# CSCI 531 Semester Project Design Document

Daniel Shebib, 9101159372

## OVERVIEW

This document contains the high-level design for an EHR audit system. The system aims to allow patients and auditors to query pre-loaded audit data from a database. The audit data is assumed to be securely and correctly generated and stored, available in a simple log file. The audit system supports the following security guarantees as per the project document:[1]

1. **Privacy**: Patient privacy should be maintained. Unauthorized entities should not be able to access audit records. This is provided by maintaining authorization constraints on accessed data within the server. SSL is used to provide confidentiality and integrity for data in transit, and AES-GCM encryption provides confidentiality and integrity for data at rest.
2. **Identification and authorization**: All system users must be identified and authenticated. All requests to access the audit data should be authorized. Identities are generated and tracked using standard SSL procedures and certificates. Users are authorized through passwords, implemented and protected with HKDF.
3. **Queries:** Authorized entities should be able to query audit records. Records are maintained unencrypted in memory (the system is assumed to be protected against injection attacks), and encrypted on disk. Queries are authorized and then sent over SSL.
4. **Immutability**: No one should be able to delete or change existing audit records without detection. Any modifications/deletions of the audit records should be detected and reported. Integrity is provided in transit by SSL. Immutability of stored records on both the client and server-side is provided by the integrity mechanism of AES-GCM. Additionally, the server uses a blockchain to historically validate all changes to the log file, which is proved by all users of the system.
5. **Decentralization**: The system should not rely on a single trusted entity to support immutability. Blockchain is used to ensure immutability. All parties must sign the blockchain to verify the log integrity as well as the order of the log entries.

Other similar systems have been suggested in the past.[2,3,4] These have been useful starting points, particularly the HyperLedger Fabric design described by C. Stamatellis et al. This design focuses on an audit log server and has a small but robust prototype of the system described.

## IMPLEMENTATION

The system is implemented as a C++ program, with a client and server library that supports the required functionality of the system. An additional client and server stub simulates typical usage of the system, and a video of an unauthorized edit to the log is included to show how immutability is enforced by the server.

### Quick Start

**Github:** https://github.com/shebib/CSCI_531_EHR_Audit

To run, load the .sln using Visual Studio. Build and run, ensuring that both the Client and Server projects are run simultaneously (this can be done by editing the Startup Project under Properties for the solution). User passwords, auditors, and log entries can be seen in the InitDatabase.cpp file, please use them to test the demo.

## External Libraries

The following external libraries and packages were used:

- https://www.codeproject.com/Articles/1000189/A-Working-TCP-Client-and-Server-With-SSL.[5] This codeproject contains a simple SSL client/server setup. It is available under an open-source license (CPOL). This was used as the implementation of SSL, specifically using TLS 1.2, and forms the client-server relationship that is the basis of this prototype
- **Crypto++** (https://www.cryptopp.com/):[6] A C++ library with robust implementations of various cryptographic algorithms. This was used as the library which implements primitives such as AES, GCM, HKDF, SHA-256, etc.
- **HyperLedger Fabric** (https://www.hyperledger.org/use/fabric)**:**[7] Though this is not currently implemented in the prototype due to language constraints, the final design of the system is expected to use Hyperledger Fabric to provide the decentralized blockchain system to guarantee authorization and integrity of the audit logs.

