**Cs585 Project 2 Documentation**

**Date: 4/25/2014**

**Team names: Luan Pham, David Smiths**

**File Included: KeyExchange.java**

**poem.txt**

**Makefile**

**Description:**

The goal of this project is to implement a secure communication by establishing a key agreement among two parties using Diffie-hellman technique. Once both parties shared their public key and agree upon the same parameters of the Diffie-Hellman, they can generate the secret keys, and use it to encrypt and decrypt ciphertext among themselves. The requirement for this project is to encrypt a poem from one computer and decrypt it from another computer using the key generated from the DH technique.

**Usage(example):**

1. Alice generates a keypair (alice, alice.pub) and sends Bob alice.pub(which include the parameters of the Diffie-hellman) (cmd: java -jar build/target.jar --gen 1024 alice)

2. Bob receives alice.pub(also DH parameters) and generates a symmetric keypair (bob, bob.pub). (cmd: java -jar build/target.jar --gen alice.pub bob)

3. Bob generates an AES key bob.aes using Alice public key and bob private key(cmd: java -jar build/target.jar --gen-aes bob alice.pub bob.aes)

4. Bob encrypts poem.txt using this key as poem.txt.aes (cmd: java -jar build/target.jar --encrypt bob.aes poem.txt poem.txt.aes) and sends bob.pub and poem.txt.aes to Alice.

5. Alice generates an AES key alice.aes using bob public key and alice private (cmd: java -jar build/target.jar --gen-aes alice bob.pub alice.aes)

6. Alice decrypts poem.txt.aes using this key as poem2.txt (cmd: java -jar build/target.jar --decrypt alice.aes poem.txt.aes poem2.txt)

Note\* the transfer of files between Bob and Alice are done using email, usb, etc.

**Implementation:**

The program was written in JAVA and we use the JAVA build in Security Library for Diffie-Hellman key agreement and AES for encryption/decryption. The program works by using Command Line Input (CLI), which takes the user input and call the appropriate function correspond to the inputs.

***Functions in KeyAgreement.java:***

**Main(String [ ] args)** - This function handles the user inputs and perform various task requested by the user. Based on the user input, it will call the appropriate function to perform the task.

- if args is empty => printUsage()

- if length(args) = 3 and args[0] = “--gen” => generate new keypair and save it to file name args[2].

*example - if args = “--gen 1024 bob.keykp”, then call generateKeyPair(1024) and save the keypair to bob.kp by calling saveKeyPair(kp,bob.kyp.*

- if length(args) = 4 and args[0] = “--gen-aes” > generate AES key and save it it to a filename args[3].

args[1] and args[2] provide the private key and the public key(of the other user). Then calling the function generateSecretKey will return an AES key.

*example- if args “--gen-aes Alice.privateKey Bob.publicKey alice.aes”, then call generateSecretKey(alice.privateKey,Bob.publicKey) will return an AES key and savit it to filename alice.aes.*

if length(args) = 4 and args[0] = “--encrypt” => encrypt the file using provided AES key from args and save it.

*for example, if args = “--encrypt alice.aes poem.txt poem.enc”, then call*

*encrypt(byte poem.txt, alice.aes) will return an ecrypted poem and save it to poem.enc.*

if length(args) = 4 and args[0] = “--decrypt” => decrypt the file using provided AES key from args and save it.

*for example, if args =”--decrypt bob.aes poem.enc poem.txt”, then call*

*decrypt(byte poem.enc, bob.aes) will return a decrypted poem and save it to*

*poem.txt.*

***public static void printUsage ()*** - Prints the usage for the application

***public static KeyPair generateKeyPair (Integer size)*** - Generates a NEW keypair for Diffie-Hellman given an integer size.

***public static KeyPair generateKeyPair (DHParameterSpec params)*** - Generates a NEW keypair for DH using existing parameters of DH.

***public static void saveKeyPair (KeyPair kp, String filename)*** - Saves the given keypair to a file pair (filename for private key, filename.pub for public key)

***public static SecretKey generateSecretKey (PrivateKey privateKey, PublicKey publicKey)*** - Generates a shared SecretKey using a (Private,Public) keypair

***public static byte[] encrypt (byte[] plaintext, SecretKey key)*** - Encrypts a byte-string of plaintext using the given SecretKey. this function will use build in class “Cipher” with parameters AES/ECB/PKCS5PADDING for encryption.

***public static byte[] decrypt (byte[] ciphertext, SecretKey key)*** *-* Decrypts a byte string using the given SecretKey. This function will use build in class “Cipher” with parameters AES/ECB/PKCS5PADDING for decryption.

