Practice 7: KEY MANAGEMENT USING PUBLIC ENCRYPTION

7.1 OVERVIEW

7.1.1 Introduction

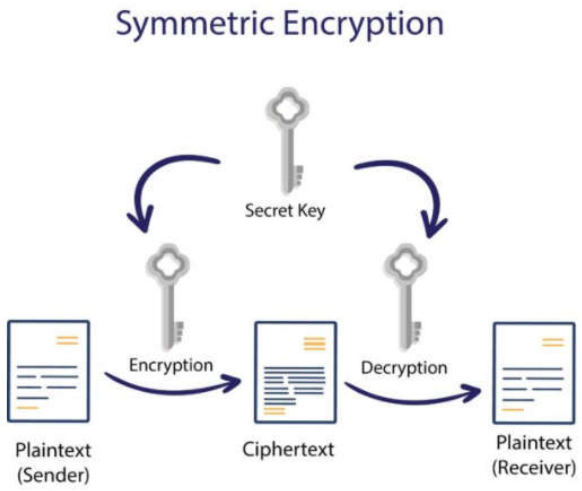
- Lab 7: Key Management using Public Encryption
- Practice time: class: 3 study hours, self-study: 3 study hours.
- Requirements: Students using Netbeans Software

7.1.2 Objective

- This course provides students with knowledge of cryptographic algorithms and how they are used in today's world.
- The content emphasizes the principles, topics, approaches, and problem solving related to the underlying technologies and architectures of the field.

7.2 CONTENTS

7.2.1 Basic knowledge



Encryption using asymmetric keys is a little more complicated than symmetric key encryption. A public key and a private key are two distinct keys that are used to encrypt and

decrypt data instead of the same key for both operations. These keys are made as a pair in order for them to relate to one another.

Description of the process:

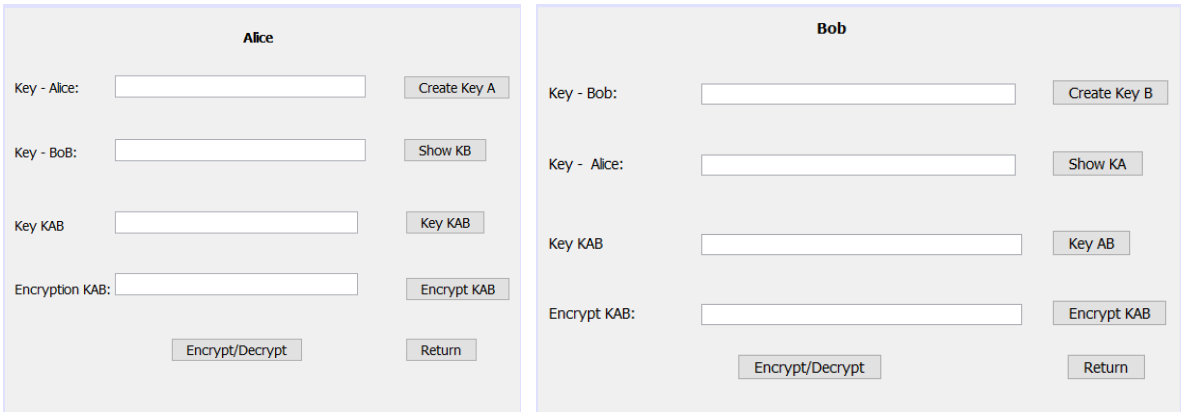
- The data is first encrypted-at-rest by a symmetric encryption key.
- The symmetric key is now encrypted by the public key of the person who the data is being sent to. That encrypted symmetric key and the ciphertext are sent to the recipient of the data.
- Once the ciphertext and key reach the recipient, the symmetric key is decrypted by that user's private key, and the ciphertext is decrypted.

7.2.2 Public Encryption

Write a program to encrypt and decrypt text with Key Management using Public Encryption. The program can perform the following functions:

- Design form follow image.
- Allow Create Key A, Key B and Key KAB.

