Contributions

| Group: Bryan Endres ID: #8 & Andrew Bradley ID: #2 | Contribution Percentage (100% Total) | Contribution details. |
|--|---------------------------------------|---|
| Andrew Bradley #2 | 50% | 1) Figured out the logic of the Server code 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. |
| Bryan Endres #8 | 50% | 1) Figured out the logic of the client code 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. |

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 tranported 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 tranported 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
 - 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 conneciton.

Part 2: Interface Design and Implementation.

```
Class ChatClient
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
                          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;
(Major)
                        "Handles communication logic"
Method #1
                        public static void main(String[] args)
                        No input
                        Sample Output:
                        ***Setup Connection***
                        SEND: data
                        ***Calculation Results***
                        RECEIVED: serverData
(Major)
                           * Converts a given string into a packet object.
Method #2
                        public static Packet getPacket(String incoming)
```

| | Sample Input - packetString = 111564987987;66549898165987;651321687987 |
|--------------------------|---|
| | Sample Output No output. Returns packet. . |
| (Major) Method #3 etc | * Converts a given packet to a String that can be sent over a network. public static String preparePacket(Packet packet) |
| | Sample Input A Packet object Sample Output |
| | No output. Returns a string representation. E.g., "111564987987;66549898165987;651321687987" |

| Class ChatClient (| <u>cont)</u> |
|----------------------|--|
| (Major) Method #4 | * Converts an RSA Key to a String representation public static String keyToString(RSAKey key) |
| | public state builty key robuilty (Korikey key) |

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

| No output. Returns a packet created with given information |
|--|
| * Check the integrity of the message, only decrypt if MAC is authentic. public static String getMessage(Packet packet, BigInteger Ks, int testCase) |
| Sample Input Packet, 498489416518949889, 3 Sample Output No output. Decrypts and returns the original message |
| |

Sample Output

Class ChatServer

```
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
```

```
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;
(Major)
                        "Handles communication logic"
Method #1
                        public static void main(String[] args)
                        No input
                        Sample Output:
                        ***Setup Connection***
                        SEND: data
                         ***Calculation Results***
                        RECEIVED: serverData
(Major)
                           * Converts a given string into a packet object.
Method #2
                         public static Packet getPacket(String incoming)
```

| | Sample Input packetString = 111564987987;66549898165987;651321687987 | |
|--------------------------|---|--|
| | Sample Output No output. Returns packet | |
| (Major) Method #3 etc | * Converts a given packet to a String that can be sent over a network. public static String preparePacket(Packet packet) | |
| | Sample Input A Packet object Sample Output | |
| | No output. Returns a string representation. E.g., "111564987987;66549898165987;651321687987" | |

| Class ChatServer (cont) | | |
|-------------------------|--|--|
| (Major) Method #4 | * Converts an RSA Key to a String representation | |
| | public static String keyToString(RSAKey key) | |

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

| Sample Output |
|--|
| No output. Returns a packet created with given information |
| |

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

Part 3: Designed/Expected Output.

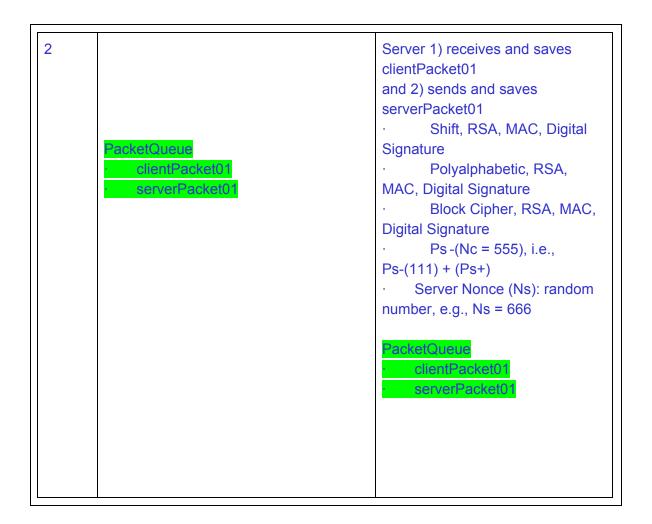
Please follow the format of Phase #2 Sample Output as shown in to Design the Expected Output for each of the Three Major Phases (which combines the phase #1 (handshake) and #2(key derivation) to

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 | Client sends clientPacket01 with the following data Shift, RSA, MAC, Digital Signature Substitution, RSA, MAC, Digital Signature Polyalphabetic, RSA, MAC, Digital Signature Client Nonce (Nc): random number, e.g., Nc = 555 PacketQueue clientPacket01 | |