## SYSTEM ARCHITECTURE

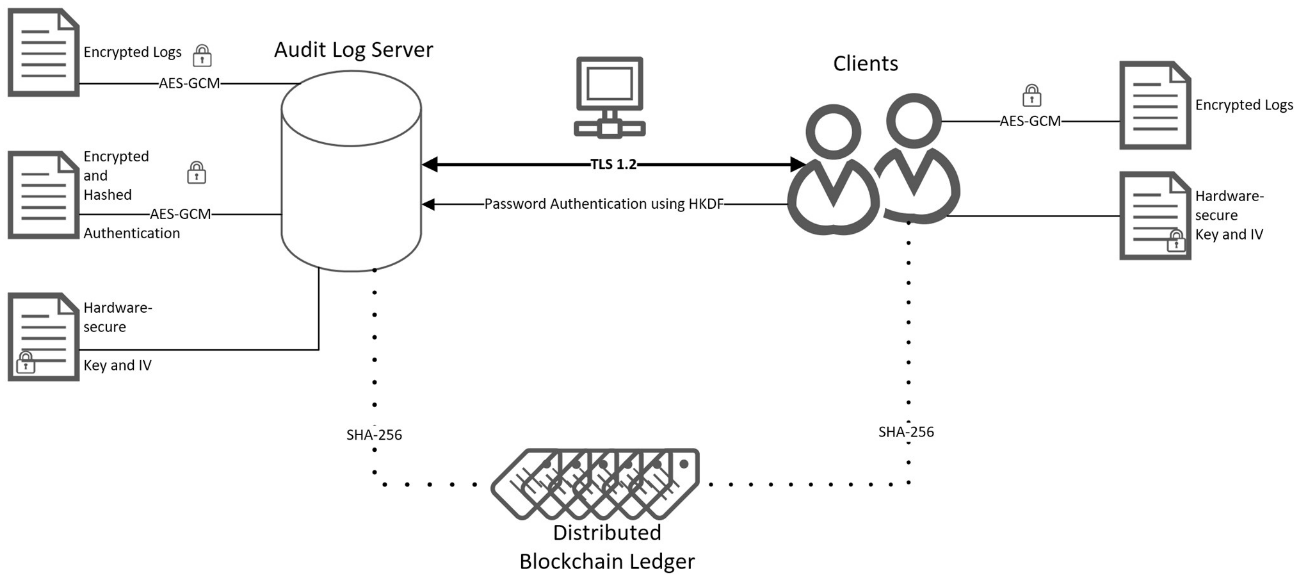
Encrypted Logs
AES-GCM
Audit Log Server
Encrypted and Hashed Authentication
AES-GCM
Hardware-secure Key and IV
TLS 1.2
Password Authentication using HKDF
Clients
AES-GCM
Encrypted Logs
Hardware-secure Key and IV
SHA-256
SHA-256
Distributed Blockchain Ledger

**Figure 1: System Architecture**

The above diagram shows the high-level architecture for the audit system. For this demo, there is only one audit log server and one (or multiple) clients connected. The more general system design will be described later as a scale-up of this prototype. Specific cryptographic constructions are discussed in the next section.

## Server

The server stores the log, handles client log accesses, and accepts log updates. It is designed with security in mind for all features. The server has a loop that allows it to handle multiple clients at a time. Each time a client initiates an SSL connection, the server runs a server instance which handles all communication and queries from the client.

At initialization, the server uses the securely stored hardware key and IV to read an AES key and IV last used to encrypt the server data files. This encryption uses AES-256 with GCM for authenticated encryption. If the authentication fails, the server emits an error and shuts down immediately, as the log files may have been tampered with by an external source. The server uses the key and IV to load the query logs as well as the authentication information for user logins.

User logins are implemented using HKDF to securely store derived password keys without storing the plaintext anywhere on the server. Each user has a 128-bit salt, which when combined with their password yields a key using HKDF. Matches between username and the derived key cause a successful login. Each user has a username, user id, and patient id. The permissions of auditors are enabled by having a special patient id of 0, which allows them to access all patient logs. Otherwise, users are limited to only access logs relating to their patient id. In the future, further tiers of permissions can be added to allow doctors and administrators to access different types of logs based on their permissions.

Once a user logs in, they can query the system for logs. Querying by patient id returns all the logs matching that patient id over the TLS connection. Auditors can query logs for any patient, patients can only query logs matching their own patient id, as enforced by the server.

Log integrity is guaranteed by the AES-GCM encryption and TLS connection: an attacker would have to either alter the AES-GCM-encrypted stored file (which is currently believed to be secure), or interfere with the data in transit over TLS1.2 (which is also currently believed to be secure).

