Michael Manzanares
Toan Ton
03/25/18

Delegation:

Michael Manzanares: Parts 1-5. Michael added parts to Client Part 6 and advised Toan on what classes should be used as template for part 6 & 7.

Toan Ton: Parts 6&7, edited Parts 1-5. Toan edited each part as well to ensure that tasks have been completed.

Part1:

The first part is to create 8 keys, four are going to be private and the other four are public. These were name for simplicity and per the assignment tasks. They were named relevant to ourselves. Two pairs were made for the server. Three pairs of signature keys were made to go along each public and private key. Since there are two of us, a pair of public and private keys were made with a pair of signature keys for each user in this team. We decided that creating another pair for each user was a good idea. There are an extra pair of keys more than required.

```
PrivateKey michaelManzprivKey = key.getPrivate();
PublicKey michaelManzpubKey = key.getPublic();
PrivateKey SignManzprivKey = key.getPrivate();
PublicKey SignManzpubKey = key.getPublic();

PrivateKey toanTonprivKey = key.getPrivate();
PublicKey toanTonpubKey = key.getPublic();
PrivateKey SignTonprivKey = key.getPrivate();
PublicKey SignTonpubKey = key.getPublic();

PublicKey serverPubKey = key.getPublic();
PrivateKey serverPvtKey = key.getPrivate();
PublicKey serverSignPubKey = key.getPublic();
PrivateKey serverSignPvtKey = key.getPrivate();



try {
   PemUtils.writePublicKey(michaelManzpubKey, "michaelManzpubKey.pem");
} catch (FileNotFoundException e) {
   System.out.println("Write Public Key: File not found Exception");
}
try {
   PemUtils.writePublicKey(SignManzpubKey, "SignManzpubKey.pem");
} catch (FileNotFoundException e) {
   System.out.println("Write Public Key: File not found Exception");
```

```
}
try {
   PemUtils.writePublicKey(toanTonpubKey, "toanTonpubKey.pem");
} catch (FileNotFoundException e) {
   System.out.println("Write Public Key: File not found Exception");
}

try {
   PemUtils.writePrivateKey(toanTonprivKey, "toanTonprivKey.pem");
} catch (FileNotFoundException e) {
   System.out.println("Write Private Key: File not found Exception");
}
try {
   PemUtils.writePublicKey(SignTonpubKey, "SignTonpubKey.pem");
} catch (FileNotFoundException e) {
   System.out.println("Write Public Key: File not found Exception");
}
try {
   PemUtils.writePublicKey(serverPubKey, "serverPubKey.pem");
} catch (FileNotFoundException e) {
   System.out.println("Write Public Key: File not found Exception");
}

try {
   PemUtils.writePrivateKey(serverPvtKey, "serverPvtKey.pem");
} catch (FileNotFoundException e) {
   System.out.println("Write Private Key: File not found Exception");
}
try {
   PemUtils.writePublicKey(serverSignPubKey, "serverSignPubKey.pem");
} catch (FileNotFoundException e) {
   System.out.println("Write Public Key: File not found Exception");
}

try {
   PemUtils.writePrivateKey(serverSignPvtKey, "serverSignPvtKey.pem");
} catch (FileNotFoundException e) {
   System.out.println("Write Private Key: File not found Exception");
}
```

Above are the 12 different keys that are generated for this assignment. As stated before four are private and four are public. Two pairs were created for the servers, one pair for the server and another pair for the servers signature. Two pairs were used as the signature to retrieve the certificate from the designated website. They were modified to include the date I submitted for the certification. In this assignment we are using the same keys for encryption and for the

signature. This is made clear that it is only for the assignment purposes, whereas it would be two different keys. Toan Ton had a pair of public/private keys and a pair of public/private signature. Same was done with Michael Manzanares. One pair was made for the Server. The other pair was for the signature server.

Question to answer: "Given a string with length of n , the base64 length will be 4[n/3]"[1]. This basically means that each char in the string is 6 bit (log2(64))=6)[2]. Every char is 1 byte or 8 bits. From the formula mentioned it is seen that a 4/3 would result in above a 1.0. It rounds up. If a 6 string.length is encoded a result would equal to 8 string.length. It rounds up to the nearest multiple of 4.

Part 2:

The certificates pertain to the users such as Manzanares had the "SignManzpubKey.pem" and "michaelManzpubkey.pem" for his certificate. Toan Ton had the contrast, "SignTonpubKey.pem" and "toanTonpubkey.pem". These certificates were verified. Screenshots will be provided below.

```
-----BEGIN INFORMATION-----
Date: 25-Mar-18
Name: MichaelManzanares
Username: mmanzanares
-----END INFORMATION-----
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCsSCmimGXetRSIzxwShYjDfV8y
ZiHZKcmS3Knvat3wnMyQTlTU2vo3M3vxbDl5IoogZq0TqLRTI7SFAz/N1Qz8d9m3
GfK9ky824fz9qqyGjJtcEekG6wzuG/ouFM1KVTbTeCLA9ojeTuQZP2Dxo8v4BWb5
3T0gn8oDnV6dDHh9AwIDAQAB
-----END PUBLIC KEY-----

-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCsSCmimGXetRSIzxwShYjDfV8y
ZiHZKcmS3Knvat3wnMyQTlTU2vo3M3vxbDl5IoogZq0TqLRTI7SFAz/N1Qz8d9m3
GfK9ky824fz9qqyGjJtcEekG6wzuG/ouFM1KVTbTeCLA9ojeTuQZP2Dxo8v4BWb5
```