Client verifies certificate, extracts server's public key

Ps+(Ps-(Nc = 555)) = 555

Client generates pre_master_secret, encrypts with server's public key, sends to server, created by using a random number

Example: pre_master_secret (pms)=
1234 and Sends clientPacket02
Ps+(1234)

PacketQueue

clientPacket01

serverPacket01

clientPacket02

4 Client extracts data from Packets

- pre_master_secret = 1234,
- \cdot Nc = 555,
- · Ns = 666,

Generate the following keys, for example

- 1. Kc = 456123420 / 2
- 2. Mc = 456123420 ^ 2
- 3. Ks = 456123420 * 2
- 4. Ms = find the next probable prime from Ks.

PacketQueue

clientPacket01

serverPacket01

clientPacket02

- 1 Server receives clientPacket02,
- Ps+(1234)
- 2. Server extracts data from Packets

$$Ps-(Ps+(1234)) = 1234$$

- re master secret = 1234,
- \cdot Nc = 555,
- · Ns = 666,

Generate the following keys, for example

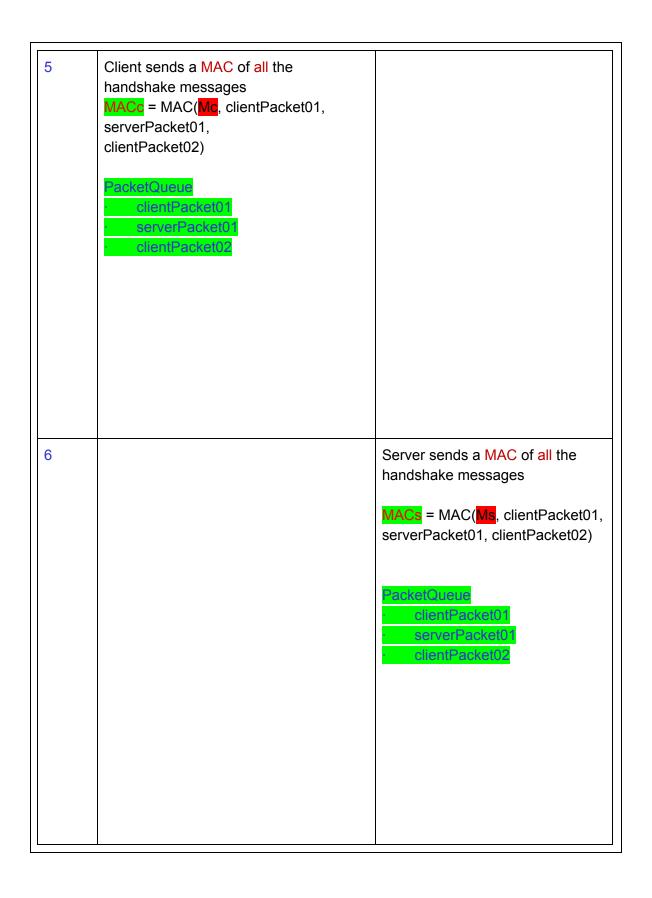
- 1. Kc = 456123420 / 2
- 2. Mc = 456123420 ^ 2
- 3. Ks = 456123420 * 2
- 4. Ms = find the next probable prime from Ks.

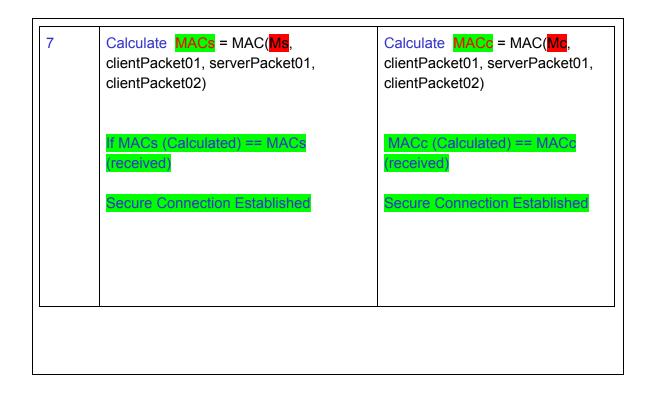
packetQueue

clientPacket01

serverPacket01

clientPacket02





| Example Output: |
|---|
| (CLIENT) |
| SEND: (User Gives Input) "hello" |
| (RECIEVER) |
| RECEIVED: hello SEND: (User Gives Input) "how are you?" |
| (CLIENT) |
| RECEIVED: how are you? |
| Etc |