Islamic University of Technology, IUT

Department of Computer Science and Engineering

Final Year Thesis
2012

Implementation of Encryption and Hash Function over SDP data in secured VoIP communication

Supervisor:

Abdullah Azfar
Assistant Professor, Dept. of CSE, IUT

Co-Supervisor:

Mahmudun Nabi
Lecturer, Dept. of CSE, IUT

Submitted By:

Md. Saleem Hasan **Student ID# 084432**
Istiak Ahmed **Student ID# 084435**

Submitted To:

Department of Computer Science and Engineering (CSE)
Islamic University of Technology (IUT)
October, 2012

Certification

This is to certify that, this thesis titled "Implementation of Encryption and Hash Function over SDP data in secured VoIP communication" is a true work of Saleem Hasan (084432) and Istiak Ahmed (084435) who successfully carried out the thesis work under the supervision of Mr. Abdullah Azfar, Assistant Professor of the Department of Computer Science and Engineering at Islamic University of Technology.

Authors:

……
Saleem Hasan
Id: 084432

……..
Istiak Ahmed
Id: 084435

Supervisor:

……..

Abdullah Azfar
Assistant Professor
Dept. of CSE, IUT

Co-Supervisor:

……..

Mahmudun Nabi
Lecturer
Dept. of CSE, IUT

Head of Department:

…….
Prof. Dr. M. A. Mottalib
Head of the Dept. of CSE, IUT.

Acknowledgement

We express our heartiest gratitude to Almighty Allah for His divine blessings, which made us possible to work on this thesis successfully.

First and foremost, we acknowledge our sincere gratitude to **Prof. Dr. M. A. Mottalib** (*Head, Department of Computer Science and Engineering, IUT*) for patronizing our work. We thank **Mr. Abdullah Azfar** (*Assistant professor, Department of Computer Science and Engineering, IUT*) and Mr**. Mahmudun Nabi** (*Lecturer, Department of Computer Science and Engineering, IUT*) for their heartiest support; their deep knowledge in the field of VoIP security influenced us to carry out this project up to this point.

Their endless patience, scholarly guidance, encouragement, valuable advice, energetic supervision and support have made this project flourish and earn a recognizable state.

We express our sincere gratitude to Islamic University of Technology (IUT) authority for providing us the necessary environment and support.

ABSTRACT

One of the most demanding aspects of the Internet now is voice communication. It can be achieved through Voice over Internet Protocol (VoIP). Although the data communication over VoIP is very reliable and secure, some of its underlying protocols are not. An example is the Session Initiation Protocol (SIP) and its underlying Session Description Protocol (SDP) which are responsible for establishing voice calls and describing the multimedia session respectively. Session Initiation Protocol (SIP) is a signaling protocol which creates a unicast and multicast session between communicating parties. This protocol does not enforce any security parameters. It uses the Session Description Protocol (SDP) to announce the session profile. The body of the SDP message is written in plaintext. Hence this exchange of information between the communicating devices is insecure and can easily be intercepted and exploited by a third party. To remedy this problem we intend in this work to create a hash of the data contained in the SDP body to ensuring data integrity. Furthermore, we intend to encrypt the information using Symmetric Key Encipherment to ensure that only the receiver can decrypt the message and implement Digital Signature to authenticate the sender.

Table of Contents

List of Figures

Abbreviations and Acronyms

DNS	Domain Name Server
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
LS	Location Services
MD5	Message Digest 5
PKI	Public Key Infrastructure
PSTN	Public Switched Telephone Network
PKI	Public Key Infrastructure
PKC	Public Key Cryptography
RFC	Request for Comments
RTP	Real Time Protocol
SIP	Session Initiation protocol
SDP	Session Description Protocol
SRTP	Secure Real Time Protocol
TLS	Transport Layer Security
TCP	Transmission Control Protocol
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
VoIP	Voice over Internet Protocol

Chapter 1: Introduction

Voice over Internet Protocol (VoIP) is one of the most popular networking protocols used nowadays. It works as telephone over the Internet. So the main objective of this protocol is to transmit voice signals between communicating devices over the Internet. VoIP protocol allows the communication to occur between two or more devices at a time. For its proper functioning VoIP have some underlying protocols of its own. For example, the Session Initiation Protocol (SIP) is an underlying protocol of the VoIP. The function of this protocol is to create a session. So, basically SIP is a signaling protocol and does not carry any actual data, rather it creates the session between two or more devices. SIP has its own underlying protocol known as the Session Description Protocol (SDP). This protocol does the function of announcing the parameters of the session or defining the session profile. The body of a SDP message is written in plain English. Now, SIP does not enforce any security measures in order to protect this SDP message from intruders. So if there is a "Man in the Middle" attack, the intruder will be able to eavesdrop the session by manipulating the SDP body. So, the created VoIP session won't be a secure one. Hence, some security measures is needed to be enforced to prevent this sort of attacks.

1.1 Motivation

VoIP is a popular scheme of communication. It is inexpensive, reliable and easy to use. Now a days people all over the world are using VoIP services for communication. VoIP protocol uses its underlying Session Initiation Protocol to initiate the sessions. And the data transmission while actual voice sessions are carried out using the Real Time Transport Protocol (RTP). The Real Time Transport Protocol is itself a secure one as it uses various security measures to protect data. But the Session Initiation part is not safe because SIP doesn't implement any security measure. So the initiation part is not safe. Hence, the entire sessions are prone to hijacking. Till now there hasn't been any significant work done to secure the session initiation part. In our work we implement a technique to enforce certain security measures to secure the session initiation part.

1.2 Common threats in VoIP

As we move from traditional telephony to VoIP, we face the inherent security issues of IP based systems. Prajwol Kumar Nakarmi, John Mattsson in their work [2] discussed about tools like Wireshark, which makes it easy to sniff and listen to the VoIP conversations if one connects to a suitable point in the network. Wireshark is an open source packet analyzer. It can be used to capture data packets from network traffic to manipulate the packets according to the intruder's will. Tools like Wireshark are easily available and can be used to implement "Man in the Middle" attack.

Anat Bremler-Barr, Ronit Halachmi-Bekel, Jussi Kangasharju in their work [4] present the unregister attack, a new kind of a denial of service attack on SIP servers. In this attack, the attacker sends a spoofed "unregister" message to a SIP server and cancels the registration of the victim at that server. This prevents the victim user from receiving any calls. We have tested common implementations of SIP servers and show that the un-register attack is easily performed on SIP servers which do not use authentication. Even on SIP servers with authentication, an attacker is able to sniff the traffic between the client and server can and still successfully attack common servers.

1.3 Objective of the Thesis

The main objective of this research is to enforce the security the current system is lacking. In order to do that, we have to find a way to encrypt the SDP data and digitally sign this data. So that if any intruder attacks the file, he/she doesn't understand the data and the user at the receiver end can also identify if the data is modified at its route.

So we have the following tasks to do:

- Ensure data integrity of SDP so that forgery of session can be detected.
- Encrypt the session data so that the data is not understandable to the intruder.
- Sign the data digitally so that the receiver can be sure of the originator of the message.

Chapter 2: Background Study

Eavesdropping refers to the phenomenon in which some unauthorized third party listens to a conversation illegally. In [1], Diego Pérez-Botero and Yezid Donoso evaluated the various cryptographic measures which are used to prevent eavesdropping in VoIP. In their work, they have discussed various countermeasure techniques like, Transport-Layer Security (TLS), Datagram Transport Layer Security (DTLS), IPSec., defined in RFC 4346, TLS provides an environment for mutual authentication (client and server), confidentiality, and integrity. One disadvantage of TLS is that it only offers confidentiality in a client-server relationship between two end points. This means that, for each signaling hop, a separate TLS connection has to be created. Thus, the lack of end-to-end confidentiality leaves SIPS vulnerable to rogue proxies. IPSec is part of the security model of IPv6. It offers protection for applications based on any transport layer protocol (i.e. UDP, TCP, SCTP, etc.) by operating in the network layer. Specifically, in a SIP environment, IPSec has the same disadvantages of TLS and DTLS when protecting the signaling path: separate secure tunnels have to be established for each hop. On the other hand Multimedia Internet Keying (MIKEY) lacks these disadvantages and is useful key generation protocol. That's why in our work in our work we won't use TLS, DTLS or IPSec.

Prajwol Kumar Nakarmi, John Mattsson and Gerald Q. Maguire Jr.in their research [2] defined IMS a generic architecture for offering multimedia services over Internet Protocol (IP). IMS is access agnostic and therefore promotes network consolidation. In an IMS domain, each device is assigned exactly one IP Multimedia Private Identity (IMPI) and one or more IP Multimedia Public Identity (IMPU). The IMPI is used for authentication, accounting, and so on. The IMPU, on the other hand is used in communications with other users. Since IMS supports both native IP based services as well as voice services over IP, it offers network operators a significant reduction in the cost of operating their networks. Moreover, since IMS uses the same protocols as the Internet, it allows developers to create applications using existing Internet protocols. It is important to note that IMS is *not* a service in itself, but rather it is a service enabler and it integrates different services (e.g. multimedia, instant messaging, presence, etc.).

2.1 Related Works

Registration Hijacking occurs when an attacker impersonates a valid UA to a registrar and replaces the legitimate registration with its own address [5]. This causes all incoming calls to be redirected to the UA registered by the attacker.

Previously, A.F.M. Sayem, Sheikh Mustaq Ahmmed and Md. Shihab Karim in their work [5] implemented SHA-256 hash algorithm on the SDP (Session Description Protocol) packets in order to secure the SIP signaling. But they only implemented the hash at the senders site and further decryption at the receiver site was not was not developed. Further, there was no performance analysis to evaluate the proposed model.

2.2 Voice over Internet protocol (VoIP)

Voice over IP (VoIP) commonly refers to the communication protocols, technologies, methodologies, and transmission techniques involved in the delivery of voice communications and multimedia sessions over Internet Protocol (IP) networks, such as the Internet [8]. As a technology it is invading enterprise, educational and government organizations. Due to its useful features, it is gaining immense popularity day by day. From a technical point of view VoIP can dramatically improve the bandwidth efficiency by exploiting advanced voice coding and compression techniques. For a connection to create VoIP uses the Session Initiation Protocol (SIP) to initiate the connection and uses Real Time Transport Protocol (RTP) to transmit the actual data during the VoIP session.

2.3 Session Initiation Protocol (SIP)

Session Initiation Protocol (SIP) is a protocol which is used to establish voice calls between two or more parties over the internet. It is an application layer protocol which is widely used for initiating and controlling communication sessions such as voice or video calls over Internet Protocol (IP). In plain words we can say SIP is the protocol used to initiate a particular session. It works with both IPv4 and IPv6. The protocol can be used for creating, modifying and

terminating two-party (unicast) or multiparty (multicast) sessions. It is only a signaling protocol and cannot be used to carry actual data.

SIP works in concert with several other protocols and is only involved in the signaling portion of a communication session. SIP clients typically use TCP or UDP to connect to SIP servers and other SIP endpoints. SIP is primarily used in setting up and tearing down voice or video calls. It also allows modification of existing calls. The modification can involve changing addresses or ports, inviting more participants, and adding or deleting media streams. The voice and video stream communications in SIP applications are carried over another application protocol, the Real-time Transport Protocol (RTP). Parameters such as: port numbers, protocols, codecs for these media streams are defined and negotiated using the Session Description Protocol (SDP) which is transported in the SIP packet body.[1][3][4]

2.4 Session Description Protocol (SDP)

The Session Description Protocol (SDP) is a format for describing streaming media initialization parameters. SDP is intended for describing multimedia communication sessions for the purposes of session announcement, session invitation, and parameter negotiation. SDP does not deliver media itself but is used for negotiation between end points of media type, format, and all associated properties. The set of properties and parameters are often called a session profile. SDP contains vulnerable information about the session in plain text which can be exploited by an Intruder. A session is described by a series of fields; one per line. The form of each field is as follows:

```
<character>=<value>
```

Within an SDP message there are three main sections, detailing the *session*, *timing*, and *media* descriptions. Each message may contain multiple *timing* and *media* descriptions.

```
Session description
        v=  (protocol version)
        o=  (originator and session identifier)
        s=  (session name)
        i=* (session information)
        u=* (URI of description)
        e=* (email address)
        p=* (phone number)
        c=* (connection information—not required if included in all media)
        b=* (zero or more bandwidth information lines)
        One or more time descriptions ("t=" and "r=" lines; see below)
        z=* (time zone adjustments)
        k=* (encryption key)
        a=* (zero or more session attribute lines)
        Zero or more media descriptions
```

```
Time description
        t=  (time the session is active)
        r=* (zero or more repeat times)
```

```
Media description, if present
        m=  (media name and transport address)
        i=* (media title)
        c=* (connection information—optional if included at
             session level)
        b=* (zero or more bandwidth information lines)
        k=* (encryption key)
        a=* (zero or more media attribute lines)
```

Figure-2-1: Parameters used in the SDP message to announce the session profile.

Several important pieces of information are mandatory within an SDP message:

- Session name.
- Time(s) the session is active.
- The media comprising the session.

- The owner/originator of the session.
- How to receive the media (addresses, ports, Formats etc).

An SDP session description includes the following media information:

- The type of media (video, audio, etc.)
- The transport protocol (RTP/UDP/IP, H.320, etc.)
- The format of the media (H.261 video, MPEG video, etc.)

Other optional information may be provided:

- Bandwidth to be used by the conference.
- The purpose of the session.
- Contact information for the person responsible for the session.
- Time zone information.
- Session attributes extending SDP

From the above discussion, we can see that SDP defines the properties and description of the originated session. SDP contains vulnerable information about the session in plain text which can be exploited by an Intruder.

And the Session Initiation Protocol (SIP) does not enforce any measure to protect this information.

So, if we summarize the entire process of data transmission, a session is initiated using SIP and session's parameters defined by using SDP is transmitted using SIP. No security measures are enforced for the protection of the SDP contents. So, at any instant of transmission an intruder can access these SDP contents and pollute it. Hence, we can conclude that the overall security of the session is at stake.

2.5 MIKEY

MIKey (Multimedia Internet Keying) is a method to exchange the security keying materials between the clients during the initiation of a call. It is a very efficient method of key exchange as it was designed to meet the requirements of initiation of secure multimedia sessions [9]. The requirements being the following

- The exchange of the parameters should be done in one round trip
- It should allow the transport in a secure establishment protocol such as SDP
- The protocol should supply end to end security for the keying material.
- It should consume low bandwidth and the computational workload should be low.

This making MIKey one of the most used mode of key exchange when dealing with Multimedia data.

MIKey supports three types of key agreements [10]:

1. Pre-shared Key : In this method a secret key which has been pre-shared between the communication parties is used to derive the keys for both the encryption and integrity protection of the MIKey message. Although this is the most cost efficient method, the problem of distributing the shared secret key makes it hard to implement.
2. Public Key Encryption: In this method a random key is generated for both encryption and integrity of the data, which is then, encrypted using the responder's public key and sent back to the responder. This method is more resource consuming but does not have the problems of distributing a shared key and has the ability to scale if a Public Key Infrastructure (PKI) is available.
3. Diffie-Hellman : This method uses a key generator to generate a fixed agreed upon length DH-key. Although this method supplies a perfect forward secrecy, it is resource consuming and cannot be used to create group keys.

2.6 MiniSIP

Minisip is a SIP (Session Initiation Protocol) user agent which can be used for make phone calls, instant messaging, and video calls to users who are connected to the same SIP network [6]. It was developed by the PhD and the Masters students of the Royal Institute of Technology (KTH, Stockholm, Sweden) and the non-KTH volunteer developers. Minisip is open-source software that runs on a Linux and windows (XP) platform and also on smart phones [2]. It is a SIP compliant program that supports multiple users on the same line. It can handle multiple voice and video calls simultaneously. It has very secured method of data transfer, using SIP(Session Initiation Protocol) as the initiation protocol and the key exchange being done using MIKey(Multimedia Internet Keying) and the data being transferred using SRTP(Secured Real-Time Transfer Protocol. Minisip is also very up-to-date since it is open-source. People from around the world can download the source code from its website (www.minisip.org) and edit it to perform their desired operations.

2.7 Several Security Techniques

Since we are clear about our objective now, we now have to identify the way to accomplish our objective. In order to accomplish our goal we will need to apply some cryptographic techniques. In this section we will discuss about some of the useful methods implemented nowadays.

2.7.1 Symmetric Key Encipherment

The concept behind public key cryptography is to provide the each of the users of the session with a pair of keys. One is a Private Key and the other is a Public key. Each user will have his own private key; which is known only to the user himself, and also a public key which is known to all the other users. The public key will be used to encrypt the message and the private key will be used to decrypt the message.

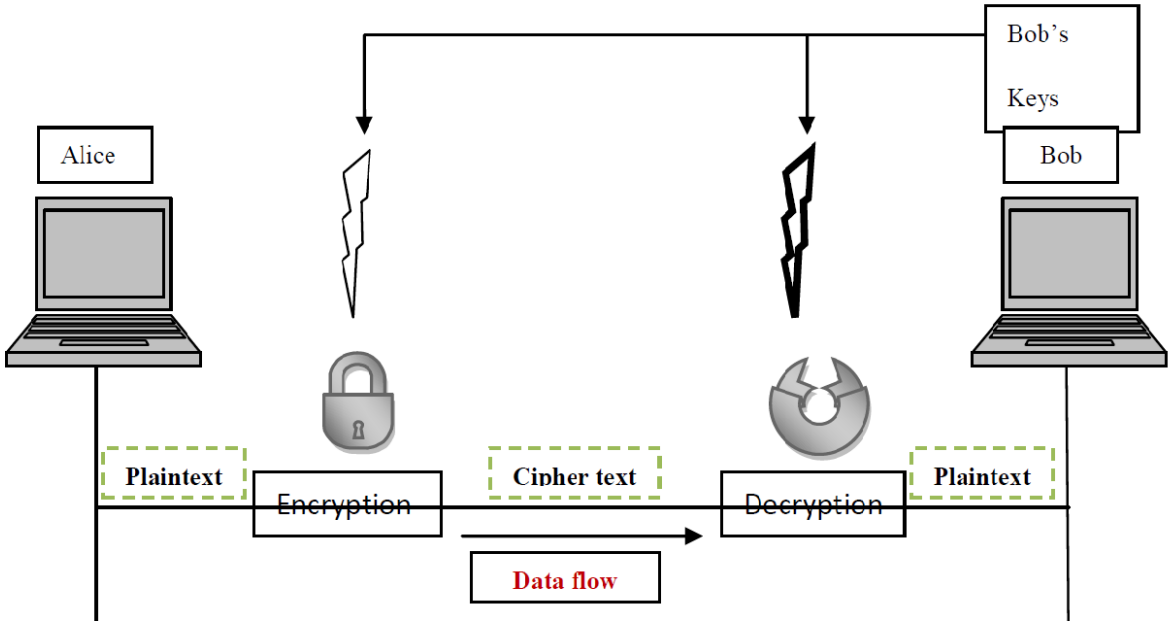

Figure-2-2: The mechanism followed in Public Key Cryptography

Suppose, Alice and Bob are two users. Alice wants to send a message to Bob. So, Alice will encrypt the message using Bob's public key and then send it to Bob. Bob will receive an encrypted message. So he will decrypt the message using his private key. Again, if Alice wanted to send the same message to Sam, then she would have to encrypt the message using Sam's public key; not Bob's public key.

The public key algorithms operate on sufficiently large numbers to make the reverse operation practically impossible. For example, RSA algorithm operates on large numbers of thousands of bits long.

2.7.2 Digital Signature

Using Digital Signature any message can be signed by the sender using his private key. At the receiving end, using the sender's public key the receiver can verify whether the received

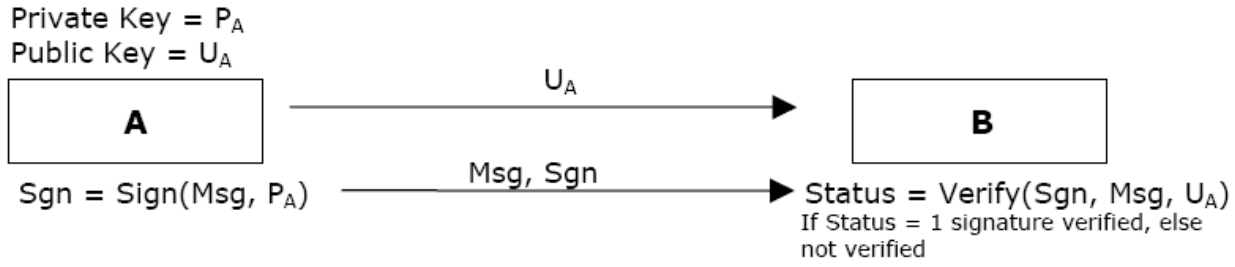

Figure-2-3: The mechanism of implementing Digital Signature

Message was signed by the intended sender. So if the message is modified by any intruder during transmission, the signature verification will fail and thus the receiver will know about the attack.

