< BLOCKCHAIN BASED CERTIFICATE AUTHORITY (CA) SYSTEM >

TECHNICAL REPORT

Start Date: **1 / NOV / 2022**

Authors : **Student1, s3850825 You Chan Lee**
Student2, s3860956 Jingxuan Long
Student3, s3869867 Yingying Guo
Student4, s3860864 Yang Feng
Student5, s3860973 Shaohui Wang

DOCUMENT CONTROL

Version #	Implemented By	Implementation Date	Reviewed By	Approval Date	Reason
1	Yang Feng, Jingxuan Long,	03/NOV/2022	Shaohui Wang	03/NOV/2022	**Writing** 1 Executive summary 2 Introduction 3 Requirements 4 Architecture 11 References
2	Yingying Guo	03/NOV/2022	Shaohui Wang	03/NOV/2022	**Writing** 5 Technical Framework 8 Test specifications 9 Test results
4	Shaohui Wang	04/NOV/2022	Yang Feng	04/NOV/2022	**Correcting** previous statement for database and testing parts, adding necessary descriptions and diagrams **Writing** 6 Implementation (Database & Blockchain) 7 Deployment Instructions
5	Jingxuan Long	07/NOV/2022	Yingying Guo	07/NOV/2022	**Correcting** 3 Requirements 4 Architecture **Writing** 10 OTHER CONSIDERATIONS

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Blockchain is defined as a distributed digital ledger that can record decentralized information such as cryptocurrency transactions. Each block contains a particular quantity of information, and the blocks are linked together in a chain according to their respective temporal order. Each server has a copy of this chain saved, so the blockchain is secure if one server in the system is operational (Gamage et al. 2020). As a result, altering the data stored in the blockchain is significantly challenging. Blockchain differs from traditional networks in two keyways: first, it is decentralized, and second, data is hard to tamper with. Based on these two features, the data stored by the blockchain is more trustworthy and legitimate, which can assist in resolving the issue of mutual mistrust (Kolb et al. 2020).

Our goal in this project is to create a system that supports the user stories we created in sprint 0. Users should first be able to register for the system. The system should, secondly, allow users to log in. The third need is that users should be able to send and receive files or messages. Fourth, users should have the ability to both encrypt and decode messages and files. Users should be able to use their keys to validate the signed versions. To safeguard user key pairs, we need integrate the system with a blockchain technology.

All the features that our client requested have been implemented. In order to deliver higher quality, we have also created a user-friendly and tidy user interface at our client's request. For a multi-user connection, we have additionally linked our system to the MongoDB. Finally, as a crucial component of the blockchain system, we have protected users' public keys. As a result, the project's final product completely met the basic aims that we had established.

1 INTRODUCTION

A blockchain is designed to store data in an open distributed database. The basic concept of blockchain is that all data is distributed across the entire network of computers, and the blockchain is completely decentralised. Transactions or records are processed not by one central administrator but by a network of users who work to verify the data and achieve a consensus (Bernard 2021). Making it hard to alter, hack, or cheat crucial data is the key motivation behind using blockchain. In this report, we will go over how to harness the advantages of the blockchain to construct a system that allows users to send and receive messages and files securely.

The team project intends to create a system that can facilitate safe user-to-user file and message transfers. We should deploy blockchain technology to provide secure transactions, such as sending and receiving encrypted or decrypted messages and files. We stored all users' public keys on a blockchain and periodically retrieved them to ensure the trustworthiness of all transactions. It is an excellent approach to store the distributed data on a blockchain because all the public keys are disseminated to other users. Users, on the other hand, saved their own private keys on local machines because they were personal information.

The project's five core implementation features are: Download keypairs, Send messages, Check messages, Send files, and Check files. Users should first be able to download their own key pairs if they misplace them. Using key pairs is mostly used for encryption, decryption, and signature. Second, users must be able to communicate with one another via the system. Different keys would be needed depending on the message type that the sender selects. Thirdly, users ought to have access to a message log. Depending on the nature of the message, different key pairs may once again be needed when the receivers check messages. The system should allow users to transfer files to other users, which brings us to the final product. Again, a separate key would be required depending on the file type. Finally, users should be able to inspect the files they have received. Depending on the type of files taken, the receivers can decode and validate using a different key.

