A common paradigm in a centralised client-server system is that a private key is held securely on the server, while the corresponding public key is distributed to all clients.

**RSA encryption in Java**

https://www.javamex.com/tutorials/cryptography/rsa\_algorithm.shtml đọc hết để hiểu RSA

**Creating an RSA key pair in Java**

- Java provides the KeyPairGenerator class for performing this task

we create an instance of KeyPairGenerator suitable for generating RSA keys;

we initialise the generator, telling it the bit length of the modulus that we require;

we call genKeyPair(), which eventually returns a KeyPair object;

we call getPublic() and getPrivate() on the latter to pull out the public and private keys.

- Pseudocode

KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");

kpg.initialize(2048); => độ dài của key càng lớn càng bảo mật nhưng trade off bằng thời gian giải mã "With every doubling of the RSA key length, decryption is 6-7 times times slower."

KeyPair kp = kpg.genKeyPair();

Key publicKey = kp.getPublic();

Key privateKey = kp.getPrivate();

We now have two Key objects. Whilst we could immediately use these to start encrypting and decrypting, in most practical uses, we want to store these two keys for later use. For that, we use a KeyFactory to pull out the components (modulus and exponents) of the keys.

**Saving the public and private key**

Typically, the private key will be placed on our server, and the public key distributed to clients. To store the key, we simply need to pull out the modulus and the public and private exponents, then write these numbers to some file (or put in whatever convenient place).

The Key interface allows us to pretend for a second that we don't need to worry about the algorithm-specific details of keys. But unfortunately, in practice we do. So there also exist "key specification" classes— RSAPublicKeySpec and RSAPrivateKeySpec in this case— with transparent methods for pulling out the parameters that make up the key. Then, a KeyFactory allows us to translate between Keys and their corresponding specification. It's a bit clumsy, but the code ends up as follows:

- Pseudocode

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

RSAPublicKeySpec pub = fact.getKeySpec(kp.getPublic(),

RSAPublicKeySpec.class);

RSAPrivateKeySpec priv = fact.getKeySpec(kp.getPrivate(),

RSAPrivateKeySpec.class);

saveToFile("public.key", pub.getModulus(),

pub.getPublicExponent());

saveToFile("private.key", priv.getModulus(),

priv.getPrivateExponent());

To save the moduli and exponents to file, we can just use boring old serialisation, since the modulus and exponents are just BigInteger objects:

- Pseudocode

public void saveToFile(String fileName,

BigInteger mod, BigInteger exp) throws IOException {

ObjectOutputStream oout = new ObjectOutputStream(

new BufferedOutputStream(new FileOutputStream(fileName)));

try {

oout.writeObject(mod);

oout.writeObject(exp);

} catch (Exception e) {

throw new IOException("Unexpected error", e);

} finally {

oout.close();

}

}

Sau khi kết thúc đoạn code ta có được 2 file public.key và private.key. Đối với file private.key chúng ta có thể dùng key đó và nhúng vào code như 1 biến or có thể lưu nó vào server( thường lưu vào sv được ưu tiên hơn vì sau này có thể sẽ có việc thay đổi private key của client)

Saving the keys to file simply makes it a bit easier to manage keys in some cases (for example, we might change keys periodically and distribute new key files to the clients; distributing a few bytes is often more practical than distributing the entire code base).

**Encryption and Decryption process**

https://www.javamex.com/tutorials/cryptography/rsa\_encryption\_2.shtml

**-- References**

https://www.javamex.com/tutorials/cryptography/asymmetric.shtml => giới thiệu asym encryption và RSA encryption

https://www.javamex.com/tutorials/cryptography/rsa\_encryption.shtml =>

https://www.javamex.com/tutorials/cryptography/rsa\_key\_length.shtml => độ dài của key liên quan đến thời gian giải mã (tham khảo cho biết)

**-- Haven't look at**

http://tutorials.jenkov.com/java-cryptography/index.html

https://introcs.cs.princeton.edu/java/99crypto/RSA.java.html

https://www.mkyong.com/java/java-asymmetric-cryptography-example/

--New terminology

Block cipher

RSA

RSA key lengths

Modulus

Java Cryptography Architecture (JCA):

JCA is structured around some central general purpose classes and interfaces. The real functionality behind these interfaces are provided by providers. Thus, you may use a Cipher class to encrypt and decrypt some data, but the concrete cipher implementation (encryption algorithm) depends on the concrete provider used.

You can implement and plugin your own providers too, but you should be careful with that. Implementing encryption correctly without security holes is hard! Unless you know what you are doing, you are probably better off using the builtin Java provider, or use a well established provider like Bouncy Castle.

Different ways to use public and private key

Using the private key for encrypting and the public for decrypting — that’s the process of signing a message/verifying that a message is coming from a particular source.

Using the public key for encrypting and the private for decrypting — that’s secure communication.