The concept of Digital Signature can be best described by the following scenario:

Four friends Bob, Pat, Doug and Susan work in an office. For security purpose, they digitally sign their messages before sending to the receiver.

Bob has been given two keys. One of Bob's keys is called a Public Key; the other is called a Private Key. Bob's Public key is available to anyone who needs it, but he keeps his Private Key to himself. Keys are used to encrypt information. Encrypting information means "scrambling it up", so that only a person with the appropriate key can make it readable again. One of Bob's two keys can encrypt data, and the other key can decrypt that data.

If Pat wants to send a message to Bob, he can encrypt the message using Bob's Public Key. Bob uses his Private Key to decrypt the message. Any of Bob's coworkers might have access to the message Pat encrypted, but without Bob's Private Key, the data is worthless.

With his private key and the right software, Bob can put digital signatures on documents and other data before sending it to others. A digital signature is a "stamp" Bob places on the data which is unique to Bob, and is very difficult to forge. In addition, the signature assures that any changes made to the data that has been signed can not go undetected. To sign a document, Bob's software will crunch down the data into just a few lines by a process called "hashing". These few lines are called a message digest. (It is not possible to change a message digest back into the original data from which it was created.)

Bob's software then encrypts the message digest with his private key. The result is the digital signature. Finally, Bob's software appends the digital signature to document. All of the data that was hashed has been signed.

At the receiving site, first, Pat's software decrypts the signature (using Bob's public key) changing it back into a message digest. If this worked, then it proves that Bob signed the document, because only Bob has his private key. Pat's software then hashes the document data into a message digest. If the message digest is the same as the message digest created when the signature was decrypted, then Pat knows that the signed data has not been changed.

Now, suppose, Doug (a disgruntled employee) wishes to deceive Pat. Doug makes sure that Pat receives a signed message and a public key that appears to belong to Bob. Unbeknownst to Pat,

Doug deceitfully sent a key pair he created using Bob's name. Short of receiving Bob's public key from him in person, how can Pat be sure that Bob's public key is authentic?

It just so happens that Susan works at the company's certificate authority center (Trusted third party). Susan can create a digital certificate for Bob simply by signing Bob's public key as well as some information about Bob.

Now Bob's co-workers can check Bob's trusted certificate to make sure that his public key truly belongs to him. In fact, no one at Bob's company accepts a signature for which there does not exist a certificate generated by Susan. This gives Susan the power to revoke signatures if private keys are compromised, or no longer needed. There are even more widely accepted certificate authorities that certify Susan.

Let's say that Bob sends a signed document to Pat. To verify the signature on the document, Pat's software first uses Susan's (the certificate authority's) public key to check the signature on Bob's certificate. Successful de-encryption of the certificate proves that Susan created it. After the certificate is de-encrypted, Pat's software can check if Bob is in good standing with the certificate authority and that all of the certificate information concerning Bob's identity has not been altered.

Pat's software then takes Bob's public key from the certificate and uses it to check Bob's signature. If Bob's public key de-encrypts the signature successfully, then Pat is assured that the signature was created using Bob's private key, for Susan has certified the matching public key. And of course, if the signature is valid, then we know that Doug didn't try to change the signed content.

2.7.3 Hash Function:

A hash function H is a transformation that takes a variable-size input m and returns a fixed-size string, which is called the hash value h.

So, h = H(m).

Properties of Hashing:

- Hashing is always one-way. That we can create a digest from a message using the hash function, but cannot reverse back to the message from the digest.
- Hashing is a one-to-one function. No two messages can have the same digest.

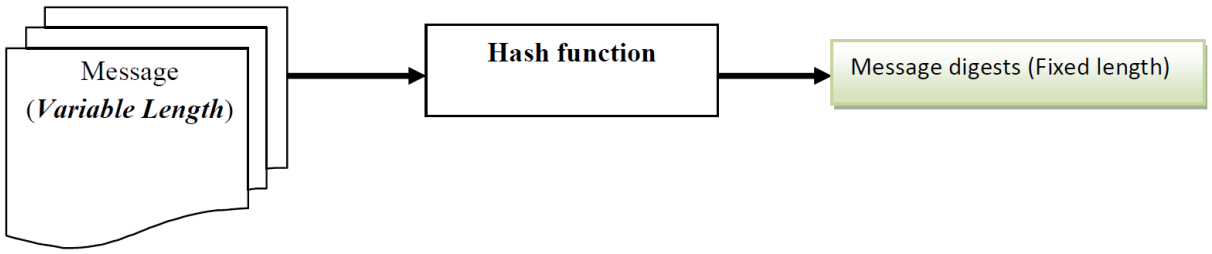

Figure-2-4: Mechanism of Hash Function

Benefits of using the Hash Function:

The main role of cryptographic Hash function is in the provision of Digital Signatures. Since hash functions are generally faster than digital signature algorithms, it is typical to compute the digital signature to some document by computing the signature on the document's hash value, which is smaller compared to the document itself.

Additionally, a digest can be made public without revealing the contents of the document from which it is derived. This is important in digital time stamping where, using hash functions, one can get a document time stamped without revealing its contents to the time stamping service.

2.7.4 Secured Hash Algorithm (SHA):

There are five approved algorithms for generating a message digest. They are:

(1) SHA-1 (2) SHA-224 (3) SHA-256 (4) SHA-384 (5) SHA-512

The following figure justifies our choice of hash algorithm from the SHA-2 family.

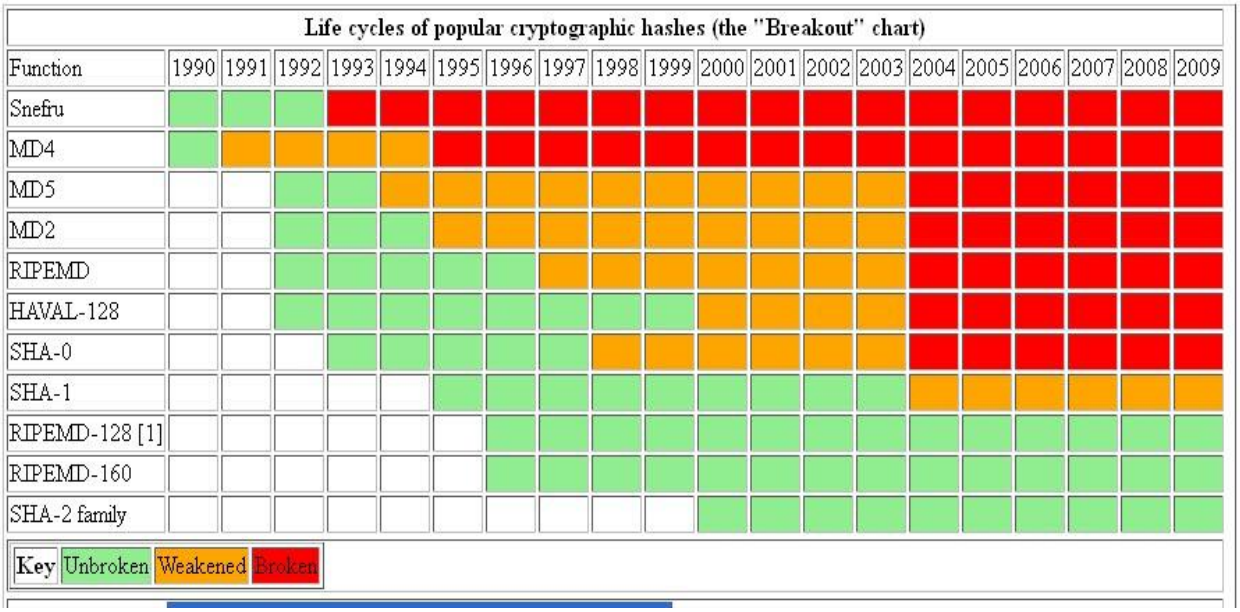

Figure-2-5: Performance of various Encryption Methods over the years.

Operation of SHA-512:

The SHA-512 algorithm is used to authenticate Debian Linux software packages. UNIX and Linux vendors are moving to using SHA-256 for secure password hashing. SHA-512 is used to ensure the integrity of the data. It creates a 512 bit digest for data of up to $2^{64'}$ bits. The data along with the digest is sent to the recipient who uses the same algorithm to recreate the digest for the data received and matches the result with the digest received. SHA-512 uses six logical functions, where each function operates on 32-bit words. The result of each function is a new 32-bit word. Padding is added to the data to make its length a multiple of 512. The padded data is

the parsed into blocks of 512 bit each, and each block is expressed as sixteen 32-bit words. Then the initial hash values are set which consists of eight 32-bit hexadecimal words. Using the sixty four 32-bit data words, eight working variables of 32-bit each and a hash values of eight 32-bit words, SHA-512 runs the algorithm to produce a 512-bit message digest.

Chapter 3: Problem Statement

3.1 Overview

The data contained in the SDP body is the main concern of this work as it gives out critical information about the communicating parties.

o = <username> <session id> <version> <network type> <address type> <address>

The “o” field describes the originator of the session, a session id and a session version number [5]. As we mentioned earlier these data are vulnerable to attacks since they are not encrypted.

Various kinds of attacks can be formulated using the above data. Figure-3.1 shows the procedure of typical Man-in-the-Middle attack.

Registration Hijacking:

Registration Hijacking is the process where an attacker impersonates a valid user agent (one of the communicating parties) to a registrar and replaces the legitimate registration with its own address. This causes all incoming calls to be sent to the user agent registered by the attacker.

Registration is normally performed using UDP, which makes it easier for the intruder to spoof requests. SDP doesn’t involve authentication and even if it does its often weak. Most registrar server does not challenge requests or only require user name and passwords, which can be defeated with traditional dictionary-style attacks.

RTP Data redirecting:

In RTP data redirection, the SDP header is modified to redirect the caller to another destination. For example, the packet is sniffed out using sniffing software that can be then used to modify and redirect the data to another user or may even be used to terminate the session.

Eavesdropping:

Like data packets, voice packets are subject to man-in-the-middle attacks where a hacker spoofs the MAC address of two parties, and forces VoIP packets to flow through the hacker's system. By doing so, the hacker can then reassemble voice packets and literally listen in to real-time conversations [11]. From this type of attack, hackers can also purloin all sorts of sensitive data and information, such as user names, passwords, and VoIP system information.

