An API for Secure Sharing of Electronic Health Records in a Public Blockchain

Maria Patricia Javier[1], Earth Wendell Lopez[1], Gabriel Luis Marcelo[1], and Katrina Ysabel Solomon[1]

Advanced Research Institute for Informatics, Computing, and Networking,
De La Salle University,
2401 Taft Avenue, Manila 1004, Philippines
katrina.solomon@dlsu.edu.ph

Abstract. Electronic Health Records (EHRs) contain information such as diagnoses, medications, and allergies of individuals stored in a digital manner. Given the sensitive nature of these information, EHR systems with improper access control may be vulnerable to attacks leading to a breach of confidentiality. Those with strict security controls, on the other hand, limit the accessibility of the records, making them difficult to share said records among various medical institutions and professionals. One approach to addressing the accessibility of EHRs, while not compromising confidentiality and access control, is by utilizing blockchain technology. The inherent security of blockchains allow it to strengthen existing EHR systems by addressing the limitations previously mentioned. In this research, an Application Programming Interface (API) was developed to integrate an EHR system to a blockchain network. The API includes functionalities such as adding records, sharing and retrieving records, and access control.

Keywords: electronic health records, blockchain, access control

1 Overview

Healthcare is undoubtedly a necessity in people's lives and with it comes a collection of data. These medical data can be stored in Electronic Health Records (EHRs). These digital records can improve the efficiency of sharing medical data throughout a medical facility. It can avoid redundancy and inaccuracy that is usually the limitation of physical health records. However, EHRs also come with its own limitations. They are commonly maintained using centralized systems which may lead to a single point of failure if not secured properly. In other cases, EHR systems tend to be too restrictive that it accessibility becomes an issue for authorized entities [1].

One major incident of an EHR system breach happened to an established healthcare company, Magellan Health, Inc. On the 12th of June 2020, Magellan health notified the public of the breach that happened by issuing a Notice of Security Incident. In this statement, Magellan Health reported what happened—a ransomware attack occurred on April 6 and was only detected by Magellan Health on April 11. The attack led to the breach of important information such as treatment information, health insurance account information, member ID, email addresses, phone numbers, and other health-related information. Despite Magellan Health not having evidence that any personal data has been misused, it should be taken into account that the leaked information may be used by the hackers with malicious intent [2] [3].

Blockchain serves as a digital ledger that can be used to hold information in a distributed manner without requiring a third-party or centralization. Among its properties, blockchains are notably known for being tamper-proof due to its immutability. Transactions done over a blockchain are only considered valid after a consensus had been reached [4]. Additionally, transaction details cannot be modified anymore once it has been validated. Blockchain rose to popularity because of its use in cryptocurrencies such as Bitcoin. This technology has since been explored and applied to other domains such as Internet of Things (IoT), business, and healthcare.

The integration of blockchain to EHR systems is a promising improvement to the current state of medical data sharing. Blockchain-based EHR systems are deployed in a decentralized manner which may prove useful in addressing the limitations of traditional centralized EHR systems. Furthermore, the existing properties of blockchains ensure that the accessibility and integrity of EHRs are upheld.

BHEEM, a proposed blockchain-based framework by [1], provides an efficient approach to storing as well as transferring or sharing of EHRs. The framework defines the constituent nodes, the IT components, as well as the blockchain contracts. Its architecture involves functionalities namely addition of a block, addition of a patient, access permission, addition of a record, retrieval of a record, and transfer of a record. This framework proposed by [1] is an established framework for blockchain although it is not particularly implemented and/or used by healthcare systems today.

While blockchain-based EHR systems do well in terms of addressing the issues of traditional EHR systems, it presents a new problem due to limitations on functionality when it comes to secure data sharing of EHRs. Most blockchain-based EHR systems are implemented in a decentralized manner posing some issues such as accidental or actual malicious access towards these health records which can cause a breach of confidentiality. According to multiple legislations such as the Health Insurance Portability and Accountability Act (HIPAA) and the Code of Federal Regulations, patients are given the right over their health information and may be the one to dictate on who can access their health information. Medical records that are shared need consent from the owner ranging from the type of data to be shared, information about the recipient, and even the period of time which the data can be accessed by the recipient [5]. Whether the electronic health record is in a traditional or digital format, the rules of its privacy and confidentiality remain the same. No healthcare data must be exposed, accessed, nor shared without the consent of the owner hence the limitation of blockchain-based EHRs towards privacy and confidentiality must be addressed.

