Network Security	Programming Project 3
CSC3055 Spring 2025 (Kissel)	Due Date: Monday March 3, 2025

Code should be stored in a private repository on GitHub under your Merrimack account. Be sure to use the source control system properly (e.g., multiple commits over the course of the project). The commit history will be used to assess individual contributions in a group project. Remember to test your solution thoroughly. Code that does not work correctly will loose credit. ***Code that does not compile will receive 0 points of credit.*** To submit your project please add me as a collaborator, my username is kisselz@merrimack.edu. Good luck!

Overview

> **!** This project is a group project and must be done in groups of size 2 or 3.

A secrets vault is a special piece of software that allows users to store security secrets such as passwords, private keys, API keys, etc. securely. The goal of a secrets vault is to be centralized repository so that secrets are note stored across a system (a problem known as *secrets sprawl*). There are many commercial implementations of secrets vaults. Some such examples include Hashicorp Vault, AWS Secrets Manager, Kapersky Vault, LastPass Vault, and many others. In this project you will implement a secrets vault in Java that will allow the user to do the following:

- Add a new service name, username, and password triple.
- Lookup a user name-password pair for a given website.
- Add a new service name and user name with a *randomly* generated password of a specific length.
- Add a new service name and private key pair.
- Lookup an Elgamal private key given a service name and output as a Base64 encoded string.
- Add a new service name and private key with a freshly generated 512-bit Elgamal key pair (the public key should be output to the screen as a Base64 string).

When the application is opened a the user will be prompted for the vault password. This password will then be used to *unseal* the vault by first decrypting the vault key (using the root key derived from the vault password) and using that vault key to decrypt the vault's contents. Before the application exits, any modifications will be written securely to the vault (a process known as *sealing*).

The Vault

The application will retain state by keeping a JSON file, called the *vault* with file name `vault.json`. The vault *securely* stores all of the secrets. The vault should be loaded automatically at start up and written to disk before exiting. If during start up the file `vault.json` is not found, you should create the file automatically (without user intervention).

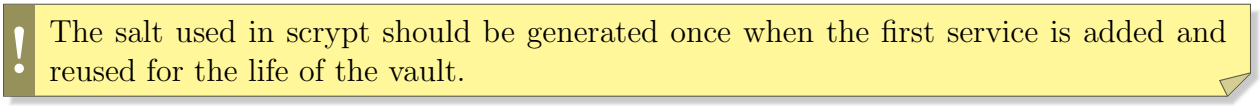
The vault contains:

- A value with key `salt` that represents a Base 64 encoded scrypt salt.
- A value with key `vaultkey` that represents the vaults cipher key as an object. The object consists of the following:
  - A value with key `iv` that represents the IV used for encryption encoded as a Base 64 string.
  - A value with key `key` that represents the key protecting the vault encoded as a Base 64 string.
- A value with key `passwords` that represents an array of username-password account objects. Each password account consists of:
  - A value with key `iv` that represents the IV used for encryption encoded as a Base 64 string.
  - A value with key `service` which represents the URL of the website.
  - A value with key `user` that represents the user name for the website.
  - A value with key `pass` which represents the encrypted password encoded as a Base 64 string.
- A value with key `privkeys` that represents an array of private key based account objects. Each private key account consists of:
  - A value key `iv` that represents the IV used for encryption encoded as a Base 64 string.
  - A value with key `service` which represents the URL of the website.
  - A value with key `privkey` that represents the private key for the website.

Beyond the adding and lookup operations on the vault, the vault must support two implicit operations *seal* and *unseal*. When the application opens, the vault is unsealed. This requires deriving the root key from a password, using the root key to decrypt the vault key, and using an AEAD cipher with the vault key to decrypt the information as it is loaded into the in-memory representation of the vault. Before the application terminates, the seal operation occurs. To seal the vault the vault key is used to encrypt the in-memory representation of the vault using an AEAD cipher, the vault key is then encrypted using the root key. The results are then written to `vault.json`.

Secrets should be encrypted and decrypted using the AEAD cipher mode `AES/GCM/NoPadding`, with a 128-bit tag size. The key used should be the vault key. The service names and user names (if applicable) should not be encrypted but, should be authenticated (read up on the `updateAAD` method of the `Cipher` class). The vault key is considered to be a special type of secret and is protected via the vault root key. The root key is derived from the vault password. When a vault is created a new random vault key is generated. If the vault already exists, the root key is used to decrypt the vault key and then the vault key is used for the remainder of the cryptographic operations.

We will derive the root key from the vault password using scrypt. You should set the cost to 2048, use the salt (IV) you generated previously, a block size of 8, a parallelization of 1. Because scrypt is not an algorithm found in one of the default providers, you will need to use the Bouncy Castle provider (`https://www.bouncycastle.org/java.html`).



Generating Passwords

To generate a random password you should generate a password of the specified length you should use `SecureRandom` to generate specified length number of *integers* that index an array of characters that consist of all upper case letters, all lower case letters, all digits, and the special characters: #, $, %, !, &, *, @, and ? (a total of 70 symbols).

If the requested password length zero or less, the generation should fail and no account should be added. The password length is less than seven, a warning message should be displayed but, the addition of the account should otherwise proceed as normal.

User Interface

Your user interface can either be a command line application or a nicely laid out GUI. When writing a GUI please use `JFileChooser`s when asking for files and use buttons, radio buttons, and check boxes for their intended purposes. If you choose the command line approach please use the command line interface (CLI) classes found in the latest release `merrimackutil` available as a JAR at `https://github.com/kisselz/merrimackutil`. The JavaDocs for this package is available at `http://cs.merrimack.edu/merrimackutil/`.

If you are writing a command line application, when reading a password make sure it is *not* displayed on the screen. Read up on the `Console` class in the `java.io` package to determine how to do this. If you are writing a GUI for your interface, investigate the `JPasswordField` class.

When you are adding new log in information the password should also be obscured using either a `JPasswordField` in the case of a GUI or using `Console` in the case of a CLI. Of

course if you are auto generating a password you simply don't display it to the screen. We do this to prevent "shoulder surfing" attacks.

Helpful Hints

I offer the following helpful hints as you embark on the project:

- I've mentioned it earlier but, **start early** and work as a team.
- For JSON and command line support download the latest release JAR file form `https://github.com/kisselz/merrimackutil`.
- The javadocs for `merrimackutil` are hosted on `https://cs.merrimack.edu/merrimackutil`.
- The javadocs for the JCE available on-line.
- The Java Security Developer's Guide is also useful `https://docs.oracle.com/en/java/javase/22/security/index.html?authuser=0`.
- I will not only be evaluating your use of security practices but, also general software design practice (use of git, general code architecture, etc. Please see the rubric).
- Remember that you should **never** hard code absolute paths into applications.

Testing

Please be sure to thoroughly test every operation in your application. You should test both the components as well as the final working program (integration testing). While it is not required, you may find it helpful to write some JUnit tests for your application. This will allow you to automate testing.

Examples

I chose to write my software as a command line application.

An example of the options screen:

```
$ java -jar dist/secrets.jar
usage:
      secrets --add --service <name> --user <uname>
      secrets --add --service <name> --user <uname> --gen <len>
      secrets --add --service <name> --key <key>
      secrets --add --service <name> --keygen
      secrets --lookup-pass <name>
      secrets --lookup-key <name>
```

```
options:
      -a, --add Adds an account.
      -s, --service Specifies the service name.
      -u, --user Specifies the user.
      -k, --key Specifies the key to store.
      -c, --keygen Adds a service with a generated key.
      -g, --gen Generate a password of given length.
      -p, --lookup-pass Displays the password information for a service.
      -r, --lookup-key Displays the key information for a service.
```

Add a new account with a username and password

```
$ java -jar dist/secrets.jar --add --service home --user kisselz
Vault Password:
Service Password:
```

Query for an account:

```
$ java -jar dist/secrets.jar --lookup-pass home
Vault Password:

User Name: kisselz
Password: password
```

Generate an account with a password that is sufficiently long.

```
$ java -jar dist/secrets.jar --add --service merrimack.edu --user alice --gen 10
Vault Password:
```

Generate an account with a password that is too short.

```
$ java -jar dist/secrets.jar --add --service yahoo.com --user alice --gen 5
Warning a password of less than 7 characters is considered insecure.
Vault Password:
```

Add a service and generate the public-private key pair (output word wrapped).

```
$ java -jar dist/secrets.jar --add --service github --keygen
Vault Password:
Public Key: MIHZMIGQBgYrDgcCAQEwgYUCQQD8poLOjhLKuibvzPcRDlJtsHiwXt7LzR60ogjzrhYXrgHzW5Gk
fm32NBPF4S7QiZvNEyrNUNmRUb3EPuc3WS4XAkBnhHGyepz0TukaScUUfbGpqvJE8FpDTWSGkx0tFCcbnjUDC3H9c9oX
kGmzLik1Yw4cIGI1TQ2iCmxBblC+eUykA0QAAkEA6WzEodanK+SBlYz5uAs7F8PcKzz0vrHW9E95ozA5wwCVgJNm7lZv
QA4ClFggmlyogZNQruHl6oL+KUmdMMEiOQ==
```

Lookup the private key in the secrets vault (output word wrapped).

```
$ java -jar dist/secrets.jar --lookup-key github
Vault Password:
Private Key: MIHZMIGQBgYrDgcCAQEwgYUCQQD8poLOjhLKuibvzPcRDlJtsHiwXt7LzR60ogjzrhYXrgHzW5Gkfm3
2NBPF4S7QiZvNEyrNUNmRUb3EPuc3WS4XAkBnhHGyepz0TukaScUUfbGpqvJE8FpDTWSGkx0tFCcbnjUDC3H9c9oXkGm
zLik1Yw4cIGI1TQ2iCmxBblC+eUykA0QAAkEA6WzEodanK+SBlYz5uAs7F8PcKzz0vrHW9E95ozA5wwCVgJNm7lZvQA4
ClFggmlyogZNQruHl6oL+KUmdMMEiOQ==
```

Add a private key to the secrets vault (input word wrapped).

```
$ java -jar dist/secrets.jar --add --service cs.merrimack.edu --key "MIHZMIGQBgYrDgcCAQEwgYUC
QQD8poLOjhLKuibvzPcRDlJtsHiwXt7LzR60ogjzrhYXrgHzW5Gkfm32NBPF4S7QiZvNEyrNUNmRUb3EPuc3WS4XAkBnh
HGyepz0TukaScUUfbGpqvJE8FpDTWSGkx0tFCcbnjUDC3H9c9oXkGmzLik1Yw4cIGI1TQ2iCmxBblC+eUykA0QAAkEA6W
zEodanK+SBlYz5uAs7F8PcKzz0vrHW9E95ozA5wwCVgJNm7lZvQA4ClFggmlyogZNQruHl6oL+KUmdMMEiOQ=="
Vault Password:
```

Project Management

Group projects require the use of good project management strategies. These skills are important to master and thus are practiced across the CS curriculum.

Source Control

In all projects in this class you will use source control. The source control system the department has chosen is `git`. In particular we will use GitHub. For this project please create a *private* GitHub repository under an account linked to your Merrimack e-mail.

As part of your grade on this project I will be requiring good use of the source control. This means your project should have several commits over the course of time.

Don't develop your entire program outside of source control and then add it to source control before submission!

Every commit should have a well worded commit message that explains what the update to the repository is all about. Please be sure to not put automatically generated files in the repository. This includes `class` files, `jar` files, and `zip` files.

Since this a group project, each member's contributions will be judged based on the commit history. For example, a group member that has one tiny commit will receive a lower grade than other members. To make your submission clear, please have a `README` in your repository that lists all of the members of the group.

Project Board & Issue Tracker

Since this is a group project, I'm requiring you to utilize the group management tools integrated into GitHub. There are two main group management tools:

1. The issue tracker which is utilized for creating issues that are directly tied to code (e.g., adding a feature, fixing a bug).

2. The project board. The project board can be used to track both planning of code (it will let you create issues) as well as other project oriented tasks.

For every project you will be *required* to use the project management. I request that you use project board to manage dates and the project schedule. You should create issues for any code that needs to be done (again can be done using the project board). By way of example adding JSON support to an object would be an issue and testing the project would be a task in the project board. Every

task should have an assigned owner and date of completion. When working with tasks they will be labeled drafts, this is OK. Only make issues out of actual code issues.

The task workflow is as follows:

1. A task is created, an owner and completion date is assigned. The task is placed in the *ready* queue. If the task is a coding task, an issue is created from the task.
2. The owner begins work on the task, the task is placed in the *in progress* queue.
3. When the owner completes the task, it should go in the done queue.

Your first task as a group is to set up the schedule. This shouldn't take more than 30 minutes to an hour of time. The use of project management tools will help me in determining the grade for each group member.

Being a Good Team Member

Part of developing software in a group is learning to be a good team member. Here are some guidelines:

- When you have agreed to complete a task, work towards completing it by the deadline. If you're struggling to complete it by the deadline, communicate with your team and update the task on the project board.
- Do not rush to finish the project as quickly as possible. Everyone codes at their own pace. If you steal work from a team mate before the internal deadline, I will deduct points from you.
- If you are done with your tasks early, feel free to offer to help other members of your team (this is called *swarming* in industry).
- Do not engage in ad-hominem attacks, discuss code issues only in terms of the code and project milestones.
- Do not fight in the repository. This could for example, be completely reverting a team members commit in a hostile way. If you all agree a commit should be reverted, that is a completely different thing.
- If you are confused about a part of the project description, talk to other members of your team. If you are all still unsure *please discuss the issue with me.*

Submitting

To submit this program, please add me as a contributor to your private GitHub repository, my username is (`kisselz@merrimack.edu`). Since this is a group project, there should only be one repository which each group member contributes to. The names of every team member (and their username, if its not their name) should appear in the `README.md` file.

Rubric

Your project will be graded based on the following criteria

- Good use of source control and project management tools (*10 points*).
- Code well architected (*10 points*).
- Password based key derivation used correctly to derive root key (*10 points*).
- Encryption and authentication of passwords and keys uses `AES/GCM/NoPadding` correctly (*15 points*).
- Vault key is used to protect all data except vault key (*10 points*).
- Vault creation does *not* require user intervention (*5 points*).
- Looking up passwords works correctly (*5 points*).
- Looking up keys works correctly (*5 points*)
- Adding a new password works correctly (*5 points*).
- Adding a new account with a random password works correctly (*10 points*).
- Adding a new key works correctly (*5 points*)
- Adding a new account with a random key works correctly (*10 points*).