

CSCI 531 Spring 2022 Semester Project

Designing a Secure Decentralized Audit System

Prasoon Gautam

For the project, option #1 was selected. I implemented a working prototype for the following goals: *privacy*, *query*, and *immutability*. The *privacy* goals also employ *authentication/identification* to identify registered users and grant access accordingly to access audit log records generated for the user. As for the exploration part, I plan to discuss Homomorphic Encryption while touching Fully Homomorphic Encryption and Merkle trees implementation.

Introduction

As mentioned in the project, Electronic Health Record (EHR) systems have gradually replaced traditional paper-based health record systems. As the data has moved to digital platform, it brings with it a myriad of security and privacy risks. The generated audit logs serve multiple functional and regulatory purposes in EHR systems. They add a layer of security by acting as a proof, storing important chronological data, and maintaining records of information exchange or operations that may have occurred over time. It therefore acts a means to back-track the data that was may have been worked upon during the transactions or operations. Due to this high importance of the log file, it should be confidential and maintained with high privacy.

For this project as the part of the option, I implemented the following goals:

- **Privacy:** As the data that we are handling is highly confidential, so to avoid any violations, we must make sure that unauthorized personnel are not able to access the any sensitive data.
- **Authorization & Identification:** Handling private data needs to be accompanied with access control. All users that are registered with the system must provide identification that permits them to access the system. For the sake of brevity and as mentioned in the project guidelines, access has been limited to *5 patients* and *2 audit* companies.
- **Queries:** Auditing a system requires a mechanism to access the secured data by the authorized personnel. Users are distributed into *Patients & Auditors* with their respective authentication mechanism and authorized query mechanism to access the record.
- **Immutability:** Whenever a transaction happens, a secure system leaves a trail of the operations. This mechanism executes the task by detecting if a user deleted or altered any existing log data. The transgression is logged into a file.

The log plays a crucial role in the audit process for the HER therefore needs to be protected. The EHR per se is not secured but the log file is secured. The system takes in the log and processes it to process the request. We assume that the audit log is provided in a .csv format ('audit_log.csv') and has the following columns:

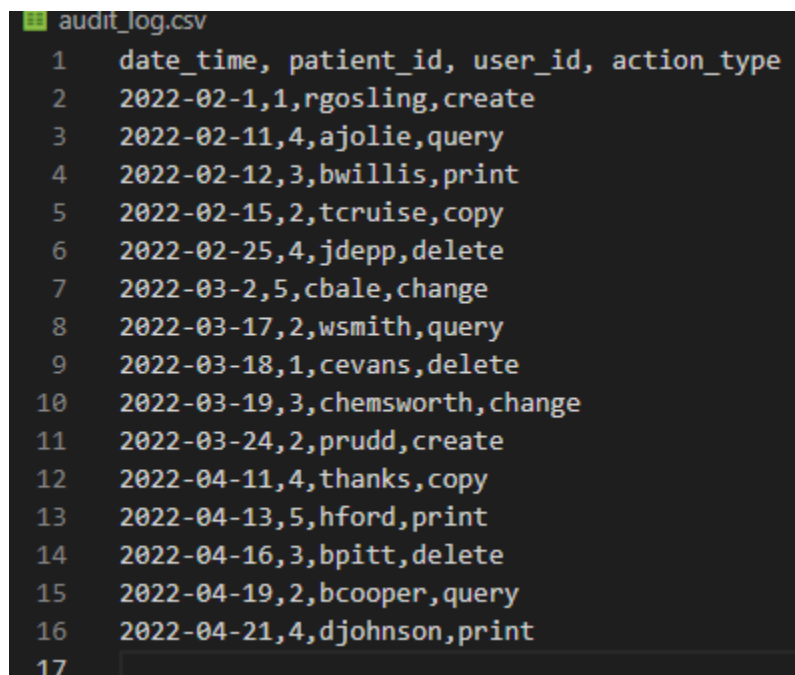
date_time: the date the data was accessed

patient_id: patient's unique ID for who the data was accessed (ID between 1-5)

user_id: username of the user who accessed the data (first name letter and last name)

action_type: action taken by the user on the data field

Following is screenshot of the audit log with 15 entries in it



```
audit_log.csv
1  date_time, patient_id, user_id, action_type
2  2022-02-1,1,rgosling,create
3  2022-02-11,4,ajolie,query
4  2022-02-12,3,bwillis,print
5  2022-02-15,2,tcruise,copy
6  2022-02-25,4,jdepp,delete
7  2022-03-2,5,cbale,change
8  2022-03-17,2,wsmith,query
9  2022-03-18,1,cevans,delete
10 2022-03-19,3,chemsworth,change
11 2022-03-24,2,prudd,create
12 2022-04-11,4,thanks,copy
13 2022-04-13,5,hford,print
14 2022-04-16,3,bpitt,delete
15 2022-04-19,2,bcooper,query
16 2022-04-21,4,djohnson,print
17
```

System Architecture

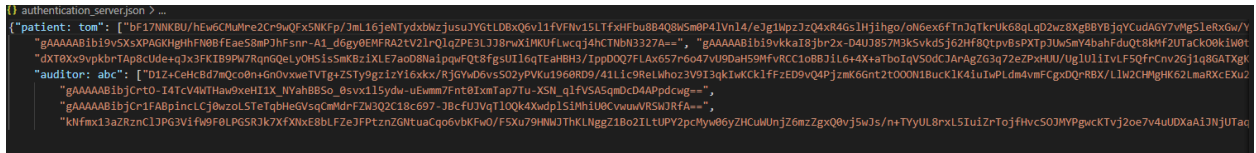
The system is made up of four files working in unison to provide several functionalities. Following are the files that exist:

- **main.py:** This file is executed when the audit system is accessed. The program with the following command: **python main.py** (Please use Python version 3.9.12 or later). The program starts with a message asking user to select an option out of the following: register a

user, query audit records, and immutability verification. The script generates a hash of the audit file through the function called from `immutability.py`. The script also encrypts the file using **Fernet encryption** (<https://cryptography.io/en/latest/fernet/>) as a library and creates a file `audit_log.csv.enc`. Fernet encryption guarantees that a message encrypted using it cannot be manipulated or read without the key. It accomplishes this goal generating a symmetric (secret) key, using AES in CBC mode and PKCS7 padding to encrypt the message, using a secure random IV to make the encryption more secure, timestamping the encryption, and finally signing the message using HMAC and SHA256 to detect any attempts to change. The encryption key is then encrypted using **RSA (PKCS1 standard with OAEP padding)** using the available libraries (RSA: https://pycryptodome.readthedocs.io/en/latest/src/public_key/rsa.html, PKCS1 OAEP: <https://pycryptodome.readthedocs.io/en/latest/src/cipher/oaep.html>). The key pair along with the encrypted symmetric key are stored in the system assuming it serves as a server stub(database where the keys are secured). After the encryption is completed, the script deletes the original audit log that came in clear and deals just with the encrypted version. The script then takes in the user input and calls other script accordingly.

- **user_authentication.py**: This file manages the authentication process in the system and is called upon various times whenever there's a need for it to process other functionalities. It has methods for registering a user, authenticating patient and auditor users. It also stores the authentication data in the *server* (`authentication_server.json` server stub). The file encrypts data using Fernet encryption and RSA encryption. Again, PKCS1 standard is used for RSA with OAEP padding. The padding improves the security of the data
 - **register_user()**: This function asks the user to provide a User ID (Username) and specify if they are a patient user or an auditor. The method then checks if the `authentication_server.json` (server stub) is present or not and loads the data. The data entered by the user is checked against the data present in the server file and sends out an error message if the User ID is already present. The method calls for another method `type ID` which releases a unique ID from the server (server stub called as `p_id.json` and `a_id.json`) according to the user type provided. A 256-bit secret key is generated and then used to encrypt the User ID and ID released from the server stub using the method `Fernetenc()`. The method also generates an RSA keypair and uses it to encrypt the secret key and create a digital signature of the combination of user ID and ID. The encrypted key, encrypted ID, encrypted user ID and digital signature are saved in the *server*(`authentication_server.json`). The method then shares the RSA keypair file and the ID with the user.

