© 2017 IJSRSET | Volume 3 | Issue 1 | Print ISSN: 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Secure End Point Data Security Using Java Application Programming Interface

Peter S. Nyakomitta[1], Dr. Solomon Ogara[2], Dr. Silvance O. Abeka[3]

[1, 2, 3]Department of School of Informatics and Innovative Systems, Jaramogi Oginga Odinga University of Science & Technology, Bondo, Kenya

ABSTRACT

Most of the current instant messaging applications such as Telegram Secret Chat transmit packets in plain text. This means that an intruder equipped with appropriate remote monitoring tools can sniff the packets being transmitted and obtain the raw packets that are being relayed across the network. However, some of them like Whatsapp and Facebook Messenger have embraced end –to-end encryption. In so doing, this encryption protects this data as it is being passed from one device to another over communication channels. Effectively, this prevents potential eavesdroppers such as telecommunication service providers, Internet service providers or the provider of the communication service from being able to access the cryptographic keys needed to decrypt the conversation. However, most messaging applications encrypt data but only between the user and the companies' servers. The consequence of this is that the service providers can pry open the data being passed across their network data anytime and access the information being passed between the communicating parties. This paper sought to develop a port-based algorithm for packet encapsulation in instant messaging sessions. This is in realization of the fact that even with end to end encryption, the instant messages are in plain text at the communicating devices. This means that eavesdropping can still happen if these instant messages are read by people other than the ones for which the communication was meant. A prototype instant messenger application was developed with end to end encryption in place, as well as message encryption at the end devices. In this way, only a party that has a decryption key can read the transmitted messages.

Keywords : Instant Messaging, packets, Whatsapp, Facebook Massager, Skype, Eavesdroppers, Port-based algorithm.

I. INTRODUCTION

Instant messaging is a rapidly growing communications means that is trendy with both home and business users. It is ideal due to its effectiveness and easy means of network-based communication. According to Wendy (2013), it has become a popular form of communication which is steadily emergent with service providers such as *WhatsApp* having more than 800 million active users. Owing to their

expansive feature sets, current mobile instant messengers have diverse applications, including group chats, sharing media files, dwelling with friends, and even fleeting encounters with strangers.

Nearly all instant messaging systems utilize the same basic client-server architecture. It normally requires that users install instant messaging clients on their client machines, which can either be desktop computers, wireless devices, smart phones, tablets or personal digital assistant (Neal, 2014). These clients then establish a communication with an instant messaging (IM) server that resides in the messaging provider's infrastructure to locate other users and exchange their required messages.

In their study, Nardi et al., (2012) note that in most cases, the messages exchanged are not sent directly from the sender's computer to the recipient's computer. Instead, these messages are sent first to an instant messaging server. From this point, these messages are relayed to the intended recipient. While the majority of

instant messaging systems employ centralized servers to broadcast all messages, some of these instant messaging systems provide peer-to-peer messaging. According to Vleck (2015), in this model, clients contact the instant messaging server to position other clients.

Thereafter, once the client chat program has managed to establish the location of its communication partner, it contacts this peer directly. In their study, Schiano and Kamm (2013) pointed out that the advantage of this scheme is that it offers better security than the client-server client scheme when both users are on the same local area network. This is mainly because the instant messages do not travel over the Internet which is a public infrastructure. However, if one user is located outside the corporate network, messages sent between machines are exposed to potential eavesdroppers, just as in the client-server-client scheme.

Despite its vast usage, the instant messaging services introduce a number of security risks if proper security measures are not applied .According to Roberts (2015), the current Internet Threat Model (ITM) including the secure socket layer (SSL) model assumes an absolutely vulnerable communication link with trusted end-points. However, this assumption of secure end-points may dent software security. This so because the present Internet environment is contaminated with malicious software (Malware) such as Trojan Horses, WORMS, botnets and viruses, compromising of a huge number of machines at any given point of time. Form this point of view, it seems reasonable to conclude that an SSL based solution is not adequate for instant messaging security.