2 System Design and Implementation

2.1 System Overview

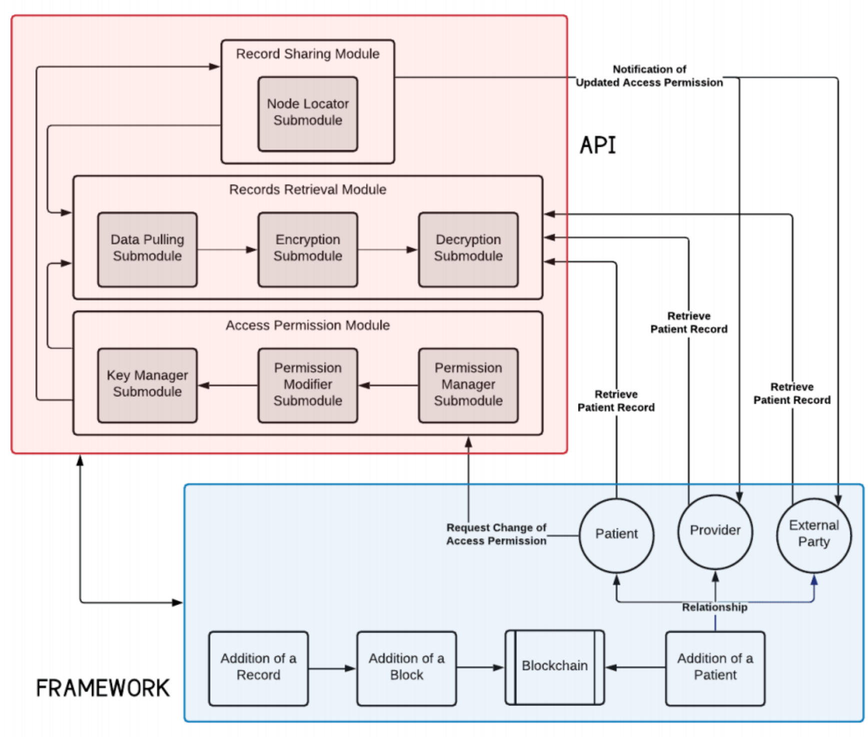


Fig. 1. System Overview

Figure 1 shows the overview of how the API gets called when integrated with the system. The system implementation mostly follows the BHEEM framework [1] only with data stored in the blockchain itself. Transactions are created through the blockchain in which data or information are stored using a smart contract.

The BHEEM framework was implemented with modifications mainly in terms of storage. It covers the blockchain creation and its storage management. It also covers the storage of the EHRs and as well as the creation of nodes and its relationships, addition of a patient to the chain, the addition of a block to the chain, and the addition of a record to a block. The API uses a smart contract to perform operations needed for secure sharing, thus every module interacts with the blockchain for every transaction or call.

The five essential functions of the API namely, data pulling, access control and permission, encryption, decryption, and record sharing will be distributed to three main modules: Access Control and Permission Module, Records Sharing Module, and Record Retrieval Module.

For the developed system, the assumption made by the researchers is that the records being handled throughout are added by the respective healthcare providers. However, ownership of said records is to be directly linked to the patients thus, giving them sole access prior to sharing it with their respective providers or chosen external parties. The Owner is to be regarded as the patient and the Receiving Node or Recipient would be either the Provider or an External Party. The following figures display a brief step-by-step process of communication among different users to allow for better understanding of how the system works upon the interaction of different entities with the proposed API for the involved functionalities. A more in-depth discussion for each process would be covered in the subsequent subsections through each of the proposed API's modules.

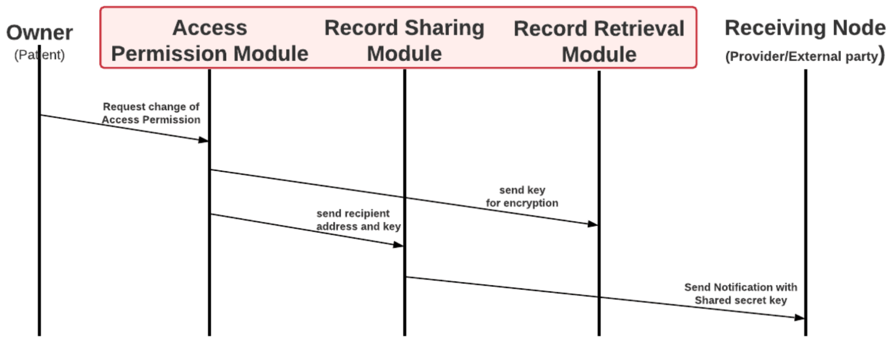

Owner
(Patient)
Access Permission Module
Record Sharing Module
Record Retrieval Module
Receiving Node
(Provider/External party)
Request change of Access Permission
send key for encryption
send recipient address and key
Send Notification with Shared secret key

Fig. 2. Record Sharing Process

Figure 2 provides a brief process of the communication when performing record sharing in the system through the API. The process starts with the owner requesting a change of access permissions through the API's Access Permission Module where after processing, would proceed with sending a key to the other two modules, the Record Sharing Module and Record Retrieval Module, and the recipient's address to Record Sharing Module only. The Record Sharing Module's responsibility would proceed with sending a notification to the Receiving Node, informing The user about the shared record where simultaneously would provide them the secret key needed for accessing the said resource.

