COSC 4P14 – Assignment 4 – 6349302

GRANT NIKE, Brock University, Canada

ACM Reference Format:
Grant Nike. 2020. COSC 4P14 – Assignment 4 – 6349302. 1, 1 (December 2020), 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 NETWORK SECURITY QUESTIONS

1.1 Question 1

Consider the polyalphabetic system shown in the figure below. Will a chosen plain-text attack that is able to get the plain-text encoding of the message "The quick brown fox jumps over the lazy dog." be sufficient to decode all messages? Why or why not?

1.1.1 Answer. No, this would not be enough to decode all messages. The string: "The quick brown fox jumps over the lazy dog." is a pangram, meaning it contains each letter in the English alphabet. If this was a monoalphabetic cipher, then the cipher-text of this string would give us the encoding for each letter in the alphabet. However this is a polyalphabetic cipher, where half(odd) of the characters are encoded with the key 5 and the other half(even) are encoded with the key 19, therefore we would not obtain the encoding for each letter of the alphabet by knowing the cipher-text of this pangram. This is because there is not one single encoding for each letter in the alphabet but two separate encodings for each letter depending on their placement in the text.

1.2 Question 2

Let's practice asymmetric encryption with RSA using very short messages. a.Using RSA, choose p=3 and q=11 , and encode the word "dot" by encrypting each letter separately.Apply the decryption algorithm to the encrypted version to recover the original plaintext message. b.Repeat part (a) but now encrypt "dot" as one message m.

1.3 Question 3

In the BitTorrent P2P file distribution protocol, the seed breaks the file into blocks, and the peers redistribute the blocks to each other. Without any protection, an attacker can easily wreak havoc in a torrent by masquerading as a benevolent peer and sending bogus blocks to a small subset of peers in the torrent.These unsuspecting peers then redistribute the bogus blocks to other peers, which in turn redistribute the bogus blocks to even more peers. Thus, it is critical for BitTorrent to have a mechanism that allows a peer to verify the integrity of a block, so that it doesn't redistribute bogus blocks. Assume that when a peer joins a torrent, it initially gets a .torrent file from a fully trusted source. Describe a simple scheme that allows peers to verify the integrity of blocks.

Author's address: Grant Nike, Brock University, 1812 Sir Isaac Brock Way, St. Catharines, ON, Canada.

2020. XXXX-XXXX/2020/12-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1.4 Question 4

Suppose Alice wants to send an e-mail to Bob. Bob has a public-private key pair(KB+,KB-), and Alice has Bob's certificate. But Alice does not have a public, private key pair. Alice and Bob (and the entire world) share the same hash functionH(·). a.In this situation, is it possible to design a scheme so that Bob can verify that Alice created the message? If so, show how with a block diagram for Alice and Bob. b.Is it possible to design a scheme that provides confidentiality for sending the message from Alice to Bob? If so, show how with a block diagram for Alice and Bob.

1.4.1 Answer A. No, it is not possible for Bob to verify Alice created a message without Alice having a public-private key pair. If Alice had the public-private key pair rather than Bob, then it would be possible. Alice could hash her message with the hash function, creating a message digest. Then Alice can sign her message digest by encrypting it with her private key, and then send the encrypted message as well as the plain-text version of the message to Bob. Bob will then apply the hash function to the plain-text message creating his own digest of the message. Bob would then request Alice's public key, and use it to decrypt the message digest sent by Alice. If the decrypted message digest matches the digest Bob created, then Alice must be who she says she is because only Alice could've encrypted the message digest with her private key.

1.4.2 Answer B. If, like in the last scenario, we assume Alice has her own public-private key pair then yes, this is possible. This will go similarly to question a, with some extra steps. Alice starts by doing the same thing as before, she will hash her message with the hash function, creating a message digest. Then Alice will again sign her message digest by encrypting it with her private key. But this time, to ensure confidentiality of the message, she will encrypt both the plain-text message and encrypted message digest with a new symmetric key. Then she will encrypt the symmetric key itself with Bob's public key, ensuring only Bob's private key can decrypt the message. Then all three parts, the new symmetric key, the now encrypted message, and the encrypted digest are all sent to Bob. Once Bob receives all of this he will use his private key the decrypt the symmetric key, then use the symmetric key to decrypt the plain-text message, and he will use first the symmetric key and then Alice's public key to decrypt the message digest. The plain-text message is hashed and compared with the decrypted digest, if they match then Alice is who she says she is.

1.5 Question 5

In a man-in-the-middle attack scenario, it is known that without sequence numbers, Trudy (a woman-in-the-middle) can wreak havoc in an SSL session by interchanging TCP segments. Can Trudy do something similar by deleting a TCP segment? What does she need to do to succeed at the deletion attack? What effect will it have?

