Directions

A. Due Date: 21 June 2025 at 11:55pm

B. The homework may be done in groups of up to two students.

What to turn in

C. Turn in all related source code (.java files) along with any headers or supplemental libraries needed to compile the code.

D. Turn in the keystore files for Alice, Bob, and the Server. Use the keystore names and passwords shown below.

E. Use the gradle.build.kts and settings.gradle files that are included in the template repository. Do not change the root project name or version. Doing so will break the autograding script.

F. Do not change the package names or the names of the classes with the main methods in them. You may add classes or methods as necessary.

G. Do not turn in a compiled JAR. The autograder will build the JAR for you automatically.

H. In addition, turn in a README.md with the following:

- Names and TZ of all students in the group
- Total number of hours spent on each part of the assignment
- Date of submission
- Reflections on the assignment's requirements (at least 100 words). What did you learn from it? What was the hardest part?

How to submit

I. Turn in your submission via the private repositories opened for you on GitHub using GitHub classroom. If you put your code somewhere else or don't use the private repository opened by GitHub Classroom you will not receive a grade!

J. Place the source code for the program in the existing module directories called src.

K. Indicate that your work is complete by performing a single commit with the text "Submitted for grading" on the repository.

- Only write "Submitted for grading" when you are done! I will take the grade from the repository from the first commit with that text.

L. **Do not** send submissions via email. Email submissions will be ignored without consideration of their merits.

Digital Certificates, Digital Signatures, and Using KeyStores

This assignment will cover three topics: Digital signatures, Key store files, and digital X.509 certificates. We'll create a small Public Key Hierarchy (PKI) with three participants: Alice, Bob, and Server. They all will have digital certificates signed by a single certificate authority (CA).

We'll put the certificates and private signing keys in Java PKCS12 KeyStore files that securely store them on disk. We'll then have Alice and Bob send signed and encrypted files to the Server who will check the signatures and decrypt the files. As in the previous assignment, the file encryption will be performed using hybrid encryption (AES/GCM + RSA).

1 PKI

We'll generate a simple PKI with four participants as shown in Figure 1. The participants are as follows:

CARoot The certificate authority that has the root public/private key pair and a CA self signed root certificate. Make the keys and certificate using openssl as shown in the recitation.

Alice A participant who has a private key (AliceKey) and a public key certificate (Alice). The public key certificate is signed by the CA root certificate as shown in the recitation. The X.500 name for Alice shall be as follows:

> CN=Alice, OU=Software Engineering, O=Kinneret College, L=Tzemach, ST=Jordan Valley, C=IL

Alice sends signed and encrypted files to the Server.

Bob A participant who has a private key (BobKey) and a public key certificate (Bob). The public key certificate is signed by the CA root certificate as shown in the recitation. The X.500 name for Bob shall be as follows:

> CN=Bob, OU=Software Engineering, O=Kinneret College, L=Tzemach, ST=Jordan Valley, C=IL

Bob sends signed and encrypted files to the Server.

Server The recipient of the signed and encrypted files from Alice and Bob. The Server has a private key (ServerKey) and a public key certificate (Server). The X.500 name for the server shall be as follows:

> CN=Server, OU=Software Engineering, O=Kinneret College, L=Tzemach, ST=Jordan Valley, C=IL

The Server receives the signed and encrypted files from Alice and Bob. It verifies their signatures and decrypts the files.

2 Key Stores

There are three key stores used in the assignment as shown in Figure 1: Alice's, Bob's, and the Server's. They have the following properties:

2.1 Alice's KeyStore

Alice's key store is called `AliceKeyStore.p12` and has the password `alicepwd` It contains the following entries (aliases are in bold):

