네트워크 보안 및 실습 (2021-1)
Term Project

사이버보안학과
201820697 김소정

1. 프로젝트 목표

대칭키 암호화를 적용한 1:1 양방향 통신 프로그램을 작성한다.

1) 소켓 프로그래밍을 통한 통신 프로그램 구현
2) 대칭키 분배를 위해 RSA 사용
3) 분배한 키를 기준으로 AES-256 암호화를 수행하여 안전한 통신 구현

2. 구현 동작

프로그램의 전체적인 동작 구조는 다음 사진과 같다.

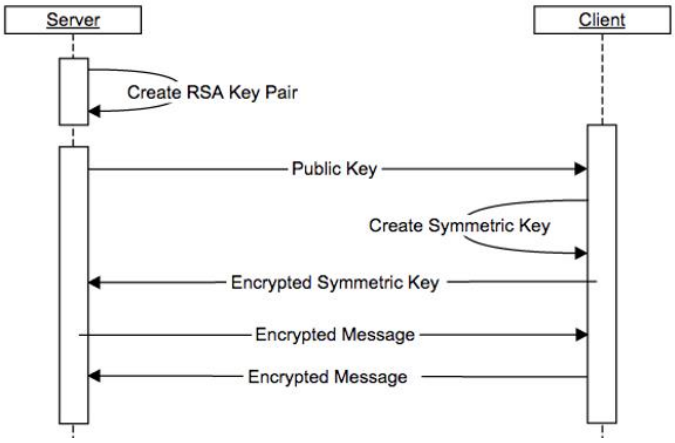

그림 1 project structure

1) #Server : Create RSA Key Pair and Send Public Key to Client

Server에서 RSA Key Pair (public key, private key)를 생성하고, Client에게 RSA Public Key를 전송한다.

```
ssol2@ssol2-virtual-machine:/mnt/hgfs/ubuntu/EncryptedChat$ java Server
> Creating RSA key Pair...
[Public Key]: MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxrC2Tp82/PDsAwLE0+Gmu+s1axcLCecyaU/meEWMRpK
5Zf5ZSP6DEl8Q+MyzBnwUjVQ2W6jYgG8OuQzDdLGv6cj9V7nyhYajWGLxuS3HOojpVRWZi3kQ2I5+pTOFbiZDz8puJTml+WtbkCnA
6IWPavd1d/nnjKLuA2w+6BNnr1N3KcAp+BU6+l/njaTUzuxyWbYQiID7Fx2L+3lOxcApDCkSqvfLpFeMInjCm3Sl3/rNWphxwIDAQ
AB
[Private Key]: MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDGsLZOnzb88OwDAsTT4aa76zVrFwsJ5zJpT+
+Zg836hXll/llI/oMSXxD4zLMGfBSNVDZbqNiAbw65DMN0sa/pyP1XufKFhqNYYvG5Lcc6iOlVFZmLeRDYjn6lM4VuJkPPym4lOaX
5a1uQKcDohY9q93V3+eeMou4DbD7oE2evU3cpwCn4FTr6X+eNpNTO7HJZthCIgPsXHYv7eU7FwCkMKRKq98ukV4wieMKbdKXf+s1a
mHHAgMBAAECggEAYJ/JWLzG4gVL0T8Ed0Hycsq0xKqFHMDJXDuWCgFQcXWWctnx54g7818v4GUk6dABaOTAIA8hM36YdzfuKXt3+8
qabvLVkTnHmX5E9aM4BP/afbt9S83BLUBdwwG/078+rYq9QG/830gEuJ1nHqtkhlITpOwLuXwA1Q78Nw5kjggLme40uKk/ojFWwL0
4aEGCMc9YPxOdbaG/+W2IaNjbAXbfpWACdlbB0x2BuwTeCj6y99mFzJLkap8v01CGbckici7rlH7CxkChOIEc2JC1JCrrYaDJHnOq
WYp97hxd+wgD+dauKPPeh0J/2v+W6NFNuICPSHqacrniwmUAdA0mgQKBgQDlkQsy8qJQePJWcR7Br5727s7j6Q9mHhOrWmJNpUyI9
VwMpXN99HgJ8OJgtWYy79HIQr7HLBWi9eHuq86AnfPiA9qoohCBxwr+bUjQTLQCNyFzwDzhDfdkuXiujI8p51Rg2klHFRvMxpVBFx
doRd6UbbmH2ruZyXA5XDFruNFEQQKBgQDdkYZb3DbMS9KLCbQYTGa4uUXU2CF/u4nj5ttj22BorALjF9vcFAv6KaVBhxrs6VoFZ1t
/kE874fYUk8P73xyHDugBhGpZrjbebO6gx2JdfyGRq3O6Yr6+8NuvMKWfXwEVluSi6RmanFrxYUtfkRoPqR8Jud23iSq6hgzDIICE
BwKBgQCnTMH9mifCP7JUXtIMU0GlMbpjDAkl4YtyGEeF0I5rTWyM5tK5Gz2T+RHxaqGVQ4L9z+6a1jzaaxkBSAIRZ4tBzo8+0AT5n
yYa72feLdgNuUHOym1TkksWA7i4dEUtQ6IRh2NZwz9kR+Cj9AEiA7FHOIO6RyTq2YvIsbBukd0oQQKBgBtXNMQrFVv6Uz2y8dU5wE
c1HYfZL7EKXEaSGypkTK4945FQRWIP2zaAYxMF7+2iEeiuy1yfWxehl3MMlNeEvYlw7nIZe+IbHEwwFvYiBGJTKLpH9NzgCHeZUxJ
4J0fWOlUGrKnpp64rcxFsFxfw2M1r6heSJujZ9TrVBcgiP5mLAoGBAM06OJE+oNRYyUchbUxMTc1wh+1bxvgnUf5ysQm9huXB1bML
s6UkHmdPzV/W/ySSP0FqBvy+8d7X3XY0hUXULcBpmk6hQLQg16utpZ/iHrOf3hUG/MsohTufKZisBfJG5q2ZR/Y9Hdx9Av+ukatt2
+k4WGSm1sewZM2byQHBO1Rj
```

그림 2 #Server : Create RSA Key Pair and Send Public Key to Client

2) #Client : Get the Public key, Create Symmetric Key and Send Encrypted Symmetric Key to Server

Client는 Server에서 전송한 RSA Public key를 받고, AES-256 Symmtric Key와 IV를 생성한다. 생성한 AES Key, IV를 RSA Public Key로 암호화하고, 암호화된 AES Key, IV를 Server에 전송한다.

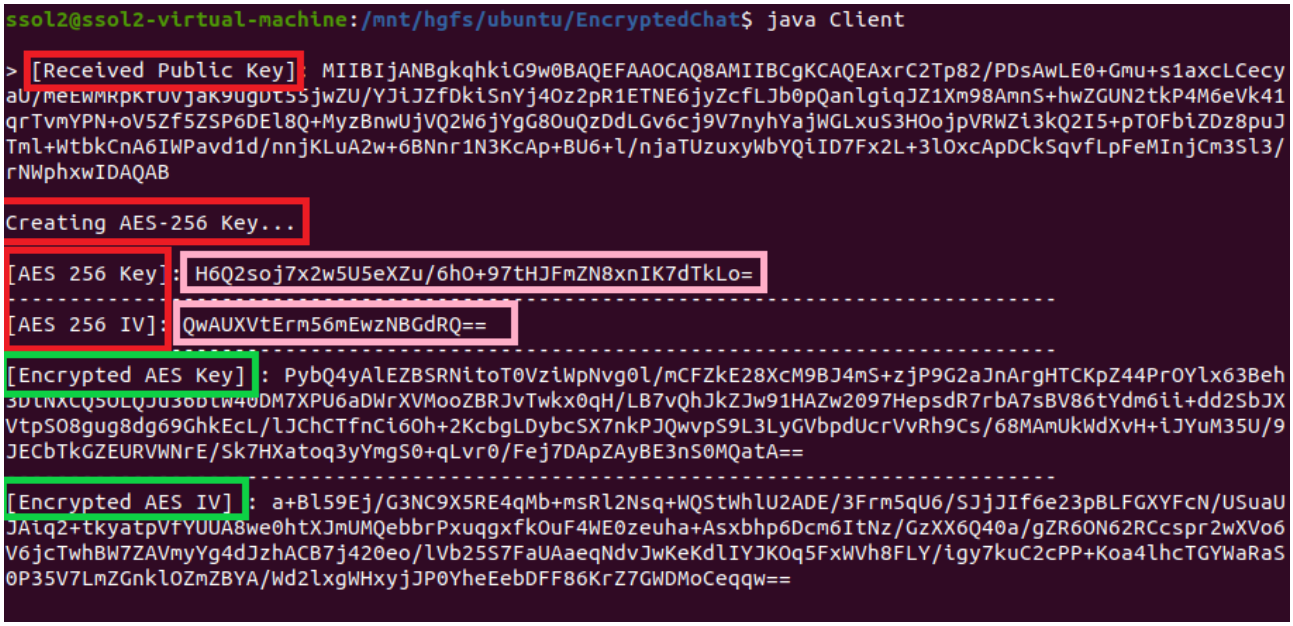

그림 3 #Client : Get the Public key, Create Symmetric Key and Send Encrypted Symmetric Key to Server

3) #Server : Get the Encrypted Symmetric Key and Decrypt Key, IV to RSA Private Key

Server는 Client에서 전송한 Encrypted Symmetric Key, IV를 받는다. 암호화된 AES Key, IV를 RSA Private Key로 복호화하면 바로 위 그림에서 Client에서 생성한 AES Key, IV와 base64 인코딩이 일치함을 확인한다. 대칭키 분배를 위해 RSA를 사용하는 과정이 성공한다.

그림 4 #Server : Get the Encrypted Symmetric Key and Decrypt Key, IV to RSA Private Key

4) #Server : Send the Encrypted Message to Client

AES-256 암호화로 Server-Client 간의 1:1 통신을 구현한다. Server에서 “Hi!”라는 메시지를 보내면, AES-256 암호화 통신으로 안전하게 Client에게 전달된다. Client는 복호화 과정을 통해 “Hi!”라는 메시지를 받는다.

그림 5 #Server : Send the Encrypted Message to Client

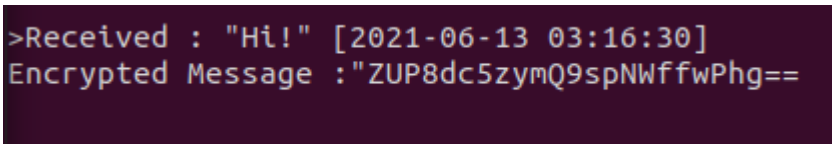

그림 6 #Client : Get the Encrypted Message

5) #Client : Send the Encrypted Message to Server

Client에서도 “Hello!”라는 메시지를 보내면, AES-256 암호화 통신으로 안전하게 Server에게 전달된다. Server는 복호화 과정을 통해 “Hello!”라는 메시지를 받는다.

그림 7 #Client : Send the Encrypted Message to Server

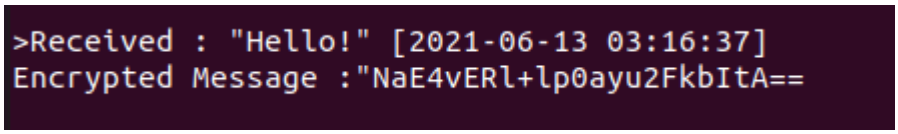

그림 8 #Server : Get the Encrypted Message

6) #Server : Sent the Encrypted Message (“exit”) to Client

Server에서 “exit” 메시지를 보내면 AES-256 암호화 통신으로 안전하게 Client에게 전달되고, Client는 복호화 과정을 통해 “exit” 메시지를 받는다. Client는 “exit” 메시지를 받으면 통신을 종료하므로 Server-Client간의 1:1 통신이 종료된다. (물론 통신 종료 전에 Client는 Server에게 “exit” 메시지를 보낸다.)