1.5.1 Answer. Yes, Trudy can do something similar by deleting a TCP segment. To succeed in this deletion attack Trudy would intercept a TCP segment and change its sequence number, which is not encrypted, to match the segment before it. This way it will be discarded as a duplicate on arrival, and not passed on to the SSL sublayer. Then increment the sequence numbers of all the TCP segments afterwards by one, so the missing segment will not be requested again. This would effect the overall message being sent as the receiver will receive incomplete data that will go all the way to the application layer, where the application may probably wont be able to properly process the data.

1.6 Question 6

Consider the example in the figure. Suppose Trudy is a woman-in-the-middle, who can insert datagrams into the stream of datagrams going from R1 and R2. As part of a replay attack, Trudy sends a duplicate copy of one of the datagrams sent from R1 to R2. Will R2 decrypt the duplicate datagram and forward it into the branch-office network? If not, describe in detail how R2 detects the duplicate datagram.

1.6.1 Answer. R2 will be able to detect the duplicate datagram. Routers R1 and R2 have established a security association and so all datagrams sent from R1 to R2 will be wrapped to form an IPSec datagram. An IPSec datagram will contain an ESP header, which in turn will contain a sequence number for the datagram. Any duplicate datagram sent by Trudy will have the same sequence number as the original datagram that is being duplicated. So when R2 checks the sequence number of the duplicate datagram, R2 will detect that it matches the sequence number of an earlier datagram and discard the datagram.

2 SECURED JAVA APPLICATION

My Java application is a client-server application(TCP sockets), with a communication channel encrypted with AES. The encryption uses a symmetric key, which is created as a shared secret by a Diffie-Hellman key exchange. This key exchange is still vulnerable to man-in-the-middle attacks, as it doesn't include a method of verifying you are talking to the right person. The key exchange doesn't have any bearing on the performance analysis and was only included for completeness. The client can send any string to the server over the encrypted channel, and the server will respond by sending the same string all uppercase over the encrypted channel.

2.1

2.2 Link To Code

Link to code in a Github Repository: https://github.com/GrantNike/COSC4P14Assignment4Code

2.3 Instructions for Compiling and Running Code

(1) Compile server with command: "javac Server.java" then compile client with command: "javac client.java"
(2) Run server file including the optional argument like so: "java Server <portNumber>". The default port number with no arguments is 4000.
(3) Then Run the client file in a separate terminal, where both arguments are required like so: "java client <hostName> <portNumber>".
(4) Once the client connects to the server, the client can send strings to the server and receive strings back over an encrypted channel.
(5) CTRL+C will close both the client and server programs. Each will have to be closed individually.

2.4 Performance Analysis

Table 1. Average End-End Delay of 30 Non-Encrypted Message Exchanges

Plain-text Message Size(Bytes)	Average End-End Delay(milliseconds)
100	0.945974
500	0.193234
1000	0.300874
5000	1.648037
10,000	0.729177
50,000	2.181547

Table 2. Average End-End Delay of 30 Encrypted Message Exchanges

Plain-text Message Size(Bytes)	Average End-End Delay(milliseconds)
100	2.361627
500	2.605507
1000	2.984567
5000	5.791111
10,000	98.097943
50,000	105.009243

2.5 Conclusions

We can see from the data presented in the two graphs above that for small message sizes the Average End-End delay is very similar between encrypted and non-encrypted channels. However as the size of the message increases, the encrypted channel had significantly higher End-End Delay than the non-encrypted channel. This can tell us that the End-End Delay of an encrypted(AES) channel doesn't scale as well with large data as a non-encrypted channel does.

