1.Description

In this project, we will discuss the generating a public-private key pair. In this project, we will create a pair using Java. The Cryptographic Algorithm we will use in this project is RSA. Asymmetric Cryptography, also known as Public Key Cryptography, is an encryption system in which two different but uniquely related cryptographic keys are used. The data encrypted using one key can be decrypted with the other. These keys are known as Public and Private Key Pair, and as the name implies the private key must remain private while the public key can be distributed.

**2.Objectives**

* To exchange a secret key surely
* To encrypt in data transmission
* To make sure the data connot be read by others during transmission

**3.Motivation**

* Data encrypted with a private key can be decrypted only with the corresponding public key.
* This is not a recommended practice to encrypt sensitive data, however, because it means that anyone with the public key, which is by definition published, could decrypt the data.
* Nevertheless, private-key encryption is useful because it means the private key can be used to sign data with a digital signature, an important requirement for electronic commerce and other commercial applications of cryptography.
* Client software such as Mozilla Firefox can then use the public key to confirm that the message was signed with the appropriate private key and that it has not been tampered with since being signed.

**4.Public-key cryptography**

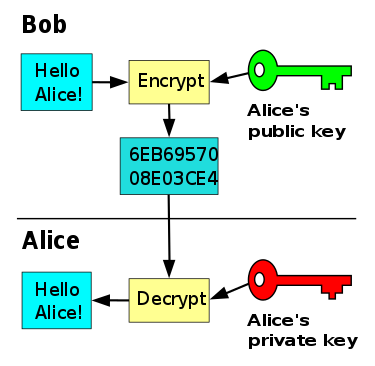
**Public-key cryptography**, or **asymmetric cryptography**, is a cryptographic system that uses pairs of keys: *public keys* which may be disseminated widely, and *private keys* which are known only to the owner. The generation of such keys depends on cryptographic algorithms based on mathematical problems to produce one-way functions. Effective security only requires keeping the private key private; the public key can be openly distributed without compromising security.

In such a system, any person can encrypt a message using the receiver's *public key*, but that encrypted message can only be decrypted with the receiver's *private key*.

Robust authentication is also possible. A sender can combine a message with a private key to create a short *digital signature* on the message. Anyone with the sender's corresponding public key can combine the same message and the supposed digital signature associated with it to verify whether the signature was valid, i.e. made by the owner of the corresponding private key.

Public key algorithms are fundamental security ingredients in modern cryptosystems, applications and protocols assuring the confidentiality, authenticity and non-repudiability of electronic communications and data storage.

* 1. **Two of the best-known uses of public key cryptography**
* *Public key encryption*, in which a message is encrypted with a recipient's public key. The message cannot be decrypted by anyone who does not possess the matching private key, who is thus presumed to be the owner of that key and the person associated with the public key. This is used in an attempt to ensure confidentiality.
* *Digital signatures*, in which a message is signed with the sender's private key and can be verified by anyone who has access to the sender's public key. This verification proves that the sender had access to the private key, and therefore is likely to be the person associated with the public key. This also ensures that the message has not been tampered with, as a signature is mathematically bound to the message it originally was made with, and verification will fail for practically any other message, no matter how similar to the original message.



In an asymmetric key encryption scheme, anyone can encrypt messages using the public key, but only the holder of the paired private key can decrypt. Security depends on the secrecy of the private key.

**5. Demonstration**

## **1. Generate a Public-Private Key Pair**

In this project, we will create a pair using Java. The Cryptographic Algorithm we will use in this project is RSA.

//GenerateKeys.java

package com.mkyong.keypair;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.security.KeyPair;

import java.security.KeyPairGenerator;

import java.security.NoSuchAlgorithmException;

import java.security.NoSuchProviderException;

import java.security.PrivateKey;

import java.security.PublicKey;

public class GenerateKeys {

private KeyPairGenerator keyGen;

private KeyPair pair;

private PrivateKey privateKey;

private PublicKey publicKey;

public GenerateKeys(int keylength) throws NoSuchAlgorithmException, NoSuchProviderException {

this.keyGen = KeyPairGenerator.getInstance("RSA");

this.keyGen.initialize(keylength);

}

public void createKeys() {

this.pair = this.keyGen.generateKeyPair();

this.privateKey = pair.getPrivate();

this.publicKey = pair.getPublic();

}

public PrivateKey getPrivateKey() {

return this.privateKey;

}

public PublicKey getPublicKey() {

return this.publicKey;

}

public void writeToFile(String path, byte[] key) throws IOException {

File f = new File(path);

f.getParentFile().mkdirs();

FileOutputStream fos = new FileOutputStream(f);

fos.write(key);

fos.flush();

fos.close();