Figure 3 shows the overall process involving a recipient's record retrieval through the API. The process of decryption would be done within the client side

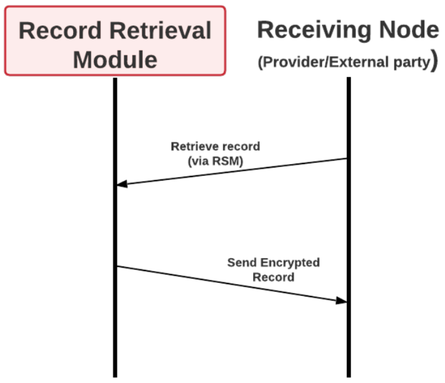


Fig. 3. Recipient's Record Retrieval Process

using the private key shared to the user upon the notification that the certain record has been shared. Assuming the private key used is valid, the recipient would finally be able to view the record in plaintext format as well. In this way, risks involving the sharing of keys over the network are minimized.

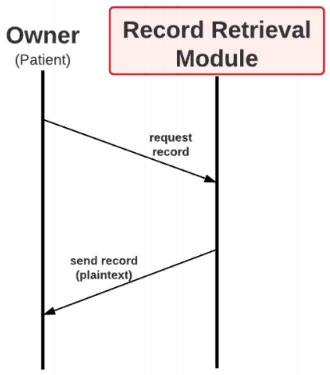


Fig. 4. Recipient's Record Retrieval Process

Figure 4 on the other hand displays a simple process in the case that the owner themselves wished to retrieve their own record. To do so, the owner simply would have to request the record through the Record Retrieval Module and would be able to easily receive it in its plaintext format as it is sent back.

Access Control and Permission Module Invoke and revoke of permissions as well as key generation happen in the Access Control and Permission Module. If the record owner is the one retrieving the record, then the owner can simply proceed to the Record Retrieval Module. On the other hand, if the owner intends to share a record, then the process starts with an owner's request for change of permission and generates the original access permission of the requested record. The module then updates the original access permission by creating a new transaction for the updated access permission and saves the recipient address which is forwarded to the Record Sharing Module. The process then generates cryptographic key and is forwarded to both the Record Retrieval Module and Record Sharing Module.

Record Sharing Module The Record Sharing Module is responsible for notifying the recipient with regards to their updated access permissions to a file. From the Access Control and Permission Module, the recipient address and private key file name is received. The module locates the recipient node and notifies the recipient that a record has been shared and will be accessible for said recipient. Along with the notification, the recipient is also provided the key via secure key exchange. The notification and the key allows the recipient to proceed to the Record Retrieval Module.

Record Retrieval Module Responsible for fetching the requested information by either the owner or recipient being shared to is the Record Retrieval Module. The module allows for both types of user to retrieve the record, varying mostly upon the need to securely provide a record for the intended recipient before completely accessing the said information. For the owner, a checking of Access Permissions for the record will first be verified where a confirmation will result in the retrieval of the record and the owner receiving it in plaintext. For recipients given access permission by the owner, the process also begins with receiving the recipient's request, verifying access permission, and pulling the requested record from the blockchain. The pulled plaintext record is to be ciphered with the owner's key in accordance with the specified encryption algorithm before being sent to the recipient. The recipient then, with the shared key, will have to decipher the encrypted record to receive the information again but in plaintext.

2.2 System Implementation

Electronic Health Record (EHR) Data Structure Due to smart contracts' limitations on the maximum number of variables to be processed in a transaction, the chosen EHR data structure was based on immunization records. These data are synthetic and was generated using Synthea. A sample EHR can be seen in Table 1.

