A security method for Gatekeeper based on Digital Authentication by H.235

SeonCheol Hwang *, SeungSoo Han **, JunYoung Lee ***,and JunRim Choi ****
* Dept of Internet & TV Broadcast, Induk Institute of Technology, Seoul, 139-749, Korea
Tel : +82-2-901-7706 Fax : +82-2-901-6884 E-mail: sthwang@induk.ac.kr
**Dept of Information Engineering, MyongJi University, YongIn, 449-728, Korea
Tel : +82-31-330-6761 Fax : +82-31-330-6470 E-mail: shan@mju.ac.kr
***Dept of Information Media Engineering, MyongJi College, Seoul, 120-776, Korea
Tel : +82-2-300-1165 Fax : +82-2-304-4832 E-mail: jylee@mail.mjc.ac.kr
****Dept of Electrical Eng. & Computer Sci., KyungPook University, DaeGu, 702-701, Korea
Tel : +82-53-950-5506 Fax : +82-53-950-5505 E-mail: jrchoi@ee.knu.co.kr

Abstract: While the needs for VoIPs(Voice over IP) encourage the commercial trials for VoIP services, there are many problems such as user authentication, blocking of illegal user and eavesdropping. In this paper, a management algorithm of registration of VoIP terminals is explained and security methods for tolling and data encryption module is designed and built up. The module structure will have the advantages of the entire development of secured gatekeeper without whole modification of gatekeeper. In order to secure the ordinary gatekeeper based on H.323 standard, user authentication and data encryption technologies are developed based on the H.235 standard and simply located over the plain H.323 stacks. The data structures for secured communications are implemented according to ASN.1 structures by H.235.

Key words: VoIP, Gatekeeper, H.323, Authentication, Eavesdropping, Security, H.235

1. INTRODUCTION

VoIP(Voice over IP) based on H.323 standard consists of four entities: a VoIP terminal, a gatekeeper, a gateway and an MCU(Multipoint Control Unit). Among them, the most important component is the gatekeeper that manages the registration of terminals, controls their bandwidths, converts IP addresses to phone numbers of PSTN(Public Switch Telephone Network)[1], does call setups and exchanges the abilities of H.245(a part of H.323). As shown above, the gatekeeper provides call control services as a kind of telephone office.

Recent developments in the data communication causes the increase of commercialized VoIP use, then companies launch service providing systems for VoIP[2]. Their focuses are the provisions of high quality Internet telephony and safe management systems in order to gather and keep more customers. In this sense, the authentication of proper customers should be necessary. Authentication in VoIP, before ITU-T announced H.235, was accomplished by companies individually. So, they could not exchange their information of customers or cross check the authentication each other. And the problems of embezzlements and eavesdropping of calls are raised too.

To solve these problems, ITU-T recommended a security protocol for Internet telephony. This recommendation is called by H.235 to provide security services such as authentication and privacy for H.323[3]. H.235 has the advantages to protect the H.323 by adding simple module on it. This kind of module structure gives advantages for developers to save efforts and time to implement security system without extra development of full systems.

We have designed protected procedures for registration of multiple VoIP terminals and security methods for call establishments and call controls based on H.235. The designed security system was developed in module style that works properly without entire changes of target system. This paper is organized as follows. The algorithms for encryption, integrity, key distribution and authenti- cation are described. The design and development of procedures for registration into gatekeeper and call establishment in secured methods are discussed. Finally, the results of secured communications are presented in ASN.1 format that was hooked during communication.

2. AUTHENTICATION METHODS

2.1. Key distribution methods

In cryptography, the key is very important to encrypt and decrypt communication data. Two kinds of key distribution methods are used in our secured gatekeeper system: a symmetric key method and a public key method. In the symmetric key method, the same key shared between two entities is used to encrypt and decrypt data. To perform the symmetric key distribution, Diffie-Hellman (DH) algorithm was used. This method provides signaling to generate a shared secret between two entities. At the end of DH exchange both the entities will possess a shared secret key. This shared key will be used to protect all data during communication.

The public key method is a kind of asymmetric key methods. This scheme uses two different keys that are a private key and a public key. In our public key method, it is assumed that an identifier and associated certificate are assigned and exchanged during subscribing processes. And the public key encryption algorithm in our system used the RSA algorithm.

