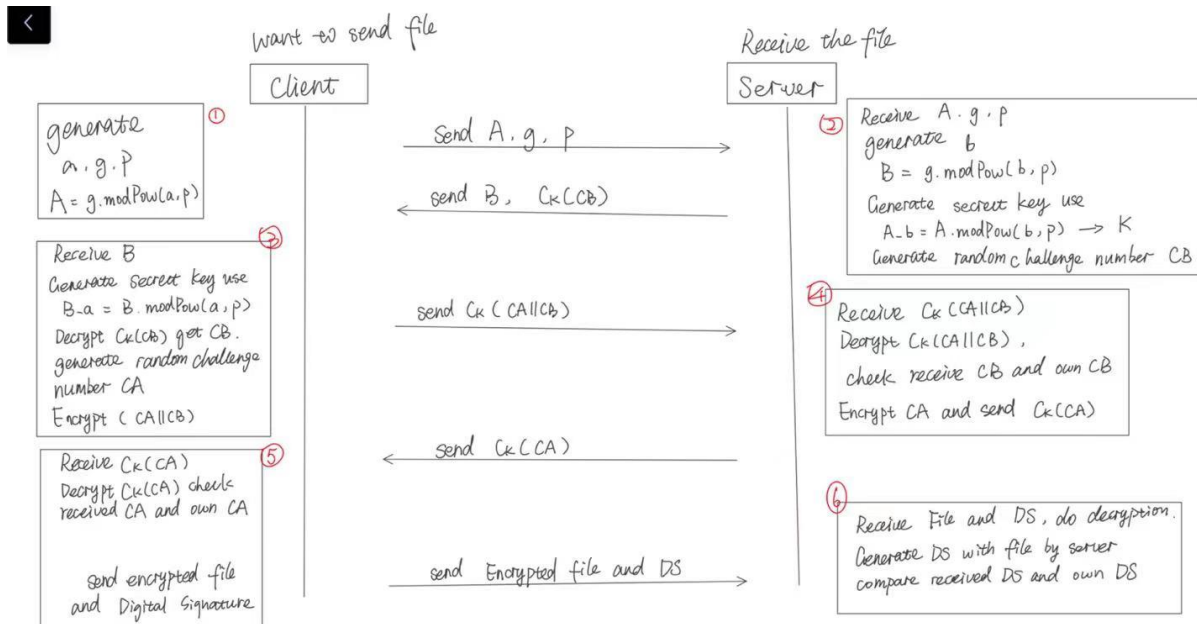


This is the read me file of **SecureFileTransfer**.

Ps: The ReadMe.docx and ReadMe.pdf have the **same content**. In case of you cannot open the file in right format.

This program does the authentication with **DH-EKE** algorithm and generate secret key in the process firstly. Then transfer the encrypted file (**AES**) and the generated digital signature with algorithm **MD5**. The server checks the DS to decide trust file or not.

The progress:



Authentication, DH-EKE:

1. Client Generate a, p, g

```
FileManage.java x FileTransferClient.java x FileTransferServer.java x MainFrame.java x MainPanel.java x DHCoder
43 private Socket client; //client socket
44 private DataOutputStream dos; //client data output stream
45 private DataInputStream dis; //client data input stream
46 private String sigA; //generated digital signature from client
47
48 private Key secretKeyA; //generated secret key of client
49
50 /**
51  * generate big integer a, p, g, calculate A
52  * @throws NoSuchAlgorithmException
53  * @throws InvalidParameterSpecException
54  */
55 private void generateAPG() throws NoSuchAlgorithmException, InvalidParameterSpecException {
56     a = DHCoder_EKE.getA();
57     BigInteger[] arr = DHCoder_EKE.getP_G();
58     p = arr[0];
59     g = arr[1];
60     A = g.modPow(a, p); // calculated public server key (A=g^a(modp))
61 }
```