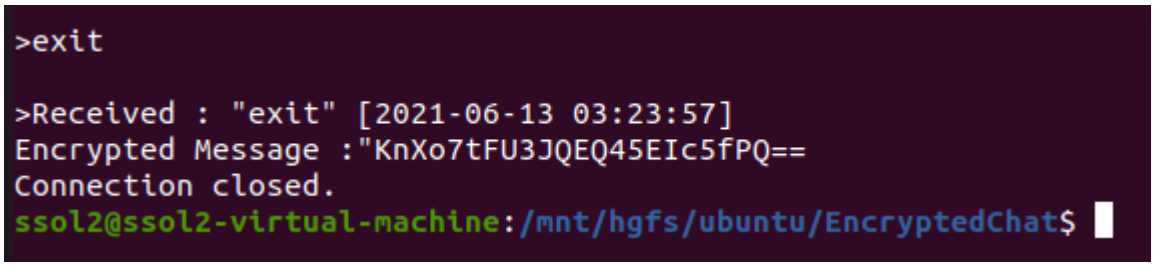

그림 9 #Server : Sent the Encrypted Message (“exit”) to Client

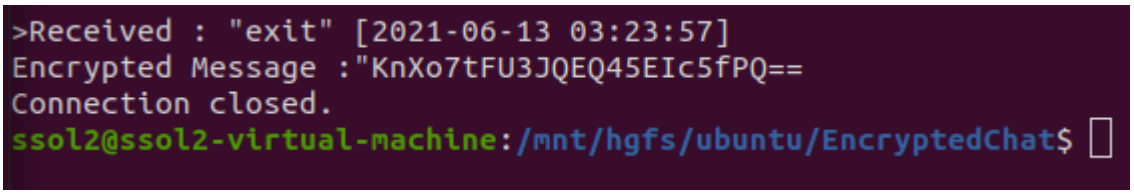

그림 10 #Client : Get the Encrypted Message (“exit”) and Connection closed.

3. 구현 설명

1) Server.c

A. 서버 생성 및 클라이언트 접속

Server socket port를 10101로 설정하고, 클라이언트 접속을 받아들인다.

```
//서버 생성
Server_Socket = new ServerSocket( port: 10101);
//클라이언트 접속
```

그림 7 Server.c

그림 11 Server.c – server, client 생성

B. 데이터 스트림 생성

Server-Client 간의 통신을 주고받은 메시지 데이터 스트림을 생성한다.

```
//데이터 스트림 생성
ObjectOutputStream Server_MSG = new ObjectOutputStream(Client_Socket.getOutputStream());
ObjectInputStream Client_MSG = new ObjectInputStream(Client_Socket.getInputStream());
```

그림 12 Server.c dataStream 생성

C. RSA 생성

RSA keyPair generator로 2048 bit의 keyPair (public key, private key)를 생성한다. 생성한 두 키를 출력하기 위해서 Byte[]로 변환한 후 Base64 인코딩을 하여 출력한다. Client에게 RSA public key를 전송한다.

```
System.out.println("> Creating RSA key Pair...");
//RSA gernerator
KeyPairGenerator KeyPair_Generator = KeyPairGenerator.getInstance("RSA");
KeyPair_Generator.initialize( keysize: 2048);
//RSA public key, private key
KeyPair Key_Pair = KeyPair_Generator.genKeyPair();
PublicKey Public_Key = Key_Pair.getPublic();
PrivateKey Private_Key = Key_Pair.getPrivate();
//RSA public key, private key base64 출력
byte[] Byte_PublicKey = Public_Key.getEncoded();
byte[] Byte_PrivateKey = Private_Key.getEncoded();
System.out.println("[Public Key]: " + Base64.getEncoder().encodeToString(Byte_PublicKey));
System.out.println("-----------------------------------------------------------------------------------------");
System.out.println("[Private Key]: " + Base64.getEncoder().encodeToString(Byte_PrivateKey));
//클라이언트에게 RSA public key 전송
Server_MSG.writeObject(Public_Key);
Server_MSG.flush();
```