}

public static void main(String[] args) {

GenerateKeys gk;

try {

gk = new GenerateKeys(1024);

gk.createKeys();

gk.writeToFile("KeyPair/publicKey",gk.getPublicKey().getEncoded());

gk.writeToFile("KeyPair/privateKey", gk.getPrivateKey().getEncoded());

} catch (NoSuchAlgorithmException | NoSuchProviderException e) {

System.err.println(e.getMessage());

} catch (IOException e) {

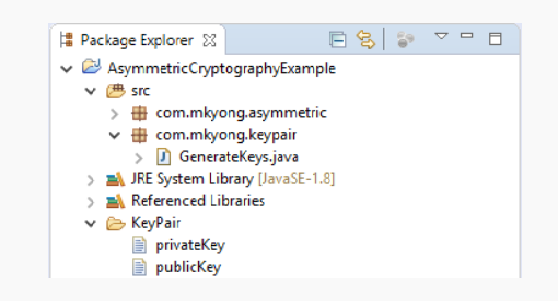
System.err.println(e.getMessage());

}

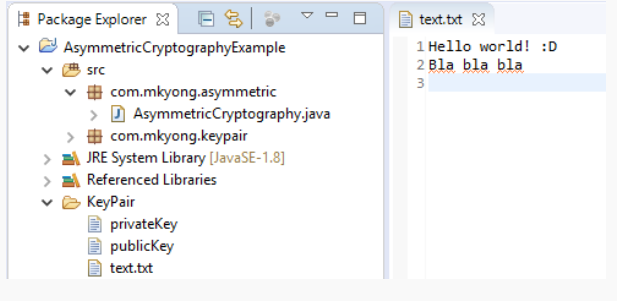
}

}

**Output:**

****

## **2. Create a text file to encrypt**



## **3. Use the Key Pair to encrypt and decrypt data**

In this project, we create a class that can load the Public and the Private keys from their files and then uses them to encrypt and decrypt a String and a File.

// AsymmetricCryptography.java

package com.mkyong.asymmetric;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.UnsupportedEncodingException;

import java.nio.file.Files;

import java.security.GeneralSecurityException;

import java.security.InvalidKeyException;

import java.security.KeyFactory;

import java.security.NoSuchAlgorithmException;

import java.security.PrivateKey;

import java.security.PublicKey;

import java.security.spec.PKCS8EncodedKeySpec;

import java.security.spec.X509EncodedKeySpec;

import javax.crypto.BadPaddingException;

import javax.crypto.Cipher;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.NoSuchPaddingException;

import org.apache.commons.codec.binary.Base64;

public class AsymmetricCryptography {

private Cipher cipher;

public AsymmetricCryptography() throws NoSuchAlgorithmException, NoSuchPaddingException {

this.cipher = Cipher.getInstance("RSA");

}

public PrivateKey getPrivate(String filename) throws Exception {

byte[] keyBytes = Files.readAllBytes(new File(filename).toPath());

PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(keyBytes);

KeyFactory kf = KeyFactory.getInstance("RSA");

return kf.generatePrivate(spec);

}

public PublicKey getPublic(String filename) throws Exception {

byte[] keyBytes = Files.readAllBytes(new File(filename).toPath());

X509EncodedKeySpec spec = new X509EncodedKeySpec(keyBytes);

KeyFactory kf = KeyFactory.getInstance("RSA");

return kf.generatePublic(spec);

}

public void encryptFile(byte[] input, File output, PrivateKey key)

throws IOException, GeneralSecurityException {

this.cipher.init(Cipher.ENCRYPT\_MODE, key);

writeToFile(output, this.cipher.doFinal(input));

}

public void decryptFile(byte[] input, File output, PublicKey key)

throws IOException, GeneralSecurityException {

this.cipher.init(Cipher.DECRYPT\_MODE, key);

writeToFile(output, this.cipher.doFinal(input));

}

private void writeToFile(File output, byte[] toWrite)

throws IllegalBlockSizeException, BadPaddingException, IOException {

FileOutputStream fos = new FileOutputStream(output);

fos.write(toWrite);

fos.flush();

fos.close();

}

public String encryptText(String msg, PrivateKey key)

throws NoSuchAlgorithmException, NoSuchPaddingException,

UnsupportedEncodingException, IllegalBlockSizeException,

BadPaddingException, InvalidKeyException {

this.cipher.init(Cipher.ENCRYPT\_MODE, key);

return Base64.encodeBase64String(cipher.doFinal(msg.getBytes("UTF-8")));