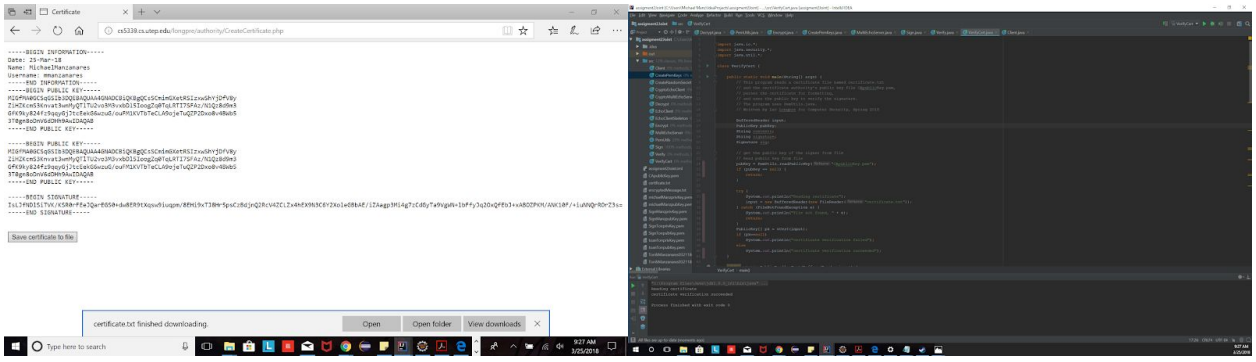
[1] 1.Namir
[2] 1.Namir

```
3T0gn8oDnV6dDHh9AwIDAQAB
-----END PUBLIC KEY-----

-----BEGIN SIGNATURE-----
IsLJfNDlSiTVK/KSR0rFEeJQarEGS0+dw8ER9tXqsw9iuqpm/8EHi9xTJ8Hr5psCzBdjnQ2RcV4ZCLZx4hEX9
N3C6Y2XoleGBbAE/iZAagp3Mi4g7zCdGyTa9VgWN+lbFfyJq2OxQfEbJ+xABOZPKM/ANK10F/+iuNNQrROrZ3
s=
-----END SIGNATURE-----

-----BEGIN INFORMATION-----
Date: 25-Mar-18
Name: Toan Ton
Username: thton
-----END INFORMATION-----
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCsSCmimGXetRSIzxwShYjDfV8y
ZiHZKcmS3Knvat3wnMyQTlTU2vo3M3vxbDl5IoogZq0TqLRTI7SFAz/N1Qz8d9m3
GfK9ky824fz9qqyGjJtcEekG6wzuG/ouFM1KVTbTeCLA9ojeTuQZP2Dxo8v4BWb5
3T0gn8oDnV6dDHh9AwIDAQAB
-----END PUBLIC KEY-----
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCsSCmimGXetRSIzxwShYjDfV8y
ZiHZKcmS3Knvat3wnMyQTlTU2vo3M3vxbDl5IoogZq0TqLRTI7SFAz/N1Qz8d9m3
GfK9ky824fz9qqyGjJtcEekG6wzuG/ouFM1KVTbTeCLA9ojeTuQZP2Dxo8v4BWb5
3T0gn8oDnV6dDHh9AwIDAQAB
-----END PUBLIC KEY-----
-----BEGIN SIGNATURE-----
Kcs/Yfzpynuls6L7zBtM0AiQtRBwQBnOaZIZgZkpH4sBfuHhUD3jpMzkEN6dxconMvqjaFfm5W
7hptD89RPclMykZiEaupfr05FzRxBI2xOrZIE+Q5V01noUYsHFgUTd2f4X80vZ4QGZt1ecVsg/
FtimmbcUmiw1HJ8ZQaVPI0Q=
-----END SIGNATURE-----
This is the certificate made for the server use in Part 6.
-----BEGIN INFORMATION-----
Date: 25-Mar-18
Name: Toan Ton
```

```
Username: thton
-----END INFORMATION-----
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCOXRyXvw9Ei+1S/ZPQPynPYwkU
yhIryPEznElw3trEgJAI56xgXBSwWk2hHv5HKZEO/1IEP+tJV6RfprGiP3+VQRH+
s+chqmNDdCFrh3CZSL0v8FRsTrT4kp2/0eVp0KT5Uj8vJtuWdpJAc0uNNGiYr+NT
7Esm6xGThhsDoo3e5QIDAQAB
-----END PUBLIC KEY-----

-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCNajyx3FCr6Amjv6gQaQOrGD7z
u/zoa6Um0MtYibodh0nI9Fgp4Le895JXGLb5Ik3uIhjyExIJnudzHC8jMsZSLIiM
zRW8YUTH67lxsE535a2we7OqCsz8BrVBJTIEEVj58ZELCaSFQzeOaDLSR/cEcZ/j
/jXW2uDQyC2gQ8c6zQIDAQAB
-----END PUBLIC KEY-----

-----BEGIN SIGNATURE-----
OqHsMzeJcz0MjJK8Q0l3usbW96U8BWYg7Kq+UYv2qqP5InNNNn14kABOkjl9qBPBSU+5PBX3xS
tMNnLxzTpAUd+RUlOtpn/8WdcNU7jHo7uZIICvP1PKWuJmGii7JCK8HruoFqldtF4xTSx7tdaI
DqYuEuh0Z9ee7k1ALcUtL9M=
-----END SIGNATURE-----
-----BEGIN INFORMATION-----
Date: 26-Mar-18
Name: MichaelManzanares
Username: mmanzanares
-----END INFORMATION-----
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCNajyx3FCr6Amjv6gQaQOrGD7z
u/zoa6Um0MtYibodh0nI9Fgp4Le895JXGLb5Ik3uIhjyExIJnudzHC8jMsZSLIiM
zRW8YUTH67lxsE535a2we7OqCsz8BrVBJTIEEVj58ZELCaSFQzeOaDLSR/cEcZ/j
/jXW2uDQyC2gQ8c6zQIDAQAB
-----END PUBLIC KEY-----

-----BEGIN PUBLIC KEY-----
```

```
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCNajyx3FCr6Amjv6gQaQOrGD7z
u/zoa6Um0MtYibodh0nI9Fgp4Le895JXGLb5Ik3uIhjyExIJnudzHC8jMsZSLIiM
zRW8YUTH67lxsE535a2we7OqCsz8BrVBJTIEEVj58ZELCaSFQzeOaDLSR/cEcZ/j
/jXW2uDQyC2gQ8c6zQIDAQAB
-----END PUBLIC KEY-----

-----BEGIN SIGNATURE-----
YwTn0lr4o0g49CYvITu6CjVEaMg0GOY2+nNVsgjXKXhnibQxkeyXJrjQ1012/Pi046oKzJ/8zg
2HaLBRyFwU0YfQhxTlGDgwI1Hyx/mFs6+7XZAOUF7AaJHzJmTQXCdQNGize0onj2F5TRdQVsm8
E4Vy6yAtyhdwtBgvEweydNU=
-----END SIGNATURE-----
```