그림 13 Server.c RSA 생성

D. RSA private key로 AES secret key, IV 복호화

Client로부터 암호화된 AES secret key, IV를 byte[] 형식으로 받는다. 잘 받은 암호화된 AES secret key, IV를 Base64 인코딩하여 출력한다. 이제 RSA private key로 암호화된 AES secret key, IV를 복호화한다. 복호화 value는 둘 다 byte[] 단위이기 때문에 복호화된 AES secret key, IV value를 바로 Base64 형식으로 출력한다.

```
//클라이언트가 보낸 암호화된 AES secret key 받음
byte[] Encrypted_AES_Key = (byte[]) Client_MSG.readObject();
System.out.println("\n> [Received AES Key] : " + Base64.getEncoder().encodeToString(Encrypted_AES_Key));
System.out.println("--------------------------------------------------------------------------------");
//클라이언트가 보낸 암호화된 AES IV 받음
byte[] Encrypted_AES_IV = (byte[]) Client_MSG.readObject();
System.out.println("> [Received AES IV] : " + Base64.getEncoder().encodeToString(Encrypted_AES_IV));
System.out.println("--------------------------------------------------------------------------------");
//암호화된 AES secret key를 RSA private key로 복호화
Cipher cipher_AES_Key = Cipher.getInstance("RSA/ECB/PKCS1Padding");
cipher_AES_Key.init(Cipher.DECRYPT_MODE, Private_Key);
Decrypted_AES_Key = cipher_AES_Key.doFinal(Encrypted_AES_Key);
//복호화한 AES secret key base64 출력
System.out.println("[Decrypted AES Key] : " + Base64.getEncoder().encodeToString(Decrypted_AES_Key));
System.out.println("--------------------------------------------------------------------------------");
//암호화된 AES IV를 RSA private key로 복호화
Cipher cipher_AES_IV = Cipher.getInstance("RSA/ECB/PKCS1Padding");
cipher_AES_IV.init(Cipher.DECRYPT_MODE, Private_Key);
Decrypted_AES_IV = cipher_AES_IV.doFinal(Encrypted_AES_IV);
//복호화한 AES IV base64 출력
System.out.println("[Decrypted IV] : " + Base64.getEncoder().encodeToString(Decrypted_AES_IV)+"\n");
```

그림 14 Server.c AES secret key, IV 복호화 by RSA private key

E. 서버 -> 클라이언트 Thread

연속적인 통신을 위해서 Thread를 이용한다. Server에서 Client로 메시지를 보내는 전용 Thread를 구축한다. Server에서 보낼 메시지를 입력을 받고 SeretKeySpec, IvParameterSpec으로 입력받은 메시지를 AES-CBC 암호화를 진행한다. 암호화된 메시지는 데이터 스트림의 writeObject() 함수를 사용하여 Client에게 전송된다.

```
//서버 -> 클라이언트 Thread
Scanner in = new Scanner(System.in);
byte[] finalDecrypted_AES_Key = Decrypted_AES_Key;
byte[] finalDecrypted_AES_IV = Decrypted_AES_IV;
Thread Server_thread = new Thread(new Runnable() {
    @Override
    public void run() {
        while (true) {
            try {
                //서버 -> 클라이언트 메시지 입력
                System.out.print("\n>");
                String msg = in.nextLine();

                //AES secret key, IV 초기화
                SecretKeySpec SecretKey_Spec = new SecretKeySpec(finalDecrypted_AES_Key, offset: 0, finalDecrypted_AES_Key.length, algorithm: "AES");
                IvParameterSpec IvParameter_Spec = new IvParameterSpec(finalDecrypted_AES_IV);
                //보낼 메시지(msg) AES-CBC 암호화
                Cipher cipher_AES_Data = Cipher.getInstance("AES/CBC/PKCS5Padding");
                cipher_AES_Data.init(Cipher.ENCRYPT_MODE, SecretKey_Spec, IvParameter_Spec);
                byte[] Encrypted_msg = cipher_AES_Data.doFinal(msg.getBytes(StandardCharsets.UTF_8));

                //클라이언트에게 암호화된 메시지(Encrypted_msg) 보냄
                Server_MSG.writeObject(Encrypted_msg);
            } catch (Exception e) {
            }
        }
    }
});
Server_thread.start();
```