Decentralization is guaranteed by the distributed blockchain ledger shared by all clients and servers of the system. This ledger contains chained SHA-256 hashes for each audit log entry. Both the server and the receiving client can check the hash of the log entry by retrieving the hash for the previous entry in the chain, appending it to the current log entry plaintext, computing the SHA-256 hash over the concatenation, and comparing the log to the chain. The result of the SHA-chain can be seen sent over SSL in the demo. In the final design, this is going to be implemented using Hyperledger Fabric, which uses more sophisticated digital signature and merkle tree functionalities in its blockchain.

## Client

Client instances are far more simple than the server. Clients initiate a connection with the server using the standard TLS1.2 handshake protocol. Note that demo clients accept a non-trusted local certificate, but the final version will pay for a CA to certify certificates in the production system. Clients are prompted for a username and password. If rejected, they are not allowed to query the system. If accepted, they may then query a patient id, which the server will return based on the clients' permissions.

To ensure data confidentiality at rest, clients save the transcript of their query records encrypted with AES-GCM. As with the server, keys and iv's are securely randomly generated and saved to a hardware-protected location for offline viewing.

Additionally, clients can ensure decentralized integrity by comparing indices and SHA-256 hashes of sent logs to the blockchain ledger.

## Source Files

The following is a description of the source files included in this project:

- **./lib:** This contains all the libraries used in this project for portability. Compiled libraries are in this directory, and included headers are in subfolders.
- **./server:** Contains all the server files. (SSLServerWrapper was modified, all other files were implemented specifically for this project)
- **./client:** Contains all the client files. (SSLClientWrapper was modified, all other files were implemented specifically for this project)
- **./InitDatabase:** Contains a single file to initialize the server database from scratch. This project should be run before the other two to simulate setting up the server and initializing the server encryption key and IV.
- **AuditCommon:** Contains generic declarations and structs used in the rest of the project.
- **CryptoInterface:** Class containing only static functions. This class is used as a wrapper around the CryptoPP library to simplify cryptographic calls.
- **ServerInstance:** An instance of the server which handles a single client. Is initialized in the wrapper and handles all server/client functionality.
- **QueryHandler:** Stores decrypted audit logs and handles queries internally within the server.
- **AuthHandler**: Stores decrypted authorization tokens and handles authentication internally within the server.
- **ServerFileHandler:** Handles reading/writing server audit log, authentication tokens, and encryption keys. Called at startup and shutdown.
- **ClientHandler:** Client instance.

## CRYPTOGRAPHIC COMPONENTS

The cryptography used in this project and prototype are intended to match current contemporary cryptographic good practice. The key cryptographic algorithms used are listed below:

**Table 1:** Cryptographic Components

| **Algorithm** | **Usage** | **Cryptographic Goals** | **Source** |
|---|---|---|---|
| TLS 1.2 | Algorithm for data transmittal between client and server | *Privacy:* Confidentiality of data in transit<br>*Integrity:* Integrity of data in transit | RFC 5246 |
| AES-256 with GCM | Used to encrypt all data at rest for both client and server | *Privacy:* Confidentiality of data at rest, for both client and server<br>*Integrity:* Integrity of data at rest, detects unauthorized editing of audit log | NIST Special Publication 800-38D |

| | | | |
|---|---|---|---|
| HKDF | Used to generate keys from username/salt/password combinations on the server-side | *Identification and Authorization:* Ensures that entities have correct credentials when accessing audit data.<br>*Privacy:* Username credentials are never stored on server (even encrypted), instead HKDF user/salt/key combinations are stored and calculated for each login attempt. | RFC 5869 |
| SHA-256 | Used in prototype to chain hashes to simulate blockchain verification | *Decentralization:* Ensures that audit log integrity can be maintained in a decentralized manner | FIPS 180-2 |
| Hyperledger Fabric | Used to hash and verify audit logs in the blockchain. | *Decentralization:* Ensures that audit log integrity can be maintained in a decentralized manner. | 7. |

## Discussion

The above set of cryptographic algorithms allow the system to provide the desired query functionality while maintaining a high standard for all cryptographic goals outlined in the introduction:

### CryptoPP

The library used to implement all cryptographic algorithms is CryptoPP.[6] This library has excellent documentation and a variety of cryptographic constructions. It also encourages a few basic primitives that enhance security, namely:

- SecBlock: A class that securely stores sensitive information in memory and ensures it is 0-wiped as soon as it goes out of context.
- Random: CryptoPP uses secure OS-specific methods to generate random keys and IV's. This prevents vulnerabilities such as those found in C++'s random methods.[8]

### SSL

TLS 1.2 was used to provide confidentiality and integrity for data in transit. Because this is a widely used standard, there are many packages that natively support it, and it is well-researched. TLS 1.3 is the newest standard and would be used in a production version of this system, as it has improved security guarantees over TLS 1.2.