- **authentication_server.json:** The encrypted data for the user registration is saved in this file which acts as a server stub for the authentication and identification. The data is stored in a dictionary format with user type and user ID (username) as key and encrypted secret key, encrypted ID, encrypted user ID and digital signature as value. Following is a



```

authentication_server.json >
{"patient": "tom": ["bf17NKKBU/hEw6CAwre2Cr9wQfXSNKFr/3mL16jeNtydxbizJusu7YgtLD6xQ6v11fVfW15LTFxHfBu8BAQ8hSm0P41Vn14/e3g1hpz1z04xR46s1Hj1hgo/oh6ex6fTnJqTkrUk68qLqD2wz8Xg8BYjgYCuDaGV7wHg51eRxEw/y",
  "gAAAAABib19vSXsXPAGKghHfN0BFaES8mP3hFann-A1_d6gy0EMFRA2LV21rQ1q2PE3L3J8rnx1MKUfLucej4hCTNBK327A==", "gAAAAABib19vXka18jbr2x-04UJ857N3ksvk45J62Hf8Qtpv8sPXTp3UwSey4baHfduQe8kHf2Uack00k1x0t",
  "dMT0X-qypkbrTAp8cUde+q3x3FKIB9Pw7RqgQelyOHS15smK8z1XLE7aoD8Na1pwFQctfgeU116qTeaHB3/1pp0Q7FLA657r6o47vU9DAH59WfVRCC1e8B31L6+4x+atBoIqV50dCJAraGZG3q72eZP4HUU/Ug1U11vLF5qfCnu26J1qR8ATXgt",
  "auditor": "abc": ["D1Z+CeHcEd7mQc0n+GnDvzeTVTg7Z5Y9g1zy16akx/RjGvYd6v5S02ypVKu1960R9/41L1c98eLmho3V913qk1wKck1FFzED9vQ4PjzmK66nt2t000H1Buck1K41u1wPLdm4vmfCgsQDqRbX/L1w2CHghK62LmaRXcExu2",
    "gAAAAABibjCrt0-I4TcV4MTHaw9xeHI1X_NYahBBS0_8svx115ydw-uEumm7fnt8IxmTap7Tu-XSN_q1FV5A5qmDcD4APdcug==",
    "gAAAAABibjCrt1FABpInclCj8wz0L5Te1qBHeGvGqCmWdrF2N3Q2C18c697-JBcFUJvqT100k4Xwdp153h1u8Cvuuu4VRSWJRfa==",
    "kNfmX13aZr2n1JPG3V1Fw9F0LPGSR3K7XfX0XkE8bLFzeJFPtzn2GNtuaCqo6vbkFw0/F5Xu79WmJThKLNggZ1Bo21LTUPY2pcMw86yZHCuUJnJ76mzZgxQ0vj5w3s/n+TyYUL8rxL5UiZrTofjHvc503JWPgucKvtJ2oe7v4uUDXaA13NjUTac"]
  }

```

screenshot of the authentication_server.json

- **p_id.json & a_id.json:** The json p_id and a_id files acting as server stub files store the ID generated for the patient and auditor users respectively. The files currently have numbers 1-5 (p_id) and 1,2 (a_id) in a list format. The register_user method calls method type_id to release an ID to use. That ID is then removed from the server files and in view of the scope of the project, can't be used again (no user deletion functionality added).
- **Fernetenc(key,raw) & Fernetdec(key,raw):** The methods encrypt, and decrypt supplied input respectively with the using the provided secret key and return the result.
- **authenticate_patient(RSAfile, id) & authenticate_auditor(RSAfile, id):** The methods authenticate the user using the supplied id and RSA keypair after confirming the user type. These methods are used to process query and immutability requests. The method first checks the availability of authentication_server.json and returns with the message that no user is present if the file is absent otherwise the data from the server file is copied. The user's presence in the file is checked and returns if its not true. The method then decrypts the user's secret key using the RSA keypair provided. The user ID and ID fields for the user are decrypted and then checked against the supplied by the user to authenticate the user. If there's a mismatch, an error is thrown.
- **query.py:** This file handles the query requests by the patients and auditors. Execution starts when the methods from the file are called in option 3 in the menu provided by main.py. Based on user input, patient_query(id) or auditor_query(id) is initiated. The methods then call other methods to decrypt the audit log and then helper methods to process the queries.
 - **patient_query(id) & auditor_query(id):** These methods are responsible to call other methods to help with the query requests. After the request process is completed, the temporary file created from the audit file is deleted.
 - **decrypt_audit_log():** This method opens the encrypted the encrypted audit_log.csv.enc, copies it contents, decrypts the contents using RSA keypair and decrypted secret key,