그림 15 Server,c Server -> Client Thread

F. 클라이언트 -> 서버 Thread

Client에서 보낸 암호화된 메시지를 byte[]로 받는다. 이후 SecretKeySpec, IvParameterSpec으로

암호화된 메시지를 AES-CBC 복호화를 한다. 복호화된 메시지가 “exit”인지 확인하는 if문을 구상한다. “exit”이라면 먼저 복호화 메시지와 암호화 메시지를 Base64 인코딩으로 출력하고, Client에게 “exit” 메시지를 보내기 위하여 “exit” 메세지를 SecretkeySpec, IvParameterSpec으로 AES-CBC 암호화하여 Client에게 보낸다. 그리고 break를 통해서 Thread의 whilte(true)를 탈출하여 Connection을 종료한다. 물론 “exit” 메시지가 아니라면, 복호화 메시지와 암호화 메시지를 Base64 인코딩으로 출력하고 Thread를 유지한다. (while(true)때문에)

```
//클라이언트 -> 서버 Thread
Socket finalClient_Socket = Client_Socket;
ServerSocket finalServer_Socket = Server_Socket;
Thread Reiv_Thread = new Thread(new Runnable() {
    @Override
    public void run() {
        while (true) {
            try {
                //클라이언트 -> 서버 메시지 받음
                byte[] Encrypted_msg = (byte[]) Client_MSG.readObject();

                //AES secret key, IV 초기화
                SecretKeySpec SecretKey_Spec = new SecretKeySpec(finalDecrypted_AES_Key, 0, finalDecrypted_AES_Key.length, "AES");
                IvParameterSpec IvParameter_Spec = new IvParameterSpec(finalDecrypted_AES_IV);
                //받은 암호화된 메시지(Encrypted_msg) AES-CBC 복호화
                Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
                cipher.init(Cipher.DECRYPT_MODE, SecretKey_Spec, IvParameter_Spec);
                byte[] msg = cipher.doFinal(Encrypted_msg);

                //Timestamp를 위한 작업..
                Date date_now = new Date(System.currentTimeMillis());
                SimpleDateFormat date_format = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
                String today = date_format.format(date_now);
                //exit 문구 검사 용 bytep[] msg -> String msg_String 제작
                String msg_String = new String(msg, "UTF-8");

                //exit 문구 검사
                if (msg_String.equals("exit")) {
                    //통신 그만 하자~
                    //복호화한, 받은 메시지(msg) 정상 출력
                    System.out.println("Received : " + "\"" + msg_String + "\" " + "[" + today + "]");
                    //받은 메시지(Encrypted_msg) base64 출력
                    System.out.println("Encrypted Message :" + "\"" + Base64.getEncoder().encodeToString(Encrypted_msg));

                    Cipher cipher_AES_Data = Cipher.getInstance("AES/CBC/PKCS5Padding");
                    cipher_AES_Data.init(Cipher.ENCRYPT_MODE, SecretKey_Spec, IvParameter_Spec);
                    byte[] Encrypted_msg_new = cipher_AES_Data.doFinal(msg_String.getBytes(StandardCharsets.UTF_8));
                    Server_MSG.writeObject(Encrypted_msg_new);
                    break;
                }
                else{
                    //복호화한, 받은 메시지(msg) 정상 출력
                    System.out.println("Received : " + "\"" + msg_String + "\" " + "[" + today + "]");
                    //받은 메시지(Encrypted_msg) base64 출력
                    System.out.println("Encrypted Message :" + "\"" + Base64.getEncoder().encodeToString(Encrypted_msg) + "\n");
                    System.out.print("\n>");
                }

            } catch (Exception e) {
            }
        }
        //통신 종료 close
        System.out.println("Connection closed.");
        try {
            Server_MSG.close();
            Client_MSG.close();
            finalClient_Socket.close();
            finalServer_Socket.close();
            System.exit(0);
        } catch (Exception e) {
        }
    }
});
Client_Thread.start();
```

그림 16 Server.c Client -> Server Thread

2) Client.c

전체적인 구조는 앞서 설명한 Server.c 와 매우 비슷하다..

A. 클라이언트 -> 서버 접속

Server.c에서의 서버 port와 동일하게 하여 서버에 접속한다. 서버 실행 대기를 위해서 while문으로 Client socket이 null인 동안에 계속 대기한다.

```
//서버 접속
while(!check) {
    try {
        //클라이언트 -> 서버 접속
        Client_Socket = new Socket("127.0.0.1", 10101);
    } catch (IOException e) {
    }
    if(Client_Socket!=null) break;
}
```

그림 17 Client.c Client 생성

B. 데이터 스트림 생성

Server-Client 간의 통신을 주고받은 메시지 데이터 스트림을 생성한다.

```
//데이터 스트림 생성
ObjectOutputStream Client_MSG = new ObjectOutputStream(Client_Socket.getOutputStream());
ObjectInputStream Server_MSG = new ObjectInputStream(Client_Socket.getInputStream());
```