***public static String readFile(String path, Charset encoding)*** - Reads a file and returns its string representation.

***public static byte[] readByteFile (String filename)*** - Reads a file and returns its byte[] representation.

***public static void writeFile(String filename, byte[] contents)*** - Writes the given byte[] to a file

**Functions Testing:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function | Input | Expected Output | Actual Output | Successful |
| printUsage | null | print Application usages | application usages | yes |
| generate KeyPair | 1024 | new DH key size 1024 | DH key size 1024 | yes |
| generate  KeyPair | DH parrameters | DH key with existing parameter | DH key with existing parameter | yes |
| generate  SecretKey | Bob public  Alice private | AES key | AES key | yes |
| generate  SecretKey | Alice public  Bob private | AES KEY | AES key | yes |
| encrypt | plainText  Alice AES key | encrypted poem | encrypted poem | yes |
| encrypt | plainText  Bob AES key | encrypted poem | encrypted poem | yes |
| decrypt | bob-Text.enc  alice aes | decrypted poem | decrypted poem | yes |
| decrypt | alice-Text.enc  bob aes | decrypted poem | decrypted poem | yes |
| read file | poem.txt | string poem | string poem | yes |
| readByteFile | poem.txt | byte poem | byte poem | yes |
| writeFile | poem.txt  string poem | poem.txt with poem | poem.txt with poem | yes |

**Codes:**

**KeyExchange.java**

/\* the purpose of this program is to generate DH keys and use it to generate a common secret key among two users. the secret key then use as an AES key from encryption and decryption. the file exchange among two users are done through email, usb, etc. this code uses java build in security library to generate DH key and secret key. It also uses the Cipher build in class with AES, ECB and PKCS5PADDING to encrypt and decrypt the text file \*/

import java.io.\*;

import java.math.BigInteger;

import java.nio.charset.Charset;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.security.\*;

import java.security.spec.\*;

import java.security.interfaces.\*;

import javax.crypto.\*;

import javax.crypto.spec.\*;

import javax.crypto.interfaces.\*;

class KeyExchange {

public static void main (String[] args) {

if (args.length == 0) {

printUsage(); // if input is empty, pring usage

}

else { // if input is -gen generate new dh key

if (args[0].equals("--gen") && args.length == 3) { // check size of agrs

// generate a new keypair

KeyPair kp;

try {

kp = generateKeyPair(Integer.parseInt(args[1]));

} catch (NumberFormatException ignore) { //catch exception

try { //generate dh key

DHPublicKey key = (DHPublicKey)KeyFactory.getInstance("DH")

.generatePublic(new X509EncodedKeySpec(readByteFile(args[1])));

kp = generateKeyPair(key.getParams());

} catch(NoSuchAlgorithmException|IOException|InvalidKeySpecException e) {

e.printStackTrace();

return;

}

}

try { //save keypair to filename args[2]

saveKeyPair(kp, args[2]);

} catch (IOException e) {

e.printStackTrace();

}

} else if (args[0].equals("--gen-aes") && args.length == 4) {

// generates the AES key using Alice's Private and Bob's Public or vice //versa

SecretKey key;

PrivateKey privateKey;

PublicKey publicKey;

try { // get public and private key, if public key belong to alice, then private key //belong to bob, vice versa

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

privateKey = kf.generatePrivate(new PKCS8EncodedKeySpec(readByteFile(args[1])));

publicKey = kf.generatePublic(new X509EncodedKeySpec(readByteFile(args[2])));

key = generateSecretKey(privateKey, publicKey); // generate secret key

writeFile(args[3], key.getEncoded()); // write to file

} catch (Exception e) {

e.printStackTrace();

return;

}

} else if (args[0].equals("--encrypt") && args.length == 4) { // encrypt file using aes //key in args

SecretKey key;

try {

key = new SecretKeySpec(readByteFile(args[1]), "AES"); //get secret AES key

byte[] encrypted = encrypt(readByteFile(args[2]), key); // encrypt the text

writeFile(args[3], encrypted); //save it to filename args[3]

} catch(IOException e) {

e.printStackTrace();

return;

}

} else if (args[0].equals("--decrypt") && args.length == 4) { // decrypt the poem using aes key

SecretKey key;

try {

key = new SecretKeySpec(readByteFile(args[1]), "AES"); // get aes key

byte[] decrypted = decrypt(readByteFile(args[2]), key); // decrypt it

writeFile(args[3], decrypted); //write it to a file

} catch(IOException e) {

e.printStackTrace();

return;

}

} else {

printUsage();

}

}

}

/\*\*

\* printUsage

\*

\* Prints the usage for the application

\*/

public static void printUsage () {

System.out.println("Usage: java -jar <jarfile> [--gen <SIZE|PUBLIC> <DEST>|" +

"--gen-aes <PRIVATE> <PUBLIC> <DEST>|" +

"--encrypt <KEY> <SRC> <DEST>|" +

"--decrypt <KEY> <SRC> <DEST>]");

}

/\*\*

\* generateKeyPair

\*

\* Generates a NEW keypair for DH. Don't use this unless no keypair

\* exists.

