatible with the version of linphone being used. Compatibility is present on the linphone website or can be inferred from the download sites which contain the appropriate libosip library versions required for the linphone version being downloaded.
- 3) Unzip and untar the libosip library downloaded from <http://ftp.gnu.org/gnu/osip/> into the usr/sip folder. Go to the usr/sip sub-directory for unpacking the files.
- 4) Go to the new subdirectory that will normally have the name of the package installed. E.g. usr/sip/libosip.
- 5) Run the “./configure” command to configure the package. Make sure to have a look at the README file to add certain options along with the command in order to configure for different requirements and specifications.

- 6) Run “make; make install” to complete installation for libosip.
  
- 7) Similarly, run the commands 3 to 6 for the linphone package available at <http://simon.morlat.free.fr/download/>
  
- 8) Download the alsa sound drivers for the sound card from [www.alsa-project.org](http://www.alsa-project.org) for successful voice communication using the Linphone client.
  
- 9) The step 8 might require changes to the /etc/module.conf file or this step can be subverted by using the “alsaconfigure” download available on the same download site as the driver download.

## **APPENDIX B**

### **Methods and Materials**

This chapter is an indicator to the determinants towards the choice of the Operating System and the open source SIP software package. The chapter then goes on to describe the architecture and design of the Libosip libraries and the use of Linphone [30], which is an application using the API's of the libosip libraries.

#### **B.1 Choice of the Operating System**

The development flexibility and the availability of all open source SIP libraries being present in source code primarily for Linux versions made Linux the most obvious choice for the operating system. Also supporting the choice was the ease with which the source code could be modified, the presence of documentation for source files that could be run on Linux and the fact that Linphone, the application using the API's in the libosip libraries was compatible only with Linux\*.

#### **B.2 Choice of the Software Libraries**

Vocal [31] and oSIP [29] are two popular open source libraries available as SIP stacks that were considered for the research. The library chosen was oSIP because of its features and compatibility with the research work. The oSIP library has been described at length later in this chapter.

---

\* Latest versions of Linphone have been made BSD compliant though no tests have been performed.

### B.2.1 oSIP - Open SIP

The oSip stack is a SIP library and a SIP stack, which is available as open software [29] under the GPL license. It provides an API for creating, parsing and modifying SIP and SDP messages. It is the initiative of Aymeric Mozard and was initiated in July 2000 [1]. It offers itself as a flexible and all encompassing software library, which can be used to build client, proxy and registrar applications.

The oSIP library can be broken up into three parts [29]-

- **The Parser** - for reading and writing the URIs, messages and the contents that define the session.
- **The Finite State Machines (FSM)** – the Transaction Layer in SIP, they use the parser for generating states and in turn are controlled by the UA.
- **The Transaction Manager** – the manager of the event or transaction queues at the finite state machines.

Apart from these three primary components, the library also uses the oSIP instance and two data contexts, the transaction context and the dialogue context. The oSIP instance is the storage for the parameters used by the SIP stack. The transaction context stores the state of the transaction whilst the dialogue context stores the state of the dialogue. The data contexts are stored in a FIFO (First In First Out) manner for better cohesion and synchronization. The FSM is not used to store any state information, only for providing the correct state that the UA should be taken to with the transaction context and event state it currently is in. This allows a single FSM to cater to several transactions at the same time since a single transaction does not take up the entire FSM.

Since there is no way for the FSM and the UA to communicate there is need for a method to communicate between them. The channel between the FSM and the UA for transferring these requests and responses is callbacks. These are stored in the oSIP instance and used by the FSM to reply to the UA about which state it needs to proceed to. Every time an oSIP instance is initiated, the UA stores its parameters in the instance.

### **B.3 Architecture of oSIP**

The OSIP libraries can be differentiated into the finite state machines, parser and the transaction manager. Each of them is further defined in the following sub-sections [29].

#### **B.3.1 Finite state machines**

The strength of the oSIP libraries lies in its 4 Finite State Machines (FSM). These machines mirror those defined in the SIP RFC 3261. Each oSIP instance consists of 4 FSM instances at a time. The FSM can be broadly differentiated according to the 2 types of transactions, INVITE and NON-INVITE. The basic reasoning behind having these 2 different kinds of transactions is that the INVITE transaction requires that it be followed by an ACK and hence need to be treated differently. The finite state machines are then differentiated depending upon whether they are initiated by the SIP client or SIP proxy server. Based upon the above classifications, the 4 FSM can now be classified as ICT (Invite Client Transaction), IST (Invite Server Transaction), NICT (Non Invite Client Transaction) and NIST (Non Invite Server Transaction).

The finite state machines are not taken up completely by a single event since they do not store the context or state of the running transaction. This allows them to be used by more than one transaction at a time. They use callbacks to talk to the UA and inform it of the next state to proceed to. They determine callbacks using the event in the transaction context queue and the callbacks issued by the UA earlier. The oSIP instance is then used by the FSM to access parameters needed.

### **B.3.2 The Parser**

The oSIP libraries use the parser to read and write the basic level of SIP messages such as Via, To, From, Contact, Cseq, Route, Record-Route, mime-version, Content-Type and Content-Length. For other headers, the values can be hard-coded and used in the finite state machine callbacks for any user-specific development.