2.3 Blockchain and Smart Contract

Blockchain processes such as creating and adding of blocks in the blockchain, mining of transactions, and verification of transactions will be done through the

Table 1. Sample Immunization Record

Field	Sample Data
Record_ID	(ID specific to record)
Date	08/01/2021
Patient_ID	0xfB13CCd7c748952eC91C7A42a3C8eD6C86d0 b97F
Birthdate	08/24/1986
Complete_Name	Kris Jacinto
Gender	M
Address	888 Hickle Ferry Suite 38
City	Springfield
Code	140
Description	Influenza seasonal injectable preservative free
Base_Cost	140.52
Healthcare_Expenses	8446.49
Healthcare_Coverage	1499.08

tool, Ganache [6]. Ganache is an Ethereum simulator that can make development for Ethereum applications easier, faster, and safer. It can provide a local blockchain that allows for developers to develop, deploy, and test projects and smart contracts such as Solidity in a safe environment and without the need to pay for gas fees provided by Ethereum transactions making it cost-effective. Additionally, since Ganache uses a local blockchain, transactions are almost instantaneous which avoids the issues of slow-paced development in an Ethereum blockchain because of its need for time to upload and deploy before contracts can be tested. Furthermore, Ganache has two versions with one being used for command-line interfaces called the ganache-CLI and the other is a full-fledged desktop application with a graphical user interface called Ganache UI — to which the researchers used the latter.

In Ganache Version 2.5.4, the addition of a block is done whenever a transaction is performed. Transactions are done with the help of a smart contract that was developed in the local environment of Ganache. Once a transaction or set of transactions has been completed, it will be added to a block pending its addition to the chain. Aside from the transactions, the block must also contain the hashed unique identifier of the previous block, hence the "chain" concept. The current block will be assigned its own unique identifier, then will undergo hashing with a nonce value. This block is then added to the chain. Both the blocks and transactions can be seen in the user interface that Ganache provides.

In mining, by default, Ganache uses its automine feature that allows for instantaneous processing of transactions, however, developers have the option to disable this. By disabling the automine feature, developers then need to input the time interval in the field that will determine the number of seconds of each block mining. Whether automine is enabled or not, transactions will not be stored in a transaction pool and will be included in the current block.

One smart contract was implemented for the system to reduce gas cost including the extra ones incurred when reading and writing to and from multiple contracts. This smart contract contains all the functions that perform the operations and calls to the blockchain as well as the storage of data as opposed to the multiple smart contract implementations. The three main information being stored in the smart contract as structs are users, records, and permissions. The permissions are stored in an array while both users and records are stored in a mapped array wherein wallet address and record ID is used as primary key, respectively.

2.4 API

The implementation of the API is deployed in a Django environment which utilizes the Django REST framework, a tool that allows developers to create Web APIs. The Django environment consists of connected files that work together to allow the API to be fully functional. The API is written using Python and utilizes HTTPS. The SSL certificate is self-signed since the API is only deployed on a local test environment. It should be noted that on actual deployment, the certificate must be obtained from a certificate authority (CA).

The API acts as a gateway for external clients to communicate with the smart contract deployed in the test blockchain and vice-versa. Responsibilities for the API include handling requests made by the user, making calls representing said requests, interacting with the smart contract, and reading the data that comes from both ends.

It utilizes the web3 library in order to communicate with the smart contract which handles the operations and calls to the blockchain and storing data in the smart contract. The crypto library is utilized for the cryptographic requirements of the system implementation, mainly focusing on AES, 3DES, and Blowfish, mostly for encrypting and decrypting records when being accessed by a valid recipient. Other libraries such as secrets and struct were also used for cryptographic purposes. The pickle library was used for saving and retrieving keys on the server. The uuid library was used for the generation of key IDs to ensure that they are unique and is considered safer than using integer-based IDs. The requests and json library was used for handling the API endpoints and its data. Lastly, the DiffieHellman library was used for performing the Diffie-Hellman key exchange algorithm for a secure key exchange.

API Requests Table 2 lists all possible requests a client can call to the API to perform functions from the smart contract.

Table 2: API Requests

Request	Method	Parameter(s)	Response
initialize	GET	N/A	Returns a message that it is connected to the blockchain. Otherwise, it returns a message stating that it failed to connect to the blockchain.
add_user	POST	address, role	Returns the value of the address and user role as JSON. Otherwise, it returns a message that the user already exists.
check_user	GET	address	Returns a message that tells that it has logged successfully. Otherwise, it returns a message that tells that log-in has failed and the address is not found.
get_current	GET	N/A	Returns the address of the current account logged in. Otherwise, it returns a JSON response error.
add_record	POST	record_date, birthdate, name, gender, address, city, vaccine_code, desecription, base_cost, expense, coverage	Returns a message that the record is successfully added with its corresponding record ID. Otherwise, it returns a message that the addition of the record is unsuccessful.