II. METHODS AND MATERIAL

A. Related Work

Many instant messaging applications only encrypt messages between the communicating parties and the instant message providers. However, recently, facebook messanger and WhatsApp started using end to end encryption. With this encryption technology, only the sender and the receiver can read what is sent, and nobody in between, not even the instant messenger provider. This is because the messages are secured with a lock, and only the recipient and the sender have the special key needed to unlock and read them. As Sanchez (2014) notes, for added protection, every message that is sent has its own unique lock and key. The advantage of this approach is that all of this happens automatically and therefore there is no need to turn on settings or set up special secret chats to secure your messages.

However, according to Mahajan t al., (2013), end-to-end encryption makes sure that data is transferred securely between the communicating endpoints. In this scenario, instead of an intruder trying to break the encryption, an he can impersonate a message recipient during key exchange phase. This can be done by substituting his public key with that of the recipient's. This essentially means that the messages are encrypted with a key known to the intruder. Moreover, after successfully decrypting the message, he can then encrypt it with a key that he shares with the actual recipient, or his public key in case of asymmetric systems, and send the message on again to avoid detection. This is in effect as a man-in-the-middle attack. Another challenge with current instant message applications is that companies normally willingly or unwillingly introduce back doors to their software (Anglano, 2014). This is meant to help subvert key negotiation or bypass encryption altogether. For instant, according to Barghuthi and Said, (2013), in 2013, was shown that Skype had a back door which allowed Microsoft to hand over their users' messages to the National Security Agency despite the fact that those messages were officially end-to-end encrypted.

A study by Bodriagov and Buchegger (2011) further revealed some authentication issues in instant message applications. Majority of the end-to-end encryption protocols comprise some form of endpoint authentication specifically to prevent man-in-the-middle-attacks. As an illustration, one could rely on certification authorities or a web of trust. An alternative technique is to generate cryptographic hashes or fingerprints based on the communicating users' public keys or shared secret keys. In this arrangement, the communicating peers compare their fingerprints using an outside (out-of-band) communication channel that guarantees integrity and authenticity of communication (but not necessarily secrecy), before starting their conversation. Hence if the fingerprints are equivalent, it is assured that there is in no man in the middle attack. When displayed for human inspection, fingerprints are usually encoded into hexadecimal strings. These strings are then formatted into groups of characters for readability. For example, a 128-bit MD5 fingerprint would be displayed as follows shown in Figure 1.

```
43:51:43:a1:b5:fc:8b:b7:0a:3a:a9:b1:0f:66:73:a8
```

Figure 1. A 128-bit MD5 Fingerprint

Some protocols display natural language representations of the hexadecimal blocks. Essentially, this is a one-to-one mapping between fingerprint blocks and words, and therefore there is no loss in entropy (Ricochet, 2014). The problem crops in when the protocol chooses to display words in the user's native (system) language. The security concern is that this can make cross-language comparisons prone to errors.

Moreover, in their study, Yusof and Abidin (2011) pointed out that the end-to-end encryption paradigm does not directly address risks at the communications endpoints themselves. Each users' communicating device can still be hacked to steal his or her cryptographic key, for example to create a man-in-the-middle-attack, or simply read the recipients' decrypted messages. Even the most perfectly encrypted communication pipe is only as secure as the mailbox on the other end. This is the gap that this paper endeavored to fill.

B. Methodology

This research work adopted the experimental research design where there was a practical design and implementation of an algorithm to protect the instant messages at the end devices as well the data in transit in communication networks. The integrated development environment chosen was *Jcreator* and the proposed algorithm was developed in Java programming language. The experimentations included the design and testing the algorithm in order to validate its performance as far as the protection of instant messages is concerned. Java was selected due to its support of networking libraries that could allow two or more systems to communicate using the concept of an IP address and a port number, which together are called socket.