The oSIP parser tries to assume the attributes of an “object-oriented C language” parser, and accordingly the oSIP objects use operations like clone, string representation, constructor and destructor. The parser is a byte-per-byte parser, which does not increment sequentially. Hence, it can parse the full message only and is unable to parse a message partially. This can slow down the process if only simple routing needs to be done. To do away with the byte-per-byte parsing, there is an option of lazy parsing that can be enabled when configuring the package.

### **B.3.3 The Transaction Manager**

The oSIP library has a transaction manager for taking care of the queues, which constitute every state machine. Transactions are used for storing the state of a single SIP

transaction. The transaction manager controls these transactions by using modules that it builds to send events from modules.

Building an application over oSIP will require building several modules including one to manage the transport. This additionally keeps oSIP independent of the transport layer. Also suggested is the construction for a module for timer management, which will control the execution of timers and decide the termination of the transaction contexts. There is already an API, which has been constructed with all the necessary modules using the oSIP libraries, Linphone.

#### **B.4 Call flow when establishing a session using the oSIP components**

When a user tries establishing a call using an API built on oSIP the call flow resembles the following:

- 1) Callback registration: The callbacks that need to be used by the API are registered with the oSIP instance.
- 2) Creation of the SIP request: The initial SIP request is created, which initializes the transaction context depending upon the type of transaction e.g. ICT, NIST etc and the request being made. This transaction context will store the state of the ongoing transaction and will be accessed by the FSM to determine the callbacks in the future.
- 3) Sending the request: A new event for the outgoing message is created which then gets added to the transaction queue. Any incidences in the future related to the transaction, like responses, error messages or timeouts generate events, which are then stored by the UA in the transaction event queue. The FSM is then informed

about the latest event in the queue, which will then force the FSM to determine a particular callback.

- 4) The FSM is a passive component and hence does not poll the transaction context queues for regular updates. Hence to inform the FSM about an event in the queue, the UA needs to keep polling and updating the FSM.
- 5) FSM then processes the event, starts the timers and finally generate callbacks to the UA.
- 6) UA sends the message according to the callback sent by the FSM, listens to arriving messages, adds the events to the transaction context and then initiates the FSM again.

## **B.5 Advantages of oSIP**

Principle features incorporated into oSIP are documented below-

- Lightweight - The greatest advantage of running oSIP is its lightweight nature- it requires very little memory and normal processing power, which allows it to run on small OS. It runs on embedded OS like VxWorks and Arm and is ideal for loading on mobile phones.
- True to the origin - It closely follows the RFC 3261, thus making it easy to interpret as well as providing the software with readymade documentation.
- Portable – it can be shipped to any platform, ranging from robust Unix and Solaris servers to embedded OS like Arm and VxWorks.

## B.6 Disadvantages of oSIP

There are some features missing in oSIP that need to be addressed. They are as enumerated below.

- Provides the user with a single API that can be developed into any of the SIP components ranging from clients to proxy servers, conference servers and registrar. It does not provide a user with specific API for each of these elements and hence it becomes laborious for a user to implement just a single SIP component.
- The SIP message is not built through a simple request or response, with it being done manually.
- There is no method of testing to check the composition of the SIP messages and the presence of the headers.

## B.7 Linphone

Linphone is free software, released under the GNU Public License, which is independently developed as a SIP application using the oSIP libraries [30]. Linphone is a comprehensive Linux based web-phone with several features. It has great troubleshooting features for developers like the ability to troubleshoot by just looking at the log generated on the command line, which illustrates the state and condition of each transaction that the videoconferencing session is passing through. This is possible because oSIP can output any SIP message or an URI as a string, which is then redirected to the terminal screen. Since Linphone runs on the oSIP library, there are no default callbacks present to speak between the UA and the FSM. However, Linphone uses the Invite Server Transaction (IST) and Non-Invite Server Transaction (NIST) for callbacks for transactions generated

by the server end. Invite Client Transactions (ICT) and Non-Invite Client Transactions (NICT) callbacks for transactions generated by the client. These callbacks have been well defined in Section 4.3.1.

### **B.7.1 Requirements for Linphone**

There are a few requirements for operating Linphone [30]:

- Linux is strongly preferred though tests have been performed on BSD and other Unix platforms.
- Gnome 1.2 needs to be installed, though not necessarily in a running state.
- Sound card
- Headphones or speakers
- Microphone

Other recommendation for running Linphone include closing any other application using the audio device as well as not using Linphone for confidential conversation since there are no security mechanisms like encryption performed on the audio stream. To be kept in mind is the fact that Linphone needs to be running in order to be able to receive calls.

### **B.7.2 Modes for Linphone**

There are three modes for operating Linphone [30]:

- A normal mode which allows us to use Linphone through the gnome menu or through command line
- As a gnome applet where the applet can be added to the gnome panel.

- As a silent daemon for non-gnome users, where the daemon can be added to the kde directory and will run in the terminal.

Sipomatic [30] is an added feature present in Linphone, which is a test program that can be used to answer call automatically for linphone. This allows a user to test whether the phone is working without having another test user or account set up.