2.2. Hash algorithm

A hash algorithm is used to check data integrity in our system. The hash algorithm calculates a message digest, called 'hash', that is a fixed-length, pseudo-random output produced by any length of input message. The message is sent with hash value. If a message is sent safely and is not changed during communication, the attached hash value is equal to the new calculated hash value of message. The hash value also is used to check non-repudiation of the message. The most common message digest produced with the distributed key is MAC(Message Authentication Code) and it is a very powerful method for message authentication. Especially the hash-based MAC is called HMAC. Two kinds of HMAC algorithm were implemented in our system: HMAC-MD5 and HMAC-SHA1.

(1) HMAC-MD5 algorithm

HMAC-MD5 algorithm is a kind of hash algorithm that calculates a message digest from a message with a key. The message digest is 128bit long data. The key used in this algorithm is not only a "cryptographic key" as used in a traditional sense but a shared secret. The size of the key is equal to or greater than L/2, where L(128bit) is the size of the hash function output. Any length of message is used in this algorithm where the message shall be divided into 512bit long data. If the data is less than 512bit, it is padded with 0. HMAC-MD5 that uses key longer than 512bit shall hash the key and then use the resultant 128bit string as the HMAC key.

(2) HMAC-SHA1 algorithm

HMAC-SHA1 is similar to HMAC-MD5 in processes. But HMAC-SHA1 uses 180bit long key and produces same length of message digest. So, SHA1 has survived cryptanalysis and comes highly recommended by the crypto community. Our system, however, truncated the 180bit hash value to 96bit according to the H.235 recommendation. This is called HMAC-SHA1-96.

2.3. Authentication Processes

The process of authentication verifies that the respondents are who they say they are. The authentication of our system is based either on a shared secret(by DH key exchange) or on public key based methods with certification. So, there are two types of authentication processes: Diffie-Hellman with optional authentication and Subscription-based authentication. Since DH method is used in point-to-point communication, it is not suitable for gatekeeper environment. Therefore our secured gatekeeper implemented the subscription-based authentication.

This method has three different variations that may be implemented depending on requirements:

(1) password-based with symmetric encryption

(2) password-based with hashing

(3) certificate-based with signatures

Authentication between Gatekeeper and endpoint generally is implemented with subscription-based methods. All RAS messages other than GRQ(Gatekeeper request) /GCF(Gatekeeper confirm) should contain the authenti- cation tokens. The token will contain the information as described in the following methods.

(1) Password-based with symmetric encryption

The encryption key is length N octets, and is formed as follows:

- If password length = N, Key = password
- If password length < N, the key is padded with zero
- If password length > N, the first N octets are assigned to the key, then the N+Mth octet of the password is XOR'd to the Mmod(N)th octet(for all octets beyond N).

Fig.1 illustrates the process of this method. GRQ means a request message to find a gatekeeper and GCF means a confirm message by the gatekeeper. In this stage, the messages are not authenticated. And then the endpoint sends and receives the request message xxQ and confirm message including 'cryptoTokens' containing 'crypto- EncryptedToken'. The 'cryptoEncryptedToken' contains an encrypted token of timestamp, random, senderID and peerID.

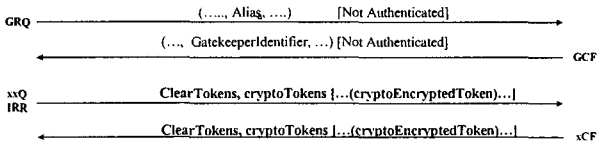

Fig.1 The procedure of password-based with symmetric encryption

(2) Password-based with hashing

This method is a kind of subscription-based method. When an endpoint registers with a gatekeeper, the password and identifier should be exchanged. The shared secret is computed by HMAC-SHA1-96 with exchanged password. This registration process is prior to the setup process between endpoints and gatekeeper. After registration, all messages will be protected by authentication and integrity. The procedures of this method are shown in Fig.2. The sender computes the hash value with the shared secret and adds it into the sending message. The recipient receives the message and then extracts the received hash value and keeps it. After that, the recipient computes the new hash value with the received message and compares the extracted hash value with the computed hash value. The message is considered uncorrupted only if both hash values are equal. If not, the authentication failed and the registration will be rejected.