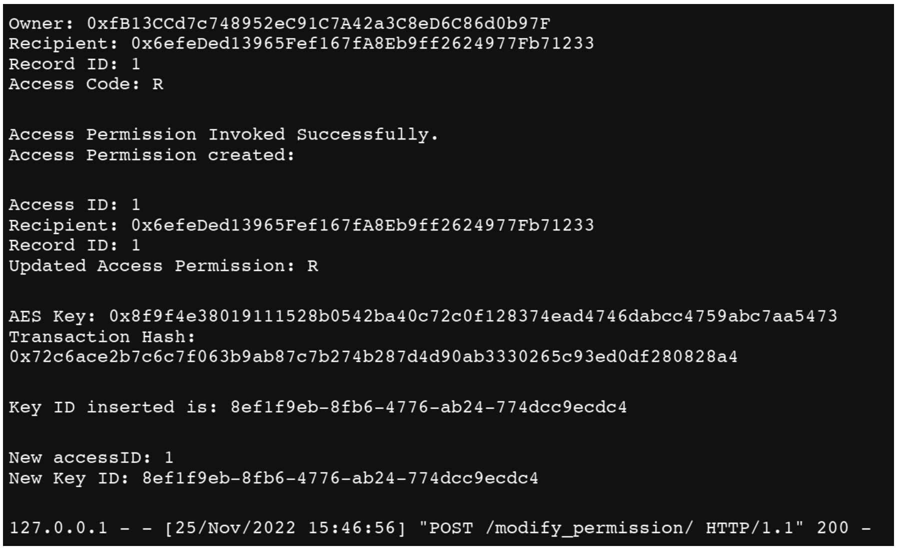
Table 2 continued from previous page

Request	Method	Parameter(s)	Response
modify_permissions	POST	owner_address, recipient_address, record_ID, permission	Returns a message that the modification of record permission is successful alongside the key ID associated with it. Otherwise, it returns a message that the modification of record permission is unsuccessful.
view_permissions	GET	address	Returns a list of current permissions that the user has which contains the access ID, record ID, and key ID. Otherwise, it returns a message that viewing of record permissions is unsuccessful.
load_user_keys	POST	address, public_key	Returns the available keys of the current address, nonce used for encryption, and the API's public key for Diffie-Hellman. Otherwise, it returns a JSON Response error.
check_owner	GET	record_ID	Returns a value depending if the current address is the record owner. Otherwise, it returns a JSON response error.
retrieve_record	GET	record_ID	Returns the record of the owner. Otherwise, it returns a JSON response error.
	POST	record_ID, address, private_key	Returns the encrypted shared record of the owner to the recipient. Otherwise, it returns a JSON response error.

3 Test Results

To determine the overall performance of the API, functionality tests were conducted. The functionality testing focuses on checking whether the expected output is achieved for the tasks of each module.

To test the API's functionality, a sample client-side application is created to interact with the API. The client requires user address registration and login in order to use the functionalities offered by the API. Divided into three modules, the Access Control functionality is under the Access Permission module, Encryption, and Data Pulling functionalities under Record Retrieval module, and Data Sharing functionality under the Record Sharing module. For this functionality test, the default encryption algorithm, AES, is selected.

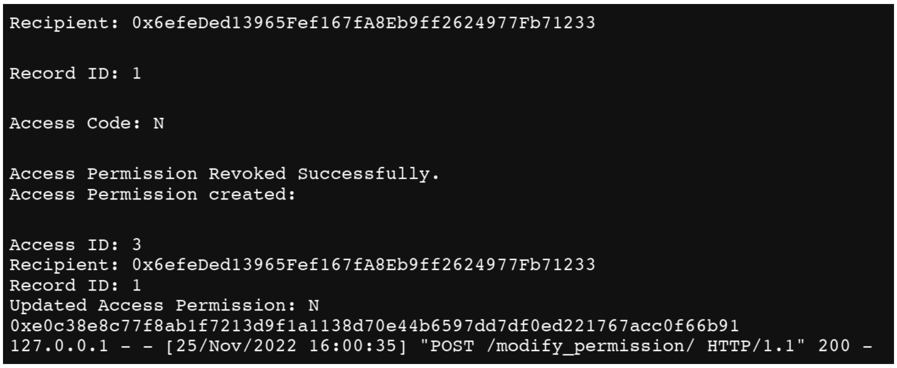


```
Owner: 0xfB13CCd7c748952eC91C7A42a3C8eD6C86d0b97F
Recipient: 0x6efeDed13965Fef167fA8Eb9ff2624977Fb71233
Record ID: 1
Access Code: R

Access Permission Invoked Successfully.
Access Permission created:

Access ID: 1
Recipient: 0x6efeDed13965Fef167fA8Eb9ff2624977Fb71233
Record ID: 1
Updated Access Permission: R

AES Key: 0x8f9f4e38019111528b0542ba40c72c0f128374ead4746dabcc4759abc7aa5473
Transaction Hash:
0x72c6ace2b7c6c7f063b9ab87c7b274b287d4d90ab3330265c93ed0df280828a4

Key ID inserted is: 8ef1f9eb-8fb6-4776-ab24-774dcc9ecdc4

New accessID: 1
New Key ID: 8ef1f9eb-8fb6-4776-ab24-774dcc9ecdc4

127.0.0.1 - - [25/Nov/2022 15:46:56] "POST /modify_permission/ HTTP/1.1" 200 -
```

