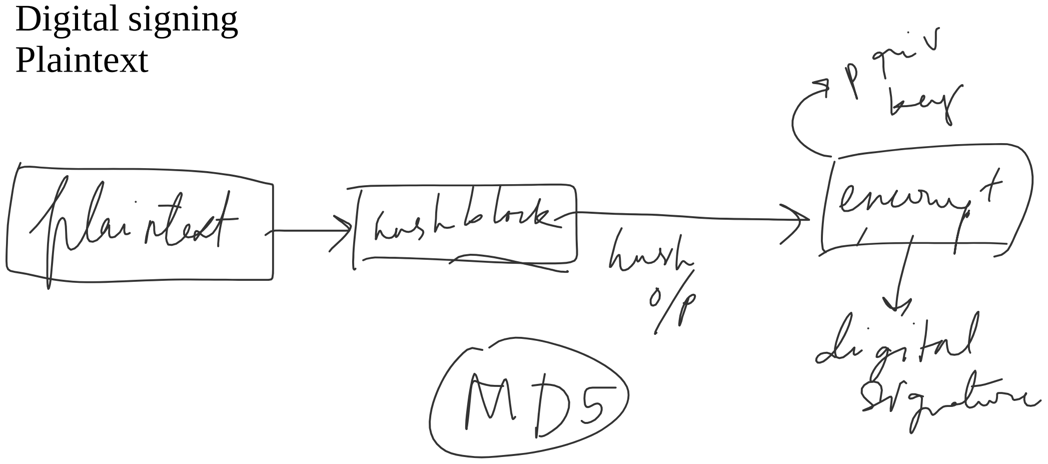
4. Lab 4

Sunday, February 21, 2021 9:24 AM

Java Socket programming
Socket:
Input stream

Digital signing
Plaintext



```
import java.security.*;
import javax.crypto.*;
public class AsymmetricCrypto {

    public static void main(String[] args) throws Exception{
        //encryption and decryption using RSA algorithm
        //KeypairGenerator object
        //generate a private and public key
        //algorithm to generate a keypair (with size) to encrypt and decrypt data

        //docs.oracle.com/javase/8/docs/api --> documentation can be used in lab 
exams (partially open)
        KeyPairGenerator keyPairGeneratorObject = KeyPairGenerator.getInstance("R
SA");
        keyPairGeneratorObject.initialize(1024);
        KeyPair keyPairObject = keyPairGeneratorObject.generateKeyPair();
        PublicKey publicKeyObject = keyPairObject.getPublic();
        PrivateKey privateKeyObject = keyPairObject.getPrivate();
        //use a Cipher object
        Cipher cipherObject = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        //padding is only for block cipher
        String toEncrypt = "We are learning about public key cryptography";
        byte[] toEncryptBytes = toEncrypt.getBytes();

        cipherObject.init(Cipher.ENCRYPT_MODE, publicKeyObject);
        byte[] encrypted = cipherObject.doFinal(toEncryptBytes);
        System.out.println("Encrypted: " + new String(encrypted));
        cipherObject.init(Cipher.DECRYPT_MODE, privateKeyObject);
        byte[] decrypted = cipherObject.doFinal(encrypted);
        System.out.println("Decrypted: " + new String (decrypted));
        }
    }
```

```
import java.security.*;
public class DigitalSigning{
    public static void main(String[] args) throws Exception{
        //find the digest of an given message
        //docs.oracle.com/javase/8/docs/api --> documentation can be used in lab
exams (partially open)

        MessageDigest messageDigestObject = MessageDigest.getInstance("MD5");
        String toFindDigest = "We are discussing digital signing.";
        byte[] toFindDigestBytes = toFindDigest.getBytes();
        byte[] digestOut = messageDigestObject.digest(toFindDigestBytes);
        String digestOutString = new String(digestOut);
        System.out.println("Digest: " + digestOutString);

        MessageDigest messageDigestObject1 = MessageDigest.getInstance("SHA-256")
;
        String toFindDigest1 = "We are discussing digital signing.";
        byte[] toFindDigestBytes1 = toFindDigest1.getBytes();
        byte[] digestOut1 = messageDigestObject1.digest(toFindDigestBytes1);
        String digestOutString1 = new String(digestOut1);
        System.out.println("Digest: " + digestOutString1);
        //when digital verification is done at receiver's end
        //both the sender and receiver choose similar digest algorithms
        //sender --> original text + signature
    }
}
```

```
import java.security.*;
import javax.crypto.*;
public class DigitalSigning2 {
    public static void main(String[] args) throws Exception{
        MessageDigest messageDigestObject = MessageDigest.getInstance("MD5");
        String toFindDigest = "We are discussing digital signing.";
        byte[] toFindDigestBytes = toFindDigest.getBytes();
        byte[] digestOut = messageDigestObject.digest(toFindDigestBytes);
        String digestOutString = new String(digestOut);
        System.out.println("Digest(MD5): " + digestOutString);

        KeyPairGenerator keyPairGeneratorObject = KeyPairGenerator.getInstance("R
SA");
        keyPairGeneratorObject.initialize(1024);
        KeyPair keyPairObject = keyPairGeneratorObject.generateKeyPair();
        PublicKey publicKeyObject = keyPairObject.getPublic();
        PrivateKey privateKeyObject = keyPairObject.getPrivate();
        //use a Cipher object
        Cipher cipherObject = Cipher.getInstance("RSA/ECB/PKCS1Padding");
```

```java
        //padding is only for block cipher

        cipherObject.init(Cipher.ENCRYPT_MODE, publicKeyObject);
        byte[] encrypted = cipherObject.doFinal(digestOut);
        cipherObject.init(Cipher.DECRYPT_MODE, privateKeyObject);
        byte[] decrypted = cipherObject.doFinal(encrypted);
    }
}
```