

Contributions

Group: Bryan Endres ID: #8 & Andrew Bradley ID : #2	Contribution Percentage (100% Total)	Contribution details.
Andrew Bradley #2	50%	<p>1) Figured out the logic of the Server code</p> <p>2) Worked with Bryan to code the project on one centralized monitor. While Bryan typed code Andrew suggested different ideas and approaches and helped debug code. At times the two split off onto their own laptops for in-depth debugging and then gathered their findings together.</p>
Bryan Endres #8	50%	<p>1) Figured out the logic of the client code</p> <p>2) Worked with Andrew to code the project on one centralized monitor. Bryan typed while Andrew gave input and also supplied his own input along the way. The two split off at times but generally used one laptop for coding the project as it was the easiest way to keep the current version maintained.</p>

Part 1: SSL Protocol Definition With Detailed Examples

1. Types of messages exchanged, e.g., request, response
 - a. Handshake message
 - b. Data Message (mostly in main)
2. Message syntax: what fields in messages & how fields are delineate
 - a. Passed packets as delimited strings using ';' in between different fields of the packet.
 - b. Delimited portions of the message using ';' and ',' as delimiters, but these were converted to BigInteger before being placed in the packet, thus keeping them from interfering with the packet field delimiters
 - c. Delimited RSA keys using '_' to differentiate from ';' and ',' in the message field.

length	type	data	MAC
--------	------	------	-----

3. Message semantics: meaning of information in fields
 - a. Client
 - i. RSAKeys are used to hold the keys for RSA encryption
 - ii. NONCE is created using a Random object
 - iii. ASCII is used to convert between Strings and BigIntegers
 - iv. PACKETS holds a list of transported packets
 - v. MAC authenticates the packets
 - vi. CASEx variables determine which ciphers are to be used
 - vii. IV is the Initialization Vector for CBC
 - viii. SUB_KEY is the substitution key for Substitution Cipher
 - b. Server
 - i. Ciphers is an array that holds the different ciphers that Sever holds
 - ii. RSAKeys are used to hold the keys for RSA encryption
 - iii. NONCE is created using a Random object
 - iv. ASCII is used to convert between Strings and BigIntegers
 - v. PACKETS holds a list of transported packets
 - vi. MAC authenticates the packets
 - vii. CASEx variables determine which ciphers are to be used
 - viii. IV is the Initialization Vector for CBC
 - ix. SUB_KEY is the substitution key for Substitution Cipher

4. Rules for when and how processes send & respond to messages

a. Rules for handshaking

1. Client sends **list of algorithms** it supports, along with **client nonce**
2. Server chooses algorithms from list; sends back: **choice + certificate + server nonce**
3. Client verifies certificate, extracts server's **public key**, generates **pre_master_secret**, encrypts with server's public key, sends to server
4. Client and server **independently** compute **encryption** and **MAC** keys from **pre_master_secret** and **nonces**
5. Client sends a **MAC** of **all** the handshake messages
6. Server sends a **MAC** of **all** the handshake messages
7. Client and server **independently** verify **the** MAC received from each other **and** establish the secure SSL Connection

b. Rules for data transfer and communication

- i. No need to use RSA at this stage, keys have already been exchanged
- ii. Used selected cipher to encrypt/decrypt messages
- iii. Used MAC to make sure that messages were not altered during transmission

c. Rules for connection closure.

- i. QUIT command initiated a 0 to be placed in front of the message which indicates that a connection closure was accepted.
- ii. All other packets were sent with a 1 in front of the message indicating that these were data records.
- iii. Upon receipt, each end device closes their connection.

Part 2: Interface Design and Implementation.