and then saving the contents in a temporary .csv file called temp_audit.csv. Decryption is again handled by Fernet symmetric encryption and RSA keypair.

- **query_helper_patient(id) & query_helper_auditor(id):** The methods process the query requests and generate a .csv file with the user requested data. The methods take in the ID as the input. query_helper_auditor(id) also prompts the auditor user to enter patient ID to access and save it in a list. The methods then open the temp_audit.csv and parse the data using the csv in-built library (<https://docs.python.org/3/library/csv.html>) and filter the data according to the ID if a patient user otherwise a list of patient IDs if an auditor user. The filtered data is then saved in a newly created .csv file named titled as “patient/auditor_[ID]_query_[date_time].csv”. Date time is generated using in-library functions (<https://docs.python.org/3/library/datetime.html>). Following are screenshots for patient and auditor queries

```
patient_1_query_2022-05-01 00:02:34.587216.csv
date_time, patient_id, user_id, action_type
2022-02-1,1,rgosling,create
2022-03-18,1,cevans,delete
```

```
auditor_1_query_2022-05-01 00:03:24.318225.csv
date_time, patient_id, user_id, action_type
2022-02-1,1,rgosling,create
2022-02-12,3,bwillis,print
2022-02-15,2,tcruise,copy
2022-03-17,2,wsmith,query
2022-03-18,1,cevans,delete
2022-03-19,3,chemsworth,change
2022-03-24,2,prudd,create
2022-04-16,3,bpitt,delete
2022-04-19,2,bcooper,query
```

- **immutability.py:** This file performs an immutability test on the file to check the integrity of the file. The mechanism works by comparing the generated hash of the files and returns a tamper alert if there's a tamper attempt along with logging the attempt. The user is authenticated using user_authentication.py before they move ahead in the process.
 - **hashFunction(filename):** This script generates a hash of the file supplied, basically the audit log. It uses SHA-256 to generate the hash of the file and returns the hash value. (<https://www.quickprogrammingtips.com/python/how-to-calculate-sha256-hash-of-a-file-in-python.html>)
 - **check_integrity.py(id, u_id):** The script introduces the user to the scenario. It then calls decrypt_audit_log() which creates a temporary audit log. We then parse the data using csv in-built libraries to a list. The file with audit log is opened and the hash of the

supplied audit log is recovered. The hash is matched with the temporary file. If they don't match, we report and return. If they match, we move further in the process and the log is printed for the user. The method then prompts user to select the line they want to access. After the edit, the line is returned in the temporary audit file. The hash of the temporary file is generated again and compared against the original hash of the supplied audit log. If it does not match, the tamper attempt is detected and logged in the .txt file "Tamper_Attempt_User_[user ID]_ID_[ID]_[date_time].trs". After the method finishes executing, the temporary audit log is deleted.

```
Tamper_Attempt_User_tom_ID_1_2022-05-01 00:14:45.471453.trs
Tamper attempt on 2022-05-01 00:14:45.471453 by User: tom with ID: 1 in the audit log.
They tried to change line 2 with entry 2022-02-11,4,ajolie,query to HAHAHAAHA.
```

- **report(u_id, id, now, line, og, new):** This method returns a string for the tamper file mentioning the User and their ID who tried to tamper with the data and the data line they attempted to tamper.

