Security for a Two-Way Text Messaging Application

Kevin Lai, Handika Handoko, Tuan Vo, John Li

ABSTRACT

The current protocol used in the two-way text messaging prototype service, UMsg, lacks proper security mechanisms, preventing practical use. To overcome the current protocol's shortcomings, we first analyze its security weaknesses. Secondly, we review various security mechanisms such as cryptographic algorithms, identity-based encryption, username/password systems, and shared key systems. Finally, from the analysis, we propose a hybrid protocol that combines various security benefits from proven security mechanisms and protocols to provide confidentiality, integrity, and availability to the UMsg service. While the hybrid protocol could be secured further, our analysis depicts that it is sufficient for use with a two-way text messaging application.

1.0 INTRODUCTION

In a recent University of British Columbia (UBC) student engineering project supervised by Dr. David G. Michelson, a prototype Personal Digital Assistant (PDA) wireless two-way text messaging system was developed to provide a novel service on UBC's ubiquitously covered wireless network. The service, named UMsg, is composed of two applications—one running on the client side and one running on the server side. The client application runs on a PalmOS-based PDA and sends messages to designated mailboxes on the server and retrieves messages from a user's mailbox.

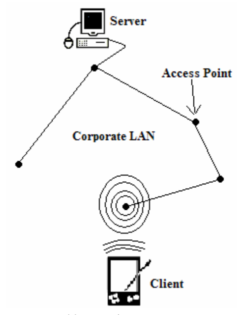


Figure 1. Client/Server Topology

While the initial UMsg prototype implements proper functionality, the service lacks any proper security mechanism to make it feasible for actual use. Therefore, in this paper we analyze the current protocol, review existing protocols, and propose a new protocol to secure the service.

2.0 CURRENT PROTOCOL

The current implementation of the UMsg application is based on a client/server topology. Outlined below are a description of the current protocol and an analysis of its security violations.

2.1 Description of Protocol

The current UMsg protocol is centrally-based, involving only client-to-server and server-to-client communication. The protocol does not require direct client-to-client communication, because the UMsg system revolves around the concept of an electronic mail system, where senders place messages on the server and the corresponding recipients retrieve them later on. To implement an authentication mechanism, the system uses a unidirectional, or "one-way", scheme, requiring only the client to send a valid username and password combination to the server [1]. Upon successful login, the client is able to access two services provided by the UMsg server—sending messages and retrieving messages.

To send a message, the current protocol requires the client to package a "Send" request in the following manner:

The character "S" || the sender's unique ID || the recipient's unique ID || text message

The notation for this communication scheme is expressed as:

Client → Server: S || senderID || recipientID || message

In the send request scheme, the senderID is the client's unique ID. The request functions as a Remote Procedure Call (RPC) to the UMsg server. Upon receiving the request, the server parses the information in the separate fields and places the message in the corresponding mailbox (recipientID). The recipient retrieves the message at a later time by transmitting a retrieve request to the server.

To retrieve messages, the current protocol requires the client to package a "Retrieve" request in the following manner:

The character "R" || the recipient's unique ID.

The notation for this communication scheme is expressed as:

Client → Server: R || recipientID

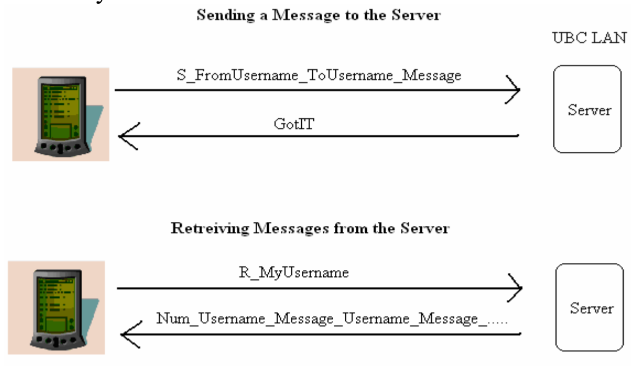
In the retrieve request scheme, the recipientID is the client's unique ID. Similar to the send request, the UMsg server parses the RPC upon receiving it and replies with all messages found in the corresponding mailbox (recipientID). The server appends a number at the beginning of the reply to indicate how many messages are included and separates individual messages with the corresponding sender IDs. All messages that are currently in the recipients mailbox will be appended to the reply, allowing the client to retrieve all his/her messages in a single package from the server. The protocol is as follows:

The number of messages || sender 1's unique ID || sender 1's message || sender 2's unique ID || sender 2's message || …

The notation for this communication scheme is expressed as:

Server → Client: # of messages || senderID 1 || senderID 1 message || senderID 2 || senderID 2 message || …

In both cases, the server plays a passive role in the system, requiring the client to initiate a request. The server does not broadcast to all clients when a new message arrives, but instead requires the clients to check for new messages periodically.



Sending a Message to the Server
UBC LAN
S_FromUsername_ToUsername_Message
GotIT
Server
Retreiving Messages from the Server
R_MyUsername
Num_Username_Message_Username_Message_.....
Server

Figure 2. Sending and Retrieving Messages

2.2 Security Violations

As evident, the current protocol does not provide any security mechanisms. The original purpose of the two-way text messaging project was to demonstrate that a PDA text-messaging application can be incorporated relatively easily into a corporate wireless LAN. With that goal accomplished, the protocol can now be analyzed for security violations that prevent the service from being used reliably.

While the current protocol provides a form of authentication via a login procedure, there is in fact no mechanism to prevent a user from sending messages as another user or retrieving messages from another user's mailbox by hacking the client application. A confidentiality violation can be seen in the following example:

User X → Server: R || User A

Upon receiving the retrieve request from User X, the server will return all messages in User A's mailbox to User X. Under the current protocol, the server is not capable of mediating every request it receives after a user has successfully logged into the system.

On the same rationale, an integrity violation occurs when a user spoofs the senderID in a send request. The following example highlights such a violation:

1. User A → Server: S || User A || User B || "Would you like to meet this afternoon around 2 pm?"
2. User B → Server: R || User B
3. Server → User B: 1 || User A || "Would you like to meet this afternoon around 2 pm?"
4. User B → Server: S || User B || User A || "Yes" (Intercepted by User X)
5. User X → Server: S || User B || User A || "No"

User A sends the message "Would you like to meet this afternoon around 2 pm?" to User B. After successfully retrieving the message from User A, User B sends the reply "Yes". Due to the malicious actions of User X, User B's message never reaches User A. Instead, User X's message, spoofed as User B, reaches User A, thus violating integrity through repudiation of source.

Finally, an availability violation exists since an attacker can flood a user's mailbox. A Denial of Service (DoS) attack such as this wastes the processing and storage resources, preventing the server from providing text messaging services to legitimate requests [2]. While the server could log the senderID to determine if a particular user is spamming mailboxes, the integrity violation mentioned above allows an attacker to send messages as various different users.

A short example of how the flooding could be accomplished is as follows:

User X → Server: S || User B || User A || "Spam!"
User X → Server: S || User C || User A || "Spam!"
User X → Server: S || User D || User A || "Spam!"
User X → Server: S || User E || User A || "Spam!"

From the analysis, one can see that the current protocol possesses many security flaws that make the UMsg application impractical for use in a wireless networking infrastructure.

3.0 ANALYSIS OF SECURITY MECHANISMS AND PROTOCOLS

Before proposing a new protocol, we first analyze various security mechanisms to determine their benefits and weaknesses. In the sections below, cryptography and identity-based encryption mechanisms are discussed, as well as username/password and shared key protocols.

3.1 Cryptography

Cryptography has always been associated with security because cryptography deals with the art of concealing information from parties that are not authorized to read it. The first use of cryptography was recorded almost 2000 years ago when Julius Caesar used a simple cryptographic algorithm to communicate with his army. Today, the Caesar cipher no longer serves its duty as various cryptanalysis techniques have improved and the standard that people have for security has growth significantly.

The implementation of cryptography requires the use of a cryptographic algorithm that does not depend on the secrecy of the algorithm. The strength of that cryptographic algorithm should rely on the secrecy of the key itself. This reasoning agrees closely to the Principle of Open Design:

> "The *principle of open design* states that the security of a mechanism should not depend on the secrecy of its design or implementation" [3].