Alice The digital certificate for Alice

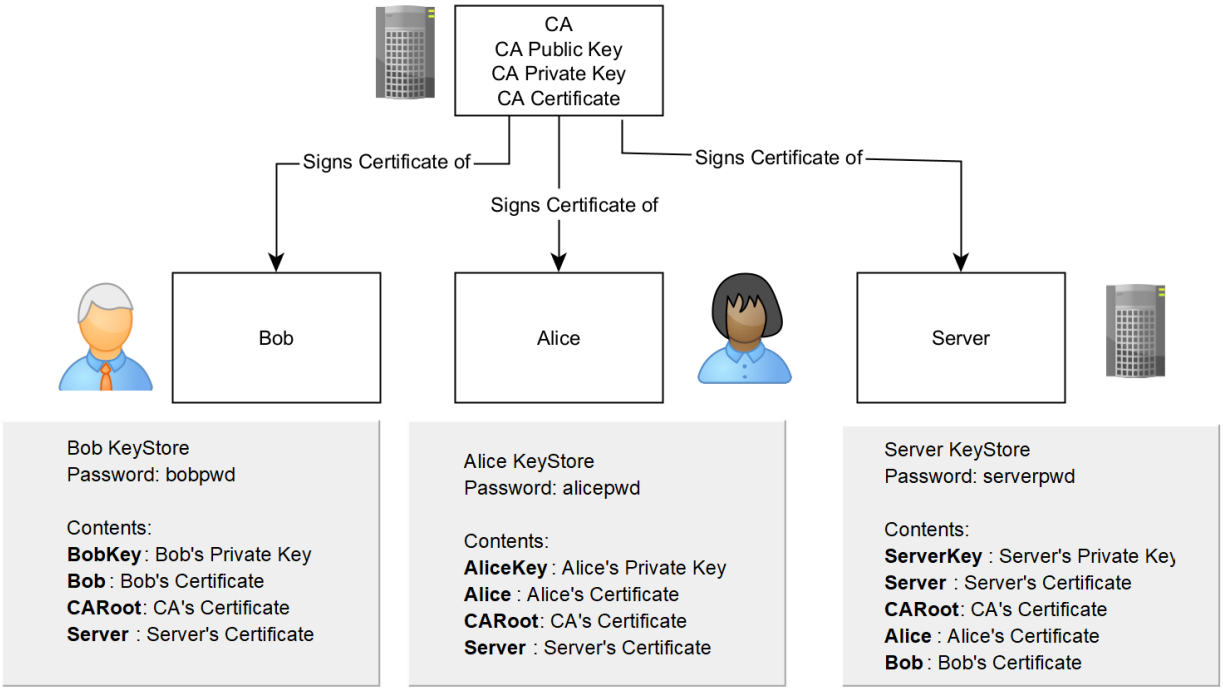

Figure 1: Public Key Infrastructure and KeyStore setup.

AliceKey The private key for Alice

Server The Server's digital certificate

CARoot The digital certificate of the certification authority root

2.2 Bob's KeyStore

Bob's key store is called `BobKeyStore.p12` has the password `bobpwd` It contains the following entries (aliases are in bold):

Bob The digital certificate for Bob

BobKey The private key for Bob

Server The Server's digital certificate

CARoot The digital certificate of the certification authority root

2.3 Server's KeyStore

The Server's key store is called `ServerKeyStore.p12` has the password `serverpwd` It contains the following entries (aliases are in bold):

Alice The digital certificate for Alice

Bob The digital certificate for Bob

Server The digital certificate for the Server

ServerKey The private key for the Server

CARoot The digital certificate of the certification authority root

3 Hybrid Encryption

When Alice or Bob send the Server a file, they first encrypt it using the provided AES key and IV. The clients support only a single encryption suite: AES/GCM/NoPadding. The key must be encrypted for the Server using RSA KeyWrap before it's sent to the Server. For the key wrap, use the Server's public key from its certificate in the KeyStore.

4 Digital Signatures

After Alice or Bob encrypt the file with AES, they create a digital signature on the encrypted file using the correct private key. The signature is then sent to the Server along with the encrypted file and wrapped key. Sign using Alice or Bob's private key as retrieved from the KeyStore.

5 Communication Steps

The Client (Alice or Bob) sends the following data to the Server:

1. The digital signature for the encrypted file
2. The wrapped AES key
3. The IV
4. The encrypted file

The Server checks the signature, unwraps the key, and decrypts the file. The process is similar to the steps used in the previous assignment with the addition of the digital signature generation and validation. The steps are summarized in Figure 2.

6 Server program

The server tool must support the following requirements:

1. The server must be a command line tool written in Java
2. The server must accept all parameters from the command line.
3. The server must support the following required command line parameters:

   ip The IP address to listen on (*e.g.* 8.1.2.3, 127.0.0.1)

   port The port to listen on (*e.g.* 1025, 5000, 5656)

   tempfile Output file to write the encrypted content to (what arrives from the client before decryption)

   outfile The output file to write the decrypted content to. It might be a path or a filename. Example value: outfiles/file1.txt

   keystore The keystore file path. The keystore must be of type PKCS12. Example valid: serverKeystore.p12

   pwd The password for the key store

   privateAlias The alias for the Server's private key in the keystore.