2. **Server:** Receive A, g, p. Generate b, calculate B. Calculate secret key from A and b.

```
FileManage.java x FileTransferClient.java x FileTransferServer.java x DHCode
103 private void receiving(Socket socket){
104     try{
105         //initialize data input stream
106         dis = new DataInputStream(socket.getInputStream());
107         String fileName = "undefined";
108         while (true){
109             //read length of the msg
110             int len = dis.readInt();
111             //read specifier of the msg
112             int specifier = dis.readInt();
113             switch (specifier){
114                 //received p from client
115                 case pIdentifier:
116                     p = new BigInteger(dis.readUTF());
117                     break;
118                 //received g from client
119                 case gIdentifier:
120                     g = new BigInteger(dis.readUTF());
121                     break;
122             }
123         }
124     }
125 }
```

```
FileManage.java x FileTransferClient.java x FileTransferServer.java x DHCode_EKE.java x MainFrame.java x MainPanel.java x MDSDS.java x
122         break;
123     }
124     //receive A from client
125     case AIdentifier:
126         //read A from client
127         A = new BigInteger(dis.readUTF());
128         b = DHCode_EKE.getA();
129         B = g.modPow(b, p); // calculated public client key (B=g^b(modp))
130         //print GUI
131         MainPanel.getInstance().addTextServer( newStr: "A from client has been received!");
132     }
133     //send B to client
134     sendStr(B.toString(), BIdentifier);
135     MainPanel.getInstance().addTextServer( newStr: "B to client has been sent!");
136     //calculate A_b
137     BigInteger calculatedA_b = A.modPow(b, p);
138     //generate secret key in server
139     secretKeyB = DHCode_EKE.generateKey(calculatedA_b.toByteArray());
140     MainPanel.getInstance().addTextServer( newStr: "B secret key has been generated.");
141 }
142 //generate random challenge number in server
143 CB = DHCode_EKE.getA();
144 //print just for test man, for security element should be deleted.
145 System.out.println("secretB: " + Base64.encodeBase64String(secretKeyB.getEncoded()));
146 //send challenge number B to client
147 sendStr(Base64.encodeBase64String(DHCode_EKE.encrypt(CB.toString().getBytes(), secretKeyB)), CBIdentifier);
148 //print in GUI
149 MainPanel.getInstance().addTextServer( newStr: "Encrypted challenge Num from B has been sent.");
150 break;
151 }
```

3. **Client:** Receive B. Generate secret key with B and a. Decrypt received CB. Generate CA. Encrypt CA||CB.

```
FileManage.java x FileTransferClient.java x FileTransferServer.java x DHCoder_EKE.java x MainFrame.java x MainPanel.java x MD5DS.java x
100 //read msg length
101 int len = dis.readInt();
102 //read msg identifier
103 int specifier = dis.readInt();
104
105 switch (specifier){
106     //receive B from server
107     case BIdentifier:
108         B = new BigInteger(dis.readUTF());
109         BigInteger calculatedB_a = B.modPow(a, p);
110         //generate secret key in client
111         secretKeyA = DHCoder_EKE.generateKey(calculatedB_a.toByteArray());
112         //print just for test man, for security element should be deleted.
113         System.out.println("secretKeyA: " + Base64.encodeBase64String(secretKeyA.getEncoded()));
114         //print in GUI
115         MainPanel.getInstance().addTextClient( newStr: "A secret key has been generated.");
116         break;
117
118     //receive challenge number B from the server
119     case CBIdentifier:
120         CB = new String(DHCoder_EKE.decrypt(Base64.decodeBase64(dis.readUTF()),secretKeyA));
121         CA = DHCoder_EKE.getA();
122         //print just for test man, for security element should be deleted.
123         System.out.println("Client received CB: " + CB);
124         System.out.println("CA: " + CA);
125         //send encrypted (CA||CB) to server
126         sendStr(Base64.encodeBase64String(DHCoder_EKE.encrypt((CA + " " + CB).getBytes(), secretKeyA)), CA_CBIdentifier);
127         //print in GUI
128         MainPanel.getInstance().addTextClient( newStr: "Encrypted CB+CA has been sent.");
129         break;
```

