# Lab 5

## Symmetric key Encryption and Signed Message Digest

## Objectives

- Be familiar with Java Cryptographic Extension (JCE).
- Learn how to use JCE to encrypt files.
- Learn how to create a signed digest of file for data integrity using JCE.

## Tasks:

1. Encrypt and decrypt text file using symmetric key (DesTextStartingCode.java)

2. Encrypt image file using symmetric key in ECB versus CBC mode (DesImageStartingCode.java)

3. Generate a signed message digest for text file, and sign the digest using asymmetric key (RSA). (DigitalSignatureStartingCode.java)

## Task 1: Encrypt&Decrypt Text File

Encrypt and decrypt the given text files (e.g., smallSize.txt) using symmetric key.

1. Read the plain file using *BufferReader*
2. Encrypt the file using *Cipher, SecretKey*
3. Print the encrypted bytes using *DatatypeConverter*
4. Decrypt the ciphertext using *Cipher*

# Task 1: Encrypt&Decrypt Text File

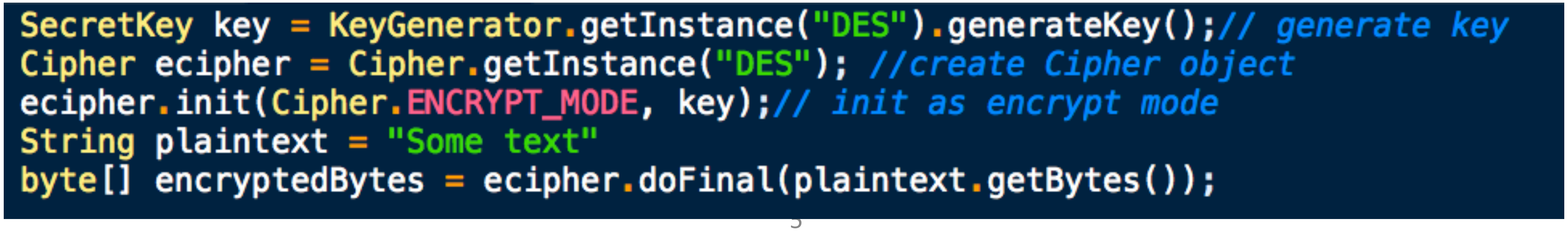
1. Read the file using *BufferReader*

   Already given in *startingcode.java*

2. Encrypt the file content using *Cipher*

   *generateKey(), create Cipher object, init Cipher and doFinal()*

Example:



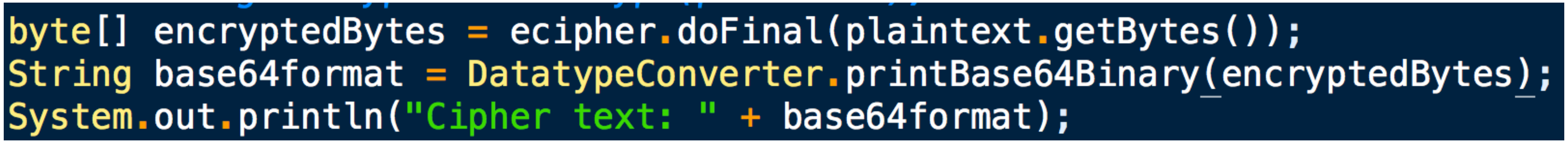
```java
SecretKey key = KeyGenerator.getInstance("DES").generateKey();// generate key
Cipher ecipher = Cipher.getInstance("DES"); //create Cipher object
ecipher.init(Cipher.ENCRYPT_MODE, key);// init as encrypt mode
String plaintext = "Some text"
byte[] encryptedBytes = ecipher.doFinal(plaintext.getBytes());
```

# Task 1: Encrypt&Decrypt Text File

3. Print the encrypted bytes using *DatatypeConverter*

As default format of return value in encryption is byte[]. You need to convert byte[] to String format if you want to print the encrypted message.