Denial of Service (DoS):

Similar to DoS attacks on data networks, VoIP DoS attacks leverage the same tactic of running multiple packet streams, such as call requests and registrations, to the point where VoIP services fail [11]. These types of attack often target SIP (Session Initiation Protocol) extensions that ultimately exhaust VoIP server resources, which cause busy signals or disconnects.

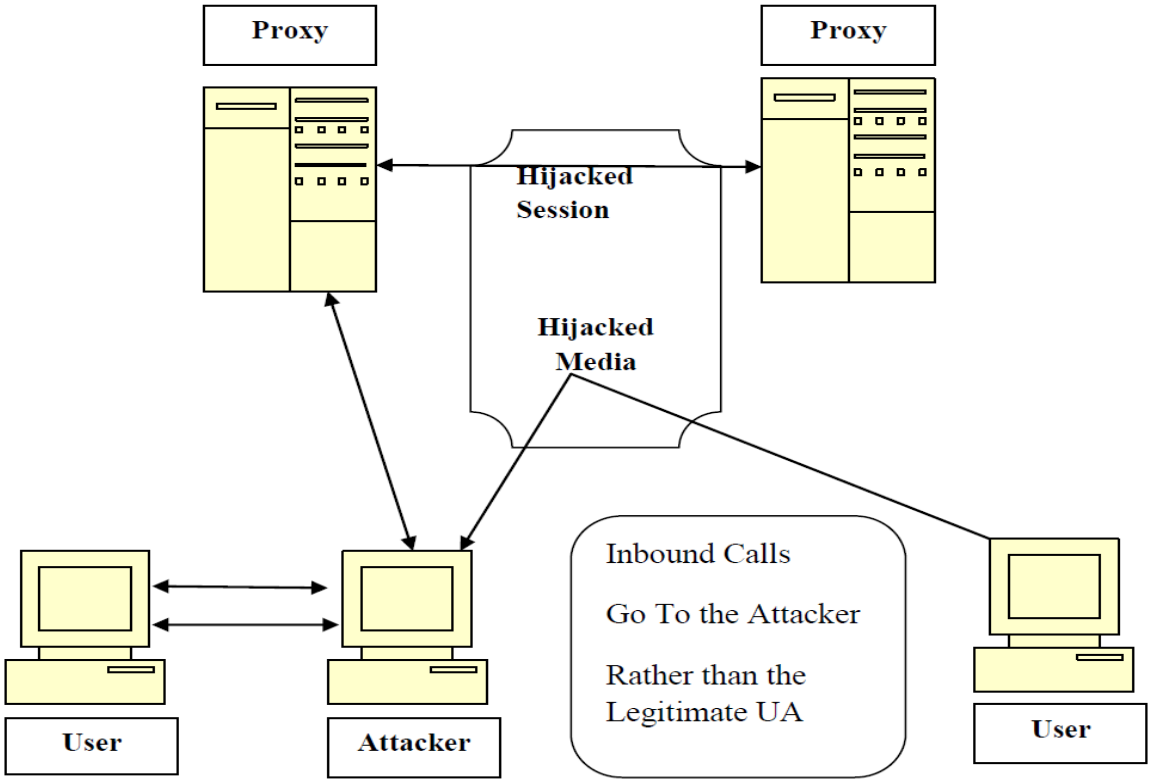

Figure-3-1: Man-in-the-Middle attack in SDP

3.2 Problems to be addressed:

- Ensure data integrity of SDP so that forgery of the session can be detected.
- Generate a hash of the data and append it to the original data.
- Encrypt the entire data so that the data is not understandable.
- Sign the encrypted data digitally so that the receiver can be sure of the originator of the message.

Chapter 4: Design and Implementation

In our design, we tried to implement the SHA-512 algorithm into Minisip to ensure data integrity. The SHA-512 hash function takes the data or message as a parameter and generates a fixed length hash for it. Then the original data and the hashed data is integrated in a field and sent together.

The reason of choosing Minisip as the client's softphone is because it is an open source soft phone client rich with security facilities. It also has a major advantage over other open source soft phones because of the massive security related development and research work it encompasses. And we have used asterisk as server to establish pc to pc soft phone network and analyzed the data. When we traced the transmitted packets using Wireshark, we found the SDP data unencrypted.

So we will use SHA-512 to create a digest of the data (session parameters). Hashing ensures the data integrity. Since hash values are irreversible, if any modification or tampering is done with the session parameters, then the hash value won't match and the receiver will know about the attack. Then we append the hash value to the original data.

The following figures explain the encryption technique:

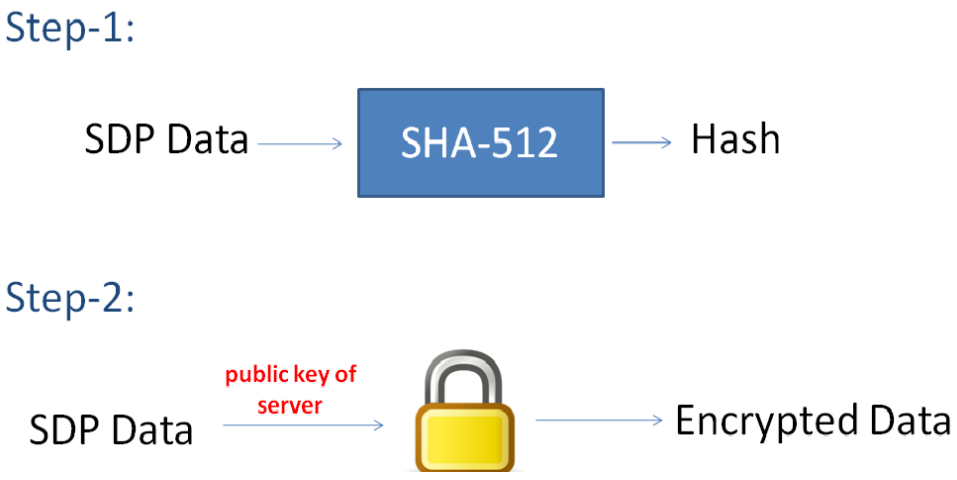

Figure 4-1: Generation of hash from SDP data and encryption of SDP data.

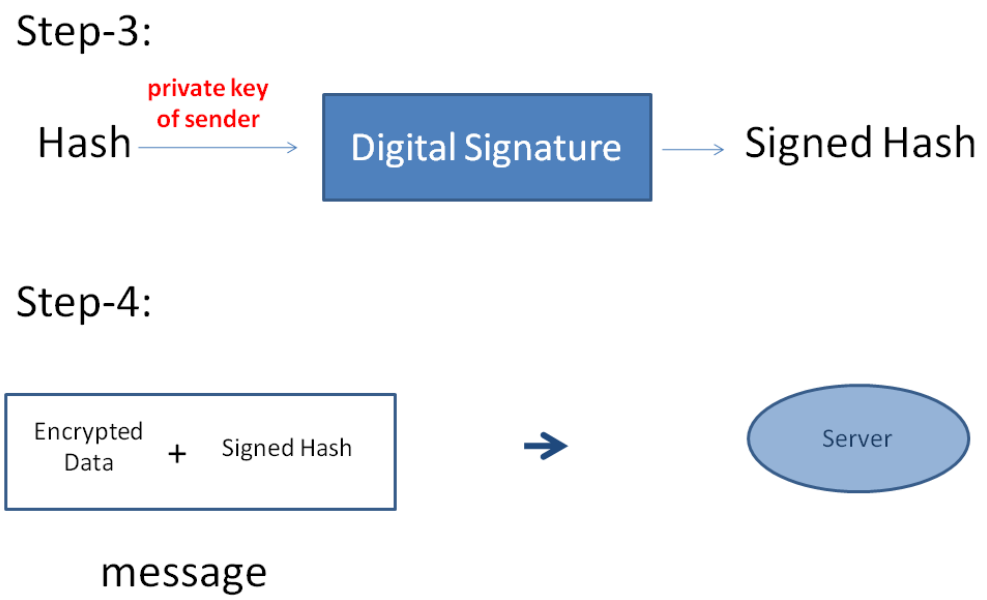

Figure 4-2: Signing the hash and appending with encrypted data and forward to server.

Upon reception of the data, the server separates the signed hash from the data to ensure the senders identity by verifying the signature. Now if the sender signature is not authenticated the packet is discarded. The rest of the message is then decrypted. Again a hash is generated from the data. Both the hashes are compared. If the two hashes match, then it is verified that there was no forgery in the route.

4.1. Environment Setup:

We have used three computers where two of them are clients and one of them is a server. All three PCs have Ubuntu 10.10 (desktop version) operating system. We install Minisip in two PCs (clients) and Opensips in the third one (server).

Installing Minisip:

We downloaded the latest version of Minisip from the source-code repository of www.minisip.org . Before installing Minisip we downloaded the required dependencies. OpenSSL and LibgladeMM were installed. Following the steps mentioned in the source code repository we downloaded and installed Minisip. The other missing packets and library files, were downloaded from the Ubuntu library repository using Synaptic Package Manager.

After finishing the setup of Minisip, we used the following commands before running Minisip.

```
sudo cp /usr/local/lib/libminisip.so.0 /usr/lib/
sudo cp /usr/local/lib/libmikey.so.0 /usr/lib/
sudo cp /usr/local/lib/libmsip.so.0 /usr/lib/
sudo cp /usr/local/lib/libmcrypto.so.0 /usr/lib/
sudo cp /usr/local/lib/libmnetutil.so.0 /usr/lib/
sudo cp /usr/local/lib/libmutil.so.0 /usr/lib/
sudo cp /usr/local/lib/libmstun.so.0 /usr/lib/
```

After that, using the command "minisip_gtkgui" runs MInisip.