A database was additionally required to store the data and files that users created. In the beginning, we used an SQLite database, but subsequently, in order to enable multiple user connections, we switched to MongoDB. It was decided to use MongoDB because it is free to use and store data in a non-structural type.

Due to our inexperience with blockchain programming at the beginning, one of the difficulties was that we had to spend a lot of time learning the technology and knowledge associated with blockchain. However, we were able to provide the groundwork for the Solidity and MetaMask environments. The other difficulty was that during the project, our customer kept pushing for additional and challenging needs, including file transactions when we had already developed message transactions in the same way. Even though the idea behind encrypting, decrypting, and signing files is the same as it is for communications, files are frequently significantly larger. Since they frequently have more bits than 4049 when converted to binary, there are several challenges in handling this problem.

Both direct internal and indirect external stakeholders are considered key stakeholders. Employees on the project are internal stakeholders. The project team's staff and developers carry out the project in accordance with the specifications and draw in additional user groups thanks to the innovative functional design and development, emphasising the platform's competitive edge that is based on blockchain technology. Customers and organisations are external stakeholders. The project's success depends on customer input and engagement, which also has an indirect impact on the platform application organization's financial health. The institutions' implementation of the project also has an indirect impact on their long-term development and future strategies.

2 REQUIREMENTS

USE CASE DIAGRAM or USE CASES

Nowadays, most of the chat software on the market includes encryption capability. Encrypted messaging programmers can provide greater security, privacy, and functionality than plain text messaging services, even though their encryption technologies and data-gathering procedures are distinct. Similarly, WhatsApp and iMessage by Apple Inc. offer chat encryption, but within their unique ecosystems (Nguyen 2021). We believe that students could be vulnerable to cyber-attacks when they communicate personal information to their instructors and university service staff. In order to preserve our privacy, we began developing this application.

