Introduction

In this report, we have written a simple program to describe EAP protocol and EAP-TLS handshake. We have used Java to implement this protocol.

Implementation

We have used the following steps in our program as shown in the figure 1.

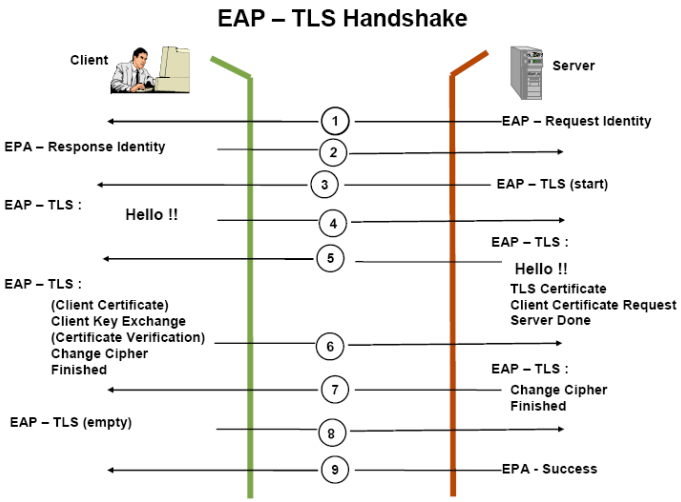

Figure 1. EAP-TLS handshake

We have used the socket programming for this client-server model. Server is listening to the port number 1000 and client connects to that port number. We have used several classes to implement this protocol.

Server Class:

Server is always listening to the port number 1000 and accepts the connection from the client and use the HandleClient class to perform the handshaking.

Code:

```
public class Server {
        public int listenPort = 1000;
        public void start() {
                ServerSocket serverSocket = null;
                try {
                serverSocket = new ServerSocket(listenPort);
                }catch (IOException ex) {
                }
                while (true) {
                        try {
                                Socket clientSocket = new Socket();
                                clientSocket = serverSocket.accept();
                                // Open a new thread handler for each request
                                HandleClient handleClient = new HandleClient(clientSocket);
                                handleClient.start();
                        } catch (Exception e) {
                        }}}
public static void main(String []args) {
                Server authenticator = new Server();
```

```
                authenticator.start();
        }}
```

HandleClient Class:
When server accepts the connection from the client and its open a thread handler for each request.

Code:

```
public class HandleClient extends Thread {
        private ObjectInputStream in = null;
        private ObjectOutputStream out = null;
        private Packet msg = null;
        private Socket sock = null;
        public HandleClient(Socket sock) {
                this.sock = sock;

/*  * We will /*
                                * Receive object from server
                                */
recieve protocol objects from the client. And also will send protocol objects to the client */
                try {
                        out = new ObjectOutputStream(this.sock.getOutputStream());
                        in = new ObjectInputStream(this.sock.getInputStream());
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
        public void sendMsg(Packet msg) {
                try {
                        out.writeObject(msg);
                } catch (Exception e) {
                        e.printStackTrace();
                }}
        /**
         * Does the EAP TLS Handshake with the client
         */
        public void run() {
                //loop until an exiting protocol message receved from the client
                while(true)
                {
                        try {
                                msg = (Packet)in.readObject();
                        } catch (Exception e) {
                        }

                        /*
                         * Parse the message and response back
                         */

                        if (msg.getType() == Config.START) {
                                Packet requestIdentity = new Packet();
                                requestIdentity.setType(Config.REQUEST_IDENTITY);
                                System.out.println("Sending : Request Identity");
                                sendMsg(requestIdentity);
                        }
```

```
else if (msg.getType() == Config.RESPONSE_IDENTITY) {
        System.out.println("Recieved : Response Identity");
        System.out.println("Sending : START");

        /*Step 3 of the diagram*/
        Packet requestIdentity = new Packet();
        requestIdentity.setControlMsg("START");
        sendMsg(requestIdentity);
}
else if ((msg.getControlMsg()).equals("HELLO!!")) {

        System.out.println("Recieved : HELLO!!");
        System.out.println("Sending : KEY EXCHANGE");

        /*
         * TLS Certificate, Client Certificate Request.
         * Server Done
         */

        /*Step 5 of the diagram*/
        Packet requestIdentity = new Packet();
        requestIdentity.setControlMsg("HELLO!!");
        sendMsg(requestIdentity);
}
else if ((msg.getControlMsg()).equals("KEY EXCHANGE")) {
        System.out.println("Recieved : KEY EXCHANGE");
        System.out.println("Sending : Cipher");
        /*Step 7 of the diagram*/
        Packet responseIdentity = new Packet();
        responseIdentity.setControlMsg("KEY EXCHANGE");
        sendMsg(responseIdentity);
}
else if ((msg.getControlMsg()).equals("EMPTY")) {
        System.out.println("Recieved : EMPTY");
        System.out.println("Sending : SUCCESS");

        /*Step 9 of the diagram*/
        Packet responseIdentity = new Packet();
        responseIdentity.setControlMsg("SUCCESS");
        sendMsg(responseIdentity);
        break;
}}}}
```

Client Class:

Client class is used to connect to perform the client operation shown in the figure 1.