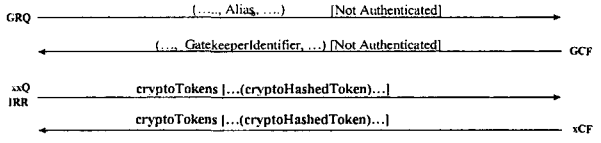

Fig.2 The procedure of password-based hashing

(3) Certificate-based with signatures

The signature security method mandates the GK-routed model and is based upon the H.245 tunneling techniques. This method does not depend on the administration of mutual shared secrets of the entities. All entities compute the digital signatures using keys extracted from certificates and send their messages with these digital signatures. There are two modes of signature security method: authentication/integrity mode and authentication-only mode. In the authentication/integrity mode, every hop re-computes security information and compares it hop-by-hop. Both authentication and integrity check are performed in this mode. In the authentication-only mode, the security information made by first hop is not changed in transit. So, this mode is called as End-to-End mode. Certificate-based with signature is shown in Fig.3.

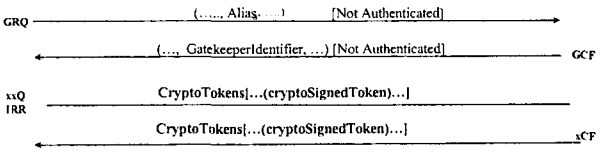

Fig.3 Certificate-based with signatures

3. SYSTEM DESIGN

3.1. Scope of security for H.323

Call signaling and control channel are secured by authentication and integrity of messages. The Media channel is not secured but the media itself is encrypted using DES(Data Encryption Standard) with symmetric key. So the media data packets consist of non-encrypted header and encrypted SDU(Service Data Unit).

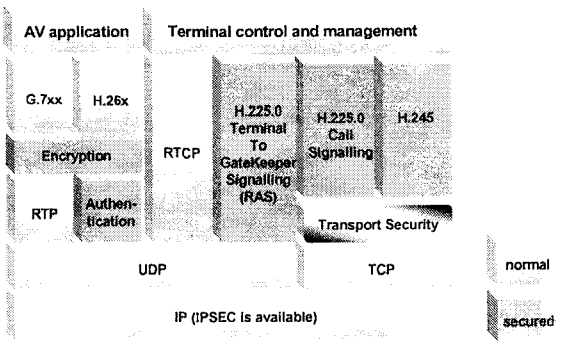

Fig.4 Overview of the scope of security area for H.323

3.2. Development of security stack based on H.235

We developed a security stack based on H.235. This stack performs the security functions at the top layer over the H.323 VoIP stack. Our security system has an advantage that is developed in module style and then works properly without entire changes of target system. The security module consists of two classes: 'TokenHandler' class and 'TokenTreater' class. According to security mode, VoIP stack decides to use the TokenHandler class or not when it constructs the PDU(Protocol Data Unit). If TokenHandler class is used, it evokes the TokenTreater class that computes an encrypted PDU following the security profiles, Procedure I/II/III. Procedure I is the password-based with hashing mode, procedure II is the certificate-based with signatures named a hop-by-hop mode and procedure III is also the certificate-based with signatures, an end-to-end mode. Fig.5 illustrates the security stack developed in this paper.

If user selects non-security mode, the PDU of H.323 will be passed by the security stack. But if user selects security mode, all PDUs will be sent to the security stack. Fig.6 illustrates more detail processes of the security stack.

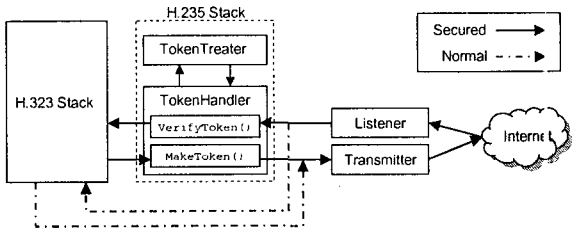

Fig.5 Overview of the developed security stack based on H.235

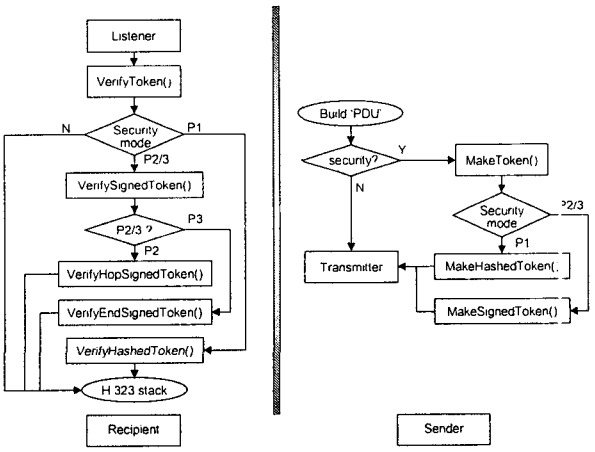

Fig.6 Detail processes of the security stack

When a PDU arrives at 'Listener' class of the recipient, it is verified by 'VerifyToken' class whether security mode or not. If it is security mode, it is classified as password-based with hashing (Procedure I), hop-by-hop signature mode (Procedure II) and end-to-end signature mode (Procedure III) according to its security mode. Procedure I evokes 'VerifyHashedToken()', procedure II evokes 'VerifyHopSignedToken()' and procedure III does 'VerifyEndSignedToken()' respectively. During this processes, authentication and message integrity is checked properly. After this processes, the PDU is sent to the H.323 stack. Consider the case where a sender

