Blockchain Technology

Assignment – 03

Aim: **Program to Generate Asymmetric key.**

Description:

1. javax.crypto.spec.SecretKeySpec:

* Constructs a secret key from the given byte array.
* This constructor does not check if the given bytes indeed specify a secret key of the specified algorithm. For example, if the algorithm is DES, this constructor does not check if key is 8 bytes long, and also does not check for weak or semi-weak keys. In order for those checks to be performed.

1. java.security.KeyPair:

* This class is a simple holder for a key pair (a public key and a private key). It does not enforce any security, and, when initialized, should be treated like a PrivateKey.

1. java.security.KeyPairGenerator:

* The KeyPairGenerator class is used to generate pairs of public and private keys. Key pair generators are constructed using the getInstance factory methods (static methods that return instances of a given class).
* A Key pair generator for a particular algorithm creates a public/private key pair that can be used with this algorithm. It also associates algorithm-specific parameters with each of the generated keys.

1. import javax.crypto.SecretKey;:

* A secret (symmetric) key. The purpose of this interface is to group (and provide type safety for) all secret key interfaces.
* Provider implementations of this interface must overwrite the equals and hashCode methods inherited from Object, so that secret keys are compared based on their underlying key material and not based on reference.

1. import java.security.PrivateKey:

* A private key. The purpose of this interface is to group (and provide type safety for) all private key interfaces.

1. java.security.PublicKey:

* A public key. This interface contains no methods or constants. It merely serves to group (and provide type safety for) all public key interfaces. Note: The specialized public key interfaces extend this interface.

1. java.security.KeyFactory:

* Key factories are used to convert keys (opaque cryptographic keys of type Key) into key specifications (transparent representations of the underlying key material), and vice versa.
* Key factories are bi-directional. That is, they allow you to build an opaque key object from a given key specification (key material), or to retrieve the underlying key material of a key object in a suitable format.

1. java.security.NoSuchAlgorithmException:

* This exception is thrown when a particular cryptographic algorithm is requested but is not available in the environment.

Code:

import java.security.spec.\*;

import java.security.KeyPair;

import java.security.KeyPairGenerator;

import javax.crypto.SecretKey;

import java.security.PrivateKey;

import java.security.PublicKey;

import java.security.KeyFactory;

import java.security.NoSuchAlgorithmException;

public class GeneratePublicPrivateKey{

    public static void generateKey(String keyAlgorithm, int numBits) throws NoSuchAlgorithmException, InvalidKeySpecException{

        KeyPairGenerator keygen = KeyPairGenerator.getInstance(keyAlgorithm);

        keygen.initialize(numBits);

        System.out.println("\nFor Algorithm: " + keyAlgorithm);

        KeyPair keypair = keygen.generateKeyPair();

        PrivateKey privateKey1 = keypair.getPrivate();

        PublicKey publicKey1 = keypair.getPublic();

        byte[] private1 = privateKey1.getEncoded();

        byte[] public1 = publicKey1.getEncoded();

        String privateFormat = privateKey1.getFormat();

        String publicFormat = publicKey1.getFormat();

        System.out.println("Private format: " + privateFormat + "\nPublic format: " + publicFormat);

        KeyFactory keyfactory = KeyFactory.getInstance(keyAlgorithm);

        EncodedKeySpec privateKeySpec = new PKCS8EncodedKeySpec(private1);

        PrivateKey privateKey2 = keyfactory.generatePrivate(privateKeySpec);

        EncodedKeySpec publicKeySpec = new X509EncodedKeySpec(public1);

        PublicKey publicKey2 = keyfactory.generatePublic(publicKeySpec);

        System.out.println("Are both private keys equal? " + privateKey1.equals(privateKey2));

        System.out.println("Are both public keys equal? " + publicKey1.equals(publicKey2));

        System.out.println("Are private key1 and public key1 equal? " + privateKey1.equals(publicKey1));

        System.out.println("Are private key2 and public key2 equal? " + privateKey2.equals(publicKey2));

    }

    public static void main(String[] args) throws NoSuchAlgorithmException, InvalidKeySpecException{

        generateKey("RSA", 1024);

        generateKey("DSA", 1024);

        generateKey("DH", 1024);

    }

}

Output:

For Algorithm: RSA

Private format: PKCS#8

Public format: X.509

Are both private keys equal? true

Are both public keys equal? true

Are private key1 and public key1 equal? false

Are private key2 and public key2 equal? false

For Algorithm: DSA

Private format: PKCS#8

Public format: X.509

Are both private keys equal? true

Are both public keys equal? true

Are private key1 and public key1 equal? false

Are private key2 and public key2 equal? false

For Algorithm: DH

Private format: PKCS#8

Public format: X.509

Are both private keys equal? true

Are both public keys equal? true

Are private key1 and public key1 equal? false

Are private key2 and public key2 equal? false