For the purpose of securing the original protocol, a few modes of operation of cryptography have been analyzed. A brief discussion of these modes of operation is as follow:

1. Electronic Code Book (ECB)
   ECB is the simplest mode of operation, but it is also the most useful one since a lot of other modes of operation are based on ECB. The idea behind ECB is the use of a shared key to encrypt plaintext (using its respective encrypting algorithm) and to decrypt ciphertext (using its respective decrypting algorithm) [4].

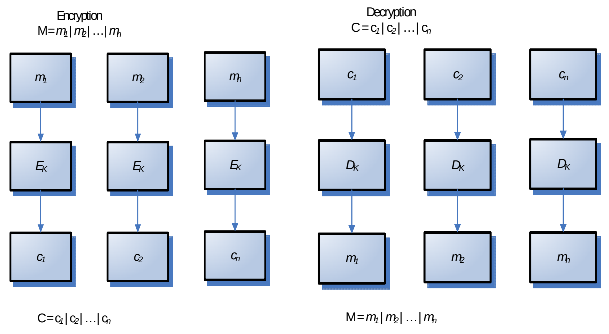


Figure 3. Encrypting and Decrypting in ECB

2. Cipher Block Chaining (CBC)
   CBC overcomes the weaknesses of ECB, such as is vulnerability to "cut-and-splice" attacks, by transforming plaintext to ciphertext in succession using a constantly modified key.

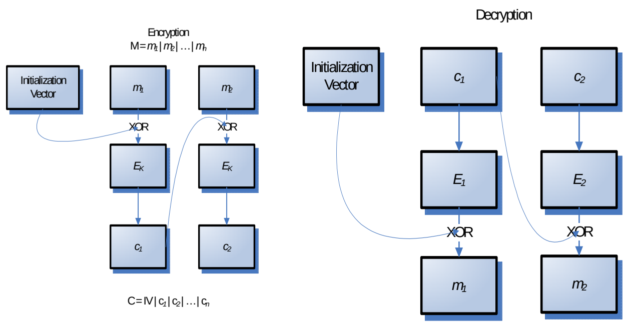


Figure 4. Encrypting and Decrypting in CBC

3. Cipher Feedback mode (CFB)
   While CBC's implementation provides much better protection than ECB, CBC still possesses a weakness in the performance sector. Because each subsequent ciphertext must be generated from the previous ciphertext, the computation of each block of ciphertext cannot be completed until a block of plaintext is completely received. CFB handles this weakness by implementing block cipher as a self synchronizing stream cipher [5].

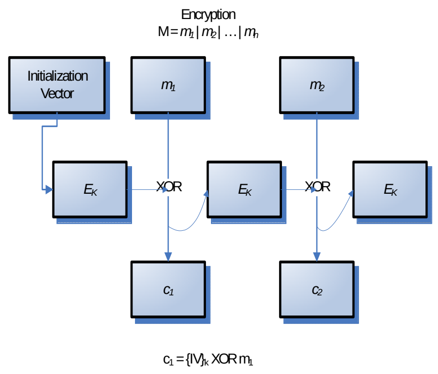


Figure 5. Encrypting in CFB

4. Output Feedback mode (OFB)
   OFB is used when a synchronous stream cipher is desired from a block cipher. It works in a similar fashion as CFB. In OFB, the output of a cipher block is XOR-ed back with the subsequent plaintext to get the subsequent ciphertext [5].

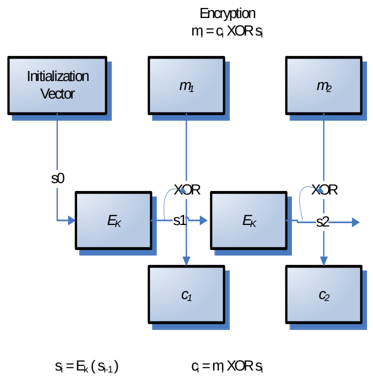


Figure 6. Encrypting in OFB

After selecting the appropriate mode of operation, there are still a few more things that must be taken care of when implementing cryptography:

a. Cryptography itself. In its implementation, we will not use our own cryptographic algorithm. The reason is because we are not capable of devising a bullet proof cryptographic algorithm
b. Key generation. The algorithm used to generate the key should not generate keys that can be deduced from a combination of plaintext and ciphertext.
c. Randomization.
d. Key management.
e. Buffer and information leakage.