<u>Class ChatClient</u>	
Define members	<pre>/** * Port number on server, if none is specified on the command line. */ static final int DEFAULT_PORT = 1728; /** * Handshake string. Each end of the connection sends this string to the * other just after the connection is opened. This is done to confirm that * the program on the other side of the connection is a CLChat program. */ static final String HANDSHAKE = "CIS435535"; /** * This character is prepended to every message that is sent. */ static final char MESSAGE = '0'; //more like the type in SSL /** * This character is sent to the connected program when the user quits. */ static final char CLOSE = '1'; //more like the type in SSL //Created a random nonce static final Random RAND = new Random(); static final BigInteger NONCE = BigInteger.valueOf(Math.abs(RAND.nextInt())); //Initialize ASCII Converter static final AsciiConverter ASCII = new AsciiConverter(); //Create Packet array list static final ArrayList<Packet> PACKETS = new ArrayList<>(); //Initialize MAC static final MAC MAC = new MAC();</pre>

	<pre>//Create RSA static final RSA RSA = new RSA(); static final RSAKey PUBLIC_KEY = RSA.getPublicKey(); static final RSAKey PRIVATE_KEY = RSA.getPrivateKey(); //Case Variables private static final int CASE1 = 1; private static final int CASE2 = 2; private static final int CASE3 = 3; private static final int CASE4 = 4; private static final int CASE5 = 5; //Initialization Vector for CBC static String IV; //Substitution Key for Substitution Cipher static char[] SUB_KEY;</pre>
(Major) Method #1	<p>“Handles communication logic”</p> <pre>public static void main(String[] args)</pre>
	No input
	<p>Sample Output:</p> <pre>***Setup Connection*** SEND: data ***Calculation Results*** RECEIVED: serverData</pre>
(Major) Method #2	<p>* Converts a given string into a packet object.</p> <pre>public static Packet getPacket(String incoming)</pre>

	<p>Sample Input</p> <ul style="list-style-type: none"> · packetString = 111564987987;66549898165987;651321687987
	<p>Sample Output</p> <ul style="list-style-type: none"> · No output. Returns packet. ·
(Major) Method #3 etc	<p><u>Method Signature and Description</u></p> <p>* Converts a given packet to a String that can be sent over a network.</p> <p>public static String preparePacket(Packet packet)</p>
	<p>Sample Input</p> <p>A Packet object</p>
	<p>Sample Output</p> <p>No output. Returns a string representation.</p> <p>E.g., “111564987987;66549898165987;651321687987”</p>

<u>Class ChatClient (cont)</u>	
(Major) Method #4	<p>* Converts an RSA Key to a String representation</p> <p>public static String keyToString(RSAKey key)</p>

	An RSAKey
	<p>No OutPut</p> <p>Returns a String Represetnation</p> <p>4648949886468_84687894651687</p>
(Major) Method #5	<p>* Converts an RSA Key String to an RSAKey</p> <p>public static RSAKey keyFromString(String keyString)</p>
	<p>Sample Input</p> <p>A string representation of the RSAKey</p>
	<p>Sample Output</p> <p>· No output. Returns RSAKey</p> <p>·</p>
(Major) Method #6	<p><u>Method Signature and Description</u></p> <p>* Encrypts the message using different forms of encryption then</p> <p>* authenticates using MAC before sending the code</p> <p>public static Packet getMessagePacket(String message, int testCase, BigInteger Kc, RSAKey serverPublicKey)</p>
	<p>Sample Input</p> <p>“Message”, 3, 9848461897987, RSAKey</p>

	<p>Sample Output</p> <p>No output. Returns a packet created with given information</p>
--	--

(Major) Method #7	<p><u>Method Signature and Description</u></p> <p>* Check the integrity of the message, only decrypt if MAC is authentic.</p> <p>public static String getMessage(Packet packet, BigInteger Ks, int testCase)</p>
	<p>Sample Input</p> <p>Packet, 498489416518949889, 3</p>
	<p>Sample Output</p> <p>No output. Decrypts and returns the original message</p>