}

public String decryptText(String msg, PublicKey key)

throws InvalidKeyException, UnsupportedEncodingException,

IllegalBlockSizeException, BadPaddingException {

this.cipher.init(Cipher.DECRYPT\_MODE, key);

return new String(cipher.doFinal(Base64.decodeBase64(msg)), "UTF-8");

}

public byte[] getFileInBytes(File f) throws IOException {

FileInputStream fis = new FileInputStream(f);

byte[] fbytes = new byte[(int) f.length()];

fis.read(fbytes);

fis.close();

return fbytes;

}

public static void main(String[] args) throws Exception {

AsymmetricCryptography ac = new AsymmetricCryptography();

PrivateKey privateKey = ac.getPrivate("KeyPair/privateKey");

PublicKey publicKey = ac.getPublic("KeyPair/publicKey");

String msg = "Cryptography is fun!";

String encrypted\_msg = ac.encryptText(msg, privateKey);

String decrypted\_msg = ac.decryptText(encrypted\_msg, publicKey);

System.out.println("Original Message: " + msg +

"\nEncrypted Message: " + encrypted\_msg

+ "\nDecrypted Message: " + decrypted\_msg);

if (new File("KeyPair/text.txt").exists()) {

ac.encryptFile(ac.getFileInBytes(new File("KeyPair/text.txt")),

new File("KeyPair/text\_encrypted.txt"),privateKey);

ac.decryptFile(ac.getFileInBytes(newFile

("KeyPair/text\_encrypted.txt")),

new File("KeyPair/text\_decrypted.txt"), publicKey);

} else {

System.out.println("Create a file text.txt under folder KeyPair");

}

}

}

**Output:**

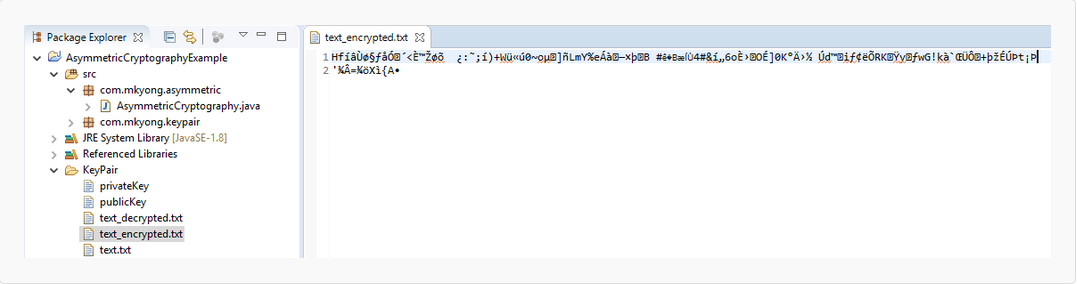
Original Message: Cryptography is fun!

Encrypted Message:

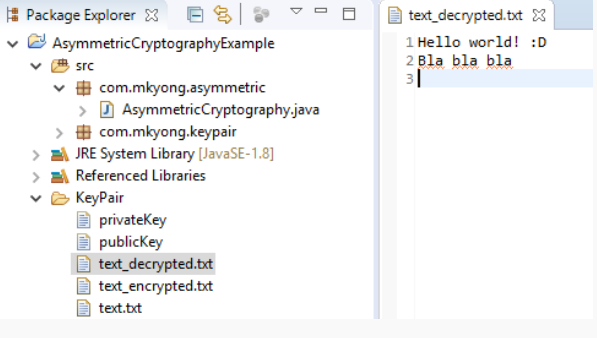
NZY+9v4AWoADVBQiFmS8ake6dG8M9v4WDf4TSPKgGPoKeJtHxEWVFGFL57qR2mbuknn8WYjfjcN+BA/RDsDQ9Q5SxQFLi8GE7A/6fdURjO+Vz3qYFaefpXccp7SnObKXnFfUNp00m5BtMSdTLEYfngF9UDBVYVFz0EZwuyP50xY=

Decrypted Message: Cryptography is fun!

**Encrypted text:**

****

**Decrypted text:**

****

**6. Conclusion**

* This project is about the encryption of the message by using the public key cryptograhpy.
* Public-key cryptography refers to encryption methods that use two keys, one for encryption and another for decryption.
* Public-key cryptography can be seen as a supercharged version of secret key cryptography.
* The current technology used to perform public-key cryptography is called RSA.
* Our project public-key cryptography is intended to transmit data securely.

**7. References**

* <https://en.wikipedia.org/wiki/Public-key_cryptography>
* https://access.redhat.com/documentation/en-US/Red\_Hat\_Certificate\_System\_Common\_Criteria\_Certification/8.1/html/Deploy\_and\_Install\_Guide/Introduction\_to\_Public\_Key\_Cryptography.html