### AES-GCM

AES is another standard block cipher. The requirements for the block cipher were to guarantee confidentiality and integrity of sensitive data stored on the client and server. This can be handled by any algorithm that provides authenticated encryption, so AES was used for encryption as it is the current industry standard, with GCM as a method of simultaneously guaranteeing integrity. AES in GCM mode is extremely powerful and is expected to be a new industry standard.[9]

### HKDF

Password storage is an important concept to get right. Password breaches have become commonplace, so it is critical to store sensitive password information in a way that prevents it from being used if it happens to be stolen. There are several key-derivation functions used for this purpose, but HKDF was chosen for its ease of use as well as its growing popularity.[10] When users create accounts, a random 128-bit salt is created and stored along with their username. The HKDF scheme was implemented as follows: `HKDF(password, salt, info),` where `info` stands for a standard string used to denote that the derivation is for password storage. This method of storage provides strong collision resistance and is time-consuming to compute, meaning that attackers who manage to access an unencrypted version of the authentication information would be unable to brute-force the original password in any reasonable computation time.

### SHA-256

To simulate the decentralized blockchain in the prototype, when returning audit logs the server will simulate a blockchain by simply chaining SHA-256 hashes for each log entry in order and returning their indices so that the client can check the integrity in decentralized manner. SHA-256 has strong collision-resistant properties and is the current standard practice. SHA-512 was tested but caused large message sizes over the demo, so SHA-256 was deemed a good compromise between security and prototype feasibility.

### Hyperledger Fabric

Decentralized integrity and authentication require complicated cryptographic protocols, and Hyperledger Fabric provides an industry-focused blockchain with high flexibility over those tied to specific cryptocurrencies.[7] With respect to an audit system, a blockchain can be used as a decentralized merkle tree, maintaining a history of log entries that can be efficiently checked by any entity by hashing log entries. Thus, for the final design of this system, every user would have access to the channel of audit log entries: patients so that they can check on their own logs and privacy, doctors/hospitals so that they can follow up on privacy breaches, and especially auditors, who need to be guaranteed that audit logs are complete, in order, and unmodified. New audit logs would be securely delivered to the audit server, which would be the only server with the permissions to sign new entries in the audit blockchain. Once signed however, the new blocks are immediately broadcasted so that all entities can hold the server accountable to the audit logs.

## Conclusion

The audit log system is designed with security in mind, thus sensitive data is almost always encrypted with AES-GCM, is transmitted with TLS, is always authenticated, and is signed and verified by a decentralized Hyperledger Fabric blockchain.

# SYSTEM LIMITATIONS AND PRODUCTIONIZATION

## Limitations

The prototype of this system is extremely limited. In particular:

- TLS certificates are not signed by any CA. This would obviously be fixed in the production version of the system by paying a trusted cert authority to sign the certificates and provide a properly authenticated SSL connection.
- TLS only works locally. Again, networking would be an obvious addition to the system.
- The system uses TLS 1.2. TLS 1.3 would be used in production for enhanced security.
- User authentication only requires a password. Though passwords are securely stored and authentication is implemented in the prototype, a production system would use an additional two-factor authentication method to improve the security of logins, as this is the new industry standard and ensures that a user's cryptographic identity cannot be simply written down on a post-it note.
- The server and client AES-key and last-used IV are stored unencrypted. In the production system, these would be stored by tightly-controlled hardware keys [[cite]], as access to these keys would allow an attacker to modify or delete the audit log maliciously. Though this would be detected by the decentralized Hyperledger Fabric network, the logs would not be recoverable in the current implementation (unless the logs themselves are encrypted with the server's public key by the server so that they are stored but only readable by the server, but this seems overkill (see next section)).
- Queries are limited to patient-id. For time constraints, only patient id's were used as query parameters to demonstrate that they work. In the production system, queries on any range of any field could easily supported by extending the current implementation.
- Queries cannot be checked using this library. In a production version, it may be desirable to provide auditors and patients with the functionality to check the retrieved logs for internal integrity – namely corroborating the access patterns of the logs with user roles to see if a breach took place.
- Hyperledger Fabric was not implemented for the prototype. Hyperledger Fabric is a considerably complex system to onboard and attach, and seems to be focused on non-C++ languages and also requires networking.

These limitations make the prototype unsuitable for production, but the prototype does meet the basic requirements of Privacy, Identification and Authorization, Queries, Immutability, and Decentralization. The following design changes would be used to productionize the full-fledged version of this HER audit system.

## Productionization

This section goes over the production version of the system design in more detail, building off of the description of the prototype above.

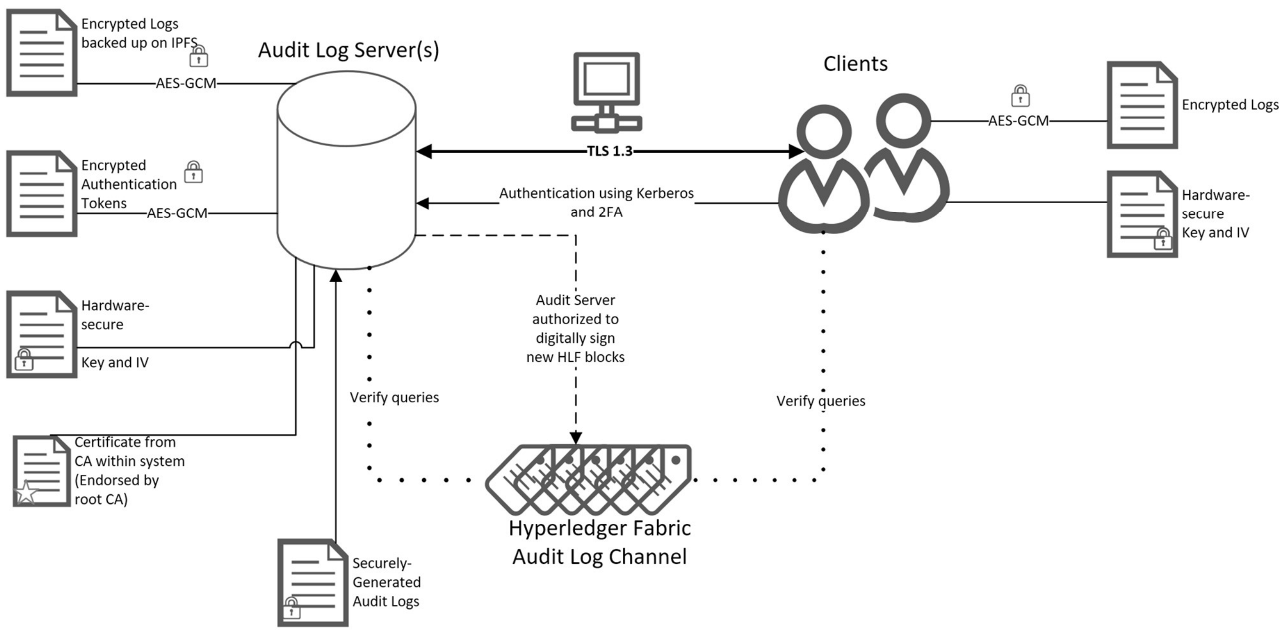
Encrypted Logs backed up on IPFS
AES-GCM
Audit Log Server(s)
Encrypted Authentication Tokens
AES-GCM
Hardware-secure Key and IV
Certificate from CA within system (Endorsed by root CA)
Securely-Generated Audit Logs
TLS 1.3
Authentication using Kerberos and 2FA
Clients
AES-GCM
Encrypted Logs
Hardware-secure Key and IV
Audit Server authorized to digitally sign new HLF blocks
Verify queries
Verify queries
Hyperledger Fabric Audit Log Channel