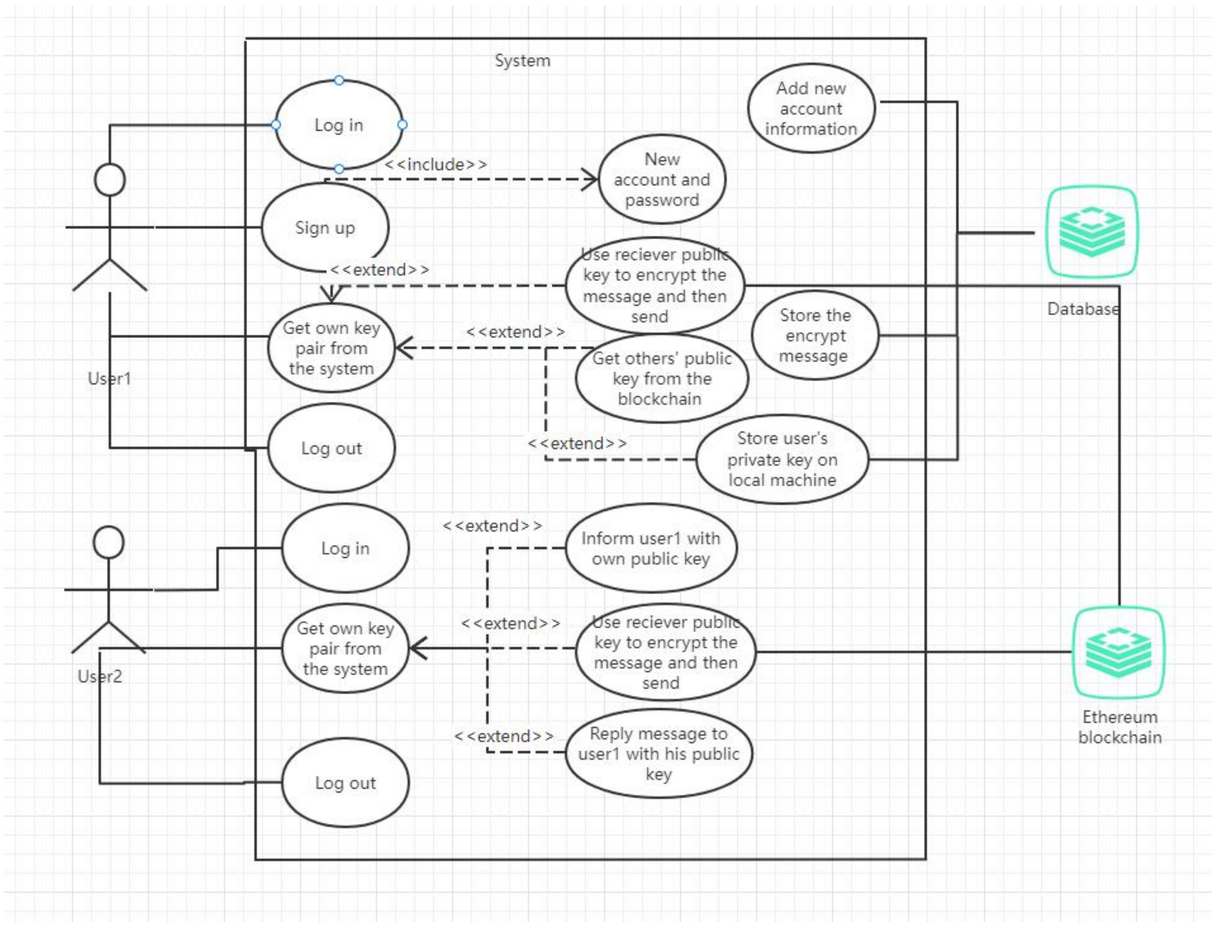

System
Log in
<<include>>
New account and password
Add new account information
Sign up
<<extend>>
Use reciever public key to encrypt the message and then send
Database
Get own key pair from the system
<<extend>>
Store the encrypt message
Get others' public key from the blockchain
User1
<<extend>>
Store user's private key on local machine
Log out
Log in
<<extend>>
Inform user1 with own public key
Get own key pair from the system
<<extend>>
Use reciever public key to encrypt the message and then send
Ethereum blockchain
User2
<<extend>>
Reply message to user1 with his public key
Log out

Figure 1. USE CASE DIAGRAM

The main operations of our system are depicted in the diagram. Users can first create a new account that has not already been added to the database and log in using their unique username and password. User 1 can sign up and log in using his or her credentials, for instance. The public and private keys that User 1 needs for other purposes are given to them. User 1 can obtain the recipient's public key from the blockchain and encrypt the message if User 1 wants to connect with other users using encrypted data. User 1 can also use their private key to sign and send their own messages, allowing the recipients to confirm the authenticity using the sender's public key. Additionally, User 1 can use the logout feature to change between accounts.

As the recipient, User 2 can log in, check his public key and private key, and use his or her own private key to decrypt the encrypted communication that User 1 created. User 2 may also utilize User 1's public key from the blockchain to encrypt and transmit the message if User 2 wants to respond to the message. We anticipate that many pupils will be able to carry out this procedure simultaneously on various machines thanks to cloud database technologies. The MongoDB is capable of storing data in binary format, including encrypted and signed information, in addition to user account and key information.