Code:

```
public class Client {

        private Socket sock = null;
        private String host = "127.0.0.1";
        private int port = 1000;

        private ObjectInputStream in = null;
        private ObjectOutputStream out = null;
```

```
	public Client() {
		/*
		 * Open a socket
		 */
	try {
	    sock = new Socket(host, port);
	   } catch (UnknownHostException e) {
			System.out.println("Terminating clinet. Could not connect to server");
			System.exit(1);
	   } catch (IOException e) {
			System.out.println("Terminating clinet. Could not connect to server");
			System.exit(1);
	   }
	   /*
	    * Create the input and output object streams
	    */
			try {
				in = new ObjectInputStream(sock.getInputStream());
				out = new ObjectOutputStream(sock.getOutputStream());
			} catch (IOException e) {
				e.printStackTrace();
			}}
	/**
	 * A basic function to send a message to the server
	 *
	 * @param protoMsg
	 */
	public void sendMsg(Packet msg) {
			try {
				out.writeObject(msg);
			} catch (Exception e) {
				System.out.println("Error: Could not send un register (CBYE) Msg to the
server !");
				e.printStackTrace();
			}
	}

	/**
	 * Does the EAP - TLS Handshake with the server
	 *
	 */
	public void start() {
			Packet requestPacket = new Packet();
			requestPacket.setType(Config.START);

			/*Initialize the request */
			sendMsg(requestPacket);

			Packet msg = null;

			while (true) {
				try {
					msg = (Packet) in.readObject();

					/*
					 * Process packet
```

```
                                        */
                                        if (msg.getType() == Config.REQUEST_IDENTITY) {

                                                System.out.println("Recieved : Request Identity");
                                                System.out.println("Sending : Response Identity");

                                                /*Step 2 of the diagram*/
                                                Packet responseIdentity = new Packet();
                                                responseIdentity.setType(Config.RESPONSE_IDENTITY);
                                                sendMsg(responseIdentity);
                                        }
                                        else if ((msg.getControlMsg()).equals("START")) {
                                                System.out.println("Recieved : START");
                                                System.out.println("Sending : HELLO!!");

                                                /*Step 4 of the diagram*/
                                                Packet responseIdentity = new Packet();
                                                responseIdentity.setControlMsg("HELLO!!");
                                                sendMsg(responseIdentity);
                                        }
                                        else if ((msg.getControlMsg()).equals("HELLO!!")) {

                                                System.out.println("Recieved : HELLO!!");
                                                System.out.println("Sending : KEY EXCHANGE");

                                                /* EAP TLS  * (Client Certificate)  * client key exchange
                                                 * (Certificate Verifvication)  * Change Cipher
                                                 * Finished  */

                                                /*Step 6 of the diagram*/
                                                Packet responseIdentity = new Packet();
                                                responseIdentity.setControlMsg("KEY EXCHANGE");
                                                sendMsg(responseIdentity);
                                        } else if ((msg.getControlMsg()).equals("KEY EXCHANGE")) {
                                                System.out.println("Recieved : Cipher Finished");
                                                System.out.println("Sending : EMPTY");

                                                /*Step 8 of the diagram*/
                                                Packet responseIdentity = new Packet();
                                                responseIdentity.setControlMsg("EMPTY");
                                                sendMsg(responseIdentity);
                                        } else if ((msg.getControlMsg()).equals("SUCCESS")) {
                                                System.out.println("Recieved : SUCCESS");
                                                System.out.println("Finished");
                                                break;
                                        }
                                } catch (SocketException e) {
                                        break;
                                } catch (IOException e) {
                                        break;
                                } catch (ClassNotFoundException e) {
                                        break;
                                } }}
public static void main(String []args) {
        Client client = new Client();
        client.start();
```

}}

Packet Class

Packet class is used to identify the request and response packets and indicate the packet format of EAP.

Code

```
public class Packet implements Serializable {
	private int code;
	private int identifier;
	private int length;
	private int reqResponseData;
	private int type;
	private String controlMsg;
	public int getCode() {
		return code;
	}
	public void setCode(int code) {
		this.code = code;
	}
	public int getIdentifier() {
		return identifier;
	}
	public void setIdentifier(int identifier) {
		this.identifier = identifier;
	}
	public int getLength() {
		return length;
	}
	public void setLength(int length) {
		this.length = length;
	}
	public int getReqResponseData() {
		return reqResponseData;
	}
	public void setReqResponseData(int reqResponseData) {
		this.reqResponseData = reqResponseData;
	}
	public int getType() {
		return type;
	}
	public void setType(int type) {
		this.type = type;
	}
	public String getControlMsg() {
		return controlMsg;
	}
	public void setControlMsg(String controlMsg) {
		this.controlMsg = controlMsg;
	}}
```

Summary

By this assignment we have got a solid background how EAP-TLS works and implemented the EAP-TLS. Our assignment code can be found here:
http://people.dsv.su.se/~islam3/IK2002/EAP_TLS_Handshake.zip