Part 3:



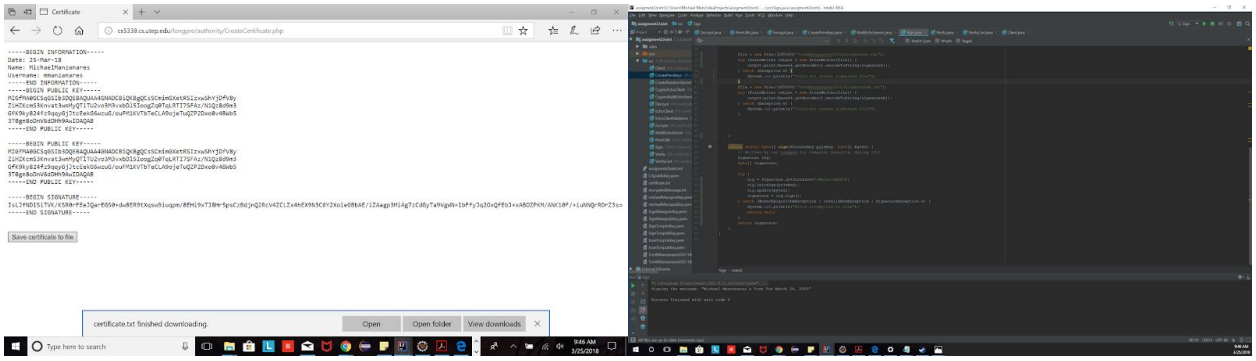
Here in the “Sign” class we utilize the private keys, with the altered signatures in accordance to the assignment.

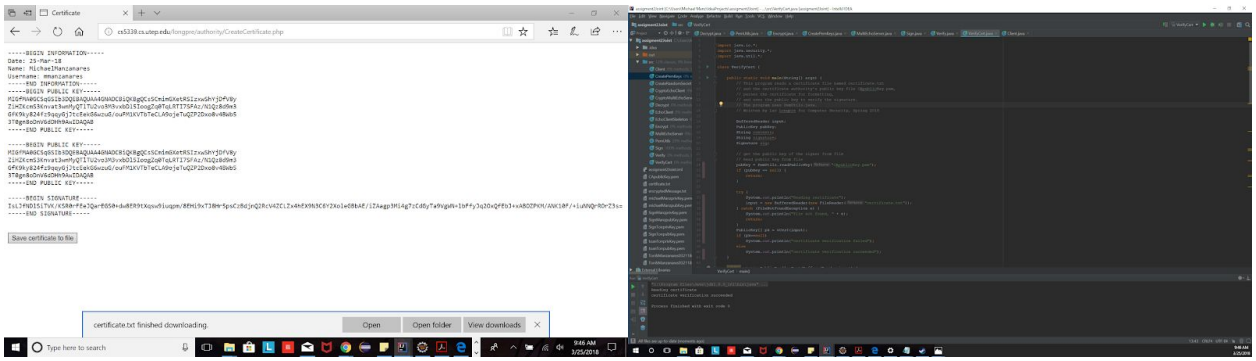
```
// Read private key from file
privKey = PemUtils.readPrivateKey("michaelManzprivKey.pem");
privKeyb = PemUtils.readPrivateKey("toanTonprivKey.pem");
privKeyc = PemUtils.readPrivateKey("SignManzprivKey.pem");
privKeyd = PemUtils.readPrivateKey("SignTonprivKey.pem");

signature = sign(privKey, messageToSign.getBytes());
signatureb = sign(privKeyb, messageToSign.getBytes());
```