While cryptography provides confidentiality by encrypting outgoing information, the algorithm alone does not provide integrity of origin or availability. Replayed messages violate integrity and also consume availability of services if used in a flooding attack.

3.2 Identity-Based Encryption

In an identity-based encryption (IBE) scheme, a cryptosystem uses an arbitrary string as a valid public key [6-7]. With an IBE system, public key distribution is no longer a concern since an entity's public key is simply a string that uniquely represents its identity. On a network infrastructure, this unique information can be a user's unique login name, IP address, or MAC address. Looking up, retrieving, or verifying public keys or certificates are no longer required. To decrypt a message, a private key for the particular message has to be retrieved from a trusted third party, often known as the private-key generator (PKG). A recipient requests a private key from the PKG to decrypt the message he/she has received. After successfully verifying the recipient's identity, the PKG uses a secret master key to generate a private key based on the arbitrary string (IP address for example) that constituted the recipient's public key [8].

Advantages of IBE are [9]:
1. Users can send messages to recipients who have not yet setup a public key
2. Improved performance and reduced complexity by eliminating public key lookup
3. Sent messages can only be read at some certain time in the future, since the private key is derived from the public key
4. A recipient's private key is valid only for a period of time and then is refreshed
5. Verification of a user can be performed by using the unique identity associated with that user

3.3 Username and Password

Passwords are one of the oldest mechanisms for enhancing security that a computer security analyst can think of. The simplistic, intuitive nature of the password system has kept its existence over time. It provides a sense of security since only users authenticated by the system can use the system and hence provides confidentiality and integrity services to a user. However, a password system is not without its drawbacks. A brute force attack using wordlists can precisely guess a password in a short period of time. Implementation of a username and password system on the two-way text messaging protocol still leaves it vulnerable to such an attack; therefore, even applying passwords does not provide a reliable form of security for the messages exchanged [10].

Despite the password system's weaknesses that make it relatively easy to compromise (provided a high performance computer), the mechanism is still widely used. With a properly chosen password, the system provides a sufficient level of security. The use of password system is analogous to the use of extra locks to your door. The extra lock that you have installed does not guarantee that nobody can enter your premises without permission. Its purpose is to increase your level of security and making the task of an adversary harder to accomplish.

3.4 Shared Key with Time Stamping

In a centralized client-server application, the server relays all communication and thus, could act as a trusted third party to clients [6]. The shared key with time stamping protocol requires the server to share a secret key with each registered client, allowing him/her to send messages encrypted with the "shared key" to the server. Using a shared key scheme, the server is capable of decrypting inbound messages received from senders and encrypting outbound intended for recipients. While the IEEE 802.11 wireless communication protocol has its own security mechanisms, a shared key system can use timestamps to assist in preventing man-in-the middle and session hijacking attacks. Time stamping also ensures that the server does not waste its resources servicing old requests sent by attackers [11]. In a shared key system, the following conditions are assumed:
1. Clients obtained the shared keys via a secure means, either from the server either in person or through another safe mechanism
2. Clocks on the mobile devices are synchronized with the server clock [1].

User A and User B use their shared keys (k_{UserA} and k_{UserB}) with the server to send and retrieve messages as follows:
1. User A → Server: UserA || {S || UserA || UserB || T || Message}k_{UserA}
2. User B → Server: UserB || {R || UserB || T}k_{UserB}
3. Server → UserB: {T || 1 || UserA || Message}k_{UserB}

User A's and User B's names are also located outside the encrypted messages so that the server can look up the corresponding shared key to decrypt incoming messages. An attacker cannot record and replay the messages because he/she will not be able to renew the time stamp in the encrypted message without the shared key. This protocol also protects a client's mailbox from being flooded with unwanted messages from attackers. Because the name of the recipient of the message is encrypted, an attacker cannot easily target a particular mailbox and flood it. Also, if an attacker records the first message and attempts to replay it multiple times in rapid succession before the timestamp is invalidated, the server is able to detect that the same message is being sent by retaining the last timestamp used by every user. If messages are sent multiple times from one IP address, the server could also take appropriate actions, such as banning further messages coming from that IP address.