Example:



```
byte[] encryptedBytes = ecipher.doFinal(plaintext.getBytes());
String base64format = DatatypeConverter.printBase64Binary(encryptedBytes);
System.out.println("Cipher text: " + base64format);
```

## Task 1: Encrypt&Decrypt Text File

4. Decrypt the encrypted bytes using *Cipher*

Similar with the encrypt process. Initialize *Cipher* as *DECRYPT_MODE* and *doFinal().*

## Task 2: Encrypt Image File

Encrypt the image file SUTD.bmp using DES (symmetric key). The image is an uncompressed 2D bitmap in RGBA format (each pixel = 32bits, same as an int).

1. Read the image into *int[][]* array using *BufferedImage*

2. Encrypt each column of *int[][]* image array using *Cipher* (Cipher takes plaintext as byte[] and produces ciphertext as byte[])

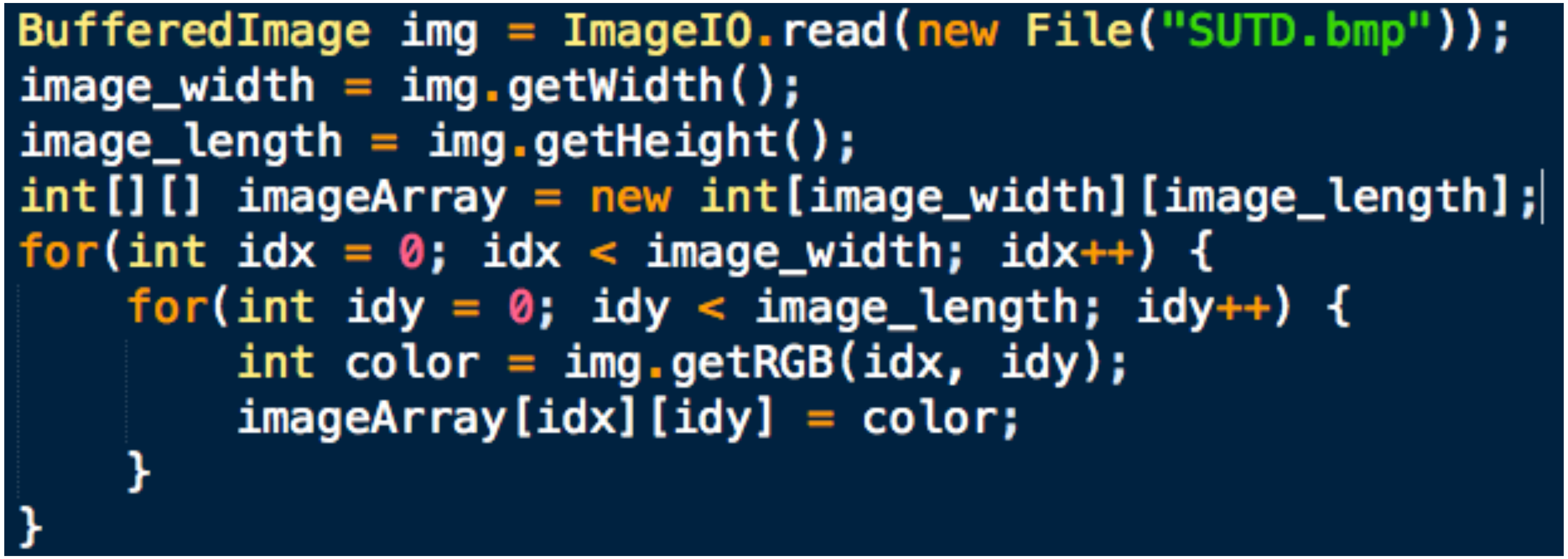
3. Write the encrypted image into a new image file using *BufferedImage*

## Task 2: Encrypt Image File

1. Read the image into *int[][]* array using *BufferedImage*

Functions: *.getWidth(), .getHeight(), .getRGB()* etc.