1. Client Design

This paper developed an algorithm that could mimic the server and clients in an instant messaging communication scenario. Figure 2 shows a snippet of the developed algorithm.

```
import java.io.*;
import java.net.*;
public class Client implements Runnable
{
        static Socket socket=null;
        static PrintStream output;
        static BufferedReader input=null;
        static BufferedReader userip=null;
        static boolean flag=false;
        public static void main(String[] args)
        {
                // int port=1234;
                 int port=1234;

                     // String host="localhost";
                      String host="localhost";

                try
                {
                        socket=new Socket(host,port);
                        userip=new BufferedReader(new InputStreamReader(System.in));
                        output=new PrintStream(socket.getOutputStream());
                        input=new BufferedReader(new InputStreamReader(socket.getInputStream()));
```

Figure 2. Java Networking Support

As shown, the first line imported the *java.io* package, which hold nearly every class employed to perform input and output (I/O) in Java. Since the instant messaging application needed input from users in form of authentication keys and message chats, it was necessary to have this package for input. Additionally, the application needed to display the information to the participants in form of message chats, hence the need for output package. All these streams represent an input source and an output destination.

The second package was that of the *java.net* package of the J2SE APIs. This package consist of a collection of classes and interfaces that provide the low-level communication details, allowing users to write programs that focus on solving the problem at hand. The concept of network *programming* was employed to writing an application programming interface that executed across multiple in which the devices are all connected to each other using a network.

The *Sockets, Printstream* and *BufferedReader* were then initialized. A socket was used as an endpoint for communication between the server and the client instant messaging applications. This socket was configured to utilize port *1234* for communication purposes. The host IP address was set to be that of the *localhost*. After this , the port and IP address were bound to the socket. The server socket waited for requests to come in over the network from the IM participants. It then performed some operation based on that request, and then returned results to the clients.

The *'input=new BufferedReader(new InputStreamReader(socket.getInputStream()));'*

statement was used to get the socket's input stream and open a *BufferedReader* on it. Readers and writers (*Printstream)* were then employed to write Unicode characters over the socket. As shown in Figure 3, if the client encountered an error in trying to access the server, it generated an error, *'Unknown host'*, followed by the IP address of the requesting client.

```
catch(Exception e)
{
            System.err.println("Unknown host"+host);
}
if(socket!=null)
{
            try
            {
                        new Thread(new Client()).start();
                        while(!flag)
                        {
                        output.println(userip.readLine());
                        }
            output.close();
            input.close();
            socket.close();
```

Figure 3. Server Contact and Connection Termination Process

However, if the server contact was successful *(if(socket!=null)),* a new thread to start conversation with the client was created (*new Thread(new Client()).start();).* From this point on, the server continues to listen to the client, get its input and pass it to other clients as confirmed by line that reads: *output.println(userip.readLine()).* To display the messages from one client to others, the following snippet in Figure 4 was employed.

However, if the server contact was successful *(if(socket!=null)),* a new thread to start conversation with the client was created (*new Thread(new Client()).start();).* From this point on, the server continues to listen to the client, get its input and pass it to other clients as confirmed by line that reads: *output.println(userip.readLine()).* To display the messages from one client to others, the following snippet in Figure 4 was employed

```
String msg;
try
{
while((msg=input.readLine())!=null)
System.out.println(msg);
flag=true;
}
catch(IOException e)
{
System.err.println("IOException" + e);
```

Figure 4. IM Display to participants

It starts by creating a string variable which could hold the chat messages. It then continues to display the chat messages from the participants until the users agree to tear down the connection as indicated by the following lines:

while((msg=input.readLine())!=null)
System.out.println(msg);

The lines below were then employed to tear down the connection:

output.close();
input.close();
socket.close();

2. Server Design

On the server side, similar packages as was the case were imported as evident in Figure 5. These were *the java.io.**, and the *java.net.** packages. In addition to these two packages, the *java.util.Scanner* package was imported to facilitate the reading of client chat messages via the keyboard.

```
import java.io.*;
import java.net.*;
import java.util.Scanner;
public class Server
{
      static ServerSocket server=null;
      static Socket socket=null;
      static ClientThread th[]=new ClientThread[10];
      public static void main(String args[])
      {
            int port=1234;
            System.out.println("Server started...");
            System.out.println("  [Press Ctrl C to terminate ]");
            try
            {
                  server=new ServerSocket (port);
            }
            catch(IOException e)
```

Figure 5. Server API Snippet

The server socket was then initialized to null and the server was also configured to use port *1234* to communicate with the client participants. If the server-client connection over this port could not be established,

an error message to this effect was generated as confirmed by the line that reads: *catch(IOException e)*. The server then proceeded to require the clients to input their username that could be used to identify the client user to other IM client participants as shown in Figure 6.

```
class ClientThread extends Thread
{
      BufferedReader input=null;
      PrintStream output=null;
      Socket socket=null;
      ClientThread th[];
      public ClientThread(Socket socket,ClientThread[] th)
      {
            this.socket=socket;
            this.th=th;
      }
      public void run()
      {
            String msg;
            String username;
            try
            {
                  input = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                  output = new PrintStream(socket.getOutputStream());
                  output.println("Kindly Enter Your UserName");
                  username = input.readLine();
                  output.println(username + ":Welcome to IM Chat.");
                  output.println("To leave IM Chat type $$");
```

Figure 6. Client Identification Username

The ability to secure the chat messages at the end points was one of the goals of this paper. As such, an API to handle this functionality was created. Considering the OSI reference model, then it could be seen that this API worked at the presentation layer as shown in Figure 7.

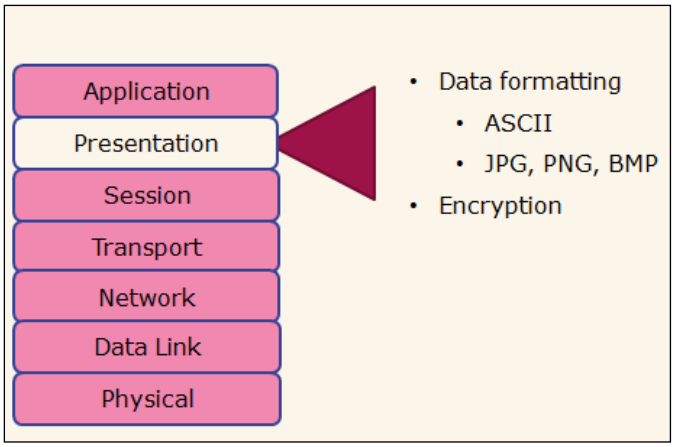


Figure 7. Application of Proposed Algorithm in OSI Model

This figure illustrates that the presentation layer is responsible for data formatting and protection, as exampled by encryption. This means that data protection schemes such as TLS and SSL work in this layer. These two security protocols are vulnerable to attacks such as the BEAST and POODLE attacks. However, the TCP protocol stack lacks the presentation layer. In fact, the application, presentation and session layers of the OSI are combined into a single layer in TCP: the application layer as shown in Figure 8.

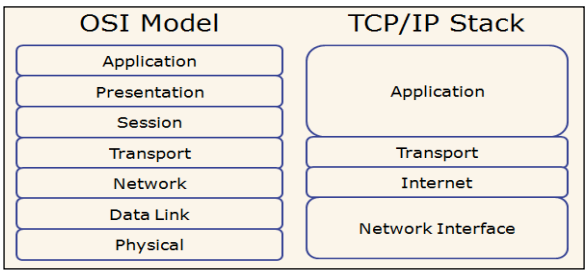


Figure 8. TCP and OSI Comparison

In TCP therefore, the data formatting functionality takes place between the transport and the application layers. In this regard, the developed algorithm was implemented between the transport layer and the application layer of the TCP so that it can provide the required end point security as shown in Figure 9.

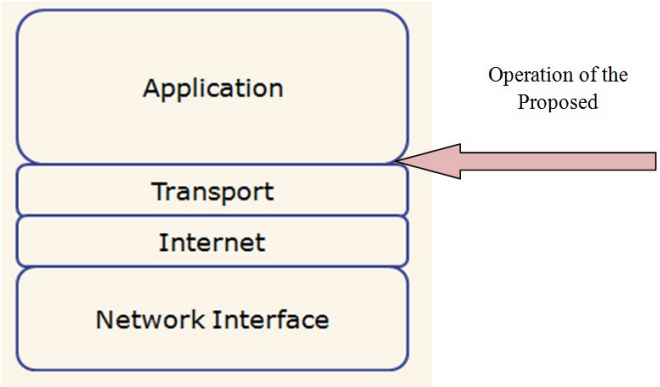


Figure 9: TCP Operation of the Proposed Algorithm

In order to ensure end point security, which is clearly missing in the end to end protection, the proposed algorithm included split knowledge type of security model where the information or privilege needed to perform an operation is divided among multiple users such that no single persons has sufficient privileges to compromise the security of an application. For this case, the split knowledge consisted of *byteArrayToHexString* transformation as shown in Figure 10.

```
try {
    // quick way to do input from the keyboard, now deprecated...
    java.io.StreamTokenizer Input=new java.io.StreamTokenizer(System.in);
    //
    System.out.print("Input your secret password : ");
    Input.nextToken();
    String hash = byteArrayToHexString(KeyGenerator.computeHash(Input.sval));
    System.out.println("the computed hash (hex string) : " + hash);
    boolean ok = true;
    String inputHash = "";
    while (ok) {
        System.out.print("Now try to enter a password : " );
        Input.nextToken();
        inputHash = byteArrayToHexString(KeyGenerator.computeHash(Input.sval));
        if (hash.equals(inputHash)){
           System.out.println("The Computed Hash Is Your Decryption Key!");
           ok = false;
           }
        else
           System.out.println("Wrong, try again...!");
    }
```

Figure 10. Split Knowledge Transformation

The first line represents the Java *StreamTokenizer* class *(java.io.StreamTokenizer)* which is used to can split the characters read from a *BufferedReader* into separate fragments. In this study, Java *StreamTokenizer* was used to move through the tokens in the underlying *BufferedReader*. This was done by calling the *nextToken()* method of the *StreamTokenizer* inside the *try….catch..*loop. After each call to *nextToken(),* the *StreamTokenizer* had several fields that could be read to see what kind of token was read and its value. This concept was employed to capture the split key that was used for client authentication. The authentication key essentially consisted a concatenation of the participants usernames and then getting the hexadecimal equivalence of such a concatenation, which in this case served as the decryption key as confirmed by Figure 11.

```
Scanner DecKey = new Scanner(System.in);
System.out.println("Please Enter The Decryption Key ");
String password = DecKey.next();
//Condition-1
if(!password.equals("1DC961149CD68AB9517C18A8F9D01DE3FBD74EDC") )
{

System.out.println("Wrong Decryption Key");
```

Figure 11. Client Authentication Process

```
while (true)
{
      msg = input.readLine();
      byte[] EnMsg = msg.getBytes("US-ASCII");
      if (msg.startsWith("$$"))
            break;
      for (int i = 0; i <= 9; i++)
            if (th[i] != null)
                  th[i].output.println("<" + username + ">" + EnMsg);
}
```

Figure 12. Administrator Chat Termination

Chatting activities during business hours. However, administrator termination does not expose the chats being exchange to the administrator. It only indicates that the chat server has been started, which can serve as a clue to the administrator that some employees are using this service. The authentication process for this proposed algorithm can be presented in pseudo code form as shown below.

1. Start
2. Determine client participants and concatenate their usernames, *S*
3. Transform the concatenated string into decryption key, *Deckey =Trans(S)*
4. Start the server
5. Client authenticate at the server using generated Decryption key, *Deckey*
6. Compare Client *Deckey* with server split knowledge, *SK*
7. If *Deckey = SK*
8. Decode message chats at end points
9. Else encipher messages at end points

The data flow diagram for this pseudocode is shown in Figure 13. Compared to the current end to end protection, the developed algorithm performs well in ensuring end point security.