Fig. 5. Server-side Record Permission Invocation

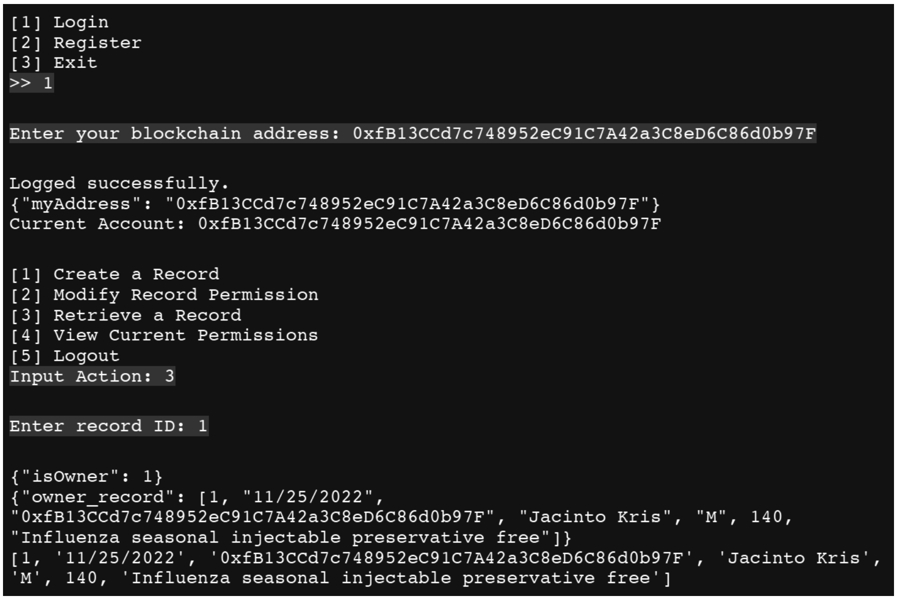
As for the access permission, the access control functionality is tested out by modifying record permissions of added records, given that the one requesting for change of access permission of a record is its owner. A record is shared to a recipient using the Invoke Permission option as shown in Figure 5. This indicates that the recipient now has (R)ead access permission for the shared record.

In the instance that an owner wants to change a recipient's access into having no access, this can be performed by choosing the Revoke Permission option. Compared to the response received when giving a (R)ead access, Figure 6, on the other hand, returns almost the same information but shows that the key ID created with the invocation has been deleted.



```
Recipient: 0x6efeDed13965Fef167fA8Eb9ff2624977Fb71233

Record ID: 1

Access Code: N

Access Permission Revoked Successfully.
Access Permission created:

Access ID: 3
Recipient: 0x6efeDed13965Fef167fA8Eb9ff2624977Fb71233
Record ID: 1
Updated Access Permission: N
0xe0c38e8c77f8ab1f7213d9f1a1138d70e44b6597dd7df0ed221767acc0f66b91
127.0.0.1 - - [25/Nov/2022 16:00:35] "POST /modify_permission/ HTTP/1.1" 200 -
```