**Figure 2:** Productionized System Design

The system will be productionized in the following way:

- There may be multiple servers handling requests, all with the same setup.
- Each server will have a valid, signed certificate connected to the rest of the EHR system, endorsed by a root authority.
- The server will store encrypted audit logs on an IPFS to ensure decentralized storage and backup.
- The server will have securely implemented hardware AES and IV keys for log encryption.
- The server will have a secure channel that securely generates and delivers audit logs in real time.
- A full implementation of Hyperledger Fabric, with a dedicated channel for audit logs that audit log servers and all users have access to.
  - Only the central audit log server can sign new blocks (representing new audit logs) to the blockchain, and it will do so in real-time as logs are delivered.
  - The server's public key will be known for identification of log origin via the HLF system.
  - The HLF signatures will contain only hashes of the logs, log information is never stored in the HLF system (even in encrypted form).
- All users will have a secure method of storing AES keys used for local caches.
  - If desired, AES keys may be stored on the audit log server and only sent to clients during a live authentication session, thus only allowing clients to use this system to analyze the logs.
- Clients and the server will have a built-in functionality to verify the contents of the audit logs themselves.
- There will be more complex authentication requirements, for example:
  - Doctor/hospital roles will allow doctor/hospital staff to access logs of patients under their domain.
- Clients (both auditors, patients, and other staff as needed) will use two-factor authentication to login, and authorization may be implemented using Kerberos

    - This allows a trusted third-party to manage the authentication between users securely, as Kerberos is widely used and trusted.
- Productionized codebase and improved UI and functionality

The above changes would allow the system to handle more users, more functionality, and with public-facing cryptographic security.

## CONCLUSION

This paper has outlined the design of a secure EHR audit system. The system provides privacy, integrity, authentication, query functionality, and decentralization. The simplified prototype provides the same guarantees and points the way towards the full-fledged production system. It should be noted that this is a small part of a larger system with many different users and use-cases, so the design is extremely preliminary. Additionally, it should be noted that no quantum-safe or zero-knowledge cryptography is used, as the former is as-yet unproven and the latter is computationally intensive and goes far beyond the cryptographic requirements of the system (and would likely only lead to unnecessary overhead with IT, administration, etc.).

## REFERENCES

1. T. Ryutov, “CSCI 531 Spring 2021 Semester Project.”
2. Charalampos Stamatellis; Pavlos Papadopoulos; Nikolaos Pitropakis; Sokratis Katsikas; William J Buchanan, “A Privacy-Preserving Healthcare Framework Using Hyperledger Fabric,” *arXiv - CS - Cryptography and Security*, 2020.
3. D Tith, JS Lee, H Suzuki, W Wijesundara, N Taira, T Obi, N Ohyama, “Application of Blockchain to Maintaining Patient Records in Electronic Health Record for Enhanced Privacy, Scalability, and Availability,” *Healthcare Informatics Research* 26 (1), 3-12, 2020.
4. M. M. Madine et al., “Blockchain for Giving Patients Control Over Their Medical Records,” in *IEEE Access*, vol. 8, 2020.
5. D Maw, “A Working TCP Client and Server with SSL,” https://www.codeproject.com/Articles/1000189/A-Working-TCP-Client-and-Server-With-SSL. Sep 24, 2020.
6. (Open-source), “Crypto++ Library 8.5,” https://www.cryptopp.com/.
7. Hyperledger, “An Introduction to Hyperledger,” https://www.hyperledger.org/wp-content/uploads/2018/08/HL_Whitepaper_IntroductiontoHyperledger.pdf
8. CryptoPP, “RandomNumberGenerator,” https://www.cryptopp.com/wiki/RandomNumberGenerator.
9. NIST, “NSA Suite B Cryptography,” https://csrc.nist.gov/projects/computer-security-objects-register/algorithm-registration#AES.
10. NIST, “Recommendation for Key-Derivation Methods in Key-Establishment Schemes,” https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Cr2.pdf.