Example:



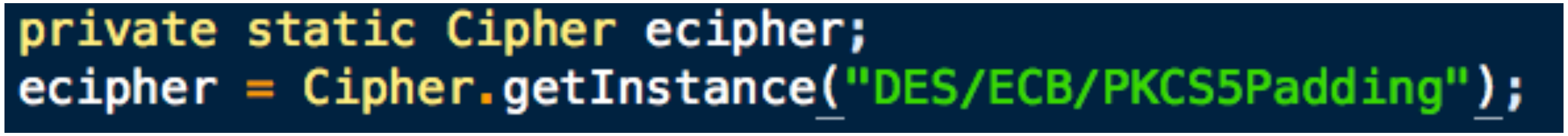
```
BufferedImage img = ImageIO.read(new File("SUTD.bmp"));
image_width = img.getWidth();
image_length = img.getHeight();
int[][] imageArray = new int[image_width][image_length];
for(int idx = 0; idx < image_width; idx++) {
    for(int idy = 0; idy < image_length; idy++) {
        int color = img.getRGB(idx, idy);
        imageArray[idx][idy] = color;
    }
}
```

## Task 2: Encrypt Image File

2. Encrypt each column of *int[][]* array using *Cipher*

Similar to the encryption in Task 1. Recall DES is a block cipher – you can select ECB or CBC mode and the padding method (PKCS5) when creating Cipher object

- ECB: code each block (64 bits) independently
- CBC (block chaining): use result of encrypting the previous block to encrypt each current block
- PKCS5: pad plaintext to multiple of 64bits, details not important



```
private static Cipher ecipher;
ecipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
```

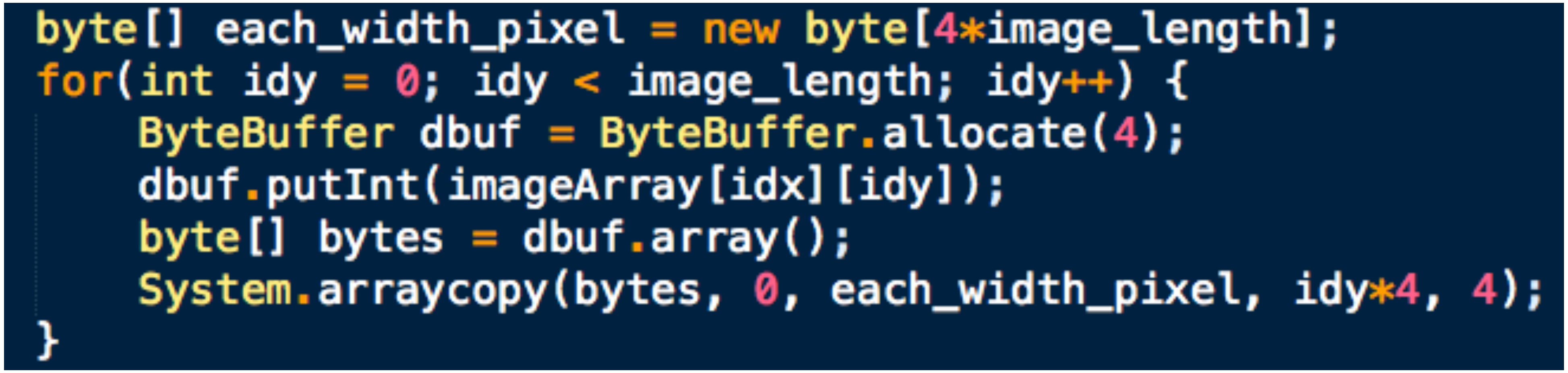
# Task 2: Encrypt Image File

2. Encrypt each column of *int[][]* array using *Cipher*

Use *ByteBuffer* to convert each column of *int[]* into *byte[]* format (Cipher operates on byte[]).