Fig. 6. Server-side Record Permission Revocation

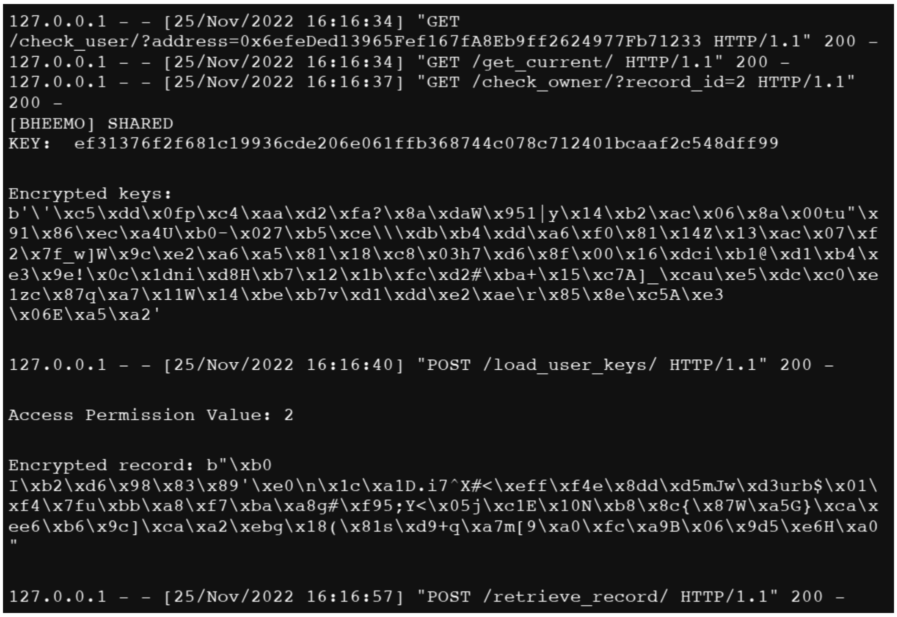


```
[1] Login
[2] Register
[3] Exit
>> 1

Enter your blockchain address: 0xfB13CCd7c748952eC91C7A42a3C8eD6C86d0b97F

Logged successfully.
{"myAddress": "0xfB13CCd7c748952eC91C7A42a3C8eD6C86d0b97F"}
Current Account: 0xfB13CCd7c748952eC91C7A42a3C8eD6C86d0b97F

[1] Create a Record
[2] Modify Record Permission
[3] Retrieve a Record
[4] View Current Permissions
[5] Logout
Input Action: 3

Enter record ID: 1

{"isOwner": 1}
{"owner_record": [1, "11/25/2022",
"0xfB13CCd7c748952eC91C7A42a3C8eD6C86d0b97F", "Jacinto Kris", "M", 140,
"Influenza seasonal injectable preservative free"]}
[1, '11/25/2022', '0xfB13CCd7c748952eC91C7A42a3C8eD6C86d0b97F', 'Jacinto Kris',
'M', 140, 'Influenza seasonal injectable preservative free']
```

Fig. 7. Client-side Record Retrieval by Owner

Retrieving a record works in two ways: one, if the owner of the record is the one retrieving it and, two, when a recipient wants to retrieve a shared record. If as an owner, the user simply needs to provide a record ID with the request. Data will be easily pulled from the blockchain and returned back in plaintext/json. Figure 7 shows this instance of an owner retrieving their own record.

For recipients, on the other hand, when wanting to access a record they have been given access to, the user will need to provide more than the record ID. Initially, the keys are first updated in cases of any permission changes prior. The user is then asked to provide the record ID, filename of his key dictionary, and



```
127.0.0.1 - - [25/Nov/2022 16:16:34] "GET
/check_user/?address=0x6efeDed13965Fef167fA8Eb9ff2624977Fb71233 HTTP/1.1" 200 -
127.0.0.1 - - [25/Nov/2022 16:16:34] "GET /get_current/ HTTP/1.1" 200 -
127.0.0.1 - - [25/Nov/2022 16:16:37] "GET /check_owner/?record_id=2 HTTP/1.1"
200 -
[BHEEMO] SHARED
KEY:  ef31376f2f681c19936cde206e061ffb368744c078c712401bcaaf2c548dff99


Encrypted keys:
b'\'\xc5\xdd\x0fp\xc4\xaa\xd2\xfa?\x8a\xdaW\x951|y\x14\xb2\xac\x06\x8a\x00tu"\x
91\x86\xec\xa4U\xb0-\x027\xb5\xce\\\xdb\xb4\xdd\xa6\xf0\x81\x14Z\x13\xac\x07\xf
2\x7f_w]W\x9c\xe2\xa6\xa5\x81\x18\xc8\x03h7\xd6\x8f\x00\x16\xdci\xb1@\xd1\xb4\x
e3\x9e!\x0c\x1dni\xd8H\xb7\x12\x1b\xfc\xd2#\xba+\x15\xc7A]_\xcau\xe5\xdc\xc0\xe
1zc\x87q\xa7\x11W\x14\xbe\xb7v\xd1\xdd\xe2\xae\r\x85\x8e\xc5A\xe3
\x06E\xa5\xa2'


127.0.0.1 - - [25/Nov/2022 16:16:40] "POST /load_user_keys/ HTTP/1.1" 200 -


Access Permission Value: 2


Encrypted record: b"\xb0
I\xb2\xd6\x98\x83\x89'\xe0\n\x1c\xa1D.i7^X#<\xeff\xf4e\x8dd\xd5mJw\xd3urb$\x01\
xf4\x7fu\xbb\xa8\xf7\xba\xa8g#\xf95;Y<\x05j\xc1E\x10N\xb8\x8c{\x87W\xa5G}\xca\x
ee6\xb6\x9c]\xca\xa2\xebg\x18(\x81s\xd9+q\xa7m[9\xa0\xfc\xa9B\x06\x9d5\xe6H\xa0
"


127.0.0.1 - - [25/Nov/2022 16:16:57] "POST /retrieve_record/ HTTP/1.1" 200 -
```

Fig. 8. Server-side Record Retrieval by Recipient

the key ID corresponding to the private key needed to decrypt the record being tried to retrieve. If all parameters requested are valid, the server will then be able to return the shared record. The server side will check if the user accessing the record has a valid access permission for said file. If the user does have access to the file, it will retrieve the record and return it to the requester. This scenario is shown in Figure 8.