4. **Server** received and decrypt CA||CB. Check CB is right or not. If check pass, encrypt and send CA.

```
FileManage.java x FileTransferClient.java x FileTransferServer.java x DHCoder_EKE.java x MainFrame.java x MainPanel.java x MD5DS.java x
149 MainPanel.getInstance().addTextServer( newStr: "Encrypted challenge Num from B has been sent.");
150 break;
151
152 //received (CA||CB) from client
153 case CA_CBIdentifier:
154     //decrypt received message
155     String[] receivedStr = new String(DHCoder_EKE.decrypt(Base64.decodeBase64(dis.readUTF()),secretKeyB)).split( regex: "\\s");
156     CA = receivedStr[0];
157     String receivedCB = receivedStr[1];
158     //print just for test man, for security element should be deleted.
159     System.out.println("receivedCA: " + CA);
160     System.out.println("receivedCB: " + receivedCB);
161     System.out.println("CB to string: " + CB.toString());
162
163     //do CB check
164     if(!receivedCB.equals(CB.toString())){
165         MainPanel.getInstance().addTextServer( newStr: "Check CB failed!");
166         socket.close();
167         break;
168     }
169     MainPanel.getInstance().addTextServer( newStr: "Check CB success!");
170
171     //send received decrypted CA to client
172     sendStr(Base64.encodeBase64String(DHCoder_EKE.encrypt(CA.getBytes(), secretKeyB)), CAIdentifier);
173     MainPanel.getInstance().addTextServer( newStr: "Encrypted CA has been sent.");
174     break;
175
```

5. Client: receive and decrypt CA, do the check. If check pass, do the file transition.

```
FileManage.java x FileTransferClient.java x FileTransferServer.java x DHCoder_EKE.java x MainFrame.java x MainPanel.java x MD5DS.java
127 //print in GUI
128 MainPanel.getInstance().addTextClient( newStr: "Encrypted CB+CA has been sent.");
129 break;
130
131 //receive challenge number A form the server
132 case CAIdentifier:
133     //decrypt received challenge number A
134     String receivedCA = new String(DHCoder_EKE.decrypt(Base64.decodeBase64(dis.readUTF()),secretKeyA));
135     //print just for test man, for security element should be deleted.
136     System.out.println("received CA: " + receivedCA);
137     //check received CA legal or not
138     if(!receivedCA.equals(CA.toString())){
139         MainPanel.getInstance().addTextClient( newStr: "Check CA failed!");
140         client.close();
141         break;
142     }
143     MainPanel.getInstance().addTextClient( newStr: "Authentication check pass.");
144     //send file
145     sendFile(filePath);
146     //generate digital signature
147     generateSigA(filePath);
148     //send digital signature to server
149     sendDS();
150     break;
151 default:
```

Symmetric Encryption: AES

1. Generate key for client and server in authentication progress:

```
FileManage.java x FileTransferClient.java x FileTransferServer.java x DHCoder_EKE.java x MainFrame.java x MainPanel.java
106 //receive B from server
107 case BIdentifier:
108     B = new BigInteger(dis.readUTF());
109     BigInteger calculatedB_a = B.modPow(a, p);
110     //generate secret key in client
111     secretKeyA = DHCoder_EKE.generateKey(calculatedB_a.toByteArray());
112     //print just for test man, for security element should be deleted.
113     System.out.println("secretKeyA: " + Base64.encodeBase64String(secretKeyA.getEncoded()));
114     //print in GUI
115     MainPanel.getInstance().addTextClient( newStr: "A secret key has been generated.");
116     break;
117
```

