האקדמית
כנרת
הנדסה, חברה ורוח

Directions

A. Due Date: 29 May 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. 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.

E. 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.

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

G. 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

H. 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!

I. Place the source code for the program in the directory called src.

J. 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.

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

Hybrid Encryption

Your task for this assignment will be to create a tool that uses RSA and AES in Java to implement a hybrid encryption protocol. The work involves adapting the AES and RSA Java command line encryption tools from recitation and adding two more features - (1) key wrap using AES key wrap mode and (2) implementing a hybrid encryption using RSA and AES.

The basic interactions are shown in Figure 1. The important keys and values mentioned in the figure as follows:

1. Bob's public key - K_{pub_B}. It's loaded from the public/private key file provided as a parameter to the client and server programs.
2. Bob's private key - K_{priv_B}. It's loaded from the public/private key file provided as a parameter to the client and server programs.
3. The file encryption ("session") key - K_S. It's provided as a parameter to the client program.
4. The file encryption initialization vector - IV. It's provided as a parameter to the client program.

The encryption steps are as shown in the figure. First the client encrypts K_S with the server's public key using RSA KeyWrap. You can see a bit more about KeyWrap at:

- `https://docs.oracle.com/en/java/javase/21/docs/api/java.base/javax/crypto/Cipher.html#WRAP_MODE`
- `https://docs.oracle.com/en/java/javase/21/docs/api/java.base/javax/crypto/Cipher.html#UNWRAP_MODE`

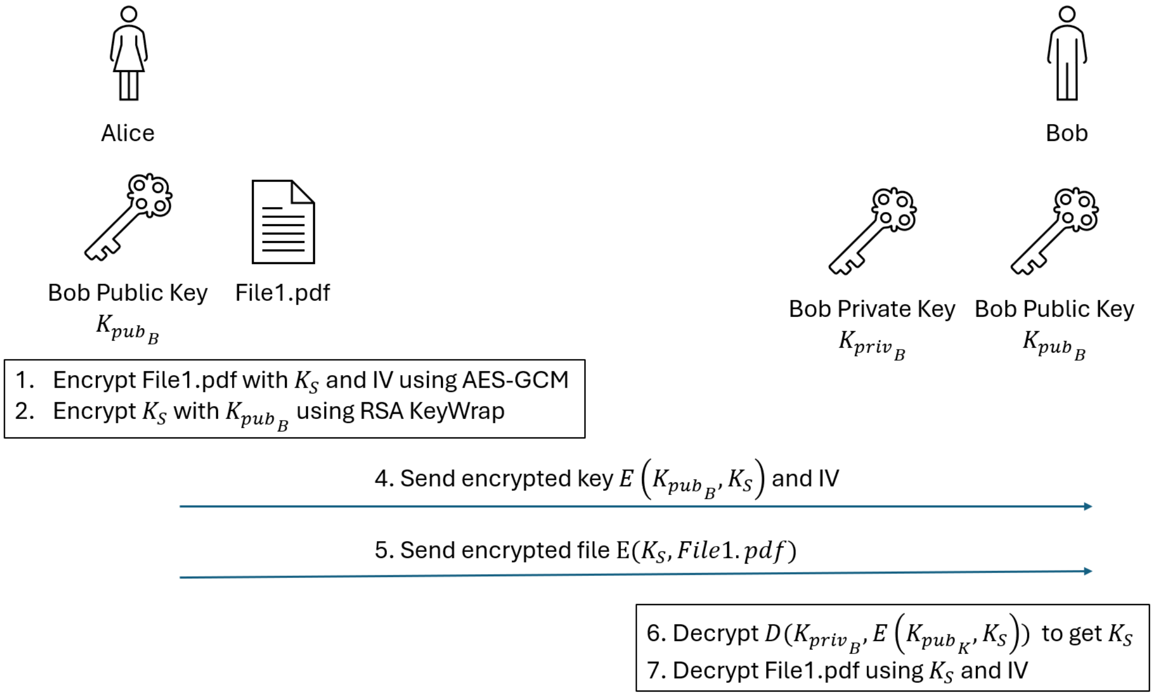
Alice
Bob
Bob Public Key K_{pub_B}
File1.pdf
Bob Private Key K_{priv_B}
Bob Public Key K_{pub_B}
1. Encrypt File1.pdf with K_S and IV using AES-GCM
2. Encrypt K_S with K_{pub_B} using RSA KeyWrap
4. Send encrypted key $E\left(K_{pub_B}, K_S\right)$ and IV
5. Send encrypted file $\mathrm{E}(K_S, File1.pdf)$
6. Decrypt $D(K_{priv_B}, E\left(K_{pub_K}, K_S\right))$ to get K_S
7. Decrypt File1.pdf using K_S and IV

Figure 1: Hybrid encryption steps

There are two tools to create - a server and a client.

1 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

   rsakey File that contains the public and private key pair for 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:  RSAFileServer -ip=s -port=p -tempfile=t -outfile=f -rsakey=rsa
   ip and port for server listening
   tempfile where encrypted file is stored
   outfile where the decrypted file is stored.
   rsakey is path to public key file
   ```
6. The tool must listen on the provided port and IP address using TCP.
7. The tool must receive the 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 decrypt the session key using RSA KeyUnwrap using cipher suite RSA/ECB/OAEPWithSHA-256AndMGF1Padding.
10. The tool must decrypt the file (in tempfile) using the decrypted session key using AES/GCM/NoPadding.
11. The tool must store the decrypted file contents in the outfile location provided as a parameter.
12. When the tool begins listening it must output the line: “Listening”
13. 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
14. If the tool successfully unwraps the key, it must output it in hexadecimal format. For example: Unwrapped key: CE178FFD3838B427B688CCF619E337F5
15. 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.
16. If the file decryption is success, it must output a success message: Decrypted file successfully
17. If the file decryption fails, the tool must output a failure message: Decryption failed
18. When the tool has completed receiving the file, wrapping, and decrypting, it must quit with the message: Stopped listening

2 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`

   rsakey File that contains the public and private key pair for 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:  RSAFileClient -dest=s -port=p -aeskey=key -iv=iv -rsakey=rsa -infile=f
   aeskey in hexadecimal format
   iv in hexadecimal format
   rsakey is path to public key file
   ```
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 using cipher suite RSA/ECB/OAEPWithSHA-256AndMGF1Padding.
8. The tool must connect to the provided port and IP address using TCP.
9. The tool must send the IV and encrypted session key to the server, followed by the encrypted contents of the `infile`.
10. If the key wrapping is successful, the tool must output the wrapped key value in hexadecimal format. Example: `Wrapped key:  80C691E9E7[...]30D`
11. If the file encryption is successful, the tool must output a success message: `Encryption succeeded`
12. If the file encryption fails, the tool must output a failure message: `Encryption failed`.
13. If the encryption is successful and the file is sent to the server successfully, the tool must output a success message: `File sent`
14. 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.

3 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.

4 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 decryptedfiles.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-rsa-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-Assignment3-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.

5 Grading notes

- Input and Output tests:
  - Success and Failure message outputs: 6 points
  - Encrypted and decrypted contents: 74 points
- Javadoc documentation: 10 points
- GitHub usage, Readme, and Reflection: 10 points