Michael Manzanares
Toan Ton
03/25/18

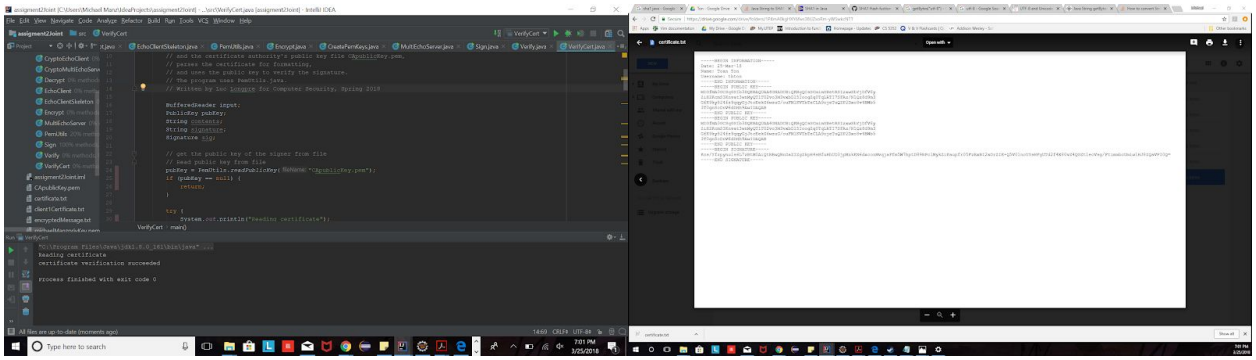
```
file = new File("Ton&Manzanares032118signature.txt");
try (PrintWriter output = new PrintWriter(file)) {
    output.print(Base64.getEncoder().encodeToString(signature));
} catch (Exception e) {
    System.out.println("Could not create signature file");
}
file = new File("Ton&Manzanares032118signatureB.txt");
try (PrintWriter output = new PrintWriter(file)) {
    output.print(Base64.getEncoder().encodeToString(signatureb));
} catch (Exception e) {
    System.out.println("Could not create signature file");
}
```

When these classes were ran, they were ran with the credentials of mmanzanares. The first three pictures of those are listed. Toan Ton are the next 3 following. Both classes were ran using both credentials or designated variables to be used.

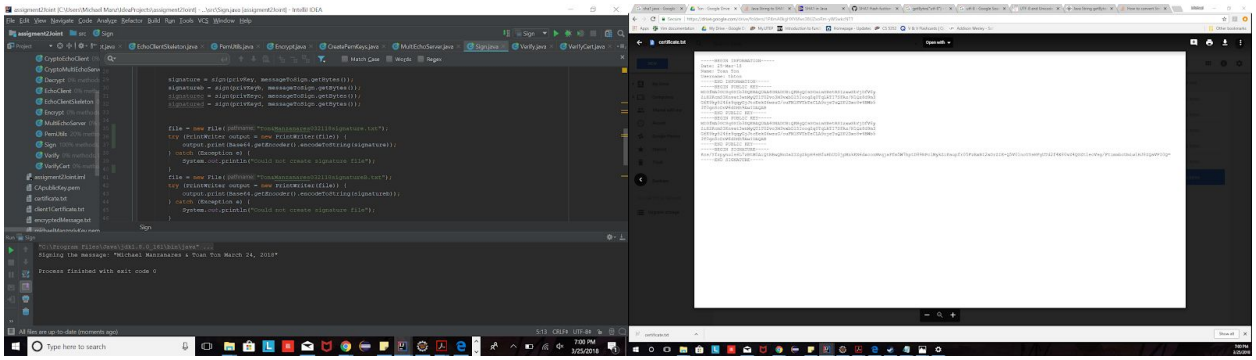


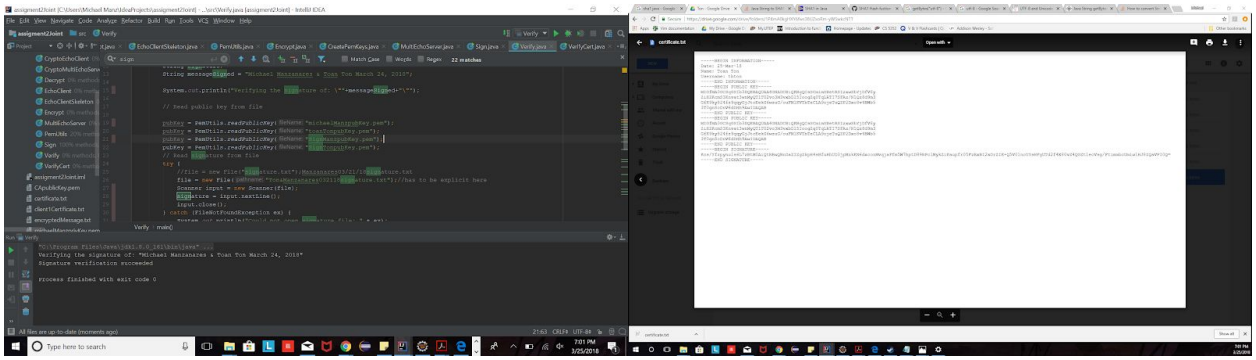


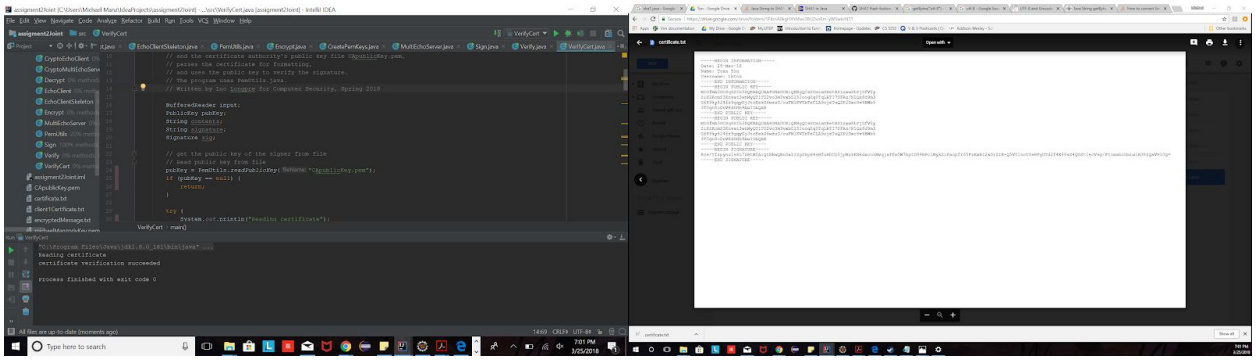
Michael Manzanares
Toan Ton
03/25/18



The above three show the three different classes Sign, Verify, and VerifyCert showing that each worked successfully. The next three pictures are the programs ran to sign, verify, and verifycert but using Toan's certificate.