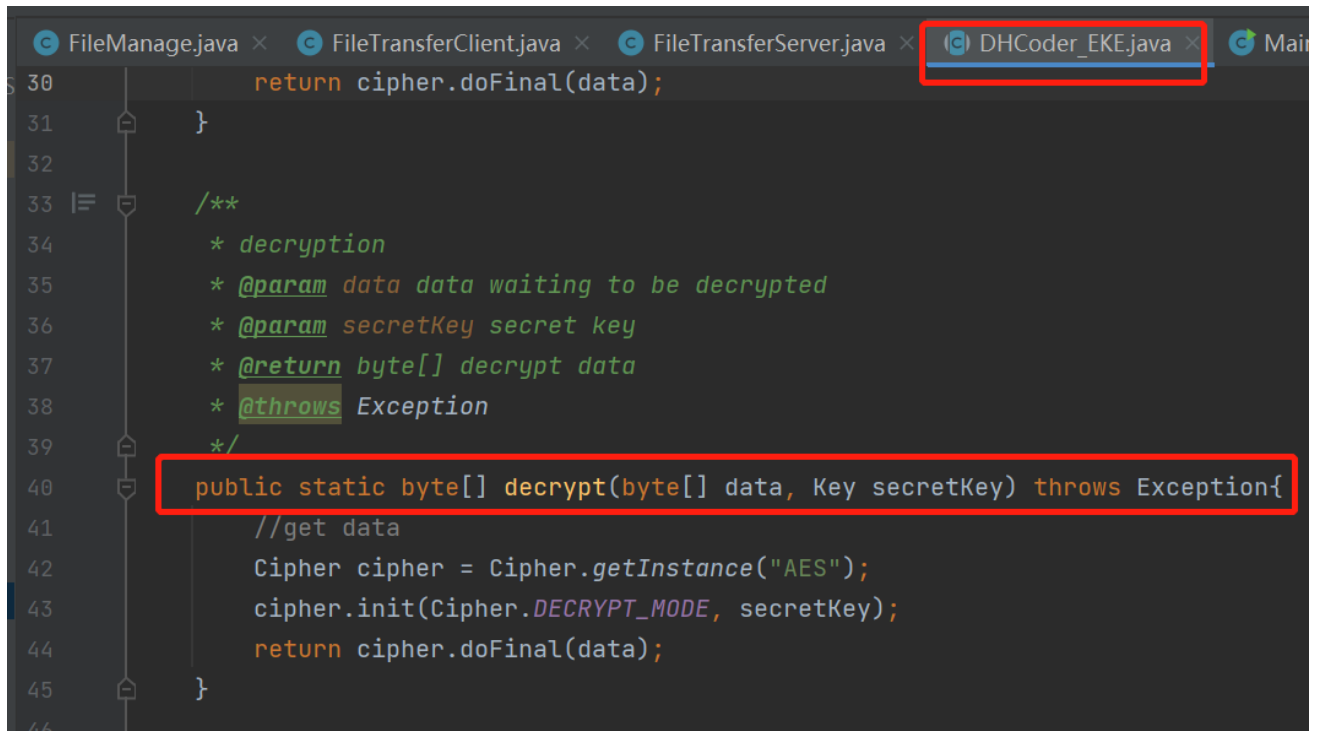
```
FileManage.java x FileTransferClient.java x FileTransferServer.java x DHCoder_EKE.java x MainFrame.java x MainPanel.java x MD5DS.java
122 //receive A from client
123 case AIdentifier:
124     //read A from client
125     A = new BigInteger(dis.readUTF());
126     b = DHCoder_EKE.getA();
127     B = g.modPow(b, p); // calculated public client key (B=g^b(modp))
128     //print GUI
129     MainPanel.getInstance().addTextServer( newStr: "A from client has been received!");
130
131     //send B to client
132     sendStr(B.toString(), BIdentifier);
133     MainPanel.getInstance().addTextServer( newStr: "B to client has been sent!");
134     //calculate A_b
135     BigInteger calculatedA_b = A.modPow(b, p);
136     //generate secret key in server
137     secretKeyB = DHCoder_EKE.generateKey(calculatedA_b.toByteArray());
138     MainPanel.getInstance().addTextServer( newStr: "B secret key has been generated.");
139
140     //generate random challenge number in server
141     CB = DHCoder_EKE.getA();
142     //print just for test man, for security element should be deleted.
143     System.out.println("secretB: " + Base64.encodeBase64String(secretKeyB.getEncoded()));
144     //send challenge number B to client
145     sendStr(Base64.encodeBase64String(DHCoder_EKE.encrypt(CB.toString().getBytes(), secretKeyB)), CBIdentifier);
146     //print in GUI
147     MainPanel.getInstance().addTextServer( newStr: "Encrypted challenge Num from B has been sent.");
148     break;
149
```

```
FileManage.java x FileTransferClient.java x FileTransferServer.java x DHCoder_EKE.java x MainFrame.java
69     arr[0] = dhSpec.getP();
70     arr[1] = dhSpec.getG();
71     return arr;
72 }
73
74 /**
75  * generate key from the input byte[]
76  * @param sharedKey
77  * @return
78  */
79 @ public static Key generateKey(byte[] sharedKey)
80 {
81     // AES supports 128 bit keys. So, just take first 16 bits of DH generated key.
82     byte[] byteKey = new byte[16];
83     for(int i = 0; i < 16; i++) {
84         byteKey[i] = sharedKey[i];
85     }
86
87     // convert given key to AES format
88     try {
89         Key key = new SecretKeySpec(byteKey, "AES");
90         return key;
91     } catch (Exception e) {
92         System.err.println("Error while generating key: " + e);
93     }
94     return null;
95 }
96 }
```