Server Setup:

We have chosen Opensips as our SIP server. We have used the following commands [15] to install the necessary dependencies:

```
# sudo apt-get install mysql-server
# sudo apt-get install gcc
# sudo apt-get install bison
# sudo apt-get install flex
# sudo apt-get install make
# sudo apt-get install sed
# sudo apt-get install tar
# sudo apt-get install build-essential libtool
# sudo apt-get install libmysqlclient15-dev
```

Then we downloaded and installed Opensips using the following commands:

```
#sudo wget http://opensips.org/pub/opensips/1.7.0/src/opensips-1.7.0_src.tar.gz
#sudo tar -xzvf opensips-1.7.0_src.tar.gz
# cd opensips-1.7.0-tls/
```

```
# sudo make all include_modules="db_mysql"
# sudo make install include_modules="db.mysql"
#sudo     cp     opensips-1.7.0-tls/modules/db_mysql/db_mysql.so
/usr/local/lib/opensips/modules
#    sudo    cp    opensips-1.7.0-tls/scripts/opensipsctl.mysql
/usr/local/lib/opensips/opensipsctl/
# sudo cp opensips-1.7.0-tls/scripts/opensipsdbctl.mysql
/usr/local/lib/opensips/opensipsctl/
```

For configuring Opensips we edit the /usr/local/etc/opensips/opensips.cfg file and uncomment the following lines:

```
log_stderror=yes
disable_tcp=yes
loadmodule "db_mysql.so"
loadmodule "auth.so"
loadmodule "auth_db.so"
modparam("usrloc", "db_mode", 2)
modparam("usrloc",                                          "db_url",
"mysql://opensips:opensipsrw@localhost/opensips")
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password")
modparam("auth_db",                                         "db_url",
"mysql://opensips:opensipsrw@localhost/opensips")
modparam("auth_db", "load_credentials", "")

if (!www_authorize("", "subscriber"))
{
www_challenge("", "0");
exit;
}
```

And comment the following lines of the same file :

```
log_stderror=no
modparam("usrloc", "db_mode", 0)
```

Then we edit the /usr/local/etc/opensips/opensipsctlrc by uncommenting the following lines:

```
SIP_DOMAIN=dns.somaliren.com
DBENGINE=MYSQL
DBHOST=localhost
DBNAME=opensips
DB_PATH="/usr/local/etc/opensips/dbtext"
DBRWUSER=opensips
DBRWPW="opensipsrw"
DBROOTUSER="root"
USERCOL="username"
LIASES_TYPE="DB"
VERIFY_ACL=1
ACL_GROUPS="local ldint voicemail free-pstn"
VERBOSE=1
```

4.2. Design Overview

The Session Description Protocol generates its headers through a function call where the function returns string type value. Then the values are assigned to their respective fields of the SDP header. Each SDP object consists of its type and a priority value. The objects are orderly placed in their respective fields according to these. In our design we first identified the function that generates the string data. Then before passing it to the SDP header, we passed the data through the SHA -512 hash generating algorithm and got the digest for each individual object. The digest was then concatenated with the object and the old object was replaced by the concatenated form. This concatenated object was then passed to the SDP header. Thus the fields in the SDP header will consist of the object itself along with the digest of each of the objects. The rest of the header was kept as it was. Then SDP header itself was encrypted using the RSA algorithm (named after its inventors Ron Rivest, Adi Shamir and Leonard Adleman). Then the encrypted header was sent to the receiver. The receiver, on receiving the encrypted header, decrypts it using his private key and can access the objects passed. For each object, the receiver finds the digest for the values and matches it to the digest it has received. If they do not match, the header is assumed to be tempered with and is rejected. Otherwise it is accepted and so the SDP data is successfully sent.

4.3. Experiment Algorithm

The hash function we used is SHA-512. The pseudo code of the function can be downloaded from the author's (Olivier Gay) website. The implementation consists of all the SHA 2 algorithms but since we do not need all of them, we just kept the SHA-512 implementation part.

The SHA-512 algorithm takes a string value as input to find its hash value. It is done through a series of steps. The algorithm can take a string of maximum length of 2^{128} bits and convert it to a 512 bit digest. The message is first padded and made of a length which is a multiple of 1024 bits. Then it is broken down into 1024 bit blocks. The digest is initialized to a predefined length of 512 bits. It is then processed with the first block of the message to find the first intermediate digest. This is then processed with the second block to find the second intermediate digest. This method is continued till the final block whose outcome is the digest for the whole message.

Since SHA-512 is word oriented, the digest is broken down into 8 words of 64 bits. During each processing with the blocks, 8 constants are used to find the outcome. Each of the processing between the digest from the previous block and the next block consists of 80 steps where three functions and several operators are used. Finally the complete digest is found after the last block is processed. All the values of the SDP header has to undergo this encryption to find its corresponding digest. [14]

The RSA algorithm is a Public Key Encipherment system named by taking the initials of its inventors Rivest, Shamir and Adleman. RSA uses two exponents, e and d where e is public and d is private. The ciphertext C is found by the formula, $C = P^e \bmod n$ from the plaintext P, and P is found by the formula $P = C^d \bmod n$ where n is a very large number created during the key generation process. [14]

Chapter-5: Result and Discussion

We used two Linux based systems to implement our design, one of which acted as the server and one of the clients and the other was the other client. We installed and configured Minisip in both the systems and installed and configured Opensips on the system acting as the server. Then we edited the systems to incorporate our algorithms. After that we made calls from each of the clients to the other and observed the communication using Wireshark Packet Tracer.

We also made calls prior to our alterations to the systems and observed them using Wireshark Packet Tracer. We observed that the packets transferred during initiation of a call had the SDP headers attached with them and the headers were visible in plaintext. Thus any intruder could intercept the packets and compromise their integrity.

After implementing our system by adding our algorithms to the systems, we observed using the packet tracer that the SDP header which formerly was being shown in plaintext was encrypted using our algorithms. Thus if any intruder was to intercept the packets, he will not have any clue about the contents of the SDP header and thus does not present any threat to the integrity of the calls.

For example in the name field of the SDP header, the name is given as

Name: abcd

For that, our encrypted system will generate the hash and encrypt the message. The encrypted message is sent to the server in the following format

1921226_1233040~12159065~11072317~6890302~2587398~1156589~2587398~12159065~15567699~3713603~5154177~

The server will take the encrypted message and decrypt it to find the actual message and verify the hash. It will then encrypt the message again to send to the server. What it sends is

66389657_59688513~12113616~47929546~21581112~13478136~65862211~13478136~12113616~910924~51000477~5566044~

The receiver upon receiving this message will decrypt it to find the original message.

Hence if any intruder intercepts the communication, he will only find a list of numbers and will not be able to decrypt the message since he does not know the keys that have been used in the encrypting mechanism.

These values were taken from the outcome of the packet tracing using Wireshark Packet Tracer.

These observations prove that the solution that we proposed to the problem of SDP header being in plaintext and open to any intrusion is effective and will stop any types of attack on the data that have been described previously.

Due to a shortage of time we were not able to conduct our proposed real time performance analysis of our system. What we intended to do was to analyze the time taken for the encryption procedure, make multiple calls and observe the performance of our system in terms of efficiency and effectiveness. We also wanted to observe the system using multiple systems linked together and analyze the secure system in a real life scenario where multiple systems communicate with each other at the same time and observe the performance.

We know that our proposed system will have a higher overhead than the current insecure system due to the encrypted techniques used. The system will need more time to initiate a call and to provide more security to the users. But if we see that the overhead is within a tolerable and acceptable limit then we can say that our proposed system is a success and we can go forward and deploy our system to the users.

Chapter 6: Conclusions and Future Plan

VoIP has gained a huge attention worldwide because of its cheap establishment, cost effective maintenance, control and flexibility and massive portability. But as it is gaining popularity, so is increasing the risks associated with it as attackers will target it for their personal benefits. So there is a growing concern about the security and vulnerability of such mechanism and more researches are being dedicated towards it as time passes.

6.1 Summary of the thesis work

SIP remains the most popular protocol to establish VoIP calls but it has certain security drawbacks. In our work, we tried to address one of its vulnerability and implemented the solution up to a point that would ensure the integrity of the sender as the data travels over the network. SDP is part of the SIP protocol that shows certain vulnerable information about a session in plain-text which might be exploited by attackers to execute Man in the Middle attack.

We know that our proposed system will have a higher overhead than the current insecure system due to the encrypted techniques used. The system will need more time to initiate a call and to provide more security to the users. But if we see that the overhead is within a tolerable and acceptable limit then we can say that our proposed system is a success and we can go forward and deploy our system to the users.

An extensive analysis has been performed in order to detect the location and origination of SDP headers and the options has been weighed on to which algorithm is to be used in order to obtain data integrity. We have ensured data integrity of SDP so that forgery of the session can be detected. We encrypted the data for security. Further we generated a hash of the data, signed the hash and appended it to encrypted data. Then the appended data plus the hash is sent to the server who decrypts it to find the required information. The server encrypts the data again before sending it to the receiver who will decrypt it to find the original data.

6.2 Future Plans

The current work can be used as a platform to make the SDP a complete secured protocol. By using public key based encryption we can ensure that the data stays hidden from the attacker. Encryption is also needed so that the attacker cannot change the hash value and even if they do, that they have no control over the process. This also ensures that RTP redirection attacks are harder to perform and require user specific research before conducting an attack.

References

[1] “VoIP Eavesdropping: A Comprehensive Evaluation of Cryptographic Countermeasures”, Diego Perez-Botero, Yezid Donoso, Second International Conference on Networking and Distributed Computing (ICNDC), 2011 , Beijing, September 2011, pages 192-196.

[2] “Evaluation of VoIP media security for smartphones in the context of IMS”, Nakarmi, P.K. Mattsson, J. ; Maguire, G.Q., Communication Technologies Workshop (Swe-CTW), 2011 IEEE Swedish, Stockholm, October 2011, pages 123-128.

[3] “VoIP secure session assistance and call monitoring via building security gateway” , * Jung-Shian Li*, * Chuan-Kai Kao, * Jau-Jan Tzeng, International Journal of Communication Systems, Volume 24, Issue 7, pages 837–851, July 2011

[4] “Unregister Attacks in SIP.”, Anat Bremler-Barr, Ronit Halachmi-Bekel, Jussi Kangasharju, 2006 2nd IEEE workshops on Secure Network Protocols, 2006. Pages 32-37.

[5] “Implementation of Secure Hash Function over SDP.” A.F.M Sayem, Sheikh Mushtaq Ahmed, Md. Shihab Karim.

[6] MiniSIP homepage, http://www.minisip.org/, last visited: April 2012.

[7] “VoIP-Info.org-A reference guide to all VoIP things”, http://www.voip-info.org/ , last visited on April 2012.

[8] http://computer.howstuffworks.com/ip-telephony.htm , last visited on April 2012.

[9] http://rfc-ref.org/RFC-TEXTS/3830/index.html , last visited on April, 2012.

[10] “RFC-3830, MIKEY:Multimedia Internet Keying” by J. Arkko , E. Carrara, F. Lindholm, M. Naslund, K. Norrman, August 2004.

[11] http://watchguardsecuritycenter.com/tag/sip-registration-hijacking/ , last visited on July, 2012.