Although this protocol provides the mechanisms for authentication and prevention of replay attack, it assumes the shared key can not be derived from the messages. However, the use of one shared key to encrypt all communication between one user and the server provides persistent attackers with the opportunity to perform statistical analysis and attempt to derive the shared key. Once a shared key is compromised, an attacker gains full access to the user's account. Furthermore, key management and distribution is still a fundamental problem with shared key systems.

4.0 PROPOSED PROTOCOL

The revised protocol for the UMsg application should provide confidentiality, integrity, and availability, but in a manner that is practical for a two-way text messaging application. In accordance, the proposed solution is a hybrid protocol consisting of elements from cryptographic mechanisms, identity-based encryption, username/password, and shared key protocols.

4.1 Description of Protocol

The proposed protocol requires an initialization stage for every request made to the server. The purpose of the initialization stage is to authenticate a user with their UBC Campus Wide Login (CWL) and provide a nonce session key for a send or receive request ("Nonce" means "used only once"). To begin a request, the client first sends a "Session Key" request to the server.

Notation
UserA_CWL = CWL username of User A
k_{CWL} = SHA1 Hash of CWL (username || password)
k_{SessA} = Session key for User A
T = Timestamp
Rand = 128-bit random number
S = Send request
R = Retrieve request

The client's request will be encrypted on the PDA using a secret key, k_{CWL}, derived from a hash of the client's CWL username and password before being sent to the server. By using the CWL information to generate the shared key, some benefits of an identity-based encryption scheme are utilized. Since the CWL information uniquely identifies a user, the server can verify his/her identity based on the information it already possesses. A complete IBE system is not required, since direct client-to-client communications do not occur when using the UMsg service. Furthermore, the problem with shared key distribution is avoided since the CWL information is pre-established by UBC's IT Services.

Upon receiving the request, the server responds with an encrypted message using the same secret key containing a session key, k_{Sess}, which can be used for one request by the client. After a request is made with the provided session key, the key is immediately invalidated.

The notation for the initial handshake stage is as follows:
1. User A → Server: UserA_CWL || {SessionKeyRequest || UserA_CWL || MAC Address || T || Rand}k_{CWL}
2. Server → User A: {SessionKeyReply || UserA_CWL || T || k_{SessA} || Rand + 1}k_{CWL}

The protocol assumes that the UMsg server has access to UBC's CWL information, since the UMsg server will be a part of UBC's network and will be managed by IT Services. The UMsg server will parse the CWL username it receives in the request, match the username to the corresponding password in its records, compute the k_{CWL} for the user, and decrypt the message. If the server is unable to successfully decrypt the message, then the request fails, indicating to the server that the client entered in their login information incorrectly.

The client is also required to send his/her device's MAC address in the protocol to provide a second level of authentication. The CWL username and MAC address sent in the request is compared against UBC's DNS server to verify the same user logged in on the wireless network is currently sending the request. If the DNS check fails, the session key request fails.

Added in this protocol is the ability for the server to authenticate itself in the reply, proving to the client that the reply came from the UMsg server. The random number sent in the original session key request is incremented by one and transmitted in the session key reply to indicate to the client that the his/her message was successfully received/decrypted by the server and that message they are receiving came directly from the server. Also included in the proposed protocol is a timestamp T to prevent replay attacks.

Sending or retrieving messages requires the use of the session key obtained in the initialization stage. The protocol for sending is as follows:
User A → Server: S || UserA_CWL || {S || T || UserA_CWL || recipient_CWL || message}k_{SessA}

The retrieve request and retrieve reply are as follows:
1. User A → Server: R || UserA_CWL || {R || T || UserA_CWL}k_{SessA}
2. Server → User A: {T || # of messages || sender1_CWL || message 1 || sender2_CWL || message 2 || …}k_{SessA}

While the proposed protocol can utilize any cryptographic algorithm, the Advanced Encryption Standard (AES), based on Rajindael's algorithm, is well established, making it the recommended choice. AES uses Cipher Block Chaining (CBC) with 128-bit keys and 128-bit block sizes for its encryption scheme [8].

4.2 Confidentiality

With the use of a cryptographic algorithm such as AES, the proposed protocol provides confidentiality, since all messages sent and received between clients and the server are encrypted before transmission. While the sender ID is still exposed in plaintext so that the server can determine what CWL hash key to use, the recipient and message itself is now in ciphertext. To further enhance the security mechanism, session keys are only used once per request, preventing previously transmitted messages from being disclosed even if an attacker compromises the current session key. The protocol assumes the CWL username and password are pieces of information only known by the user, thus constituting a secret key. In addition, UBC's CWL server is assumed to store passwords using a publicly known hash, such as SHA1, that can be used on the client side to generate k_{CWL}.

4.3 Integrity

To preserve integrity and non-repudiation of source, the proposed protocol allows the UMsg server to provide a first level authentication mechanism by verifying that it can decrypt the message with its CWL hash key. A successful decryption of the message indicates to the server that the message was transmitted by the owner of the CWL account, since the protocol assumes that only the owner of the account knows the password. In conjunction, UBC's DNS server acts as a second authentication mechanism by comparing the CWL information and MAC address transmitted in a session key request with the information in its DNS records. The DNS check also provides non-repudiation of source for the two-way text messaging application, since only the CWL user logged into UBC's wireless network can be using the service. The CWL information is checked by the UMsg server and the DNS server.

For the client to verify that the integrity of the session key reply is preserved, the UMsg server increments the random number in the original request and sends the result in the reply. Upon receiving the reply, the client decrypts the message and verifies that he/she received the random number sent out in the original request, but incremented by one.

The man-in-the-middle attack is avoided by using nonce session keys. While an attacker can listen to communications in an attempt to crack the session key, the key is updated for every request, making the protocol much more difficult to break. Both the re-use of session keys and timestamps allow the server to detect and prevent replay attacks.

4.4 Availability

In terms of availability, the protocol protects recipient mailboxes from being flooded by allowing the server to detect and discard replayed messages based on the re-use of a session key. To prevent attackers from flooding the server with session key requests, the DNS server can determine the IP address of the sender and deny access after reaching a threshold of failed session key requests. This check can be done irregardless of whether or not the attacker spoofs the CWL username on the request.

4.5 Other Security Benefits

In addition to providing confidentiality, integrity, and availability, the hybrid protocol also provides various other security benefits based on security design principles. Including timestamps in messages provides forward security, while updating the session key for every request provides backward security [12]. In addition, the DNS server's authentication check on the UBC wireless login and the UMsg's authentication check on the application login provide a form of multi-level authentication. The complete mediation principle is put into effect by requiring a new session key to be obtained for every send or receive request. Even after a user has successfully logged in, every request is first checked by the server to ensure that the user's information has not been tampered with after logging in. The protocol conforms to an open design principle, where its security relies only on keeping the key secret and not the algorithm. In this case, the key is the CWL password for a particular account. The hash function, SHA1, used to generate the CWL username and password hash key, k_{CWL}, and the encryption algorithm, AES, are both publicly known. Finally, a separation of privileges principle is adhered to requiring both the DNS server and the UMsg server to grant access to a user. Both servers must grant you access before the text messaging service can be utilized.

5.0 CONCLUSION AND FUTURE WORK

The proposed hybrid protocol achieves a suitable combination of security benefits from various different mechanisms and protocols for the UMsg service. While the proposed protocol could implement additional layers of security, it is sufficient considering the relative simplicity of the prototype UMsg service. Cryptographic algorithms in conjunction with shared keys provide confidentiality, while a pseudo-IBE scheme using the UBC CWL information provides integrity. Availability is preserved through the nonce session keys and a server-side mechanism for handling flood and replay attacks.