System In-use screenshots

```
C:\Users\pgaut\Desktop\Spring 22\Applied Crypto\Semester Project\Project\trial run>python main.py

*****
Welcome to the Secure Decentralized Audit System
Please enter an option out of the following:
1. Register a User
2. Query audit records
3. Immutability verification
4. Exit System
Choice:
```

System starts

```
*****
Welcome to the Secure Decentralized Audit System
Please enter an option out of the following:
1. Register a User
2. Query audit records
3. Immutability verification
4. Exit System
Choice: 1
Please enter the name to be registered as User_ID: Tom
Please specify if you are a patient or an auditor: patient
The patient user has been registered.
Your patient ID is: 1
The assigned RSA keypair has been stored in: Tom.pem

*****
Welcome to the Secure Decentralized Audit System
Please enter an option out of the following:
1. Register a User
2. Query audit records
3. Immutability verification
4. Exit System
Choice: 1
Please enter the name to be registered as User_ID: EY
Please specify if you are a patient or an auditor: auditor
The auditor user has been registered.
Your patient ID is: 1
The assigned RSA keypair has been stored in: EY.pem
```

Register as user and auditor

```
*****
Welcome to the Secure Decentralized Audit System
Please enter an option out of the following:
1. Register a User
2. Query audit records
3. Immutability verification
4. Exit System
Choice: 2
Please enter the user ('patient' or 'auditor') you are attempting to query as: patient
Please provide your RSA key file: Tom.pem
Please enter your patient ID: 1
You have been successfully authenticated as a registered user.
Created file: patient_1_query_2022-05-01 22;19;12.448667.csv

*****
Welcome to the Secure Decentralized Audit System
Please enter an option out of the following:
1. Register a User
2. Query audit records
3. Immutability verification
4. Exit System
Choice: 2
Please enter the user ('patient' or 'auditor') you are attempting to query as: auditor
Please provide your RSA key file: EY.pem
Please enter your auditor ID: 1
You have been successfully authenticated as a registered user.
Please enter the patient ID/s you want to access (separated with a space): 1 2 3
Created file: auditor1_query_2022-05-01 22;19;48.181952.csv
```

Requesting query by user and auditor

```
*****
Welcome to the Secure Decentralized Audit System
Please enter an option out of the following:
1. Register a User
2. Query audit records
3. Immutability verification
4. Exit System
Choice: 2
Please enter the user ('patient' or 'auditor') you are attempting to query as: patient
Please provide your RSA key file: Tom.pem
Please enter your patient ID: 2
Digital signature mismatch. Access denied

*****
Welcome to the Secure Decentralized Audit System
Please enter an option out of the following:
1. Register a User
2. Query audit records
3. Immutability verification
4. Exit System
Choice: 2
Please enter the user ('patient' or 'auditor') you are attempting to query as: auditor
Please provide your RSA key file: EY.pem
Please enter your auditor ID: 2
Digital signature mismatch. Access denied
```

Invalid login by user and auditor

```

*****
Welcome to the Secure Decentralized Audit System
Please enter an option out of the following:
1. Register a User
2. Query audit records
3. Immutability verification
4. Exit System
Choice: 3
Please enter the user ('patient' or 'auditor') you are attempting to query as: patient
Please provide your RSA key file: Tom.pem
Please enter your patient ID: 1
You have been successfully authenticated as a registered user.
We will now walkthrough a scenorio where system detects changes made to audit log and report it
Instance of Audit Log created
Hashes match. We will proceed
Following is the data from the Audit Log instance created
date_time, patient_id, user_id, action_type
2022-02-1,1,rgosling,create
2022-02-11,4,ajolie,query
2022-02-12,3,bwillis,print
2022-02-15,2,tcruise,copy
2022-02-25,4,jdepp,delete
2022-03-2,5,cbale,change
2022-03-17,2,wsmith,query
2022-03-18,1,cevans,delete
2022-03-19,3,chemsworth,change
2022-03-24,2,prudd,create
2022-04-11,4,thanks,copy
2022-04-13,5,hford,print
2022-04-16,3,bpitt,delete
2022-04-19,2,bcooper,query
2022-04-21,4,djohnson,print
Select a line number between 1-16 to edit: 3
Following line will be edited: ['2022-02-12', '3', 'bwillis', 'print']
Enter the new value for the line: deleted
FILE TAMPERING DETECTED. ATTEMPT LOGGED IN FILE Tamper_Attempt_User_Tom_ID_1_2022-05-01 22;21;45.390704.txt

```

File tamper attempt by auditor

```

*****
Welcome to the Secure Decentralized Audit System
Please enter an option out of the following:
1. Register a User
2. Query audit records
3. Immutability verification
4. Exit System
Choice: 4
Thank you for using the system!