2. The encryption

```
FileManage.java x FileTransferClient.java x FileTransferServer.java x DHCoder_EKE.java x MainFrame.java
12 private static final String KEY_ALGORITHM = "DH";
13 /**
14  * Symmetric key algorithm: AES
15  */
16 private static final String SELECT_ALGORITHM = "AES";
17 //Djdk.crypto.KeyAgreement.legacyKDF=true
18
19 /**
20  * Encryption
21  * @param data data waiting encryption
22  * @param secretKey secret key
23  * @return byte[] encrypted data
24  * @throws Exception
25  */
26 public static byte[] encrypt(byte[] data, Key secretKey) throws Exception{
27     //encrypt data
28     Cipher cipher = Cipher.getInstance("AES");
29     cipher.init(Cipher.ENCRYPT_MODE, secretKey);
30     return cipher.doFinal(data);
31 }
32 }
```

3. The decryption

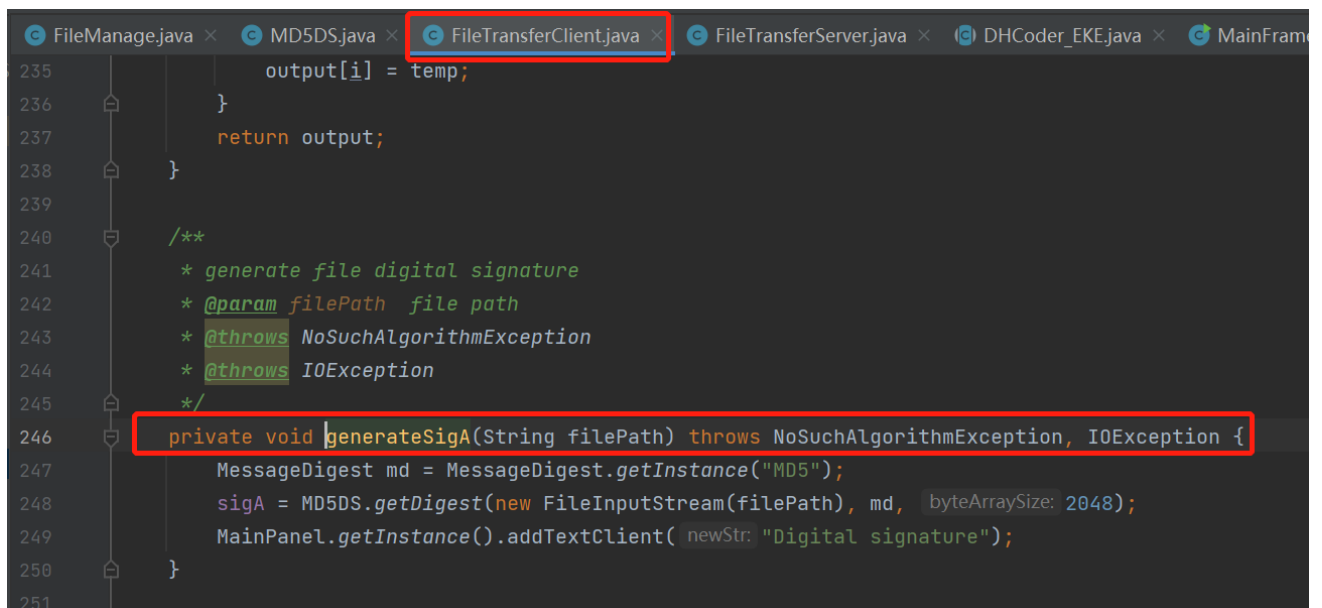


The screenshot shows an IDE with several tabs: FileManage.java, FileTransferClient.java, FileTransferServer.java, DHCoder_EKE.java (selected and highlighted with a red box), and MainFrame.java. The code in DHCoder_EKE.java is as follows:

```
30     return cipher.doFinal(data);
31 }
32
33 /**
34  * decryption
35  * @param data data waiting to be decrypted
36  * @param secretKey secret key
37  * @return byte[] decrypt data
38  * @throws Exception
39  */
40 public static byte[] decrypt(byte[] data, Key secretKey) throws Exception{
41     //get data
42     Cipher cipher = Cipher.getInstance("AES");
43     cipher.init(Cipher.DECRYPT_MODE, secretKey);
44     return cipher.doFinal(data);
45 }
46
```