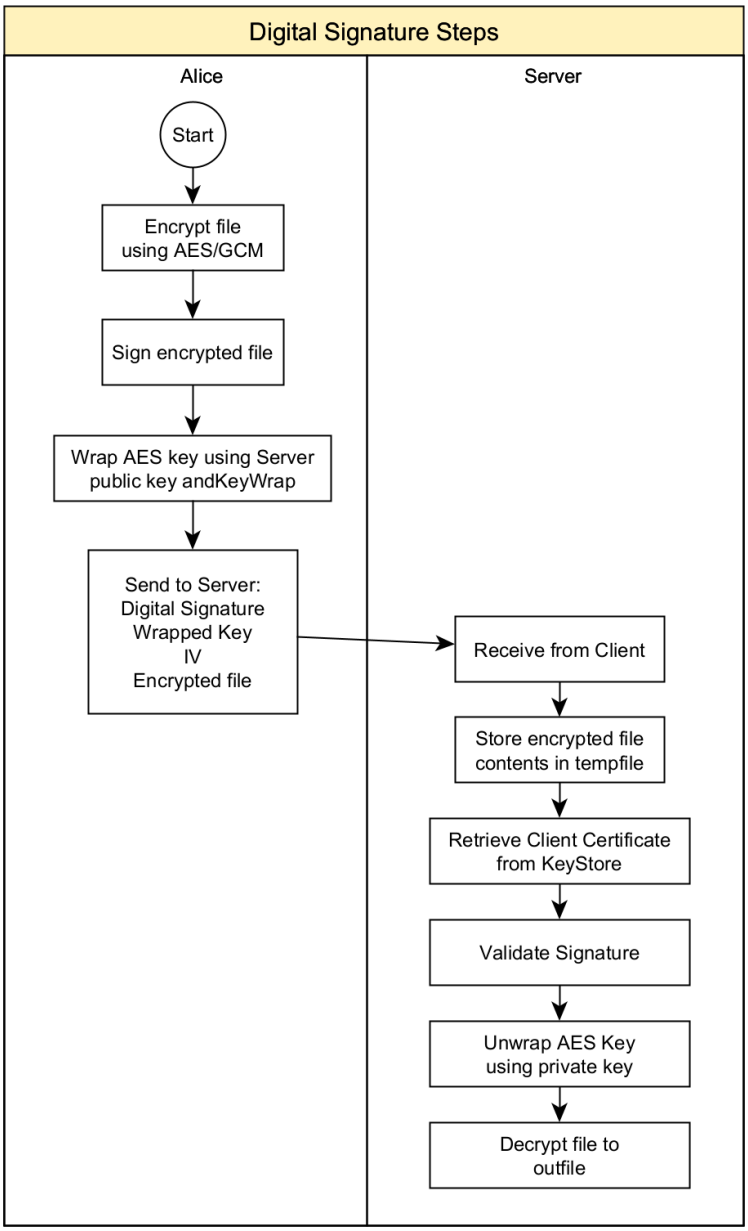

Figure 2: Digital signature and encryption process

publicAlias The alias for the client certificate for the client that will contact the server.

4. The parameters may be provided in any order.
5. If any parameter is missing or incorrect, the tool must quit with an error and show a usage message:

```
Usage:  CertificateSigningServer-5785 -ip=s -port=p -tempfile=t -outfile=f -keystore=k
-pwd=password -privateAlias=privA -publicAlias=pubA
ip and port for server listening
tempfile where encrypted file is stored
outfile where the decrypted file is stored
password is the keystore password
keystore must be of type PKCS12
privateAlias is the alias for the private key in the keystore
publicAlias is the alias for the public key in the keystore
```

6. The tool must listen on the provided port and IP address using TCP.
7. The tool must receive the digital signature, encrypted session key, IV, and encrypted file from the client.
8. The tool must store the encrypted file (without the session key and IV) in the tempfile location provided as a parameter.
9. The tool must validate the digital signature using the certificate with the key store alias given as the publicAlias parameter.
10. The tool must decrypt the session key using RSA KeyUnwrap using cipher suite RSA/ECB/OAEPWithSHA-256AndMGF1Padding.
11. The tool must decrypt the file (in tempfile) using the decrypted session key using AES/GCM/NoPadding.
12. The tool must store the decrypted file contents in the outfile location provided as a parameter.
13. When the tool begins listening it must output the line: "Listening"
14. When the tool receives incoming TCP connection it must output the incoming IP and port. For example: Received connection from /127.0.0.1:48876
15. The server must verify the signature using the SHA256withRSA algorithm.
16. If the signature is valid, the server must output a success message along with the X.500 name of the signer. For example:

```
Signature valid from subject CN=Alice, OU=Software Engineering, O=Kinneret College,
L=Tzemach, ST=Jordan Valley, C=IL
```

17. If the signature fails, the server must output a failure message with the X.500 name of the expected signer. Example:

```
Error:  Signature invalid for subject CN=Bob, OU=Software Engineering, O=Kinneret
College, L=Tzemach, ST=Jordan Valley, C=IL
```