\*/

public static KeyPair generateKeyPair (Integer size) {

// Genererate parameters for DH

try {

AlgorithmParameterGenerator paramGen

= AlgorithmParameterGenerator.getInstance("DH");

paramGen.init(size);

AlgorithmParameters params = paramGen.generateParameters();

DHParameterSpec dhSpec = (DHParameterSpec)params.getParameterSpec(DHParameterSpec.class);

KeyPairGenerator kpGen = KeyPairGenerator.getInstance("DH");

kpGen.initialize(dhSpec); //genderate keypair

return kpGen.generateKeyPair();

} catch (NoSuchAlgorithmException|InvalidParameterSpecException|InvalidAlgorithmParameterException e) {

e.printStackTrace();

return null;

}

}

/\*\*

\* generateKeyPair

\*

\* Generates a NEW keypair for DH using existing parameters.

\*/

public static KeyPair generateKeyPair (DHParameterSpec params) {

try {

KeyPairGenerator gen = KeyPairGenerator.getInstance("DH");

gen.initialize(params);

return gen.generateKeyPair(); //return dh key

} catch(NoSuchAlgorithmException|InvalidAlgorithmParameterException e) {

e.printStackTrace();

return null;

}

}

/\*\*

\* saveKeyPair

\*

\* Saves the given keypair to a file pair (filename for private, filename.pub for public)

\*/

public static void saveKeyPair (KeyPair kp, String filename)

throws IOException {

writeFile(filename, kp.getPrivate().getEncoded());

writeFile(filename + ".pub", kp.getPublic().getEncoded());

}

/\*\*

\* generateSecretKey

\*

\* Generates a shared SecretKey using a (Private,Public) keypair

\*/

public static SecretKey generateSecretKey (PrivateKey privateKey, PublicKey publicKey) {

try {

KeyAgreement ka = KeyAgreement.getInstance("DiffieHellman");

ka.init(privateKey); //one user private key

ka.doPhase(publicKey, true); // the other user public key

return ka.generateSecret("AES");

} catch(NoSuchAlgorithmException|InvalidKeyException e) {

e.printStackTrace();

return null;

}

}

/\*\*

\* encrypt

\*

\* Encrypts a byte-string of plaintext using the given SecretKey

\*/

public static byte[] encrypt (byte[] plaintext, SecretKey key) {

try {

Cipher c = Cipher.getInstance("AES/ECB/PKCS5Padding"); // using build in Aes,ecb,pad

c.init(Cipher.ENCRYPT\_MODE, key);

return c.doFinal(plaintext);

} catch (NoSuchAlgorithmException|InvalidKeyException|IllegalBlockSizeException|

NoSuchPaddingException|BadPaddingException e) {

e.printStackTrace();

return null;

}

}

/\*\*

\* decrypt

\*

\* Decrypts a byte string using the given SecretKey

\*/

public static byte[] decrypt (byte[] ciphertext, SecretKey key) {

try {

Cipher c = Cipher.getInstance("AES/ECB/PKCS5Padding");

c.init(Cipher.DECRYPT\_MODE, key);

return c.doFinal(ciphertext);

} catch (NoSuchAlgorithmException|InvalidKeyException|IllegalBlockSizeException|

NoSuchPaddingException|BadPaddingException e) {

e.printStackTrace();

return null;

}

}

/\*\*

\* readFile

\*

\* Reads a file and returns its string representation.

\*/

public static String readFile(String path, Charset encoding)

throws IOException {

//byte[] encoded = Files.readAllBytes(Paths.get(path));

byte[] encoded = Files.readAllBytes(Paths.get(path));

return new String(encoded, encoding);

}

/\*\*

\* readByteFile

\*

\* Reads a file and returns its byte[] representation.

\*/

public static byte[] readByteFile (String filename)

throws IOException {

return Files.readAllBytes(Paths.get(filename));

}

/\*\*

\* writeFile

\*

\* Writes the given byte[] to a file

\*/

public static void writeFile(String filename, byte[] contents)

throws IOException {

FileOutputStream fos = new FileOutputStream(new File(filename));

fos.write(contents);

fos.close();

}

}//end of class

**Makefile:**

CLASS\_DIR=build/class/

JARFILE=build/target.jar

SRC\_DIR=src/

MAIN\_CLASS=KeyExchange

.PHONY: src

jar: $(JARFILE)

$(JARFILE): src

jar cvfe $(JARFILE) $(MAIN\_CLASS) -C $(CLASS\_DIR) .

src:

@$(MAKE) -C $(SRC\_DIR)

clean:

rm -rf build