Pseudo-code for the hybrid protocol can be found in the Appendix. Our analysis hopes to depict how the proposed protocol addresses the various security weaknesses of the original protocol. Future work involves merging the hybrid protocol and evaluating its effectiveness through actual hands-on testing.

6.0 APPENDIX

6.1 Client Pseudo-code

```
// SessionKey+Random#+UsernamePassword+TimeStamping
// Client Side

String T;                   // timestamp
String Rand;                // random number
String myCWL = getMyCWL()   // CWL user naem
String myPassword = getMyPassword(); // CWL password
String MACAddress = getMacAddress(); // MAC address
String kCWL = SHA1Hash(myCWL + myPassword); //
obtain CWL key by hashing CWL login and password

Struct packet

// The "+" sign represents a way to concatenate
// strings so that the final string
// could be disassembled later

Main()
{
  String sessionKey;
  String userCommand = getUserCommand();

  While(userCommand != "quit"){
    if(userCommand == sendMsg or userCommand ==
    retrieveMsg){

      Rand = (String) getRandomNumber();
      T = getCurrentTime();

      sessionKey = Handshaking();
      if(sessionKey = error)
        Report error and break;

      if(userCommand == sendMsg)
        sendPacket=
        "S"+myCWL+encrypt(Ksession,"S"+T+UserA_CWL+r
        eceiverCWL+Message);
      else
        sendPacket=
        "R"+myCWL+encrypt(Ksession,"R"+T+UserA_CWL);

      replyPacket = fromServer();
      reply = decrypt(Ksession,replyPacket));
      if( reply == error )
        report server reply error and break
      if(userCommand == sendMsg)
        process the acknowledgement reply of the
        server
      else
        process the reply and update the database
        and user interface with the new messages
        retrieved
    }
    else
      process other userCommands on the client......

    userCommand = getUserCommand(); // get new
                                    // UserCommand
  }
}

// Handshaking is used to obtain the session key
// from the server
String Handshaking()
{
  packet replyPkg;
  packet packet1 =
    encrypt(kCWL,"SessionKeyRequest"+UserA_CWL+MACAd
    dress+T+Rand);
  packet sendPkg = myCWL+packet1;

  toServer(sendPkg);
  replyPkg = fromServer();
  return getSessionKeyDecrypt(kCWL, sendPkg,
  replyPkg);
}

// encrypt a message
packet encrypt(Key, String msg)
{
  Encrypt the msg with the given key
}

// Retrieve session key from the reply package
Key getSessionKeyDecrypt(Key key1,packet packet1,
packet packet2)
{
  packet decPacket1 = decrypt(key1,packet1);
  packet decPacket2 = decrypt(key1,packet2);

  if((decPacket1.T and decPacket2.T are within 30
  seconds) &&
  (decPacket1.Random==decPacket2.Random+1))
    return dePacket2.Ksession;
  else
    return error
}

// decrypt a package
String decrypt(Key key1,packet packet1)
{
  Decrypt the packet with given key and disassembled
  to different components (i.e. T, Ksession, ID,
  etc)
}
```