wishes to transmit a PDU. According to the security mode, the sender transmits the PDU without adding any security component or sends it into 'MakeToken()' for making security components. If the security mode is procedure I, the PDU is passed to 'MakeHashedToken()'. And if the security mode is procedure II or III, the PDU is handed to 'MakeSignedToken()'.

4. RESULTS AND DISCUSSION

4.1. GUI(Graphic User Interface) result

Our system consists of a Gatekeeper and a user terminal for VoIP that is designed in GUI. The user terminal can call directly by pushing numbers and select a GK-route mode. When select a GK-route mode, users should register with the secured GK and choose one among the security profile I/II/III. Fig.7 shows the sample GUI results of our secured VoIP system.

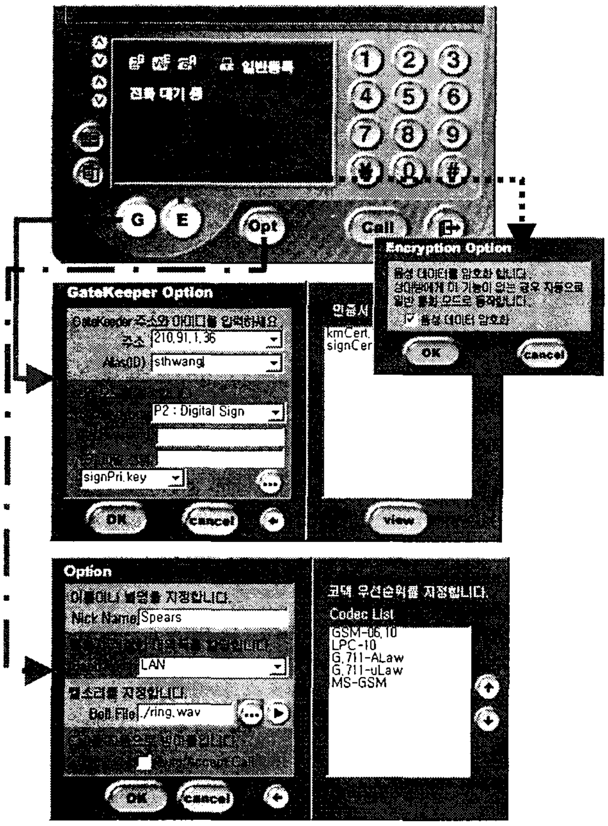

Fig.7 The sample GUI results of our secure VoIP

4.2. Results of Registration

When an entity successfully finds out a gatekeeper, it tries to register with the gatekeeper. In this stage, the entity sends 'registrationRequest(RRQ)' to the gatekeeper and receives 'registrationConfirm(RCF)'. The following shows the result of cryptoToken of RRQ and RCF of three procedures: Procedure I, II and III.