A CLIENT.JAVA

```
import java.io.*;
import java.net.*;
import java.security.*;
import javax.crypto.*;
import java.security.SecureRandom;
import java.util.Base64;
import java.util.Random;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

/*
Grant Nike
6349302
Dec 6th
```

```
Client class requests a connection with a given host on a given port.
    Once connected the client and server will perform
a Diffie-Hellman key exchange to produce a shared secret key. The client
    can then send AES encrypted strings to the server,
and recieve and decrypt strings from the server.
*/
public class client {
    public static void main(String[] args) throws Exception {
        if (args.length < 2) { //User must include host name and port
    number of server as arguments
            System.err.println(
                "Usage: java client <host name> <port number>");
            System.exit(1);
        }
        String host = args[0];//Name of host is first argument
        int port = Integer.parseInt(args[1]);//Port number is second
    argument

        try (
            Socket socket = new Socket(host, port);//Create new socket
    for connection with server
            OutputStream outputStream = socket.getOutputStream();
            InputStream inputStream = socket.getInputStream();
            PrintWriter out = //Sends strings to server
                new PrintWriter(outputStream,true);
            BufferedReader in =//Recieves strings from server
                new BufferedReader(
                    new InputStreamReader(inputStream));
            BufferedReader stdIn =//Allows user to type a continuous
    stream of input
                new BufferedReader(
                    new InputStreamReader(System.in));
        ) {
            System.out.println("Connection made with server!");//Lets
    user know a connection has been established
            // Create public and private key pair for Diffie-Hellman key
    exchange
            KeyPairGenerator keyGenerator = KeyPairGenerator.getInstance(
    "EC");
            keyGenerator.initialize(128);
            KeyPair kp = keyGenerator.genKeyPair();
            PublicKey publickey = kp.getPublic();
            KeyAgreement keyAgreement = KeyAgreement.getInstance("ECDH");
            keyAgreement.init(kp.getPrivate());
            //Receive server's public key
            ObjectInputStream objectInputStream = new ObjectInputStream(
    inputStream);
            PublicKey serverPublicKey = (PublicKey) objectInputStream.
    readObject();
            //Send public key to server
            ObjectOutputStream objectOutputStream = new
    ObjectOutputStream(outputStream);
            objectOutputStream.writeObject(publickey);
            //Create new shared secret key which will serve as the
    symmetric key for AES encryption
            keyAgreement.doPhase(serverPublicKey, true);
            byte[] key = keyAgreement.generateSecret();
            //Get initial vector from server
            byte[] IV = Base64.getDecoder().decode(in.readLine());
            //Run performance analysis if test parameter was entered
            if(args.length == 3){
                long averageDelay = 0;
                int test_plaintext_size = Integer.parseInt(args[2]);
                for(int i = 0;i<30;i++){
                    //Create test string using letters and numbers
```

```
                    int L = 48; int R = 122;
                    String testInput = new Random().ints(L, R + 1)
                    .filter(j -> (j <= 57 || j >= 65) && (j <= 90 || j >=
 97))
                    .limit(test_plaintext_size)
                    .collect(StringBuilder::new, StringBuilder::
appendCodePoint, StringBuilder::append)
                    .toString();
                    //Start timing delay
                    long start = System.nanoTime();
                    String encryptedMessage = Base64.getEncoder().
encodeToString(encrypt(testInput.getBytes(),key,IV));
                    //Send test string to server
                    out.println(encryptedMessage);
                    //Read and decrypt server response
                    String serverMessage = in.readLine();
                    String decryptedMessage = decrypt(Base64.getDecoder()
.decode(serverMessage),key,IV);
                    //Stop timing delay
                    long stop = System.nanoTime();
                    long delay = stop-start;
                    averageDelay += delay;
                    System.out.println("End-End Delay: "+delay/1000000.0+
" milliseconds");
                }
                System.out.println("Average End-End Delay over 30 runs: "
+(averageDelay/30.0)/1000000.0+" milliseconds");
            }
            String userInput;
            while ((userInput = stdIn.readLine()) != null) { //Allow user
 to send strings to server continuously
                long start = System.nanoTime();
                String encryptedMessage = Base64.getEncoder().
encodeToString(encrypt(userInput.getBytes(),key,IV));
                out.println(encryptedMessage);
                String serverMessage = in.readLine();
                if(serverMessage == null){
                    System.out.println("Server closed connnection.");
                    break;
                }
                String decryptedMessage = decrypt(Base64.getDecoder().
decode(serverMessage),key,IV);
                long stop = System.nanoTime();
                long RTT = stop-start;
                System.out.println("Server: " + decryptedMessage);
                System.out.println("RTT: "+RTT/1000000.0+" milliseconds")
;
                System.out.println("Server Encoded: "+serverMessage);
                if(userInput.equals("quit") || userInput.equals("n")){//
Stop sending input if user quits connection
                    break;
                }
            }
        } catch (UnknownHostException e) { //Host can't be found
            System.err.println("Host: " + host + " is unknown.");
            System.exit(1);
        } catch (IOException e) { //No IO from our side or host
            System.err.println("Couldn't get I/O for the connection to "
+
                host);
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

```
    public static byte[] encrypt (byte[] plaintext,byte[] key,byte[] iv )
     throws Exception{
        //Create instance of cipher to encrypt text
        Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");
        //Create key spec and initial vector spec
        SecretKeySpec key_spec = new SecretKeySpec(key, "AES");
        IvParameterSpec iv_spec = new IvParameterSpec(iv);
        //Put cipher in encrypt mode
        c.init(Cipher.ENCRYPT_MODE, key_spec, iv_spec);
        //Encrypt text
        byte[] ciphertext = c.doFinal(plaintext);

        return ciphertext;
    }

    public static String decrypt (byte[] ciphertext, byte[] key,byte[] iv
    ) throws Exception{
        //Create instance of cipher to decrypt given text
        Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");
        //Create key spec and initial vector spec
        SecretKeySpec key_spec = new SecretKeySpec(key, "AES");
        IvParameterSpec iv_spec = new IvParameterSpec(iv);
        //Put cipher in decrypt mode
        c.init(Cipher.DECRYPT_MODE, key_spec, iv_spec);
        //Decrypt text
        byte[] plaintext = c.doFinal(ciphertext);

        return new String(plaintext);
    }
}
```

B SERVER.JAVA

```
import java.io.*;
import java.net.*;
import java.security.*;
import javax.crypto.*;
import java.security.SecureRandom;
import java.util.Base64;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

