Directions

A. Due Date: 8 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.

Combining Hashing and Encryption

Your task for this assignment will be to create a Java encryption and hashing tool. The work involves completing the complete the command line Java hashing and HMAC tool from recitation and adding two more features - (1) encryption using CBC, CTR, and GCM from the previous assignment and (2) sending encrypted files over sockets.

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)

   hashalg The hash algorithm to use for the hashing or MAC creation. You must support the following algorithms - MD5, SHA1, SHA256, SHA512, SHA3-256, SHA3-512, HmacSHA256, HmacSHA512, HmacSHA3-256, HmacSHA3-512

   cipher The cipher suite to use for the encryption. You must support the following combinations: AES/CBC/PKCS5Padding, AES/CTR/NoPadding, AES/GCM/NoPadding

   key The encryption/decryption key to use. It must be provided in hexadecimal format. Example key value: `6B696C6C696E67796F756775796A6F6E`

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

   hmackey If an HMAC algorithm is chosen for the hashalg parameter, a key for it must be provided. It must be provided in hexadecimal format. Example value: `abababab`

   taglength If GCM is chosen as for the cipher mode, a tag length must be provided. It must be provided as a positive integer. Example values: 96, 128.
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:  AES-Hashing-Server-5785 -ip=ip -port=p -hashalg=ha -cipher=c -key=k -outfile=f
   -hmackey=hk -taglength=t
   ```
6. The tool must listen on the provided port and IP address using TCP.
7. The tool must receive the IV and MAC or hash digest from the client.
8. The tool must verify that the hash digest or MAC are correct using the provided algorithms.
9. If the hash digest and MAC are correct, the tool must decrypt the data received and put it in the file path provided (outfile).
10. If the hash digest and MAC are not correct, the tool should not attempt to decrypt the data.
11. If the hash or MAC are correct and the decryption was successful, the tool must print a success message with the output file, the received and calculated digests, and a valid indication. Sample success message:

```
outfiles/Test1.txt is valid received digest 759334242E56CC5D8CC1B1AD4AFD572D computed
digest 759334242E56CC5D8CC1B1AD4AFD572D
Decryption completed on file outfiles/Test1.txt
```

12. If the hash or MAC are incorrect, the tool must print a failure message with the attempted output file, the received and calculated digests, and a failure indication. Sample failure message:

```
outfiles/Test53-AES-GCM-NoPadding-decrypted-mountain-interval.txt is bad received
digest EFFA2B8F9B24CC43468C55CE650BE5200E9E76C9676B116E70D12FC054D8EBA7 computed
digest 7CA9666126B31C8C7A1586D0CAB15E5B19A72DA5B859A7BD691D571AB98E8A26
```

13. If the hash or MAC are correct and the decryption failed, the tool must print a message with the output file, the received and calculated digests, a valid indication for the digests, and a failure message for the decryption. Sample message:

```
outfiles/Test1.txt is valid received digest 759334242E56CC5D8CC1B1AD4AFD572D computed
digest 759334242E56CC5D8CC1B1AD4AFD572D
Decryption failed on file outfiles/Test1.txt
```

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)

   hashalg The hash algorithm to use for the hashing or MAC creation. You must support the following algorithms - MD5, SHA1, SHA256, SHA512, SHA3-256, SHA3-512, HmacSHA256, HmacSHA512, HmacSHA3-256, HmacSHA3-512

   cipher The cipher suite to use for the encryption. You must support the following combinations: AES/CBC/PKCS5Padding, AES/CTR/NoPadding, AES/GCM/NoPadding

   key The encryption/decryption key 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 encrypt, calculate a hash digest or MAC on, and the send to the server. It might be a path or a filename. Example value: tests/file1.txt

   hmackey If an HMAC algorithm is chosen for the hashalg parameter, a key for it must be provided. It must be provided in hexadecimal format. Example value: `abababab`

   taglength If GCM is chosen as for the cipher mode, a tag length must be provided. It must be provided as a positive integer.

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:  AES-Hashing-Client-5785 -dest=ip -port=p -hashalg=ha -cipher=c -key=k
-iv=iv -infile=f -hmackey=hk -taglength=t
```

6. The tool must connect to the provided port and IP address using TCP.
7. The tool must send the IV and MAC or hash digest to the server, followed by the contents of the infile.
8. The tool must use an Encrypt-then-MAC or Encrypt-then-Hash technique.
9. Once the file has been fully processed and sent, the tool must output a success message with the cipher used, the digest or MAC calculated, the file name, and the destination sent to. Sample success message:

```
Encryption completed on file AES/CBC/PKCS5Padding
File encrypted and hashed.  Digest:  759334242E56CC5D8CC1B1AD4AFD572D
tests/mountain-interval.txt sent to /127.0.0.1:5000
```

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 file decryptedfiles.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-hashing-test-script.sh. Those are the tests that the autograder will run. Do not modify the file.

Some versions of Java do not support SHA3. I have configured the autograder and autobuilder scripts to use a version of Java on GitHub that does support SHA3. If you receive "algorithm not supported" errors while testing, please message me.

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