(1) Procedure I mode

registrationRequest ======>

```
registrationRequest {
  requestSeqNum = 2
    protocolIdentifier = 0.0.8 2250.0.2
    discoveryComplete = TRUE
    callSignalAddress = 1 entries {

- --- <omitting. > -----
    cryptoTokens = 1 entries {
      [0]=nestedcryptoToken cryptoHashedToken {
        tokenOID = 0.0.8.235.0.1.1        ===> "A" : RAS/H.225.0 is hashed
        hashedVals = {
          tokenOID  =  0.0.8.235.0.1.5     ===>  "T"  :  ClearToken  for
message security
          timeStamp = 1027908669
          dhkey = {
            halfkey =
11111101011101100011011000100010010100101010100000010001110101001110111111100
0010101110010100010110011010111000000111111000101010010101110111111001111010
1001101010101101011100001011011000110101011011111000011100110100111110000110
0001110111000101110100111100100010111011010101111011110010011001001010011100
1000010110010110101100011111101001011100001000110111111100101001011100010000
0011111100110000110101010011011001010110111001001001100001101001001010100001
110010111000100011101111111110101000101010000100011010 11
            modSize =
1001000100000010110010000011000111101110001101100000011111101100110000100010
0100001101111111100011111011001111010110100101001001101011000111101010101011
0011001010101100101011011110100111000010101011110000111000100001101101111100
0101001011110111011011010000111001011000001001111000000011010100111100110010
1011100011001011111101110000110010001101111110110011101011011000110000001110
1010110010110110100101101000101100001001101101110101001001010011110110101010
01110110001000100100100110010100101001001111001010001101
            generator =
0000001010001000110101111110011001010011101011110111001011000101100011000000
1000010010110100011001101111100111110010111011000100100111000101110010010010
0010000110010101101101111110010101011000101111111011101000100100111110101110
0101100111011100101101110001001011100010110011001110100110011111001100010000
1111111001110111100101111101111011011001001010100100010010101011001110100 10
1001000000001011010101001100010110111010010101100011001100010100000100010011
000010101110101000111001011110000000001001101101011 00010
          }
          random = 624937534
          generalID =  19 characters {
            004f 0070 0065 006e 0067 0061 0074 0065   Opengate
            003a 0020 0068 0073 0074 005f 006d 0061   . hst_ma
            0072 0063 0068                            rch
          }
        }
        token = {
          algorithmOID = 0.0.8.235.0.1.6        ===> "U" : HMAC-SHA1-96 is
used
          paramS = {
          }
          hash =
01001101001011100100000010111001001100011011010000010110101010000011010000
1101111011011100110010 1
        }
      }
    }
    keepAlive = FALSE
    willSupplyUUIEs = FALSE
  }
```

※ registrationConfirm <======

```
    gkclient.cxx(164)      H225    Started RAS thread
     h323pdu.cxx(1392)      H225RAS  Received PDU:
registrationConfirm {
    requestSeqNum = 2
    protocolIdentifier = 0.0.8.2250.0.2
    callSignalAddress = 1 entries {

----- <omitting..> -----

    }
    timeToLive = 600
    cryptoTokens = 1 entries {
      [0]=nestedcryptoToken cryptoHashedToken {
        tokenOID = 0.0.8.235.0.1.1        ===> "A" : RAS/H.225.0 is hashed
        hashedVals = {
          tokenOID = 0.0.8.235.0.1.5      ===> "T" : ClearToken for
message security
          timeStamp = 1027908572
          dhkey = {
            halfkey =
1110001001111110000100010001000010110111101111111111000100111110111100101001
0000100110001110010101110110000000101010101101001110000011010010000111001100 0
0001011111111000010110011111011101110101000011100100101001100100111010000010
1010010110110011100111011100100000101001010100011001101011110101000110001010
0111111110000110100111110010010101110100001010011000010100000110100110100001
1100000011000000000111011110001011010011111001001010110011111100110110011101 1
11000010010101110011101100001100000101110111111010010110 1
            modSize =
1001000100000010110010000011000111101110001101100000011111101100110000100010
0100001101111111100011111011001111010110100101001001101011000111101010101011
0011001010101100101011011110100111000010101011110000111000100001101101111100
0101001011110111011011010000111001011000001001111000000011010100111100110010
1011100011001011111101110000110010001101111110110011101011011000110000001110
1010110010110110100101101000101100001001101101110101001001010011110110101010
01110110001000100100100110010100101001001111001010001101
            generator =
0000001010001000110101111110011001010011101011110111001011000101100011000000
1000010010110100011001101111100111110010111011000100100111000101110010010010
0010000110010101101101111110010101011000101111111011101000100100111110101110
0101100111011100101101110001001011100010110011001110100110011111001100010000
1111111001110111100101111101111011011001001010100100010010101011001110100 10
1001000000001011010101001100010110111010010101100011001100010100000100010011
000010101110101000111001011110000000001001101101011 00010
          }
          random = 624937535

----- <omitting..> -----

        token = {
         algorithmOID = 0.0.8.235.0.1.6     ===> "U" : HMAC-SHA1-96 is used
          paramS = {
          }
          hash =
0000001001101000011111110110010011001100010011000110011011111100111100000 11
110000101011111 1000
        }
      }
    }
    willRespondToIRR = FALSE
  }
TokenHandler.cxx(442)      ######## This PDU have a GOOD crypto-Token.
######## Authenticated
```

(2) Procedure II mode

In this case, only RRQ is shown as following because RCF has similar structure as RRQ. The digital signature is shown in place of hashing value.