Digital Signature: MD5

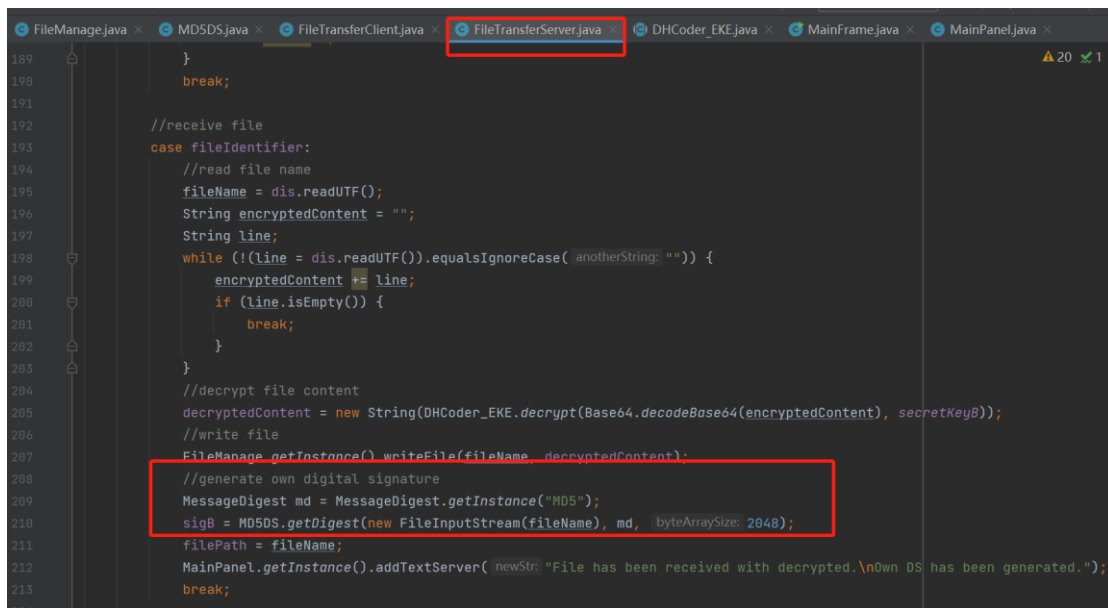
1. Generate DS in client:



The screenshot shows an IDE with several tabs: FileManage.java, MD5DS.java, FileTransferClient.java (selected and highlighted with a red box), FileTransferServer.java, DHCoder_EKE.java, and MainFrame.java. The code in FileTransferClient.java is as follows:

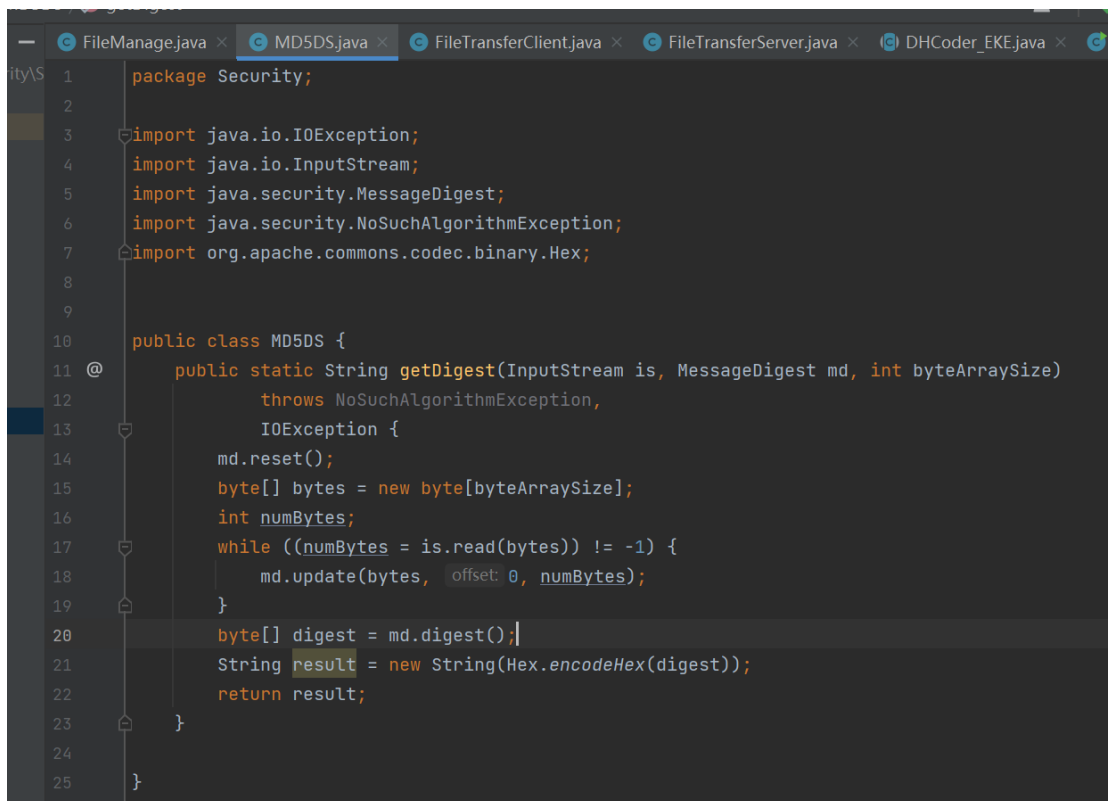
```
235     output[i] = temp;
236 }
237     return output;
238 }
239
240 /**
241  * generate file digital signature
242  * @param filePath file path
243  * @throws NoSuchAlgorithmException
244  * @throws IOException
245  */
246 private void generateSigA(String filePath) throws NoSuchAlgorithmException, IOException {
247     MessageDigest md = MessageDigest.getInstance("MD5");
248     sigA = MD5DS.getDigest(new FileInputStream(filePath), md, byteArraySize: 2048);
249     MainPanel.getInstance().addTextClient(newStr: "Digital signature");
250 }
251
```

2. Generate DS in server:



```
189 }
190 break;
191
192 //receive file
193 case fileIdentifier:
194     //read file name
195     fileName = dis.readUTF();
196     String encryptedContent = "";
197     String line;
198     while (! (line = dis.readUTF()).equalsIgnoreCase("anotherString: "")) {
199         encryptedContent += line;
200         if (line.isEmpty()) {
201             break;
202         }
203     }
204     //decrypt file content
205     decryptedContent = new String(DHCoder_EKE.decrypt(Base64.decodeBase64(encryptedContent), secretKeyB));
206     //write file
207     FileManager.getInstance().writeFile(fileName, decryptedContent);
208     //generate own digital signature
209     MessageDigest md = MessageDigest.getInstance("MD5");
210     sigB = MD5DS.getDigest(new FileInputStream(fileName), md, byteArraySize: 2048);
211     filePath = fileName;
212     MainPanel.getInstance().addTextServer( newStr: "File has been received with decrypted.\nOwn DS has been generated.");
213     break;
```

3. MD5DS class



```
1 package Security;
2
3 import java.io.IOException;
4 import java.io.InputStream;
5 import java.security.MessageDigest;
6 import java.security.NoSuchAlgorithmException;
7 import org.apache.commons.codec.binary.Hex;
8
9
10 public class MD5DS {
11     @
12     public static String getDigest(InputStream is, MessageDigest md, int byteArraySize)
13         throws NoSuchAlgorithmException,
14         IOException {
15         md.reset();
16         byte[] bytes = new byte[byteArraySize];
17         int numBytes;
18         while ((numBytes = is.read(bytes)) != -1) {
19             md.update(bytes, offset: 0, numBytes);
20         }
21         byte[] digest = md.digest();
22         String result = new String(Hex.encodeHex(digest));
23         return result;
24     }
25 }
```