4 Conclusion

The research aimed to develop an API to improve the use of public blockchain technology for healthcare by addressing limitations in security or privacy issues in terms of sharing sensitive data and information such as Electronic Health Records (EHRs) by adding a layer of security using a cryptographic algorithm to the data at rest prior to transit. This implementation eliminates the aforementioned issues of traditional EHRs being prone to tampering and general inaccessibility. Furthermore, blockchain-based EHRs utilizes the strengths of blockchain as one needs to compromise all that has the ledger for data to be possibly compromised. The decentralized characteristic of blockchain allows for user discretion in sharing their own healthcare data with the benefit of blockchain's immutability. Along with the benefits of blockchain, the API's functionalities were tested using a client application developed by the researchers. Ganache was used as a testing environment for the blockchain, Remix for the smart contract, Django framework for the application development, and Web3 for the connection between the application code, the smart contract, and the blockchain. The API functionalities: record addition, record sharing, and record retrieval were tested with various API calls designed to communicate through the blockchain with a smart contract were successful. Records from the blockchain can be added, shared given that the user gives permission to another user, and accessed by users that have been granted to view records which can be checked with the client application.

As the study focused on API development as a possible solution to the problem, the blockchain environment was not much focused on which left the gas fees and practical cost of implementation out of consideration such as details on upkeeping user balances. Input validation for records were not taken into consideration as the researchers assumed correct input for testing purposes. Also, as the study follows the existing framework wherein access control on roles were not discussed in detail, the implementation offers different roles for a user but has the same access on functionalities regardless if the user registered as patient, a doctor, or a healthcare provider. Since the study followed in utilizing blockchain for storing data, it's immutable characteristic also offers advantages with it being tamper-proof but also comes with disadvantages such as not being able to modify erroneous inputted data and smart contract's limitation on maximum number of variables it can handle in a transaction, given that the records being stored are healthcare data which are usually composed of several data fields.

The implementation also utilizes the Django REST API framework which provides flexibility and ease of implementation, as well as test coverage of source code, for the necessary client-server setup to test out the API itself. With this, there is an option to utilize a database for the web app implementation which the researchers decided not to focus on as exploration of the blockchain technology as a storage for healthcare data is intended. Other options for storing EHR data include other hospital's systems with their own servers and Cloud-based EHRs. These other options also come with its advantages and disadvantages such as EHR systems being prone to theft or data being tampered while cloud-based solutions become more expensive as the need for storage increases. The best choice for data storage relies on the use case of the API depending on the needs of the system that will use it. It could also be possible to implement a hybrid storage which involves both the blockchain and database that can utilize both its strengths together to create a better system overall.

References

1. Vora, J., Nayyar, A., Tanwar, S., Tyagi, S., Kumar, N., Obaidat, M. & Rodrigues, J. BHEEM: A Blockchain-Based Framework for Securing Electronic Health Records. *2018 IEEE Globecom Workshops (GC Wkshps)*. pp. 1-6 (2018)
2. Cyber autopsy series: Phishing attack on magellan health. *GlobalSign*. (2020,10), https://www.globalsign.com/en/blog/cyber-autopsy-series-phishing-attack-magellan-health (visited on 08/18/2023)

3. Davis, J. Magellan health settles for $1.43m after Data Breach, delayed notification. *SC Media*. (2022,9), https://www.scmagazine.com/analysis/magellan-health-settles-for-1-43m-after-data-breach-delayed-notification (visited on 08/18/2023)
4. Yaga, D., Mell, P., Roby, N. & Scarfone, K. Blockchain Technology Overview. *CoRR*. **abs/1906.11078** (2019), http://arxiv.org/abs/1906.11078
5. Dubovitskaya, A., Xu, Z., Ryu, S., Schumacher, M. & Wang, F. Secure and trustable electronic medical records sharing using blockchain. (2017,8)
6. Ganache. *Ganache - Truffle Suite*. https://trufflesuite.com/ganache/ (visited on 08/18/2023)