<p><u>Class ChatServer</u></p>

Define members

```
/**
 * Port number on server, if none is specified on the command line.
 */
static final int DEFAULT_PORT = 1728;

/**
 * Handshake string. Each end of the connection sends this string to the
 * other just after the connection is opened. This is done to confirm that
 * the program on the other side of the connection is a CLChat program.
 */
static final String HANDSHAKE = "CIS435535";

/**
 * This character is prepended to every message that is sent.
 */
static final char MESSAGE = '0'; //more like the type in SSL

/**
 * This character is sent to the connected program when the user quits.
 */
static final char CLOSE = '1'; //more like the type in SSL

//Created a random nonce
static final Random RAND = new Random();
static final BigInteger NONCE =
BigInteger.valueOf(Math.abs(RAND.nextInt()));

//Initialize ASCII Converter
static final AsciiConverter ASCII = new AsciiConverter();

//Create Packet array list
static final ArrayList<Packet> PACKETS = new ArrayList<>();

//Initialize MAC
static final MAC MAC = new MAC();

//Create RSA
static final RSA RSA = new RSA();
static final RSAKey PUBLIC_KEY = RSA.getPublicKey();
static final RSAKey PRIVATE_KEY = RSA.getPrivateKey();

//Case Variables
```

	<pre> private static final int CASE1 = 1; private static final int CASE2 = 2; private static final int CASE3 = 3; private static final int CASE4 = 4; private static final int CASE5 = 5; static final String[] CIPHERS = { "2", "3", "4", "5" }; //Initialization Vector for CBC static String IV; //Substitution Key for Substitution Cipher static char[] SUB_KEY; </pre>
(Major) Method #1	<p>“Handles communication logic”</p> <pre> public static void main(String[] args) </pre>
	No input
	<p>Sample Output:</p> <pre> ***Setup Connection*** SEND: data ***Calculation Results*** RECEIVED: serverData </pre>
(Major) Method #2	<p>* Converts a given string into a packet object.</p> <p>.</p> <pre> public static Packet getPacket(String incoming) </pre>

	<p>Sample Input</p> <ul style="list-style-type: none"> · packetString = 111564987987;66549898165987;651321687987
	<p>Sample Output</p> <ul style="list-style-type: none"> · No output. Returns packet. ·
(Major) Method #3 etc	<p><u>Method Signature and Description</u></p> <p>* Converts a given packet to a String that can be sent over a network.</p> <p>public static String preparePacket(Packet packet)</p>
	<p>Sample Input</p> <p>A Packet object</p>
	<p>Sample Output</p> <p>No output. Returns a string representation.</p> <p>E.g., “111564987987;66549898165987;651321687987”</p>

<u>Class ChatServer (cont)</u>	
(Major) Method #4	<p>* Converts an RSA Key to a String representation</p> <p>public static String keyToString(RSAKey key)</p>

	An RSAKey
	<p>No OutPut</p> <p>Returns a String Represetnation</p> <p>4648949886468_84687894651687</p>
(Major) Method #5	<p>* Converts an RSA Key String to an RSAKey</p> <p>public static RSAKey keyFromString(String keyString)</p>
	<p>Sample Input</p> <p>A string representation of the RSAKey</p>
	<p>Sample Output</p> <p>· No output. Returns RSAKey</p> <p>·</p>
(Major) Method #6	<p><u>Method Signature and Description</u></p> <p>* Encrypts the message using different forms of encryption then</p> <p>* authenticates using MAC before sending the code</p> <p>public static Packet getMessagePacket(String message, int testCase, BigInteger Kc, RSAKey serverPublicKey)</p>
	<p>Sample Input</p> <p>“Message”, 3, 9848461897987, RSAKey</p>

	<p>Sample Output</p> <p>No output. Returns a packet created with given information</p>
--	--

(Major) Method #7	<p><u>Method Signature and Description</u></p> <p>* Check the integrity of the message, only decrypt if MAC is authentic.</p> <p>public static String getMessage(Packet packet, BigInteger Ks, int testCase)</p>
	<p>Sample Input</p> <p>Packet, 498489416518949889, 3</p>
	<p>Sample Output</p> <p>No output. Decrypts and returns the original message</p>

Part 3: Designed/Expected Output.