❖ **Step 1: Design Form:**



❖ **Step 2: Create class with name: Crypto.java and write code for initialization function:**

```
public static final String toHexString(byte[] block)
{
        StringBuffer buf = new StringBuffer();
        int len = block.length;

        for (int i = 0; i < len; i++)
        {
                byte2hex(block[i], buf);
                if (i < len-1)
                {
                        buf.append(":");
                }
        }
        return buf.toString();
}

public static final void byte2hex(byte b, StringBuffer buf)
{
        char[] hexChars = { '0', '1', '2', '3',
                                        '4', '5', '6', '7',
                                        '8', '9', 'A', 'B',
                                        'C', 'D', 'E', 'F' };

        int high = ((b & 0xf0) >> 4);
        int low = (b & 0x0f);
        buf.append(hexChars[high]);
        buf.append(hexChars[low]);
}
```

❖ **Step 3: Write an event handler function for Alice:**

3.1 Variable environment:

```
public class Alice extends javax.swing.JFrame {

    KeyAgreement aliceKeyAgree;
    PublicKey bobPubKey;
    SecretKey aliceDesKey;
    Cipher aliceCipher;
    public Alice() {
        initComponents();
    }
```

3.2 Button Create Key A:

```
private void btnCreateKeyAActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        AlgorithmParameterGenerator paramGen=AlgorithmParameterGenerator.getInstance("DH");
        paramGen.init(512);
        AlgorithmParameters params=paramGen.generateParameters();
        DHParameterSpec dhSkipParamSpec=(DHParameterSpec) params.getParameterSpec(DHParameterSpec.class);
        System.out.println("Generating a DH Keypair...");
        KeyPairGenerator aliceKpairGen = KeyPairGenerator.getInstance("DH");
        aliceKpairGen.initialize(dhSkipParamSpec);
        KeyPair aliceKpair = aliceKpairGen.generateKeyPair();

        System.out.println("Initializing the KeyAgreement Engine with DH private key");
        aliceKeyAgree= KeyAgreement.getInstance("DH");
        aliceKeyAgree.init(aliceKpair.getPrivate());

        byte[] alicePubKeyEnc= aliceKpair.getPublic().getEncoded();
        FileOutputStream fos =new FileOutputStream("D:/A.pub");
        fos.write(alicePubKeyEnc);
        fos.close();
        txtKeyA.setText(alicePubKeyEnc.toString());
    } catch (Exception e) {
    }
}
```

3.3 Button Show Key B:

```
private void btnShowKhoaBActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        FileInputStream fis=new FileInputStream("D:/B.pub");
        byte[] bkeyP=new byte[fis.available()];
        fis.close();
        txtKeyB.setText(bkeyP.toString());
    } catch (Exception e) {
    }
}
```

3.4 Button Encrypt KAB

```
private void btnEncryptKABActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        aliceKeyAgree.doPhase(bobPubKey, true);
        aliceDesKey = aliceKeyAgree.generateSecret("DES");
        txtEncryptionKAB.setText(aliceDesKey.toString());
        BufferedWriter bw=null;
        String fileName="D:\\KeyA.txt";
        bw=new BufferedWriter(new FileWriter(fileName));
        bw.write(aliceDesKey.toString());
        bw.close();

    } catch (Exception e) {
    }
}
```

3.5 Button KeyAB

```
private void btnKeyABActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        FileInputStream fis=new FileInputStream("D:/B.pub");
        byte[] bobPubKeyEnc=new byte[fis.available()];
        fis.read(bobPubKeyEnc);
        fis.close();

        KeyFactory aliceKeyFac=KeyFactory.getInstance("DH");
        X509EncodedKeySpec x509KeySpec=new X509EncodedKeySpec(bobPubKeyEnc);
        bobPubKey= aliceKeyFac.generatePublic(x509KeySpec);
        System.out.println("Executing PHASE1 of key agreement...");
        aliceKeyAgree.doPhase(bobPubKey, true);
        byte[] aliceSharedSecret=aliceKeyAgree.generateSecret();

        System.out.println("Key KAB: secret (DEBUG ONLY): " +CryptoUtil.toHexString(aliceSharedSecret));
        txtKeyAB.setText(CryptoUtil.toHexString(aliceSharedSecret));
    } catch (Exception ex) { }
}
```