※ registrationRequest ======>

```
  h323pdu.cxx(1415)    H225RAS  Sending PDU:
 registrationRequest {
   requestSeqNum = 2
   protocolIdentifier = 0.0.8 2250 0.2
   discoveryComplete = TRUE
   callSignalAddress = 1 entries {

...   <omitting> ......

   cryptoTokens = 1 entries {
     [0]=nestedcryptoToken cryptoSignedToken {
       tokenOID  =  0.0.8.235.0.1.2          ===>  "B"  :  ClearToken
authentication-only
       token = {
         toBeSigned = {
            tokenOID      =      0.0.8 235 0.1.7  ===>     "S"     :
authentication/integrity
           timeStamp = 1027908900                    /non-repudiation of the
message
           dhkey = {
             halfkey =
[illegible]
             modSize =
[illegible]
             generator =
[illegible]
           }
           random = [illegible]
           generalID = 19 characters {
             004f 0070 0065 006e 0067 0061 0074 0065   Opengate
             003a 0020 0068 0073 0074 005f 006d 0061   : hst_ma
             0072 0063 0068                            rch
           }
         }
        algorithmOID  =  1.2.840 113549 1 1.4   ===>  "V"  :  RSA-MD5
digital
        paramS = {                                          signature

        }
                               signature
[illegible]
      }
    }
   }
   keepAlive = FALSE
   willSupplyUUIEs = FALSE
 }

.. ... <omitting> ......
```

(3) Procedure III mode

The result of the Procedure III was same as that of the Procedure II except 'tokenID=0.0.8.235.0.1.7' being replaced by 'tokenID=0.0.8.235.0.1.3'. It is shown as following.

```
.... . ....<omitting>..................... ... ....
   cryptoTokens = 1 entries {
     [0]=nestedcryptoToken cryptoSignedToken {
       tokenOID = 0.0.8.235.0.1.2
       token = {
         toBeSigned = {
           tokenOID = 0.0 8 235.0.1 3     ===> "R" : end-to-end
           timeStamp = 1027909268                   authentication-
only
           random = 2123044011
           generalID = 19 characters {
. . . .        <omitting ............ ... . ..........
```

5. CONCLUSION

We have designed secured procedures for registration of multiple VoIP terminals and security methods for call establishments and call controls based on H.235. The designed security system was developed in module style that works properly without entire changes of target system. The algorithms for encryption, integrity, key distribution and authentication have been implemented and verified according to RFC2104[4] and RFC2404[5]. The procedures for registration into gatekeeper and call establishment in secured methods have been designed and developed. These procedures were defined in the recommendation H.235 and implemented as a structure of ANS.1. Finally, the results of secured communications have been verified and showed that performed correctly. The communication tests were accomplished by using hooked data while call was connected. The results of this tests showed that the security components of PDU was correctly computed and could provide the authentication of the entities and the efficient protection of data. The next step of this study will be to design an entire key distribution system based on X.509[6] and a full management system of clients for tolling system.

References

[1] ITU-T H.323, Packet-based multimedia communications systems, ITU-T, 2000.

[2] D. Minoli and E. Minoli, Delivering Voice over IP Networks, Wiley Computer Publishing, 1998.

[3] ITU-T H.235, Security and encryption for H-Series (H.323 and other H.245-based) multimedia terminals, ITU, 2000

[4] H. Krawczyk, M. Bellare, and R. Canetti, HMAC: Keyed-Hashing for Message Authentication, Internet Engineering Task Force, Request for Comments(RFC) 2104, February, 1997.

[5] C.Madson and R. Glenn, The Use of HMAC-SHA-1-96 within ESP and AH, Internet Engineering Task Force, Request for Comments(RFC) 2404, November, 1998.

[6] R. Housley et al: Internet X.509 Public Infrastructure Certificate and CRL Profile, RFC2459, Internet Engineering Task Force, 1999.

[7] H. Schulzrinne and J. Rosenberg, "Internet telephony: Architecture and protocols – anIETF perspective," Computer Networks and ISDN Systems, Vol.31, pp.237-255, Feb.1999.

[8] J. Rosenberg and H. Schulzrinne, "An RTP payload format for user multiplexing," Internet Draft, Internet Engineering Task Force, Jan. 1998.

[9] Versit Consortium, "Computer telephony integration(CTI) encyclopedia," Tech. Rep. Release 1.0, Versit Consortium, Oct. 1996.

[10] M. Handley, H. Schulzrinne, E. Scholler, J. Rosenberg, "SIP: Session Initiation Protocol", RFC2543, IETF, March 1999.