그림 18 Client.c dataStream 생성

C. RSA Public Key 수신

Server에서 보낸 RSA Public Key를 받고, Base64 인코딩 출력을 위해 byte[]로 바꾼 후 출력한다.

```
//서버가 보낸 public key 받음
PublicKey Public_Key = (PublicKey)Server_MSG.readObject();
//RSA public key base64 출력
byte[] Byte_PublicKey = Public_Key.getEncoded();
System.out.println("\n> [Received Public Key]: " + Base64.getEncoder().encodeToString(Byte_PublicKey)+"\n");
```

그림 19 Client.c RSA public key 수신

D. AES 생성 및 AES Secret Key, IV 전송

AES key generator로 256 bit의 secret key를 생성하고, IvParmeterSpec을 이용하여 random IV를 생성한다. 생성한 두 값을 출력하기 위해서 Byte[]로 변환한 후 Base64 인코딩을 하여 출력한다. Server에게 생성한 AES secret key와 IV를 전달하기 위해서 수신했던 RSA Public key로 암호화하여 안전하게 전달한다. 암호화한 AES secret key, IV를 Base64로 출력한다.

```
System.out.println("Creating AES-256 Key...\n");
//AES gernerator
KeyGenerator Key_Generator = KeyGenerator.getInstance("AES");
Key_Generator.init(256);
//AES secret key 생성
Secret_Key = Key_Generator.generateKey();
//AES radom IV 생성
SecureRandom random = new SecureRandom();
byte[] ivData = new byte[16];
random.nextBytes(ivData);
IvParameter_Spec = new IvParameterSpec(ivData);
//AES secret key, IV base64 출력
byte[] Byte_SecretKey = Secret_Key.getEncoded();
System.out.println("[AES 256 Key]: "+Base64.getEncoder().encodeToString(Byte_SecretKey));
System.out.println("-------------------------------------------------------------------------------------------");
System.out.println("[AES 256 IV]: "+Base64.getEncoder().encodeToString(IvParameter_Spec.getIV()));
System.out.println("-------------------------------------------------------------------------------------------");
//AES secret key를 RSA public key로 암호화
Cipher cipher_AES_Key = Cipher.getInstance("RSA/ECB/PKCS1Padding");
cipher_AES_Key.init(Cipher.ENCRYPT_MODE, Public_Key);
byte[] Encrypted_AES_Key = cipher_AES_Key.doFinal(Byte_SecretKey);
//암호화된 AES secret key base64 출력
System.out.println("[Encrypted AES Key] : "+Base64.getEncoder().encodeToString(Encrypted_AES_Key));
System.out.println("-------------------------------------------------------------------------------------------");
//AES IV를 RSA public key로 암호화
Cipher cipher_AES_IV = Cipher.getInstance("RSA/ECB/PKCS1Padding");
cipher_AES_IV.init(Cipher.ENCRYPT_MODE, Public_Key);
byte[] Encrypted_AES_IV = cipher_AES_IV.doFinal(IvParameter_Spec.getIV());
//암호화된 AES IV base64 출력
System.out.println("[Encrypted AES IV] : "+Base64.getEncoder().encodeToString(Encrypted_AES_IV)+"\n");
//서버에게 암호화된 AES secret key, IV 전송
Client_MSG.writeObject(Encrypted_AES_Key);
Client_MSG.flush();
Client_MSG.writeObject(Encrypted_AES_IV);
Client_MSG.flush();
```

그림 20 Client.c AES secret key, IV 생성 및 전송

E. 클라이언트 -> 서버 Thread

연속적인 통신을 위해서 Thread를 이용한다. Client에서 Server로 메시지를 보내는 전용 Thread를 구축한다. Client에서 보낼 메시지를 입력을 받고 AES Secret key와 IvParameterSpec으로 입력받은 메시지를 AES-CBC 암호화를 진행한다. 암호화된 메시지는 데이터 스트림의 writeObject() 함수를 사용하여 Server에게 전송된다

```
//클라이언트 -> 서버 Thread
Scanner in = new Scanner(System.in);
SecretKey finalSecret_Key = Secret_Key;
IvParameterSpec finalIvParameter_Spec = IvParameter_Spec;
Thread Client_Thread = new Thread(new Runnable() {
    @Override
    public void run() {
        while(true) {
            try {
                //클라이언트 -> 서버 메시지 입력
                System.out.print("\n>");
                String msg = in.nextLine();

                //보낼 메시지(msg) AES-CBC 암호화
                Cipher cipher_AES_Data = Cipher.getInstance("AES/CBC/PKCS5Padding");
                cipher_AES_Data.init(Cipher.ENCRYPT_MODE, finalSecret_Key, finalIvParameter_Spec);
                byte[] Encrypted_msg = cipher_AES_Data.doFinal(msg.getBytes(StandardCharsets.UTF_8));

                //서버에게 암호화된 메시지(Encrypted_msg) 보냄
                Client_MSG.writeObject(Encrypted_msg);
            }catch (Exception e) {
            }
        }
    }
});
Client_Thread.start();
```