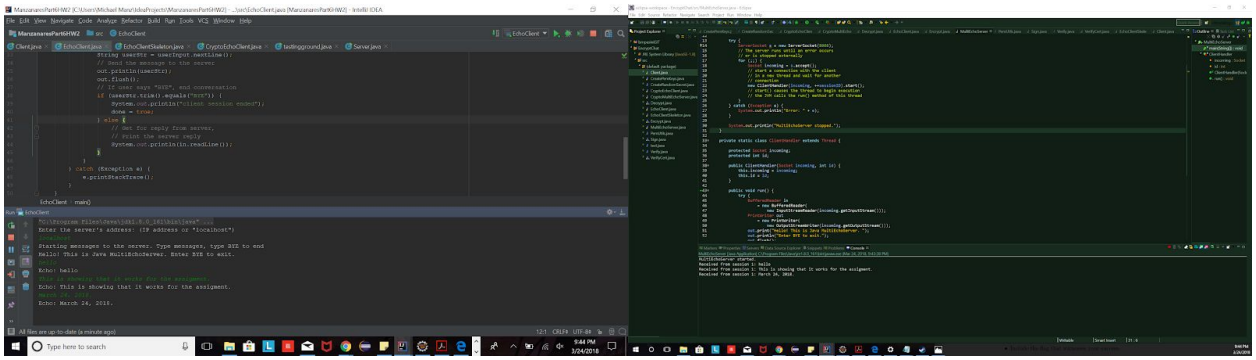






The screenshot to the left shows the certificate that was retrieved from the website provided by Dr. Longpre.

Part 4:



The screenshots show both the server and client (non “Crypto”). This one shows the client/server relationship. A message is sent to the server and it echos. Other users are able to be added. I use different IDE’s; intelliJ, Eclipse and Netbeans. They would echo to the designated user(or the IDE that is using.) It is a good basis to better understand the “Cyrpto” classes that include the server and client.

Part 5:

In the “CryptoMultiEcho” we were tasked with encrypting the message once it has been decrypted. First in order to get the encryption running in this particular class it was required to initialize

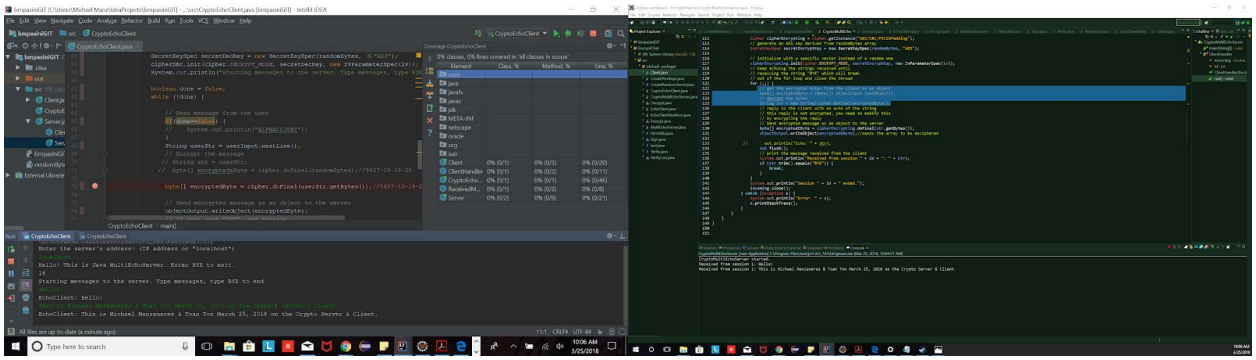
```
//CryptoMultiEchoServer
byte[] iv = (byte[]) objectInput.readObject();
// we will use AES encryption, CBC chaining and PCS5 block padding
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
// generate an AES key derived from randomBytes array
SecretKeySpec secretKey = new SecretKeySpec(randomBytes, "AES");

// initialize with a specific vector instead of a random one
cipher.init(Cipher.DECRYPT_MODE, secretKey, new IvParameterSpec(iv));
Cipher cipherEncrypting = Cipher.getInstance("AES/CBC/PKCS5Padding");
// generate an AES key derived from randomBytes array
SecretKeySpec secretEncryptKey = new SecretKeySpec(randomBytes, "AES");

// initialize with a specific vector instead of a random one
cipherEncrypting.init(Cipher.ENCRYPT_MODE, secretEncryptKey, new
IvParameterSpec(iv));

byte[] encryptedByte = (byte[]) objectInput.readObject();
```

```
// decrypt the bytes
String str = new String(cipher.doFinal(encryptedByte));
// reply to the client with an echo of the string
// this reply is not encrypted, you need to modify this
// by encrypting the reply
// Send encrypted message as an object to the server
byte[] encryptedEByte = cipherEncrypting.doFinal(str.getBytes());
objectOutput.writeObject(encryptedEByte);//sends the array to be deciphered
```