/*
Grant Nike
6349302
Dec 6th
Server class begins listening for a client connection on given port, if
    no port is given default is 4000.
Once connected the client and server will perform a Diffie-Hellman key
    exchange to produce a shared secret key. The Server
initiates the key exchange.The Server will then listen for AES encrypted
    strings from the client, decrypt them, and send back
the same String in all uppercase and AES encrypted.
*/
public class Server {
    public static void main(String[] args) throws Exception {
        //Set port to arguement if one was given, otherwise default port
    is 4000
        int port = (args.length != 1)? 4000 : Integer.parseInt(args[0]);
```

```
        ServerSocket serverSocket = new ServerSocket(port);//Create new
    server socket for connection with a client
        System.out.println("Server listening on port: " + port); //Lets
    user on server side know server is running
        try (
            Socket clientSocket = serverSocket.accept();
            OutputStream outputStream = clientSocket.getOutputStream();
            InputStream inputStream = clientSocket.getInputStream();
            PrintWriter out =
                new PrintWriter(outputStream, true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(inputStream));
        ) {
            System.out.println("Connection made with client on port: "+
    clientSocket.getPort());//Lets server side know a connection has been
     made with a new client
            // Create public and private key pair for Diffie-Hellman key
    exchange
            KeyPairGenerator keyGenerator = KeyPairGenerator.getInstance(
    "EC");
            keyGenerator.initialize(128);
            KeyPair kp = keyGenerator.genKeyPair();
            PublicKey publickey = kp.getPublic();
            KeyAgreement keyAgreement = KeyAgreement.getInstance("ECDH");
            keyAgreement.init(kp.getPrivate());
            // Create initial vector for encryption
            byte[] IV = new byte[16];
            SecureRandom random = new SecureRandom();
            random.nextBytes(IV);
            //Send public key to client
            ObjectOutputStream objectOutputStream = new
    ObjectOutputStream(outputStream);
            objectOutputStream.writeObject(publickey);
            //Receive client's public key
            ObjectInputStream objectInputStream = new ObjectInputStream(
    inputStream);
            PublicKey clientPublicKey = (PublicKey) objectInputStream.
    readObject();
            //Create new shared secret key which will serve as the
    symmetric key for AES encryption
            keyAgreement.doPhase(clientPublicKey, true);
            byte[] key = keyAgreement.generateSecret();
            //Send initial vector to client
            out.println(Base64.getEncoder().encodeToString(IV));
            //Start reading client input
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                String decodedMessage = decrypt(Base64.getDecoder().
    decode(inputLine),key,IV);
                System.out.println("Client Encoded: "+inputLine);
                System.out.println("Client: " + decodedMessage);//Print
    what client said to server side
                decodedMessage = decodedMessage.toUpperCase();
                out.println(Base64.getEncoder().encodeToString(encrypt(
    decodedMessage.getBytes(),key,IV)));//Echos back everthing the client
     says
            }
            serverSocket.close();//Make sure to close socket when done
    with it
        } catch (IOException e) { //Couldn't listen on port provided
            System.out.println("Couldn't listen on port: "+ port);
            System.out.println(e.getMessage());
        }
    }
```

```
    public static byte[] encrypt (byte[] plaintext,byte[] key,byte[] iv )
     throws Exception{
        //Create instance of cipher to encrypt message
        Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");
        //Create key spec and initial vector spec
        SecretKeySpec key_spec = new SecretKeySpec(key, "AES");
        IvParameterSpec iv_spec = new IvParameterSpec(iv);
        //Put cipher in encrypt mode
        c.init(Cipher.ENCRYPT_MODE, key_spec, iv_spec);
        //Encrypt message
        byte[] ciphertext = c.doFinal(plaintext);

        return ciphertext;
    }

    public static String decrypt (byte[] ciphertext,byte[] key,byte[] iv)
     throws Exception{
        //Create instance of cipher to encrypt text
        Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");
        //Create key spec and initial vector spec
        SecretKeySpec key_spec = new SecretKeySpec(key, "AES");
        IvParameterSpec iv_spec = new IvParameterSpec(iv);
        //Put cipher in decrypt mode
        c.init(Cipher.DECRYPT_MODE, key_spec, iv_spec);
        //Decrypt text
        byte[] plaintext = c.doFinal(ciphertext);

        return new String(plaintext);
    }
}
```