그림 21 Client.c Client -> Server Thread

F. 서버 -> 클라이언트 Thread

Server에서 보낸 암호화된 메시지를 byte[]로 받는다. 이후 AES Secret key와 IvParameterSpec으로 암호화된 메시지를 AES-CBC 복호화를 한다. 복호화된 메시지가 “exit”인지 확인하는 if문을 구상한다. “exit”이라면 먼저 복호화 메시지와 암호화 메시지를 Base64 인코딩으로 출력하고, Client에게 “exit” 메시지를 보내기 위하여 “exit” 메세지를 AES Secret key, IvParameterSpec으로 AES-CBC 암호화하여 Server에게 보낸다. 그리고 break를 통해서 Thread의 whilte(true)를 탈출하여 Connection을 종료한다. 물론 “exit” 메시지가 아니라면, 복호화 메시지와 암호화 메시지를 Base64 인코딩으로 출력하고 Thread를 유지한다. (while(true)때문에)

```
//서버 -> 클라이언트 Thread
Socket finalClient_Socket = Client_Socket;
Thread Server_Thread = new Thread(new Runnable() {
    @Override
    public void run() {
        while(true) {
            try {
                //서버 -> 클라이언트 메시지 받음
                byte[] Encrypted_msg = (byte[])Server_MSG.readObject();

                //받은 암호화된 메시지(Encrypted_msg) AES-CBC 복호화
                Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
                cipher.init(Cipher.DECRYPT_MODE, finalSecret_Key, finalIvParameter_Spec);
                byte[] msg = cipher.doFinal(Encrypted_msg);

                //Timestamp를 위한 작업..
                Date date_now = new Date(System.currentTimeMillis());
                SimpleDateFormat date_format = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
                String today = date_format.format(date_now);

                //byte[] msg -> String msg_String
                String msg_String = new String(msg,"UTF-8");

                //exit 문구 검사
                if(msg_String.equals("exit")) {
                    //통신 그만 하자~
                    //복호화한, 받은 메시지(msg) 정상 출력
                    System.out.println("Received : " + "\"" + msg_String + "\" " + "[" + today + "]");
                    //받은 메시지(Encrypted_msg) base64 출력
                    System.out.println("Encrypted Message :"+ "\""+Base64.getEncoder().encodeToString(Encrypted_msg));

                    Cipher cipher_AES_Data = Cipher.getInstance("AES/CBC/PKCS5Padding");
                    cipher_AES_Data.init(Cipher.ENCRYPT_MODE, finalSecret_Key, finalIvParameter_Spec);
                    byte[] Encrypted_msg_new = cipher_AES_Data.doFinal(msg_String.getBytes(StandardCharsets.UTF_8));
                    Client_MSG.writeObject(Encrypted_msg_new);
                    break;
                }
                else{
                    //통신 계속 하자~
                    //복호화한, 받은 메시지(msg) 정상 출력
                    System.out.println("Received : " + "\"" + msg_String + "\" " + "[" + today + "]");
                    //받은 메시지(Encrypted_msg) base64 출력
                    System.out.println("Encrypted Message :"+ "\""+Base64.getEncoder().encodeToString(Encrypted_msg)+"\n");
                    System.out.print("\n>");

                }


            }catch (Exception e) {
            }
        }
        //통신 종료 close
        System.out.println("Connection closed.");
        try {
            Server_MSG.close();
            Client_MSG.close();
            finalClient_Socket.close();
            System.exit(0);
        }catch (Exception e) {
        }
    }
});
Server_Thread.start();
```

그림 22 Server -> Client Thread