In the “CryptoEchoClient”, the contrast had to be done in order to modify the class to be able to decrypt the encrypted byte[] that contains the message being passed between the two classes.

```
//CryptoEchoClient
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
// generate an AES key derived from randomBytes array
SecretKeySpec secretKey = new SecretKeySpec(randomBytes, "AES");
cipher.init(Cipher.ENCRYPT_MODE, secretKey);
// the initialization vector was generated randomly
// transmit the initialization vector to the server
// no need to encrypt the initialization vector
// send the vector as an object
byte[] iv = cipher.getIV();
objectOutput.writeObject(iv);
Cipher cipherDEc = Cipher.getInstance("AES/CBC/PKCS5Padding");
// generate an AES key derived from randomBytes array
SecretKeySpec secretDecKey = new SecretKeySpec(randomBytes, "AES");
cipherDEc.init(Cipher.DECRYPT_MODE, secretDecKey, new IvParameterSpec(iv));
System.out.println("Starting messages to the server. Type messages, type BYE to end");
String userStr = userInput.nextLine();
// Encrypt the message
// String str = userStr;
byte[] encryptedByte = cipher.doFinal(userStr.getBytes());//5637-15-18-20

// Send encrypted message as an object to the server
objectOutput.writeObject(encryptedByte);
// If user says "BYE", end session
//       byte[] encryptedRecByte = (byte[]) objectInput.readObject();
if (userStr.trim().equals("BYE")) {
   System.out.println("client session ended");
   done = true;
} else {
```

```
byte[] encryptedRByte = (byte[]) objectInput.readObject();//retrieve the object
String str = new String(cipherDEc.doFinal(encryptedRByte));//Decrypts the encrypted byte
System.out.println("EchoClient: " + str);
```



Part 6:

```
//try {
                    byte[] encryptedServerBytes = (byte[])
objectInput.readObject();
                    System.out.println("Encrypted Bytes recieved");
                    byte[] signatureServerBytes = (byte[])
objectInput.readObject();
                    System.out.println("Signature Bytes recieved");
                    byte[] decryptedServerBytes = decrypt(pvtDecrypKey,
encryptedServerBytes);
                    System.out.println("Bytes Decrypted");
                    if(verify(pkpair[1], decryptedServerBytes,
signatureServerBytes)){
                          System.out.println("Verification Success");
                    }
                    else {
                          System.out.println("Verification Failed");
                    }

                    byte[] clientRandomBytes = new byte[8];
                    new Random().nextBytes(clientRandomBytes);
                    byte[] encryptedBytes = encrypt(pkpair[0],
clientRandomBytes);
                    System.out.println("Sending encrypted bytes to server");
                    objectOutput.writeObject(encryptedBytes);

                    MessageDigest digest = MessageDigest.getInstance("SHA-256");
                    byte[] hashedRandomBytes = digest.digest(clientRandomBytes);
```

```
                    byte[] signatureBytes = sign(pvtSignKey, hashedRandomBytes);
                    objectOutput.writeObject(signatureBytes);
                    System.out.println("Sending Signature to server");

                    byte[] sharedSecret = new byte[16];
                    System.arraycopy(decryptedServerBytes, 0, sharedSecret, 0,
8);
                    System.arraycopy(clientRandomBytes, 0, sharedSecret, 8, 8);


                    try {//This part was added handle bringing in the decrypt
mode.  Encrypt Mode had be initiated
                   // we will use AES decryption, CBC chaining and PCS5 block
padding
                           byte[] initializationVector = (byte[])
objectInput.readObject();
                           System.out.println("IV Recieved");
                    cipherDec = Cipher.getInstance("AES/CBC/PKCS5Padding");
                    // generate an AES key derived from randomBytes array
                    SecretKey secretKey = new SecretKeySpec(sharedSecret, "AES");
                    cipherDec.init(Cipher.DECRYPT_MODE, secretKey, new
IvParameterSpec(initializationVector));
                    System.out.println("Decrypt Mode on");
                } catch (NoSuchAlgorithmException | NoSuchPaddingException |
InvalidKeyException e) {
                    System.out.println("error setting up the AES decryption");
                    return;
                }


//decrypting
  // Wait for reply from server,
                           encryptedServerBytes = (byte[])
objectInput.readObject();
                           // will need to decrypt and print the reply to the
screen
                           String plaintext = new
String(cipherDec.doFinal(encryptedServerBytes), "UTF-8");
                           System.out.println(plaintext);
//added methods that helped
public static PublicKey[] vCert(BufferedReader input) {
             PublicKey pubKey;
             String contents;
             String encPubKey;
```

```
            String sigPubKey;
            String signature;
            Signature sig;

            // get the certificate and signature
            try {
                  String line = input.readLine();
                  if (!"-----BEGIN INFORMATION-----".equals(line)) {
                        System.out.println("expecting:-----BEGIN
INFORMATION-----");
                        System.out.println("got:" + line);
                        return null;
                  }
                  contents = line + "\r\n";
                  line = input.readLine();
                  while (line != null && !"-----END
INFORMATION-----".equals(line)) {
                        contents += line + "\r\n";
                        line = input.readLine();
                  }
                  contents += line + "\r\n";
                  line = input.readLine();
                  if (line != null && !"-----BEGIN PUBLIC
KEY-----".equals(line)) {
                        System.out.println("expecting:-----BEGIN PUBLIC
KEY-----");
                        System.out.println("got:" + line);
                        return null;
                  }
                  encPubKey = "";
                  while (line != null && !"-----END PUBLIC
KEY-----".equals(line)) {
                        contents += line + "\r\n";
                        encPubKey += line + "\r\n";
                        line = input.readLine();
                  }
                  contents += line + "\r\n";
                  encPubKey += line + "\r\n";
                  line = input.readLine();
                  while (line != null && line.trim().length() == 0) {
                        contents += line + "\r\n";
                        line = input.readLine();
                  }
                  if (!"-----BEGIN PUBLIC KEY-----".equals(line)) {
```

```
                        System.out.println("expecting:-----BEGIN PUBLIC
KEY-----");
                        System.out.println("got:" + line);
                        return null;
                  }
                  sigPubKey = "";
                  while (line != null && !"-----END PUBLIC
KEY-----".equals(line)) {
                        contents += line + "\r\n";
                        sigPubKey += line + "\r\n";
                        line = input.readLine();
                  }
                  contents += line + "\r\n";
                  sigPubKey += line + "\r\n";
                  line = input.readLine();
                  while (line != null && line.trim().length() == 0) {
                        contents += line + "\r\n";
                        line = input.readLine();
                  }
                  if (!"-----BEGIN SIGNATURE-----".equals(line)) {
                        System.out.println("expecting:-----BEGIN
SIGNATURE-----");
                        System.out.println("got:" + line);
                        return null;
                  }
                  signature = input.readLine();
                  line = input.readLine();
                  if (!"-----END SIGNATURE-----".equals(line)) {
                        System.out.println("expecting:-----END
SIGNATURE-----");
                        System.out.println("got:" + line);
                        return null;
                  }
            } catch (IOException e) {
                  System.out.println("error occurred while reading the
certificate, " + e);
                  return null;
            }
            System.out.println("Extracting Public Keys from Server");
            PublicKey[] pkpair = new PublicKey[2];
            // construct the encryption public key retrieved from the
certificate
            pkpair[0] = PemUtils.constructPublicKey(encPubKey);
            // construct the signature public key retrieved from the
```

```
certificate
            pkpair[1] = PemUtils.constructPublicKey(sigPubKey);

            try {
                  // get the public key of the signer from file
                  // Read public key from file
                  pubKey = PemUtils.readPublicKey("CApublicKey.pem");
                  if (pubKey == null) {
                        return null;
                  }
                  // verify the signature
                  sig = Signature.getInstance("SHA1withRSA");
                  sig.initVerify(pubKey);
                  sig.update(contents.getBytes());
                  if (sig.verify(Base64.getDecoder().decode(signature))) {
                        //Signature verification succeeded
                        System.out.println("Signature verification success");
                        return pkpair;
                  } else {
                        //Signature verification failed
                        return null;
                  }
            } catch (NoSuchAlgorithmException | InvalidKeyException |
SignatureException e) {
                  System.out.println("error occurred while trying to verify
signature" + e);
                  return null;
            }
      }

      public static byte[] decrypt(PrivateKey privKey, byte[]
encryptedByteArray) {
            // decrypts a byte array using a private key
            // and returns the decryption as a byte array

            try {
                  Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
                  cipher.init(Cipher.DECRYPT_MODE, privKey);
                  return cipher.doFinal(encryptedByteArray);
            } catch (NoSuchAlgorithmException | NoSuchPaddingException |
InvalidKeyException | IllegalBlockSizeException | BadPaddingException e) {
                  System.out.println("error while decrypting the message");
                  return null;
            }
```

```
	}

	public static boolean verify(PublicKey pubKey, byte[] message, byte[]
signature) {
		// Written by Luc Longpre for Computer Security, Spring 2018
		try {
			Signature sig = Signature.getInstance("SHA1withRSA");
			sig.initVerify(pubKey);
			MessageDigest digest = MessageDigest.getInstance("SHA-256");
			byte[] hashedmessage = digest.digest(message);
			sig.update(hashedmessage);
			return sig.verify(signature);
		} catch (NoSuchAlgorithmException | InvalidKeyException |
SignatureException e) {
			System.out.println("problem verifying signature: " + e);
		}
		return false;
	}

	public static byte[] encrypt(PublicKey pubKey, byte[] bytes) {
		// encrypts a byte array using a public key
		// and returns the encryption as a byte array

		Cipher cipher;
		byte[] encryptedByteArray;

		try {
			cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
			cipher.init(Cipher.ENCRYPT_MODE, pubKey);
		} catch (NoSuchAlgorithmException | NoSuchPaddingException |
InvalidKeyException e) {
			System.out.println("Could not initialize encryption");
			return null;
		}
		try {
			return cipher.doFinal(bytes);
		} catch (IllegalBlockSizeException | BadPaddingException e) {
			System.out.println("Encryption error");
			return null;
		}
	}

	public static byte[] sign(PrivateKey privKey, byte[] bytes) {
		// Written by Luc Longpre for Computer Security, Spring 2018
```

```
            Signature sig;
            byte[] signature;

            try {
                  sig = Signature.getInstance("SHA1withRSA");
                  sig.initSign(privKey);
                  sig.update(bytes);
                  signature = sig.sign();
            } catch (NoSuchAlgorithmException | InvalidKeyException |
SignatureException e) {
                  System.out.println("Error attempting to sign");
                  return null;
            }
            return signature;
      }
```

The decryption_mode had to be brought into the application. Along with handling decrypting the message from the server. All source code will be included in the zip file. The server file was also created in order to communicate with the client per accordance to the proper protocol assigned. The server format was derived from the cryptomultiserver. The same methods were brought into the server class; vCert, encrypt, sign, decrypt and verify. These methods helped accomplish the goal.

```
System.out.println("Getting Client Certificate");
                        pkpair = vCert(in);

                        byte[] serverRandomBytes = new byte[8];
                        new Random().nextBytes(serverRandomBytes);
                        byte[] encryptedBytes = encrypt(pkpair[0],
serverRandomBytes);
                        System.out.println("Sending encrypted bytes to
client");
                        objectOutput.writeObject(encryptedBytes);

                        MessageDigest digest =
MessageDigest.getInstance("SHA-256");
                        byte[] hashedRandomBytes =
digest.digest(serverRandomBytes);
                        byte[] signatureBytes = sign(pvtSignKey,
hashedRandomBytes);
                        objectOutput.writeObject(signatureBytes);
                        System.out.println("Sending Signature to client");

                        byte[] encryptedClientBytes = (byte[])
```

```
objectInput.readObject();
                        System.out.println("Encrypted Bytes recieved");
                        byte[] signatureClientBytes = (byte[])
objectInput.readObject();
                        System.out.println("Signature Bytes recieved");
                        byte[] decryptedClientBytes = decrypt(pvtDecrypKey,
encryptedClientBytes);
                        System.out.println("Bytes Decrypted");
                        if(verify(pkpair[1], decryptedClientBytes,
signatureClientBytes)){
                              System.out.println("Verification Success");
                        }
                        else {
                              System.out.println("Verification Failed");
                        }

                        byte[] sharedSecret = new byte[16];
                        System.arraycopy(serverRandomBytes, 0, sharedSecret,
0, 8);
                        System.arraycopy(decryptedClientBytes, 0,
sharedSecret, 8, 8);

 // Read message from the user
                           byte[] encryptedUserMessage =
(byte[])objectInput.readObject();
                           // Encrypt the message
                           String plaintext = new
String(cipherDec.doFinal(encryptedUserMessage),"UTF-8");
                           System.out.println(plaintext);
                           if(plaintext == "BYE") {
                              break;
                           }
                           // Send encrypted message as an object to the
server
                           System.out.println("Message Recieved");
                           String systemMessage = "Message Recieved";
                           byte[] encryptedServerBytes =
cipherEnc.doFinal(systemMessage.getBytes());
                           objectOutput.writeObject(encryptedServerBytes);
```

```
Server [Java Application] C:\Program Files\Java\jre1.8.0_141\bin\javaw.exe (Mar 26, 2018,
Server started.
Getting Client Certificate
Extracting Public Keys from Client
Signature verification success
Sending encrypted bytes to client
Sending Signature to client
Encrypted Bytes recieved
Signature Bytes recieved
Bytes Decrypted
Verification Success
Encrypt Mode on
IV Recieved
Decrypt Mode on
Starting Communication Loop
Apple
Message Recieved
Mango
Message Recieved
Pie
Message Recieved
BYE
Message Recieved
Connection Closed
Session 1 ended.
```

```
<terminated> Client [Java Application] C:\Program Files\Java\jre1.8.0_141\bin\javaw.exe (Mar 26, 2018,
Extracting Public Keys from Server
Signature verification success
Encrypted Bytes recieved
Signature Bytes recieved
Bytes Decrypted
Verification Success
Sending encrypted bytes to server
Sending Signature to server
IV Recieved
Decrypt Mode on
Encrypt Mode on
Starting messages to the server. Type messages, type BYE to end
Apple
Message Recieved
Mango
Message Recieved
Pie
Message Recieved
BYE
client session ended
```

Part 7:

Flag for Toan Ton:

```
EchoClientSkeleton [Java Application] C:\Program Files\Java\jre1.8.0_141\bin\javaw.exe (Mar 26, 2018, 1:21:39 AM)
Extracting Public Keys from Server
Signature verification success
Verification Success
IV Recieved
Decrypt Mode on
Starting messages to the server. Type messages, type BYE to end
flag
flag: Npljlh8ZPK
```

Flag for Michael Manzanares:

```
EchoClientSkeleton [Java Application] C:\Program Files\Java\jre1.8.0_141\bin\javaw.exe (Mar 26, 2018, 1:26:16 AM)
Extracting Public Keys from Server
Signature verification success
Verification Success
IV Recieved
Decrypt Mode on
Starting messages to the server. Type messages, type BYE to end
flag
flag: EGUWqJOl3q
```

Feedback:

Team dynamic:

The parts were delegated to each member based on amount of work. Each part was shared in a google drive folder (link available upon request). Any changes that were made, were available real time. The ability to count on others to do the part is a big part of group work. It seems most practical manners are done in teams. Allows the delegation of tasks. We used Steam chat to communicate.

Michael: I liked the tutorial method it seemed to me. The different classes that were made available gave an insight to see how proper coding is done particularly on computer security. I always wonder if my code is poor in sense if it is organized or if I used the correct method. Seeing professors code allowed seeing how proper coding is.

Toan: The assignment was very good hands on experience for learning how protocols work. I learned a lot this assignment and believe overall it's a good assignment. I would only say that the instructions should be clear from the start to avoid any confusion in the future, as well as making sure all the correct files are there.

References:

1. Namir, Royri. "Base64 Length Calculation?" String - Base64 Length Calculation? - Stack Overflow, 24 Sept. 2015, stackoverflow.com/questions/13378815/base64-length-calculation.
2. java, Nathan. "Exception When Calling MessageDigest.getInstance('SHA256')." *Java - Exception When Calling MessageDigest.getInstance("SHA256") - Stack Overflow*, 9 Oct. 2017,

stackoverflow.com/questions/12642742/exception-when-calling-messagedigest-getinstancesha256/12642834.