    1) convert one int (4 bytes) at a time using *ByteBuffer*

    2) concatenate all the converted bytes together using *System.arraycopy().*



```
byte[] each_width_pixel = new byte[4*image_length];
for(int idy = 0; idy < image_length; idy++) {
    ByteBuffer dbuf = ByteBuffer.allocate(4);
    dbuf.putInt(imageArray[idx][idy]);
    byte[] bytes = dbuf.array();
    System.arraycopy(bytes, 0, each_width_pixel, idy*4, 4);
}
```

## Task 2: Encrypt Image File

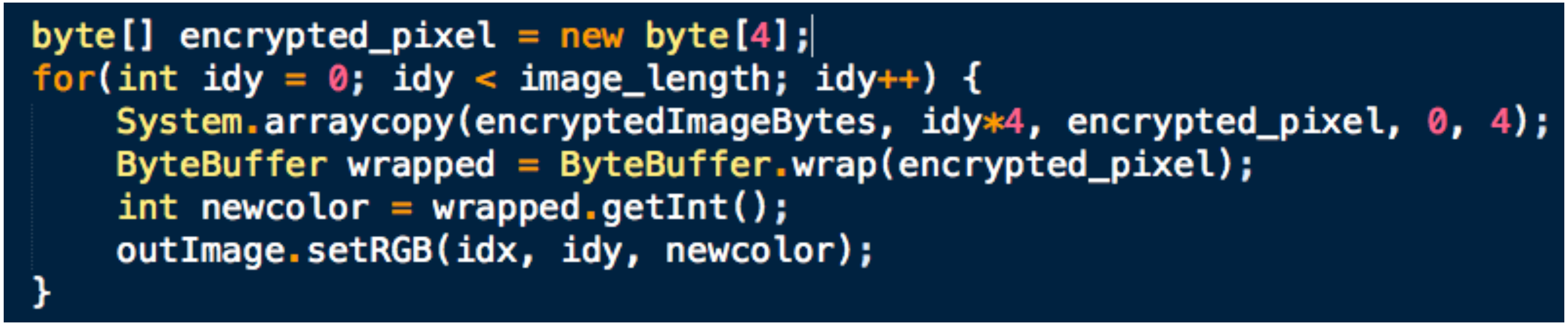
3. Append the encrypted column of pixels into new image file using *BufferedImage.*

Use *ByteBuffer* and *System.arraycopy()* to convert the ciphertext (*byte[]*) into column of pixels (*int[]*).

Then, similar with the first step, use:

```
public void setRGB(int x, int y, int rgb)
```

to set the RGB value of each pixel of output image.



```
byte[] encrypted_pixel = new byte[4];
for(int idy = 0; idy < image_length; idy++) {
    System.arraycopy(encryptedImageBytes, idy*4, encrypted_pixel, 0, 4);
    ByteBuffer wrapped = ByteBuffer.wrap(encrypted_pixel);
    int newcolor = wrapped.getInt();
    outImage.setRGB(idx, idy, newcolor);
}
```

## Task 3: Signed Message Digest

Create the signed message digest for text files (e.g., smallSize.txt) using MD5, and sign the digest using RSA (asymmetric key).

1. Read the file using *BufferReader*
2. Create signed message digest for file content using *MessageDigest*
3. Encrypt the digest by private key using *Cipher*
4. Decrypt the digest by public key using *Cipher*

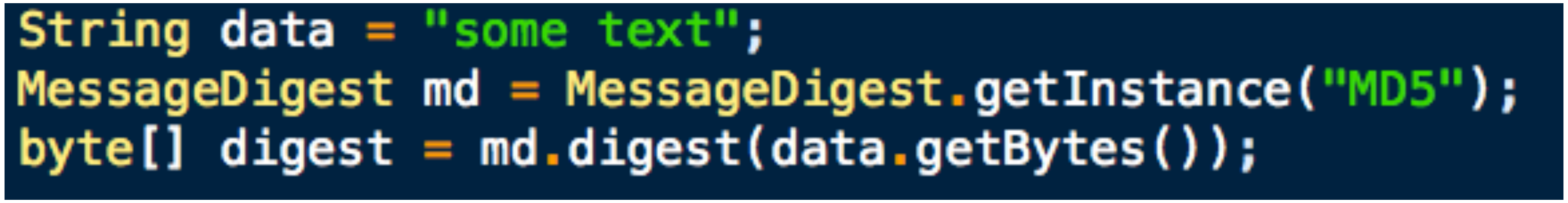
# Task 3: Signed Message Digest

1. Read the file using *BufferReader*

Already given in *startingcode.java*

2. Create signed message digest for file using *MessageDigest*

Using *MessageDigest* class and functions:

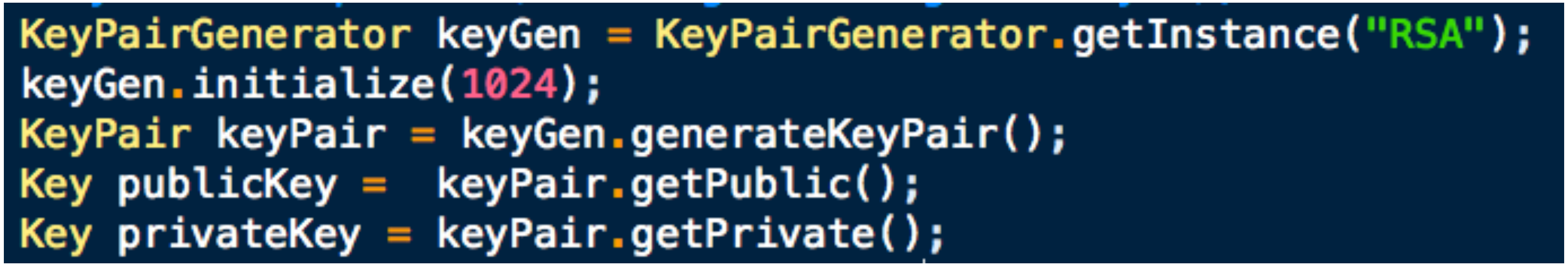


```
String data = "some text";
MessageDigest md = MessageDigest.getInstance("MD5");
byte[] digest = md.digest(data.getBytes());
```

# Task 3: Signed Message Digest

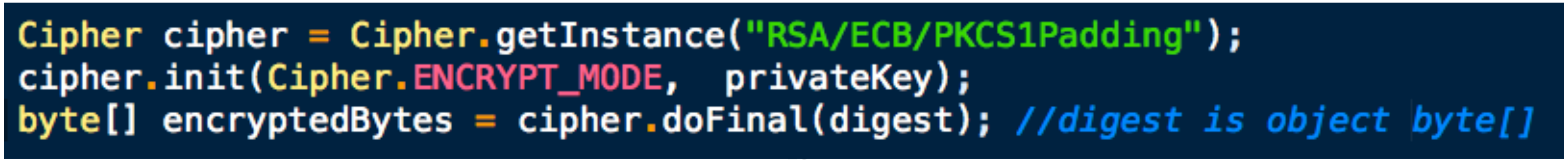
3. Encrypt the digest by private key using *Cipher*

You need to generate a pair of private and public keys using KeyPairGenerator, KeyPair, Key and functions:



```
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
keyGen.initialize(1024);
KeyPair keyPair = keyGen.generateKeyPair();
Key publicKey =  keyPair.getPublic();
Key privateKey = keyPair.getPrivate();
```

Then encrypt the digest using private key. Same as task 1, use Cipher object. But configure the object to do RSA.



```
Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
cipher.init(Cipher.ENCRYPT_MODE,  privateKey);
byte[] encryptedBytes = cipher.doFinal(digest); //digest is object byte[]
```

## Task 3: Signed Message Digest

4. Decrypt the signed digest by public key using *Cipher*

Similar to step 3, decrypt the encrypted digest using the generated public key.

## Requirement

1. A document which illustrates outputs of the codes and the answers for all the questions in handout.

2. Completed  source codes for Task1/Task2/Task3.

3. Encrypted .BMP images in Task2 (both ECB and CBC modes).