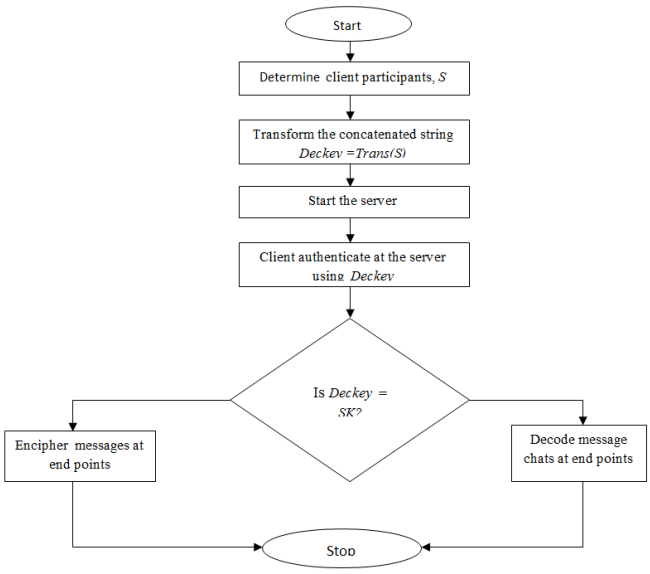


Figure 13. End Point Data Security Port-Based Data Flow Diagram

III. RESULTS AND DISCUSSION

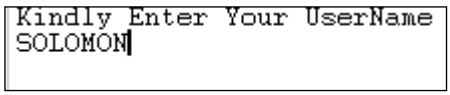
In this section, the results obtained from the development and implementation of a port-based instant messaging end point protection algorithm are presented. To start off the chat process, the server was started as sown in Figure 14.

```
Server started...
  [Press Ctrl C to terminate ]
```

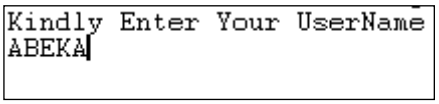
Figure 14. Chat Activation

As this figure shows, upon successful compilation, the server's Java console indicates that the server has actually been started. It additionally gives the administrator an option to terminate the chat process by pressing the *Ctrl C* keys.

After the server activation process, the clients were required to enter their usernames. These usernames were to be used to identify the participants during the cat process and also to be part of the split knowledge needed to form the decryption key. Figure 15 (a) and (b) shows these processes.



(a)



(b)

Figure 15. Client Usernames Capture

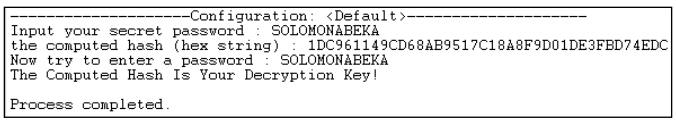


Figure 16. Decryption Key Formation

The first line shows the concatenated plaintext string while the second line shows the transformed string that would later be used as the decryption key for decoding the instant chats. To ensure that the users not easily forget this key, the application requires that they confirm the entry of this string. The second last line serves to inform the users that key

1DC961149CD68AB9517C18A8F9D01DE3FBD74EDC would be their decryption key.

The proposed algorithm was to ensure that the instant messaging chats are protected on the interfaces of the end devices. This required that the data be in a transformed format unless valid decryption keys are entered by whoever wants to access these messages. Figure 17 shows clients authenticating themselves at the server with wrong decryption keys.

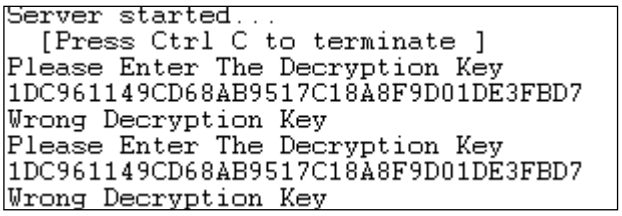


Figure 17. Client Authenticating With Wrong Decryption Keys

Note that the entered key*1DC961149CD68AB9517C18A8F9D01DE3FBD7* does not match the generated decryption key of *1DC961149CD68AB9517C18A8F9D01DE3FBD74EDC*. Therefore, the application generates two error messages for the two clients, *ABEKA* and *SOLOMON*. Consequently, these IM participants can never receive the other party's message in plaintext. Instead, the message arrives while it is transformed as shown in Figure 18.

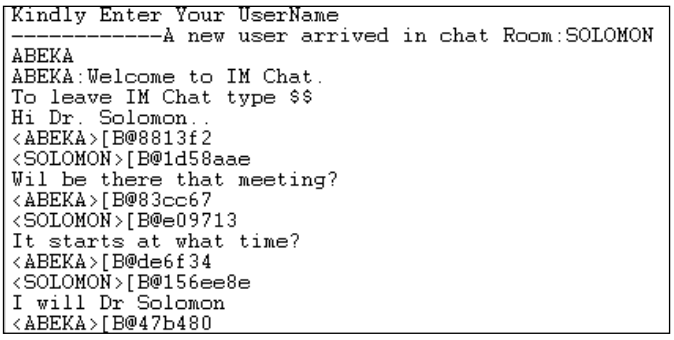


Figure 18. Transformed IM Chats- Client_1

This figure shows that the current user is *ABEKA* and the communication recipient is SOLOMON. Since this client entered an invalid decryption key, then the IM chats arriving from SOLOMON, and IM chats directed to SOLOMON will be in a transformed format. To validate this, the IM chats arriving to SOLOMON were also observed as shown in Figure 19.

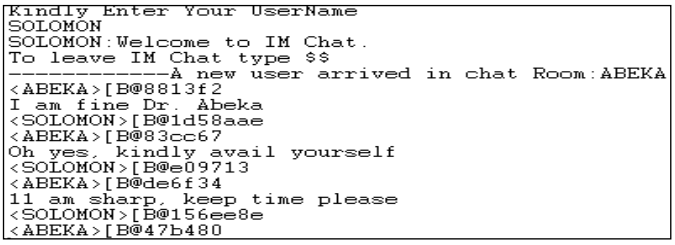


Figure 19. Transformed IM Chats- Client_2

This figure confirms that *SOLOMON* receives transformed messages from *ABEKA*. Additionally, *SOLOMON* sends transformed messages to *ABEKA*. To receive and send messages in human understandable format, both clients must use their split knowledge of the decryption key so that they can decode each other's message chats. Figure 20 shows the clients authenticating with correct decryption keys.

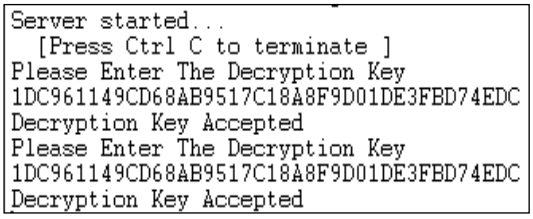


Figure 20. Clients Authenticating With Valid Decryption Keys

With proper entry of the split knowledge keys, the clients engage in plaintext kind of communication as shown in Figure 21. Therefore, the developed algorithm

managed to protect masquerading where a chat participant tries to assume the identity of another chat participant so as to get the content of the other party. This would be difficult as it will require that the intruder possess part of the split knowledge.

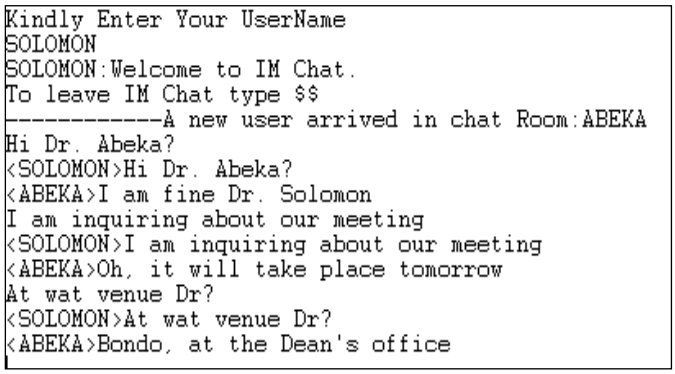


Figure 21. Client Chatting In Plaintext

In this way, end point security for the chats that arrive on the end device interface are protected from the snooping eyes of intruders. Te effect of this is that chat participants are assured that they are engaging in a chat session with true clients and not some masqueraders.

IV. CONCLUSION

Instant messaging applications have emerged as convenient means of communication among individuals both within and outside the organizations. Due to the prevalence of the so called bring your own device (BYUOD), organizations are increasingly allowing these devices access to company resources. This means that critical sensitive information may be residing in employee devices. This means that this information must be sufficiently protected from intruders. Since instant message applications are one f the avenues that can be employed to divulge organization's sensitive data, the developed algorithm sought to protect the data in the end point devices against intruders.