18. If the tool successfully unwraps the key, it must output it in hexadecimal format. For example: `Unwrapped key:  CE178FFD3838B427B688CCF619E337F5`
19. If the tool fails in wrapping the key, it must output an error message: "Error unwrapping key:" followed by the error from the associated Exception object.
20. If the file decryption is success, it must output a success message: Decrypted file successfully
21. If the file decryption fails, the tool must output a failure message: Decryption failed

22. When the tool has completed receiving the file, verifying, unwrapping, and decrypting, it must quit with the message: Stopped listening

7 Client program

The client tool must support the following requirements:

1. The client must be a command line tool written in Java
2. The client must accept all parameters from the command line.
3. The client must support the following required command line parameters:

   dest The IP address to connect to (*e.g.* 8.1.2.3, 127.0.0.1)

   port The port to send to (*e.g.* 1025, 5000, 5656)

   aeskey The session key (K_S) to use. It must be provided in hexadecimal format. Example key value: `6B696C6C696E67796F756775796A6F6E`

   iv The initialization vector to use for the encryption

   infile The file to encrypt and the send to the server. It might be a path or a filename. Example value: `tests/file1.txt`

   keystore The keystore file path. The keystore must be of type PKCS12. Example valid: aliceKeystore.p12

   pwd The password for the key store

   privateAlias The alias for the client's private key in the keystore.

   publicAlias The alias for the server certificate in the keystore.
4. The parameters may be provided in any order.
5. If any parameter is missing or incorrect, the tool must quit with an error and show a usage message:

   ```
   Usage:  CertificateSigningClient-5785 -dest=s -port=p -aeskey=key -iv=iv -publicAlias=pubA
   -privateAlias=privA -infile=f -keystore=k -pwd=pass
   aeskey in hexadecimal format
   iv in hexadecimal format
   keystore must be of type PKCS12
   password is the keystore password
   privateAlias is the alias for the private key in the keystore
   publicAlias is the alias for the public key in the keystore
   ```
6. The tool must encrypt the file using the provided AES session key and IV using AES/GCM/NoPadding.
7. The tool must encrypt the AES session key using KeyWrap and cipher suite RSA/ECB/OAEPWithSHA-256AndMGF1Padding.
8. The tool must wrap the AES key using the public key with the keystore alias provided as a parameter.
9. The tool must sign the encrypted file using the suite SHA256withRSA and the private key with the keystore alias provided as a parameter.
10. The tool must connect to the provided port and IP address using TCP.
11. The tool must send the digital signature, IV, and encrypted session key to the server, followed by the encrypted contents of the `infile`.

12. If the signing is successful, the tool must output the digital signature value in hexadecimal format. Example: `Encrypted file signature: 6D6328D5[...]497`
13. If the key wrapping is successful, the tool must output the wrapped key value in hexadecimal format. Example: `Wrapped key: 80C691E9E7[...]30D`
14. If the file encryption is successful, the tool must output a success message: `Encryption succeeded`
15. If the file encryption fails, the tool must output a failure message: `Encryption failed`.
16. If the encryption is successful and the file is sent to the server successfully, the tool must output a success message: `File sent`
17. If the file is not sent for some reason, the tool must output an error: "`Input or output error:`" followed by the error from the associated Exception object.

8 Code documentation

Add Javadoc documentation to every method and class. The documentation for methods must include:

- A 1-2 sentence summary of the method's purpose
- @param entries for all parameters, including what they are used for
- @return entry with a 1-2 sentence description of the return value
- @throws entries for any exceptions thrown.

The documentation for classes must includes:

- A 2-3 sentence description of the class' purpose
- @author the code's author
- @version a version for the class. Update on every commit to the repository.

9 Submission notes and autograding

Use the assignment starter repository for your code. The repository includes test files in the tests directory and expected output file hashes in the files decrypted-files.sha256 and encrypted-files.sha256. Don't change the input files. The repository contains autograding tests that will cover most of the grading for the assignment.

Some of the tests are designed to produce successful results. Others are designed to produce failure notices. The full grading tests can be found in the file automated-cert-test-script.sh. Those are the tests that the autograder will run. Do not modify the file.

Sample output files for client and server are in 448-Assignment4-Output-Sample-Files.zip to help you. Tool output strings and error messages for client and server can be found in the ClientConstants and ServerConstants classes.

10 Grading notes

- Input and Output tests:
  - Success and Failure message outputs: 10 points
  - KeyStore and certificate contents: 30 points

  - Encrypted and decrypted contents: 40 points
- Javadoc documentation: 10 points
- GitHub usage, Readme, and Reflection: 10 points