3.6 Button Encrypt/Decrypt:

```
private void btnEncryptDecryptActionPerformed(java.awt.event.ActionEvent evt) {
    DESCS des=new DESCS();
    des.setVisible(true);
}
```

3.7 Button Return:

```
private void btnReturnActionPerformed(java.awt.event.ActionEvent evt) {
        NewJFrame n=new NewJFrame();
       n.setVisible(true);

}
```

Step 4: Write an event handler function for Alice:

```
public class Bob extends javax.swing.JFrame {

    KeyAgreement bobKeyAgree;
    PublicKey alicePubKey;
    SecretKey bobDesKey;
    Cipher bobCipher;
    public Bob() {
        initComponents();
    }
```

4.1 Button Create Key B:

```
private void btnCreateKeyBActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        boolean read=false;
        while(!read){
            try {
                FileInputStream fis=new FileInputStream("D:/A.pub");
                fis.close();
                read=true;

            } catch (Exception ex) {}
        }
        FileInputStream fis =new FileInputStream("D:/A.pub");
        byte[] alicePubKeyEnc=new byte[fis.available()];
        fis.read(alicePubKeyEnc);
        fis.close();
        KeyFactory bobKeyFac=KeyFactory.getInstance("DH");
        X509EncodedKeySpec x509KeySpec =new X509EncodedKeySpec(alicePubKeyEnc);
        alicePubKey=bobKeyFac.generatePublic(x509KeySpec);
        DHParameterSpec dhParamSpec =( (DHPublicKey)alicePubKey).getParams();
        System.out.println("Generate DH keypair...");
        KeyPairGenerator bobKpairGen =KeyPairGenerator.getInstance("DH");
        bobKpairGen.initialize(dhParamSpec);
        KeyPair bobKpair=bobKpairGen.generateKeyPair();
        System.out.println("initializing KeyAgreement engine...");
        bobKeyAgree=KeyAgreement.getInstance("DH");
        bobKeyAgree.init(bobKpair.getPrivate());
        byte[] bobPubKeyEnc=bobKpair.getPublic().getEncoded();
        FileOutputStream fos=new FileOutputStream("D:/B.pub");
        fos.write(bobPubKeyEnc);
        fos.close();
        txtkhoab.setText(bobPubKeyEnc.toString());
    } catch (Exception e) {
    }
}
```

4.2 Button Show KA:

```
private void btnShowKAActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        FileInputStream fis=new FileInputStream("D:/A.pub");
        byte[] akeyP=new byte[fis.available()];
        fis.read(akeyP);
        fis.close();
        txtkhoaa.setText(akeyP.toString());
    } catch (Exception e) {
    }
}
```

4.3 Button Key KAB:

```
private void btnKeyABActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        bobKeyAgree.doPhase(alicePubKey, true);
        byte[] bobSharedSecret = bobKeyAgree.generateSecret();
        System.out.println("Key KAB : shared secret (DEBUG ONLY)"+ CryptoUtil.toHexString(bobSharedSecret));
        txtkhoachung.setText(CryptoUtil.toHexString(bobSharedSecret));
    } catch (Exception e) {
    }
}
```

4.4 Button Encrypt KAB:

```
private void btnEncryptKABActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        bobKeyAgree.doPhase(alicePubKey, true);
        bobDesKey = bobKeyAgree.generateSecret("DES");
        txtmahoakab.setText(bobDesKey.toString());
        BufferedWriter bw=null;
        String fileName="D:\\KhoaB.txt";
        bw=new BufferedWriter(new FileWriter(fileName));
        bw.write(bobDesKey.toString());
        bw.close();

    } catch (Exception e) {
    }
}
```

4.5 Button Encrypt/Decrypt:

```
private void btnEncryptDecryptActionPerformed(java.awt.event.ActionEvent evt) {
    DESCS des=new DESCS();
    des.setVisible(true);
}
```

4.6 Button Return:

```
private void btnReturnActionPerformed(java.awt.event.ActionEvent evt) {
        NewJFrame n=new NewJFrame();
    n.setVisible(true);

}
```

Result: