Project 1: Encrypted File System

- **Due:** 03/09/2025 (Sun) 11:59pm

Introduction

In a traditional file system, files are usually stored on disks unencrypted. When the disks are stolen by someone, the contents of those files can be easily recovered by a malicious adversary.

Encrypted file systems (EFS) have been developed to prevent such leakages. In an EFS, all files in a disk are encrypted. Nobody can decrypt the files without knowing the required secret. Therefore, even if the disk is stolen or an adversary can read files stored on the disk, the contents of the files are kept confidential. Modern operating systems, such as Linux, MacOS, and Windows, support EFS.

In this project, you are asked to implement a simulated version of EFS in Java. More specifically, you will need to implement several library functions, which simulate the functionalities of an EFS.

Getting started

If you have not finished setting up your project environment, please follow the instructions in Project Setup to set it up first.

We expect you will work on this project locally in your own machine because it requires a graphical user interface (GUI). The project server only provides a text-based interface (i.e., SSH), we recommend using it only for a testing purpose in this project.

Although not recommended, a GUI interface can be enabled in the project server if you use SSH with X11 forwarding. But it may not be very responsive.

```
[host] $ ssh -X NetID@csa-chk22b.utdallas.edu
```

To start the project in your machine, install the Java Development Kit (JDK) following these instructions.

The files you will need for this and subsequent projects in this course are distributed using the Git version control system. To learn more about Git, take a look at the Git user's manual, or, if you are already familiar with other version control systems, you may find this CS-oriented overview of Git useful.

You can access the repository via our private GitLab server, and you can start with forking this repository to your own namespace. Make sure to connect to the VPN if you are accessing it from outside the campus.

```
$ git clone ssh://git@s3lab.utdallas.edu:2224/NetID/infosec.git
Cloning into infosec...
$ cd infosec
$ git checkout project1
```

Git allows you to keep track of the changes you make to the code. For example, if you are finished with one of the tasks, and want to checkpoint your progress, you can *commit* your changes by running:

```
$ git commit -am 'my solution for project 1 task X'
Created commit 60d2135: my solution for project 1 task X
 1 files changed, 1 insertions(+), 0 deletions(-)
$
```

This *commit* will store your progress locally. If you would like to store that in our GitLab server, you may do *push* by running:

```
$ git push
Counting objects: 3, done.
Delta compression using up to 24 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 269 bytes | 269.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To ssh://git@s3lab.utdallas.edu:2224/cxk200010/infosec.git
   a56269d..da4c4ea  project1 -> project1
```

You can keep track of your changes by using the `git diff` command. Running git diff will display the changes to your code since your last commit, and `git diff origin/project1` will display the changes relative to the initial code supplied for this project. Here, `origin/project1` is the name of the git branch with the initial code you downloaded from our server for this project.

Hand-in procedure

You will turn in your project by pushing your progress to the repository and tag the final version of the project.

When you are ready to hand in your project code and report, please place the report in a file called `report-project1.pdf` in the top level of your directory before handing in your work. After that, add your report to the Git repository with `git add report-project1.pdf` and `git commit`.

If you have obtained help of any kind while working on this project, make sure to write the names or URLs of your sources in `references-project1.txt` in the top level of your directory, and add it to the repository with `git add references-project1.txt` and `git commit`.

Make sure your code builds successfully in the project server since that is where we will build and grade your code. Run the following command to check if your code builds successfully in the server.

```
$ cd ~/infosec
$ make
$ make test
Build test was successful!
```

After checking your code builds successfully in the project server, add any new files you have created into the Git repository, commit, and push. For example,

```
$ git add src/NewFile.java
$ git commit -am 'project1 complete'
$ git push
```

Tag your commit as `project1-final` and push the tag to the repository to submit your progress. **We only grade the commit with this tag.**

```
$ git tag project1-final
$ git push
$ git push origin --tags

# if you want to change the final tag,
$ git tag -d project1-final # this will delete the local tag
Deleted tag 'project1-final' (was 75411c7)
$ git push origin :refs/tags/project1-final # this will delete the remote tag
To ssh://s3lab.utdallas.edu:2224/cxk200010/infosec.git
- [deleted]         project1-final

$ git tag project1-final
$ git push
$ git push origin --tags
```

Project Requirements

Functionality requirements

To simulate a file system, files are stored in blocks of fixed size. More specifically, a file will be stored in a directory which has the same name as the file. The file will be split into trunks and stored into different physical files. That is, a file named `/abc.txt` is stored in a directory `/abc.txt/`, which includes one or more physical files: `/abc.txt/0`, `/abc.text/1`, and so on. Each physical file is of size exactly 1024 bytes, to simulate a disk block. You need to implement the following functions:

- `void create(String file_name, String user_name, String password)`: Create a file that can be opened by someone who knows both user name and password. Both user name and password are ASCII strings of at most 128 bytes.
- `String findUser(String file_name)`: Return the user name associated with a file.
- `int length(String file_name, String password)`: Return the length of a file, provided that the given password matches the one specified in creation. If it does not match, your code should throw an exception.
- `byte[] read(String file_name, int starting_position, int length, String password)`: Return the content of a file for a specified segment, provided that the given password matches. If it does not match, your code should throw an exception.
- `void write(String file_name, int starting_position, byte[] content, String password)`: Write new content into a file at a specified position, provided that the given password matches. If it does not match, the file should not be changed and your code should throw an exception.
- `void cut(String file_name, int length, String password)`: Cut a file to be the specified length, provided that the password matches. If it does not match, no change should occur and your code should throw an exception.
- `boolean check_integrity(String file_name, String password)`: Check that a file has not been modified outside the EFS interface. If someone has modified the file content or meta-data

without using the write call, return False; otherwise, return True, provided that the given password matches. If it does not match, your code should throw an exception.

Security requirements

We have the following security requirements.

- **Meta-data storage:** You will need to store some meta-data for each file. In a file system, this is naturally stored in the i-node data structure. In this project, we require that meta-data are stored as part of the physical files. You will need to decide where to put such data (e.g. at the beginning of the physical files), and also what cryptographic operations need to be performed to the meta-data. Naturally the first physical file would contain some meta-data, however, you can also store meta-data in other physical files.
- **User authentication:** You need to ensure that if the password does not match, reading and writing will not be allowed. You thus need to store something that is derived from the password; however, you should make it as difficult as possible for an adversary who attempts to recover the password from the stored information (perhaps using a dictionary attack).
- **Encryption keys:** In an EFS, we can choose to use one single key to encrypt all the files in the file system; or we can choose to encrypt each file using a different key. In this project, we choose the latter approach, in order to reduce the amount of data encrypted under one key.
- **Encryption algorithm and modes:** You are required to use AES with 128-bit block size and 128-bit key size. The code for encrypting/decrypting one block (i.e. 128 bits) is provided. When you encrypt/decrypt data that are more than one blocks, you are required to use CTR, the Counter mode. You will need to decide how to generate the IV's (initial vectors). For encryption, you can treat a file as a message, treat a chunk (stored in one physical file) as a message, or choose some other design.
- **Adversarial model:** We assume that an adversary may read the content of files stored on the disk from time to time. In particular, a file may be written multiple times, and the adversary may observed the content on disk between modifications. Your design and implementation should be secure against such an adversary.
- **File length:** You design should also hide the length of the file as much as possible. The number of physical files used for a file will leak some information about the file length; however, your design should not leak any additional information. That is, if files of length 1,700 bytes and 1,800 bytes both need 2 physical files, then an adversary should not be able to tell which is the case.
- **Message authentication:** We want to detect unauthorized modification to the encrypted files. In particular, if the adversaries modify the file by directly accessing the disk containing the EFS, we want to detect such modifications. Modification to other meta-data such as the user or file length should also be detected. Message Authentication Code (MAC) can help. You need to decide what specific algorithm to use, and how to combine encryption and MAC.

Efficiency requirements

We also have the following two efficiency requirements.

- **Storage:** We want to minimize the number of physical files used for each file.
- **Speed:** We want minimize the number of physical files accessed for each read or write operation. That is, if an write operation changes only one byte in the file, we want to access as small a number of physical files as possible, even if the file is very long.

These two efficiency goals may be mutually conflicting. You need to choose a design that offers a balanced tradeoff.

EFS Implementation (50 pts)

You are asked to complete `EFS.java` in the `src` directory. Specifically, you need to implement the functions as described in Functionality requirements. **Please do not modify the other source files.** If there is indeed a need to modify any of them, please describe it in detail in your report.

We provide a text editor that you can use to verify your EFS design and implementation. To build your code and use the editor, run the following commands in your machine with `make`:

```
$ make run
```

If your machine does not have `make`, you can build and run your code using JDK directly:

```
$ cd src
$ javac EFS.java Sample.java Editor.java Config.java PasswordIncorrectException.java Utility.java
$ java Editor
```

In your implementation, if the password is incorrect, throw `PasswordIncorrectException`. For other exceptions/errors, you can either throw an exception (other than `PasswordIncorrectException`), or handle it yourself.

You are not allowed to use encryption/decryption/hash/MAC functions in Java library. However, we provide utility functions for you to use in `Utility.java`. Since class `EFS` is derived from class `Utility`, you can directly invoke them. Using these functions is encouraged but not required. The functions are listed as follows:

- `void set_username_password()`: You can set/reset the user name and password by invoking this

function.

- `byte[] read_from_file(File file)` : This function will return all the content of a given file as a byte array. It will throw an exception if the file cannot be read.
- `void save_to_file(byte[] s, File file)` : This function will write a given buffer `s` to a given file (overwrite). It will throw an exception if the file cannot be written.
- `File set_dir()` : It will allow you to select a directory. The return value is the chosen directory. You can choose a new directory by typing the full path. If nothing is chosen, it will return `null` .
- `byte[] encrypt_AES(byte[] plainText, byte[] key)` : This function will return AES encryption result of a given plaintext using a given key.
- `byte[] decrypt_AES(byte[] cipherText, byte[] key)` : This function will return AES decryption result of a given ciphertext using a given key.
- `byte[] hash SHA256(byte[] message)` : This function will return SHA256 result of a given message.
- `byte[] hash SHA384(byte[] message)` : This function will return SHA384 result of a given message.
- `byte[] hash SHA512(byte[] message)` : This function will return SHA512 result of a given message.
- `byte[] secureRandomNumber(int randomNumberLength)` : This function will return a random number vector of a given length in bytes. You can assume this is a secure random number generator.

Project Report (50 pts)

In your report, please include the following. You can edit `report-project1.tex` that we provide and generate `report-project1.pdf` using LaTeX. Feel free to use a word processing program (e.g., MS Word) if you are more comfortable with that. Make sure only the PDF file is submitted with the file name `report-project1.pdf` (but not the docx file).

Design explanation (20 pts)

- **Meta-data design:** Describe the precise meta-data structure you store and where. For example, you should describe in which physical file and which location (e.g., in `/abc.txt/0` , bytes from 0 to 127 stores the user name, bytes from 128 to ... stores ...).
- **User authentication:** Describe how you store password-related information and conduct user authentication.
- **Encryption design:** Describe how files are encrypted, how files are divided up, and how encryption is performed. Explain why your design ensures security even though the adversary can read each stored version on the disk.
- **File length hiding:** Describe how your design hides the file length to the degree that is feasible without increasing the number of physical files needed.
- **Message authentication:** Describe how you implement message authentication, and in

particular, how this interacts with encryption.
- **Efficiency:** Analyze the storage and speed efficiency of your design. Describe a design that offers maximum storage efficiency. Describe a design that offers maximum speed efficiency. Explain why you chose your particular design.

Pseudo code (10 pts)

Provide pseudo-code for the functions `create`, `length`, `read`, `write`, `check_integrity`, and `cut`. From your description, any crypto detail should be clear. Basically, it should be possible to check whether your implementation is correct without referring to your source code. You may want to describe how password is checked separately since it is used by several of the functions.

Design variations (20 pts)

Answer the following questions:

1. Suppose that the only write operation that could occur is to append at the end of the file. How would you change your design to achieve the best efficiency (storage and speed) without affecting security?
2. Suppose that we are concerned only about adversaries who steal the disks. That is, the adversary can read only one version of the same file. How would you change your design to achieve the best efficiency?
3. Can you use the CBC mode in the EFS? Describe how your design would change if the CBD mode is used and analyze the security and efficiency of the resulting design.
4. Can you use the ECB mode in the EFS? Describe how your design would change if the ECB mode is applied and analyze the security and efficiency of the resulting design.

Sample Program

We provide a sample program named `Sample.java` in the `src` directory. You are encouraged to start this project by reading this source file.

Please notice that in this sample, files are not encrypted. Also, it does not implement any optimization to improve the EFS efficiency. You will need to handle the efficiency requirements yourself.

Feel free to use your own program to verify your implementation if you want.

The detailed steps in each function are described as follows:

- `void create(String file_name, String user_name, String password)`:

1. Create a directory named as the name of the file you want to create (e.g. `/abc.txt/`).
2. Create the first block and use the whole block as meta-data (`/abc.txt/0`).
3. Write `0` into the first block. It means the file length is 0.
4. Write user name into the first block.

- `String findUser(String file_name)`: Find the user name from the first block (`/abc.txt/0`).
- `int length(String file_name, String password)`: Find the length of file from the first block (`/abc.txt/0`).
- `byte[] read(String file_name, int starting_position, int length, String password)`:
  1. Compute the block that contains the start position. Assume it is n_1.
  2. Compute the block that contains the end position. Assume it is n_2.
  3. Sequentially read in the blocks from n_1 to n_2 (`/abc.txt/` n_1, `/abc.txt/` $(n_1 + 1)$, ... , `/abc.txt/` n_2).
  4. Get the desired string.
- `void write(String file_name, int starting_position, byte[] content, String password)`:
  1. Compute the block that contains the start position. Assume it is n_1.
  2. Compute the block that contains the end position. Assume it is n_2.
  3. Sequentially write into the blocks from n_1 to n_2 (`/abc.txt/` n_1, `/abc.txt/` $(n_1 + 1)$, ... , `/abc.txt/` n_2).
  4. Update the first block (meta data) for the length of file if needed.
- `void cut(String file_name, int length, String password)`:
  1. Find the number of block needed. Assume it is n.
  2. Envoke `write` to update block n (`/abc.txt/` $(n + 1)$).
  3. Remove the redundant blocks if necessary.
  4. Update the first block (meta data) for the length of file.
- `boolean check_integrity(String file_name, String password)`: N/A.