```

System exits

System Assumptions and Limitations

1. The implementation does not support decentralization as option 1 was selected.
2. Fernet encryption is used for encrypting all the information. As Fernet uses a timestamp in it token to encrypt data, the encryption value generated is unique. This creates a problem for generating digital signatures for user login information as the generated value will always be different. Therefore, the digital signature is first decrypted, and the obtained value is matched with the supplied information for validation.

3. As we use RSA for authentication, this brings up few restrictions like user cannot perform a trusted-realm authentication because RSA is used every time. Due to RSA being used, something like an identifier or alias cannot be used for login purposes.
4. As I am directly asking user for their credentials, this introduces Man-in-the-Middle attacks scenarios in my implementation and the supplied keys can be captured.
5. Using Fernet encryption simplifies integrity as a Fernet token has a HMAC filed in it. This 256-bit HMAC is the concatenation of version, timestamp, IV, ciphertext (encrypted using AES in CBC mode padded with PKCS7 algorithm) and signed using the signing key section of the generated Fernet key.

Cryptographic schemes exploration and Applicability

- **Homomorphic Encryption:** Homomorphic encryption algorithms are a type of encryption schemes that are designed to allow user to perform operations of the encrypted data itself. The encryption starts with plaintext message, m . Our target is to perform some operation on it, but the challenge is that this message shouldn't be exposed so it is safer to encrypt the message using enc before performing any operation on it. The cipher text generated after encryption is transformed into some other value with some other function. The output value is an encrypted message which can be decrypted back to the original message if the attempted operation was performed on it. This process stops any possibilities of data leakage. Instead of using Fernet encryption for the encryption of the data used like the audit file in .csv format, or the userID, ID and digital signature for registered users in authentication_server.json, Homomorphic Encryption can be used. Using it can eliminate the process of decrypting message and files every time to perform any required operations on it thereby securing the data and maintaining a higher form of security on them.
- **Zero-knowledge proofs:** Zero-knowledge proof is a concept by which one party, the prover, can prove to another party, the verifier, that a given statement is true while avoiding conveying any additional information apart from the fact that the given statement is true. This is a good concept to be used in the process of authenticating users without revealing any information like ID or any keys. This proof-of-knowledge concept can eliminate any sensitive information leaks during a possible man-in-the-middle attack.
- **Merkle Tree:** Merkle trees are data implementation structure where the leaves of the tree are the chunks of a file and the parents of these leaves being the hashes of the concatenation. A simple verification on a chunk of file, which was detected in audit trail, by recalculating the

Merkle root and comparing the values on the way can tell us if the chunk was tampered with or not. This functionality can be implemented in the mechanism to check the immutability of the audit log as any changes to file will leave an audit trail which is just the chunk of data that was tampered with and captured.

- **Elliptic-curve cryptography:** Elliptic-curve cryptography (ECC) is a concept of public-key cryptography that employs algebraic structures of elliptic curves over finite fields and thereby generating smaller keys to provide security. This reduced key size can directly benefit the storage and transmission capabilities. Implementing ECC in the system will reduce the key size generation. This will also power the functionality for smaller devices, and it will be simpler to implement and will require less computing power while providing same level of security as RSA.

References

<https://avinetworks.com/glossary/elliptic-curve-cryptography/>

Merkle Tree: CSCI 531, Lecture 6, Authenticated Encryption, by Dr. Tatyana Ryutov

<https://hackernoon.com/eli5-zero-knowledge-proof-78a276db9eff>

https://en.wikipedia.org/wiki/Zero-knowledge_proof

<https://brilliant.org/wiki/homomorphic-encryption/>