[12] http://www.youdzone.com/signature.html, last visited July, 2012.

[13] http://www.ouah.org/ogay/sha2 , last visited August, 2012.

[14] “Cryptography and Network Security” , by Behrouz A. Forouzan, published by The McGraw-Hill Company.

[15] “SomaliREN 2 team, Opensips Installation & Configuration Manual – Version 1.0” [Online]. Available: http://vm199.xen.ssvl.kth.se/csdlive/sites/default/files/projects/Opensips%20Installation%20&%20Configuration%20Manual%20-%20Version%201.0.pdf; Last viewed September 12, 2012.

Appendix A

SHA-512 Implementation

```
/*
 * FIPS 180-2 SHA-224/256/384/512 implementation
 * Last update: 02/02/2007
 * Issue date:  04/30/2005
 *
 * Copyright (C) 2005, 2007 Olivier Gay <olivier.gay@a3.epfl.ch>
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *      documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the project nor the names of its contributors
 *     may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
```

```
 *
 * THIS SOFTWARE IS PROVIDED BY THE PROJECT AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE PROJECT OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */



/*
 * FIPS 180-2 SHA-224/256/384/512 implementation
 * Last update: 02/02/2007
 * Issue date:  04/30/2005
 *
 * Copyright (C) 2005, 2007 Olivier Gay <olivier.gay@a3.epfl.ch>
```

```
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *      documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the project nor the names of its contributors
 *     may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE PROJECT AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.    IN NO EVENT SHALL THE PROJECT OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
```

```
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS

 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)

 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT

 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY

 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF

 * SUCH DAMAGE.

 */

//

#ifndef SHA2_H

#define SHA2_H


#define SHA224_DIGEST_SIZE ( 224 / 8)

#define SHA256_DIGEST_SIZE ( 256 / 8)

#define SHA384_DIGEST_SIZE ( 384 / 8)

#define SHA512_DIGEST_SIZE ( 512 / 8)

#define SHA256_BLOCK_SIZE  ( 512 / 8)

#define SHA512_BLOCK_SIZE  (1024 / 8)

#define SHA384_BLOCK_SIZE  SHA512_BLOCK_SIZE

#define SHA224_BLOCK_SIZE  SHA256_BLOCK_SIZE

#ifndef SHA2_TYPES

#define SHA2_TYPES

typedef unsigned char uint8;

typedef unsigned int  uint32;

typedef unsigned long long uint64;
```

```
#endif
#ifdef __cplusplus
extern "C" {
#endif

typedef struct {
    unsigned int tot_len;
    unsigned int len;
    unsigned char block[2 * SHA512_BLOCK_SIZE];
    uint64 h[8];
} sha512_ctx;

typedef sha512_ctx sha384_ctx;
void sha512_init(sha512_ctx *ctx);
void sha512_update(sha512_ctx *ctx, const unsigned char
*message,
                   unsigned int len);
void sha512_final(sha512_ctx *ctx, unsigned char *digest);
void sha512(const unsigned char *message, unsigned int len,
            unsigned char *digest);

#ifdef __cplusplus
}
#endif

#endif /* !SHA2_H */

//
```

```
#if 0
#define UNROLL_LOOPS /* Enable loops unrolling */
#endif
#include<string>
#include<iostream>
using namespace std;
//#include <string.h>

//#include "sha2.h"

#define SHFR(x, n)    (x >> n)
#define ROTR(x, n)   ((x >> n) | (x << ((sizeof(x) << 3) - n)))
#define ROTL(x, n)   ((x << n) | (x >> ((sizeof(x) << 3) - n)))
#define CH(x, y, z)  ((x & y) ^ (~x & z))
#define MAJ(x, y, z) ((x & y) ^ (x & z) ^ (y & z))

#define SHA512_F1(x) (ROTR(x, 28) ^ ROTR(x, 34) ^ ROTR(x, 39))
#define SHA512_F2(x) (ROTR(x, 14) ^ ROTR(x, 18) ^ ROTR(x, 41))
#define SHA512_F3(x) (ROTR(x,  1) ^ ROTR(x,  8) ^ SHFR(x,  7))
#define SHA512_F4(x) (ROTR(x, 19) ^ ROTR(x, 61) ^ SHFR(x,  6))

#define UNPACK32(x, str)                      \
{                                             \
    *((str) + 3) = (uint8) ((x)      );       \
    *((str) + 2) = (uint8) ((x) >>  8);       \
    *((str) + 1) = (uint8) ((x) >> 16);       \
    *((str) + 0) = (uint8) ((x) >> 24);       \
```

```
}

#define PACK32(str, x)                          \
{                                               \
    *(x) =   ((uint32) *((str) + 3)      )     \
           | ((uint32) *((str) + 2) <<  8)     \
           | ((uint32) *((str) + 1) << 16)     \
           | ((uint32) *((str) + 0) << 24);    \
}

#define UNPACK64(x, str)                        \
{                                               \
    *((str) + 7) = (uint8) ((x)      );         \
    *((str) + 6) = (uint8) ((x) >>  8);         \
    *((str) + 5) = (uint8) ((x) >> 16);         \
    *((str) + 4) = (uint8) ((x) >> 24);         \
    *((str) + 3) = (uint8) ((x) >> 32);         \
    *((str) + 2) = (uint8) ((x) >> 40);         \
    *((str) + 1) = (uint8) ((x) >> 48);         \
    *((str) + 0) = (uint8) ((x) >> 56);         \
}

#define PACK64(str, x)                          \
{                                               \
    *(x) =   ((uint64) *((str) + 7)      )     \
           | ((uint64) *((str) + 6) <<  8)     \
           | ((uint64) *((str) + 5) << 16)     \
```

```
           | ((uint64) *((str) + 4) << 24)    \
           | ((uint64) *((str) + 3) << 32)    \
           | ((uint64) *((str) + 2) << 40)    \
           | ((uint64) *((str) + 1) << 48)    \
           | ((uint64) *((str) + 0) << 56);   \
}

/* Macros used for loops unrolling */
#define SHA512_SCR(i)                         \
{                                             \
    w[i] =  SHA512_F4(w[i -  2]) + w[i -  7]  \
          + SHA512_F3(w[i - 15]) + w[i - 16]; \
}




#define SHA512_EXP(a, b, c, d, e, f, g ,h, j)               \
{                                                           \
    t1 = wv[h] + SHA512_F2(wv[e]) + CH(wv[e], wv[f], wv[g]) \
         + sha512_k[j] + w[j];                              \
    t2 = SHA512_F1(wv[a]) + MAJ(wv[a], wv[b], wv[c]);       \
    wv[d] += t1;                                            \
    wv[h] = t1 + t2;                                        \
}
uint64 sha512_h0[8] =
            {0x6a09e667f3bcc908ULL, 0xbb67ae8584caa73bULL,
```

```
             0x3c6ef372fe94f82bULL, 0xa54ff53a5f1d36f1ULL,
             0x510e527fade682d1ULL, 0x9b05688c2b3e6c1fULL,
             0x1f83d9abfb41bd6bULL, 0x5be0cd19137e2179ULL};

uint64 sha512_k[80] =
            {0x428a2f98d728ae22ULL, 0x7137449123ef65cdULL,
             0xb5c0fbcfec4d3b2fULL, 0xe9b5dba58189dbbcULL,
             0x3956c25bf348b538ULL, 0x59f111f1b605d019ULL,
             0x923f82a4af194f9bULL, 0xab1c5ed5da6d8118ULL,
             0xd807aa98a3030242ULL, 0x12835b0145706fbeULL,
             0x243185be4ee4b28cULL, 0x550c7dc3d5ffb4e2ULL,
             0x72be5d74f27b896fULL, 0x80deb1fe3b1696b1ULL,
             0x9bdc06a725c71235ULL, 0xc19bf174cf692694ULL,
             0xe49b69c19ef14ad2ULL, 0xefbe4786384f25e3ULL,
             0x0fc19dc68b8cd5b5ULL, 0x240ca1cc77ac9c65ULL,
             0x2de92c6f592b0275ULL, 0x4a7484aa6ea6e483ULL,
             0x5cb0a9dcbd41fbd4ULL, 0x76f988da831153b5ULL,
             0x983e5152ee66dfabULL, 0xa831c66d2db43210ULL,
             0xb00327c898fb213fULL, 0xbf597fc7beef0ee4ULL,
             0xc6e00bf33da88fc2ULL, 0xd5a79147930aa725ULL,
             0x06ca6351e003826fULL, 0x142929670a0e6e70ULL,
             0x27b70a8546d22ffcULL, 0x2e1b21385c26c926ULL,
             0x4d2c6dfc5ac42aedULL, 0x53380d139d95b3dfULL,
             0x650a73548baf63deULL, 0x766a0abb3c77b2a8ULL,
             0x81c2c92e47edaee6ULL, 0x92722c851482353bULL,
             0xa2bfe8a14cf10364ULL, 0xa81a664bbc423001ULL,
             0xc24b8b70d0f89791ULL, 0xc76c51a30654be30ULL,
```

```
                0xd192e819d6ef5218ULL, 0xd69906245565a910ULL,
                0xf40e35855771202aULL, 0x106aa07032bbd1b8ULL,
                0x19a4c116b8d2d0c8ULL, 0x1e376c085141ab53ULL,
                0x2748774cdf8eeb99ULL, 0x34b0bcb5e19b48a8ULL,
                0x391c0cb3c5c95a63ULL, 0x4ed8aa4ae3418acbULL,