6.2 Server Pseudo-code

```
// SessionKey+Random#+UsernamePassword+TimeStamping
// Server Side

String T;                    // timestamp
String Rand;                  // random number
String myPassword = getMyPassword();    // CWL
password
String MACAddress = getMacAddress();    // MAC
address of user
String kCWL;

Struct packet;

// The "+" sign represents a way to concatenate
// strings so that the final string
// could be disassembled later

Main()
{
  String sessionKey;
  String ServerCommand = getServerCommand();
  While(ServerCommand != "quit"){

    // receive request from the client over the
    // network
    fromClientPacket = fromClient();
    T = getCurrentTime();

    // client is requesting a new session key
    if(userCWLExist(extractFirstItem(fromClientPacke
    t)))
    {
      kCWL =
      hashtable.lookup(extractFirstItem(fromClientPa
      cket));

      ClientRequest =
      decrypt(kCWL,fromClientPacket); //includes
      errorchecking

      if(check(ClientRequest.MACAddress) &&
      (ClientRequest.T is within 30 seconds of T)){

        // arbitrary random key
        sessionKey =
        SHA1Hash(ClientRequest.userCWL,ClientRequest
        .T,ClientRequest.Random+1);

        sessionKeyHashtable.put(userCWL,sessionKey);

        replymsg =
        encrypt(kCWL,SessionKeyReply+myCWL+T+session
        Key+ClientRequest.Random+1);

        toClient("SessKeyReply"+replymsg);

      }
      else
        report error and break
    }

    // If user want to send new messsages to server
    // or retrieve new messages from the server
    else if(extractFirstItem(fromClientPacket) ==
    "R" || extractFirstItem(fromClientPacket) ==
    "S")
    {
      // get an existing session key for the client
      // from the hash table
      // using the userCWL
      sessionKey =
      sessionKeyHashtable.lookup(extractSecondItem(f
      romClientPacket));

      if(sessionKey == error)
        report error and break

      ClientRequest =
      decrypt(sessionKey,fromClientPacket);

      if(ClientRequest != error && (ClientRequest.T
      is within 30 sec of T))
      {
        // check if encrypted field match
        // non-encrypted field
        if(ClientRequest.SendRetrieveField ==
        extractFirstItem(fromClientPacket)){
          if(ClientRequest.SendRetrieve == "S")
            inbox.put(ClientRequest.receipient,
            ClientRequest.msg);
          // Client wants to retrieve new messages
          else
          {
            replymsg =
            encrypt(sessionKey,T+inbox.getTotalMsgN
            umber);

            ClientRequest.userCWL)+inbox.getNewestM
            sg(ClientRequest.userCWL));

            inbox.deleteNewestMsg(ClientRequest.use
            rCWL);
            toClient(replymsg);
          }
          // delete the session key
          sessionKeyHashtable.delete(ClientRequest.
          userCWL);
        }
      }
      else
        report invalid package and break
    }
    else
      report invalid package and break

    ServerCommand = getServerCommand();
  }
}

Hashtable functions: lookup(), put(),delete();
Inbox functions: getTotalMsgNumber(),
getNewestMsg(), put(), delete();
```

7.0 REFERENCES

[1] Needham, Roger M., "Using Encryption of Authentication in Large Networks of Computers," In *Communications of the ACM*, Vol. 21, pp. 993-994, 1978.

[2] Northcutt, Stephen & Novak, Judy, "Denial of Service," Network Intrusion Detection 3rd Edition, Sams, 2002.

[3] Bishop, Matt, Computer Security: Art and Science, Addison Wesley, 2003.

[4] Swawinatha, Tara M. et. al., "Wireless Security and Privacy: Best Practices and Design Techniques," Addison Wesley, 2002.

[5] Schneier, Bruce, Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C, John Wiley & Sons, Inc., 1996.

[6] Appenzeller, Guido & Lynn, Ben, "Minimum-Overhead IP Security using Identity Based Encryption," Stanford University.

[7] Shamir, A, "Identity-Based Cryptosystems and Signature Schemes," In *Advances in Cryptology - Crypto '84*, Lecture Notes in Computer Science, Vol. 196, Springer-Verlag, pp. 47-53, 1984.

[8] Boneh, D. & Franklin, M., "Identity based encryption from the Weil pairing," In *Proceedings of Crypto '2001, Lecture Notes in Computer Science*, Vol. 2139, Springer-Verlag, pp. 213-229, 2001.

[9] Stading, Tyron, "Secure Communication in a Distributed System Using Identity Based Encryption," IBM.

[10] Vines, Russel Dean, "Wireless Security Essentials: Defending Mobile Systems from Data Piracy," Indianapolis, Wiley Publishing, 2002.

[11] "Wireless Network Security," Proxim Corporation, Whitepaper, 2003.

[12] Abdulla, Michel et. al., "Forward-Secure Threshold Signature Schemes," University of California, p. 6, 2002.

Bachmann, Glenn, PalmOS Programming, Sams, 2002.

Pogue, David, Palm Pilot: The Ultimate Guide, O'Reilly, 1999.