V. REFERENCES

[1]. M. Wendy (2013). ZDNet UK; Instant messaging boosts business.

[2]. H. Neal (2014). Semantic Security Response; Threats to Instant Messaging.

[3]. B. Nardi, S. Whittaker, E. Bradner (2012). Interaction and outeraction: instant messaging in action. Proceedings of the ACM Conference on Computer Supported Cooperative Work, Philadelphia, Pennsylvania, USA

[4]. T. Vleck (2015) . Instant Messaging on CTSS and Multics. Multicians.org.

[5]. J. Schiano, C. Kamm (2013). The character, functions and styles of instant messaging in the workplace. Proceedings of the ACM Conference on Computer Supported Cooperative Work , New Orleans, Louisiana, USA.

[6]. P. Roberts (2015). IDG News Service; MSN Messenger Worm Steals Game Keys W32/Rodok-A or Henpeck worm used via IM, then plant trojan to lift game.

[7]. Sanchez, J., (2014). Malicious Threats, Vulnerabilities and Defenses in WhatsApp and Mobile Instant Messaging Platforms.

[8]. Mahajan, A., Dahiya, M., Sanghvi, H., (2013). Forensic Analysis of Instant Messenger Applications on Android Devices. Int. J. Comput. Appl. 68, 38–44.

[9]. Anglano, C.,(2014). Forensic analysis of WhatsApp Messenger on Android smartphones. Digit. Investig. 11, 201–213.

[10]. Barghuthi, N.B. Al, Said, H., (2013). Social Networks IM Forensics: Encryption Analysis. J. Commun. 8.

[11]. Bodriagov, O., Buchegger, S.(2011). Encryption for peer-to-peer social networks. In: Proceedings - IEEE International Conference on Privacy, Security, Risk and Trust and IEEE International Conference on Social Computing, PASSAT/SocialCom 2011. pp. 1302–1309.

[12]. Ricochet P. (2014). Anonymous and serverless instant messaging that just works.

[13]. Yusof, M.K., Abidin, A.F.A., (2011). A secure private instant messenger. In: 17th Asia-Pacific Conference on Communications.

Authors Bibliography

Pursuing Msc. In Information Technology security and Audit from Jaramogi Oginga Odinga University of Science and Technology School of Informatics and Innovative System (JOOUST). His research interest is on cloud computing, Virtualization, Computer Networks Security, Modelling and simulation. He has published numerous research articles covering areas such as Analysis of VMware Hypervisor Security, IM security in a virtual Environment, Simulation of LIFI technology for secure data propagation among others. He is a career banker with bias in e-banking systems.

Dr. Solomon O. Ogara, B.Sc. (Egerton), B.Sc. (Arizona), M.Sc. (Dakota), Ph.D. (North Texas) is currently the Chairperson of the Department of Computer Science & Software Engineering, Jaramogi Oginga Odinga University Of Science And Technology. He has worked as an assistant professor of computer information system at Livingstone College. He has taught different computer information systems and networking courses including: Introduction to Computer Information Systems; Object Oriented Programming; Decision Support & Business Intelligence, Computer Architecture & Organization; System Analysis and Design, Web Design using HTML5; Database Management, Enterprise Network Design, Wired, Optical and Wireless Communications; Voice/VoIP Administration; Operating Systems with UNIX and Windows Server; Data, Privacy and Security; Principles of Information Security.

Dr. Silvance O. Abeka is currently a Senior Lecturer and a Dean- School of Informatics and Innovative Systems of Jaramogi Oginga Odinga University of Science and Technology. He worked previously as a Director- Institute of Open and Distance Learning at Africa Nazarene University and also as a Dean- Faculty of Applied Science and Technology of Kampala International University- Dar es Salaam Collage. He holds a PhD in Management Information System (MIS), Masters of Science in Computer Science from University of Da es Salaam and Master of Business Administration (Information Technology) from Kampala International University. His research interests include IT innovation adoption, open source software study, IT offshoring, Management Information Systems, Foundations of Network and System Security, Impact of Digital Technologies on Society, Networking Protocols and Topologies, Web-Design and E- Learning Technologies.