                0x5b9cca4f7763e373ULL, 0x682e6ff3d6b2b8a3ULL,
                0x748f82ee5defb2fcULL, 0x78a5636f43172f60ULL,
                0x84c87814a1f0ab72ULL, 0x8cc702081a6439ecULL,
                0x90befffa23631e28ULL, 0xa4506cebde82bde9ULL,
                0xbef9a3f7b2c67915ULL, 0xc67178f2e372532bULL,
                0xca273eceea26619cULL, 0xd186b8c721c0c207ULL,
                0xeada7dd6cde0eb1eULL, 0xf57d4f7fee6ed178ULL,
                0x06f067aa72176fbaULL, 0x0a637dc5a2c898a6ULL,
                0x113f9804bef90daeULL, 0x1b710b35131c471bULL,
                0x28db77f523047d84ULL, 0x32caab7b40c72493ULL,
                0x3c9ebe0a15c9bebcULL, 0x431d67c49c100d4cULL,
                0x4cc5d4becb3e42b6ULL, 0x597f299cfc657e2aULL,
                0x5fcb6fab3ad6faecULL, 0x6c44198c4a475817ULL};



/* SHA-256 functions */

/* SHA-512 functions */


void sha512_transf(sha512_ctx *ctx, const unsigned char
*message,
                   unsigned int block_nb)

{
```

```
    uint64 w[80];
    uint64 wv[8];
    uint64 t1, t2;
    const unsigned char *sub_block;
    int i, j;

    for (i = 0; i < (int) block_nb; i++) {
        sub_block = message + (i << 7);

#ifndef UNROLL_LOOPS
        for (j = 0; j < 16; j++) {
            PACK64(&sub_block[j << 3], &w[j]);
        }

        for (j = 16; j < 80; j++) {
            SHA512_SCR(j);
        }

        for (j = 0; j < 8; j++) {
            wv[j] = ctx->h[j];
        }

        for (j = 0; j < 80; j++) {
            t1 = wv[7] + SHA512_F2(wv[4]) + CH(wv[4], wv[5],
wv[6])
                + sha512_k[j] + w[j];
            t2 = SHA512_F1(wv[0]) + MAJ(wv[0], wv[1], wv[2]);
            wv[7] = wv[6];
```

```
            wv[6] = wv[5];
            wv[5] = wv[4];
            wv[4] = wv[3] + t1;
            wv[3] = wv[2];
            wv[2] = wv[1];
            wv[1] = wv[0];
            wv[0] = t1 + t2;
        }

        for (j = 0; j < 8; j++) {
            ctx->h[j] += wv[j];
        }
#else
        PACK64(&sub_block[  0], &w[ 0]); PACK64(&sub_block[  8],
&w[ 1]);
        PACK64(&sub_block[ 16], &w[ 2]); PACK64(&sub_block[ 24],
&w[ 3]);
        PACK64(&sub_block[ 32], &w[ 4]); PACK64(&sub_block[ 40],
&w[ 5]);
        PACK64(&sub_block[ 48], &w[ 6]); PACK64(&sub_block[ 56],
&w[ 7]);
        PACK64(&sub_block[ 64], &w[ 8]); PACK64(&sub_block[ 72],
&w[ 9]);
        PACK64(&sub_block[ 80], &w[10]); PACK64(&sub_block[ 88],
&w[11]);
        PACK64(&sub_block[ 96], &w[12]); PACK64(&sub_block[104],
&w[13]);
        PACK64(&sub_block[112], &w[14]); PACK64(&sub_block[120],
&w[15]);
```

```
        SHA512_SCR(16);      SHA512_SCR(17);      SHA512_SCR(18);
SHA512_SCR(19);

        SHA512_SCR(20);      SHA512_SCR(21);      SHA512_SCR(22);
SHA512_SCR(23);

        SHA512_SCR(24);      SHA512_SCR(25);      SHA512_SCR(26);
SHA512_SCR(27);

        SHA512_SCR(28);      SHA512_SCR(29);      SHA512_SCR(30);
SHA512_SCR(31);

        SHA512_SCR(32);      SHA512_SCR(33);      SHA512_SCR(34);
SHA512_SCR(35);

        SHA512_SCR(36);      SHA512_SCR(37);      SHA512_SCR(38);
SHA512_SCR(39);

        SHA512_SCR(40);      SHA512_SCR(41);      SHA512_SCR(42);
SHA512_SCR(43);

        SHA512_SCR(44);      SHA512_SCR(45);      SHA512_SCR(46);
SHA512_SCR(47);

        SHA512_SCR(48);      SHA512_SCR(49);      SHA512_SCR(50);
SHA512_SCR(51);

        SHA512_SCR(52);      SHA512_SCR(53);      SHA512_SCR(54);
SHA512_SCR(55);

        SHA512_SCR(56);      SHA512_SCR(57);      SHA512_SCR(58);
SHA512_SCR(59);

        SHA512_SCR(60);      SHA512_SCR(61);      SHA512_SCR(62);
SHA512_SCR(63);

        SHA512_SCR(64);      SHA512_SCR(65);      SHA512_SCR(66);
SHA512_SCR(67);

        SHA512_SCR(68);      SHA512_SCR(69);      SHA512_SCR(70);
SHA512_SCR(71);

        SHA512_SCR(72);      SHA512_SCR(73);      SHA512_SCR(74);
SHA512_SCR(75);

        SHA512_SCR(76);      SHA512_SCR(77);      SHA512_SCR(78);
SHA512_SCR(79);


        wv[0] = ctx->h[0]; wv[1] = ctx->h[1];
```

```
        wv[2] = ctx->h[2]; wv[3] = ctx->h[3];
        wv[4] = ctx->h[4]; wv[5] = ctx->h[5];
        wv[6] = ctx->h[6]; wv[7] = ctx->h[7];
        j = 0;
        do {
            SHA512_EXP(0,1,2,3,4,5,6,7,j); j++;
            SHA512_EXP(7,0,1,2,3,4,5,6,j); j++;
            SHA512_EXP(6,7,0,1,2,3,4,5,j); j++;
            SHA512_EXP(5,6,7,0,1,2,3,4,j); j++;
            SHA512_EXP(4,5,6,7,0,1,2,3,j); j++;
            SHA512_EXP(3,4,5,6,7,0,1,2,j); j++;
            SHA512_EXP(2,3,4,5,6,7,0,1,j); j++;
            SHA512_EXP(1,2,3,4,5,6,7,0,j); j++;
        } while (j < 80);

        ctx->h[0] += wv[0]; ctx->h[1] += wv[1];
        ctx->h[2] += wv[2]; ctx->h[3] += wv[3];
        ctx->h[4] += wv[4]; ctx->h[5] += wv[5];
        ctx->h[6] += wv[6]; ctx->h[7] += wv[7];
#endif /* !UNROLL_LOOPS */
    }
}
```

```
void sha512(const unsigned char *message, unsigned int len,
            unsigned char *digest)
{
    sha512_ctx ctx;
    sha512_init(&ctx);
    sha512_update(&ctx, message, len);
    sha512_final(&ctx, digest);
          //cout << digest <<"\n";
}
void sha512_init(sha512_ctx *ctx)
{
#ifndef UNROLL_LOOPS
    int i;
    for (i = 0; i < 8; i++) {
        ctx->h[i] = sha512_h0[i];
    }
#else
    ctx->h[0] = sha512_h0[0]; ctx->h[1] = sha512_h0[1];
    ctx->h[2] = sha512_h0[2]; ctx->h[3] = sha512_h0[3];
    ctx->h[4] = sha512_h0[4]; ctx->h[5] = sha512_h0[5];
    ctx->h[6] = sha512_h0[6]; ctx->h[7] = sha512_h0[7];
#endif /* !UNROLL_LOOPS */

    ctx->len = 0;
    ctx->tot_len = 0;
}
```

```
void sha512_update(sha512_ctx *ctx, const unsigned char
*message,
                   unsigned int len)
{
    unsigned int block_nb;
    unsigned int new_len, rem_len, tmp_len;
    const unsigned char *shifted_message;

    tmp_len = SHA512_BLOCK_SIZE - ctx->len;
    rem_len = len < tmp_len ? len : tmp_len;

    memcpy(&ctx->block[ctx->len], message, rem_len);

    if (ctx->len + len < SHA512_BLOCK_SIZE) {
        ctx->len += len;
        return;
    }

    new_len = len - rem_len;
    block_nb = new_len / SHA512_BLOCK_SIZE;
    shifted_message = message + rem_len;
    sha512_transf(ctx, ctx->block, 1);
    sha512_transf(ctx, shifted_message, block_nb);
    rem_len = new_len % SHA512_BLOCK_SIZE;
    memcpy(ctx->block, &shifted_message[block_nb << 7],
```

```
            rem_len);

    ctx->len = rem_len;
    ctx->tot_len += (block_nb + 1) << 7;
}

void sha512_final(sha512_ctx *ctx, unsigned char *digest)
{
    unsigned int block_nb;
    unsigned int pm_len;
    unsigned int len_b;
#ifndef UNROLL_LOOPS
    int i;
#endif
    block_nb = 1 + ((SHA512_BLOCK_SIZE - 17)
                    < (ctx->len % SHA512_BLOCK_SIZE));
    len_b = (ctx->tot_len + ctx->len) << 3;
    pm_len = block_nb << 7;

    memset(ctx->block + ctx->len, 0, pm_len - ctx->len);
    ctx->block[ctx->len] = 0x80;
    UNPACK32(len_b, ctx->block + pm_len - 4);

    sha512_transf(ctx, ctx->block, block_nb);

#ifndef UNROLL_LOOPS
    for (i = 0 ; i < 8; i++) {
```

```
        UNPACK64(ctx->h[i], &digest[i << 3]);
    }
#else
    UNPACK64(ctx->h[0], &digest[ 0]);
    UNPACK64(ctx->h[1], &digest[ 8]);
    UNPACK64(ctx->h[2], &digest[16]);
    UNPACK64(ctx->h[3], &digest[24]);
    UNPACK64(ctx->h[4], &digest[32]);
    UNPACK64(ctx->h[5], &digest[40]);
    UNPACK64(ctx->h[6], &digest[48]);
    UNPACK64(ctx->h[7], &digest[56]);
#endif /* !UNROLL_LOOPS */
}

//#include<cstring.h>
//#include<cstdio>
//using namespace std;

string sha512digest(string data){
     //cout <<data <<"\n";
     string digestdata = "";
     unsigned char * message = new unsigned char [data.size()];
     message[data.size()] = 0;
     memcpy (message, data.c_str(), data.size());
     unsigned char digest[SHA512_DIGEST_SIZE] = "";
     //cout << digest << "\n";
     sha512(message, strlen((char *) message) , digest);
```

```
     digestdata = (char *) digest;
     //cout<< digest << "\n";
     return digestdata;
}



int main(void)
{
     string username = "1234";
     username = sha512digest(username);

     cout << username;
    return 0;
}
```