Please follow the format of Phase #2 Sample Output as shown in to Design the Expected Output for each of the **Three** Major Phases (which combines the phase #1 (handshake) and #2(key derivation) to
Phase #1 Handshaking and Secure Connection Establishment)

***Phase # and Step # with purpose and results are required in the program output**

Phase #1: Handshaking and Secure Connection Establishment

Step #	Client	Server
1	<p>Client sends clientPacket01 with the following data</p> <ul style="list-style-type: none">· Shift, RSA, MAC, Digital Signature· Substitution, RSA, MAC, Digital Signature· Polyalphabetic, RSA, MAC, Digital Signature· Client Nonce (Nc): random number, e.g., Nc = 555 <p>PacketQueue</p> <p>1. clientPacket01</p>	

2	<div data-bbox="332 420 609 535"> <div>PacketQueue</div> <div>clientPacket01</div> <div>serverPacket01</div> </div>	<p>Server 1) receives and saves clientPacket01 and 2) sends and saves serverPacket01</p> <ul style="list-style-type: none"> · Shift, RSA, MAC, Digital Signature · Polyalphabetic, RSA, MAC, Digital Signature · Block Cipher, RSA, MAC, Digital Signature · Ps-(Nc = 555), i.e., Ps-(111) + (Ps+) · Server Nonce (Ns): random number, e.g., Ns = 666 <div data-bbox="885 819 1161 934"> <div>PacketQueue</div> <div>clientPacket01</div> <div>serverPacket01</div> </div>
---	---	---

3	<p>Client verifies certificate, extracts server's public key</p> <p>· $P_s + (P_s - (N_c = 555)) = 555$</p> <p>Client generates pre_master_secret, encrypts with server's public key, sends to server, created by using a random number</p> <p>Example: pre_master_secret (pms)= 1234 and Sends clientPacket02</p> <p>· $P_s + (1234)$</p> <p>PacketQueue</p> <ul style="list-style-type: none">clientPacket01serverPacket01clientPacket02	
---	--	--

4	<p>Client extracts data from Packets</p> <ul style="list-style-type: none"> · pre_master_secret = 1234, · Nc = 555, · Ns = 666, <p>Generate the following keys, for example</p> $1234 * 555 * 666 = 456123420$ <ol style="list-style-type: none"> 1. Kc = $456123420 / 2$ 2. Mc = $456123420 ^ 2$ 3. Ks = $456123420 * 2$ 4. Ms = find the next probable prime from Ks. <p>PacketQueue</p> <ul style="list-style-type: none"> · clientPacket01 · serverPacket01 · clientPacket02 	<p>1 Server receives clientPacket02,</p> <ul style="list-style-type: none"> · Ps+(1234) <p>2. Server extracts data from Packets</p> $Ps-(Ps+(1234)) = 1234$ <ul style="list-style-type: none"> · re_master secret = 1234 , · Nc = 555, · Ns = 666, <p>Generate the following keys, for example</p> $1234 * 555 * 666 = 456123420$ <p>Take a factor of 2</p> <ol style="list-style-type: none"> 1. Kc = $456123420 / 2$ 2. Mc = $456123420 ^ 2$ 3. Ks = $456123420 * 2$ 4. Ms = find the next probable prime from Ks. <p>packetQueue</p> <ul style="list-style-type: none"> · clientPacket01 · serverPacket01 · clientPacket02
---	--	--

5	<p>Client sends a MAC of all the handshake messages</p> <p>MAC_c = MAC(Mc, clientPacket01, serverPacket01, clientPacket02)</p> <p>PacketQueue</p> <ul style="list-style-type: none"> clientPacket01 serverPacket01 clientPacket02 	
6	<p>Server sends a MAC of all the handshake messages</p> <p>MAC_s = MAC(Ms, clientPacket01, serverPacket01, clientPacket02)</p> <p>PacketQueue</p> <ul style="list-style-type: none"> clientPacket01 serverPacket01 clientPacket02 	

7	<p>Calculate MACs = MAC(Ms, clientPacket01, serverPacket01, clientPacket02)</p> <p>If MACs (Calculated) == MACs (received)</p> <p>Secure Connection Established</p>	<p>Calculate MACc = MAC(Mc, clientPacket01, serverPacket01, clientPacket02)</p> <p>MACc (Calculated) == MACc (received)</p> <p>Secure Connection Established</p>
---	---	--

Example Output:

(CLIENT)

SEND: (User Gives Input) "hello"

(RECIEVER)

RECEIVED: hello

SEND: (User Gives Input) "how are you?"

(CLIENT)

RECEIVED: how are you?

Etc.....