Appendix B

Implementation of RSA

```
/*

This algorithm can encrypt a message < n where n is product of
two prime numbers....

*/


#include<iostream>

#include<math.h>

#include<string>

using namespace std;

class RSA

{

long long int n;

long long int p;

long long int q;
```

```
long long int m;
long long int phi;
long long int e;
long long int d;
public:

void generatePandQ();
long long int modInverse(long long int , long long int);
bool isPrime(long long int );
void display();
void generateEandD();
long long int gcd(long long int , long long int );
void extEuclidean(long long int , long long int , long long int
, long long int );
long long int encryption(long long int );
long long int decryption(long long int );
};

void RSA::display()
{
cout << "n =\t" << n << endl;
cout << "p =\t" << p <<endl;
cout << "q =\t" << q << endl;
cout << "phi =\t" << phi << endl;
cout << "e =\t" << e << endl;
cout << "d =\t" << d << endl;
}
void RSA::generatePandQ(){
```

```
     long long int start = 0, limit = 0;
// lets generate a prime number between start and limit
start = ( 1 << 12 );     //

limit = ( 1 << 15 );     //

long long int count = 0;

for(long long int i = start+1; i < limit ; i++){

     if(isPrime(i)){

          count++;
          if(count == 1)
               p = i;
          else
          {
               q = i;
               break;

          }
     }
}

//p = 7;
//q = 11;
n = p * q;
```

```
     phi = ( p - 1 )*( q - 1);



}
void RSA::generateEandD(){


     long long int start = phi >> 5; // just take half of the
phi value as a start value ..
     long long int limit = phi;
     //   cout << "start is " << start << endl;
     //   cout << "limit is " << limit << endl;
          for(; start < limit ; start++){


          if(gcd( phi, start) == 1){
               //cout << " gcd 1 found\n";
               e = start;
               break;
               }
     }
     //e = 13;
     //   cout << "e is = " << e << endl;


     long long int y;
     y = modInverse(e, phi);
     //   cout << " x value is = " << x << endl;
     //   cout << " y value is = " << y << endl;
     d = y;
     //cout << y;
```

```
}



long long int RSA::modInverse( long long int a, long long int n)
{
     long long int i = n, v = 0, d = 1;
     while (a>0) {
          long long int t = i/a, x = a;
          a = i % x;
          i = x;
          x = d;
          d = v - t*x;
          v = x;
          }
     v %= n;


     if (v<0)
          v = (v+n)%n;
     return v;


}


long long int RSA::gcd( long long int a, long long int b){


     long long int temp = 0;
     while(b != 0){
          temp = a;
```

```
            a = b;
            b = temp % b;
            }
      return a;

}

bool RSA::isPrime( long long int x){
      if(x % 2 == 0)
      return false;
      //long long int lim = sqrt(x);
      for( long long int i = 3; i < x ; i += 2){
            if( x % i == 0)
                  return false;
      }

      return true;
}

long long int RSA::encryption( long long int msg){

      long long int i = e;
      long long int e_msg = 1;
      while(i){
            i--;
            e_msg =   (e_msg * msg) % n;
            }
```

```
     return e_msg % n;

}

long long int RSA::decryption(long long int e_msg){

     long long int i = d;
     long long int d_msg = 1;
     while(i){
          i--;
          d_msg = (d_msg * e_msg) % n;
          }
     return d_msg % n;
}

class RSA2
{
long long int n;
long long int p;
long long int q;
long long int m;
long long int phi;
long long int e;
long long int d;
public:

void generatePandQ();
```

```
long long int modInverse(long long int , long long int);
bool isPrime(long long int );
void display();
void generateEandD();
long long int gcd(long long int , long long int );
void extEuclidean(long long int , long long int , long long int
, long long int );
long long int encryption(long long int );
long long int decryption(long long int );
};

void RSA2::display()
{
cout << "n =\t" << n << endl;
cout << "p =\t" << p <<endl;
cout << "q =\t" << q << endl;
cout << "phi =\t" << phi << endl;
cout << "e =\t" << e << endl;
cout << "d =\t" << d << endl;

}

void RSA2::generatePandQ(){

          //0.printf("*****");
     long long int start = 0, limit = 0;
     // lets generate a prime number between start and limit
     start = ( 1 << 12 );     //
```

```
limit = ( 1 << 15 );     //

long long int count = 0;

for(long long int i = start+1; i < limit ; i++){

     if(isPrime(i)){

          count++;
          if(count == 1)
               p = i;
          else
          {
               q = i;
               break;

          }
     }
}

//p = 7;
//q = 11;
n = p * q;
phi = ( p - 1 )*( q - 1);
```

```
}
void RSA2::generateEandD(){


     long long int start = phi >> 5; // just take half of the
phi value as a start value ..
     long long int limit = phi;
     //   cout << "start is " << start << endl;
     //   cout << "limit is " << limit << endl;
          for(; start < limit ; start++){


          if(gcd( phi, start) == 1){
               //cout << " gcd 1 found\n";
               e = start;
               break;
               }
     }
     //e = 13;
     //   cout << "e is = " << e << endl;



     long long int y;
     y = modInverse(e, phi);
     //   cout << " x value is = " << x << endl;
     //   cout << " y value is = " << y << endl;



     d = y;
     //cout << y;
```

```
}
long long int RSA2::modInverse( long long int a, long long int
n) {
     long long int i = n, v = 0, d = 1;
     while (a>0) {
          long long int t = i/a, x = a;
          a = i % x;
          i = x;
          x = d;
          d = v - t*x;
          v = x;
          }
     v %= n;

     if (v<0)
          v = (v+n)%n;
     return v;

}




long long int RSA2::gcd( long long int a, long long int b){

     long long int temp = 0;
     while(b != 0){
          temp = a;
          a = b;
```

```
          b = temp % b;
          }
     return a;

}

bool RSA2::isPrime( long long int x){
     if(x % 2 == 0)
     return false;
     //long long int lim = sqrt(x);
     for( long long int i = 3; i < x ; i += 2){
          if( x % i == 0)
               return false;
     }

     return true;

}

long long int RSA2::encryption( long long int msg){
     long long int i = e;
     long long int e_msg = 1;
     while(i){
          i--;
          e_msg =   (e_msg * msg) % n;
          }
     return e_msg % n;
```

```
}

long long int RSA2::decryption(long long int e_msg){

     long long int i = d;
     long long int d_msg = 1;
     while(i){
          i--;
          d_msg = (d_msg * e_msg) % n;
          }
     return d_msg % n;
}


int main(){

     //printf("**");
     string message;
     RSA obj;

     obj.generatePandQ();

     obj.generateEandD();
     string s;
     obj.display();
     cout << "Enter the Message : ";
     cin >> message;
```

```
	long long int d_msg = 0;
	long long int e_msg = 0;
	long long int temp;
	for(int i = 0; i<message.size(); i++){
		temp = message[i];
		e_msg = obj.encryption(temp);
		cout<<e_msg<<endl;
		d_msg = obj.decryption(e_msg);
		cout<<d_msg<<endl;

	}




	//message = 9798;

//	e_msg = obj.encryption(message);

	//cout << "Encrypted message is = " << e_msg << endl;
	//long long int d_msg = 0;
	//d_msg = obj.decryption(e_msg);
	//cout << "Decrypted message is = " << d_msg << endl;

	return 0;

}
```

Appendix C

Implementation of Digital Signature

```
/*

This algorithm can encrypt a message < n where n is product of
two prime numbers....

*/


#include<iostream>

#include<math.h>

using namespace std;

class DS

{

long long int n;

long long int p;

long long int q;

long long int m;

long long int phi;

long long int e;

long long int d;

public:


void generatePandQ();

long long int modInverse(long long int , long long int);
```

```
bool isPrime(long long int );

void display();

void generateEandD();

long long int gcd(long long int , long long int );

void extEuclidean(long long int , long long int , long long int
, long long int );

long long int encryption(long long int );

long long int decryption(long long int );

};


void DS::display()

{

cout << "n =\t" << n << endl;

cout << "p =\t" << p <<endl;

cout << "q =\t" << q << endl;

cout << "phi =\t" << phi << endl;

cout << "e =\t" << e << endl;

cout << "d =\t" << d << endl;


}

void DS::generatePandQ(){


          //0.printf("*****");

     long long int start = 0, limit = 0;

     // lets generate a prime number between start and limit

     start = ( 1 << 12 );     //


     limit = ( 1 << 15 );     //
```

```
	long long int count = 0;

	for(long long int i = start+1; i < limit ; i++){

		if(isPrime(i)){

			count++;
			if(count == 1)
				p = i;
			else
			{
				q = i;
				break;

			}
		}
	}

	//p = 7;
	//q = 11;
	n = p * q;
	phi = ( p - 1 )*( q - 1);


}
void DS::generateEandD(){
```

```
    long long int start = phi >> 5; // just take half of the
phi value as a start value ..
    long long int limit = phi;
    //   cout << "start is " << start << endl;
    //   cout << "limit is " << limit << endl;
         for(; start < limit ; start++){

         if(gcd( phi, start) == 1){
              //cout << " gcd 1 found\n";
              e = start;
              break;
              }
    }
    //e = 13;
    //   cout << "e is = " << e << endl;



    long long int y;
    y = modInverse(e, phi);
    //   cout << " x value is = " << x << endl;
    //   cout << " y value is = " << y << endl;



    d = y;
    //cout << y;
}
```

```
long long int DS::modInverse( long long int a, long long int n)
{
     long long int i = n, v = 0, d = 1;
     while (a>0) {
          long long int t = i/a, x = a;
          a = i % x;
          i = x;
          x = d;
          d = v - t*x;
          v = x;
          }
     v %= n;

     if (v<0)
          v = (v+n)%n;
     return v;

}



long long int DS::gcd( long long int a, long long int b){

     long long int temp = 0;
     while(b != 0){
          temp = a;
```

```
            a = b;
            b = temp % b;
            }
      return a;

}

bool DS::isPrime( long long int x){
      if(x % 2 == 0)
      return false;
      //long long int lim = sqrt(x);
      for( long long int i = 3; i < x ; i += 2){
            if( x % i == 0)
                  return false;
      }

      return true;

}

long long int DS::encryption( long long int msg){

      long long int i = d;
      long long int e_msg = 1;
      while(i){
            i--;
            e_msg =   (e_msg * msg) % n;
```

```
        }
    return e_msg % n;

}

long long int DS::decryption(long long int e_msg){

    long long int i = e;
    long long int d_msg = 1;
    while(i){
        i--;
        d_msg = (d_msg * e_msg) % n;
        }
    return d_msg % n;
}

int main(){

    //printf("**");
    long long int message;
    DS obj;

    obj.generatePandQ();

    obj.generateEandD();

    obj.display();
```

```
    //cout << "Enter the Message : ";
    //cin >> message;
    message = 4931571;
    long long int e_msg = 0;

    e_msg = obj.encryption(message);

    cout << "Encrypted message is = " << e_msg << endl;
    long long int d_msg = 0;
    d_msg = obj.decryption(e_msg);
    cout << "Decrypted message is = " << d_msg << endl;

    return 0;

}
```