FUNCTIONAL REQUIREMENTS SPECIFICATION

Req. ID	Functional Requirement Name	Description	Priority
1	Generation of a pair of RSA key	Users can download their own RSA key pair for encryption, signing, decryption, and validation	High
2	Encrypting and signing	Users can encrypt or sign their messages as a sender, and then send the messages to receivers	High
3	Decrypting and validating	The users, as the receivers, can decrypt the encrypted data with their own private key, and validate the signed data with the sender's public key from the blockchain	High

Figure 2. Functional requirements specification

NON-FUNCTIONAL REQUIREMENTS SPECIFICATION

Req. ID	Functional Requirement Name	Description	Priority
1	Multi-users	The system should allow multi users to log in and share information in various places at the same time.	High
2	Realtime	The system should allow the sender's message to be stored in the database immediately and the receiver can receive the message at the same time. A cloud-based database must be used.	High

Figure 3. Non-functional requirements specification

3 ARCHITECTURE

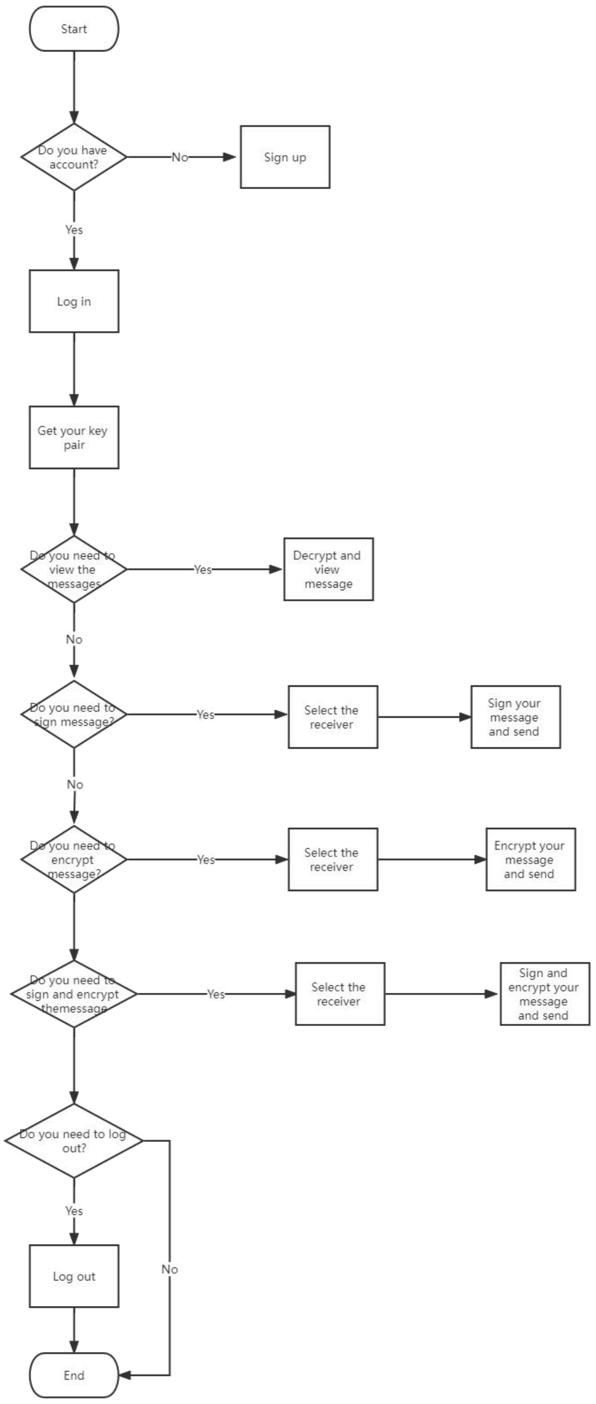

Start
Do you have account?
No
Sign up
Yes
Log in
Get your key pair
Do you need to view the messages
Yes
Decrypt and view message
No
Do you need to sign message?
Yes
Select the receiver
Sign your message and send
No
Do you need to encrypt message?
Yes
Select the receiver
Encrypt your message and send
Do you need to sign and encrypt themessage
Yes
Select the receiver
Sign and encrypt your message and send
Do you need to log out?
Yes
Log out
No
End

Figure 4. Architecture diagram

According to the demands of our customer, as illustrated in **Figure 4**, the flowchart displays every potential function that users may carry out. Users can use all five features once they have joined up and logged in to the system until they log out.

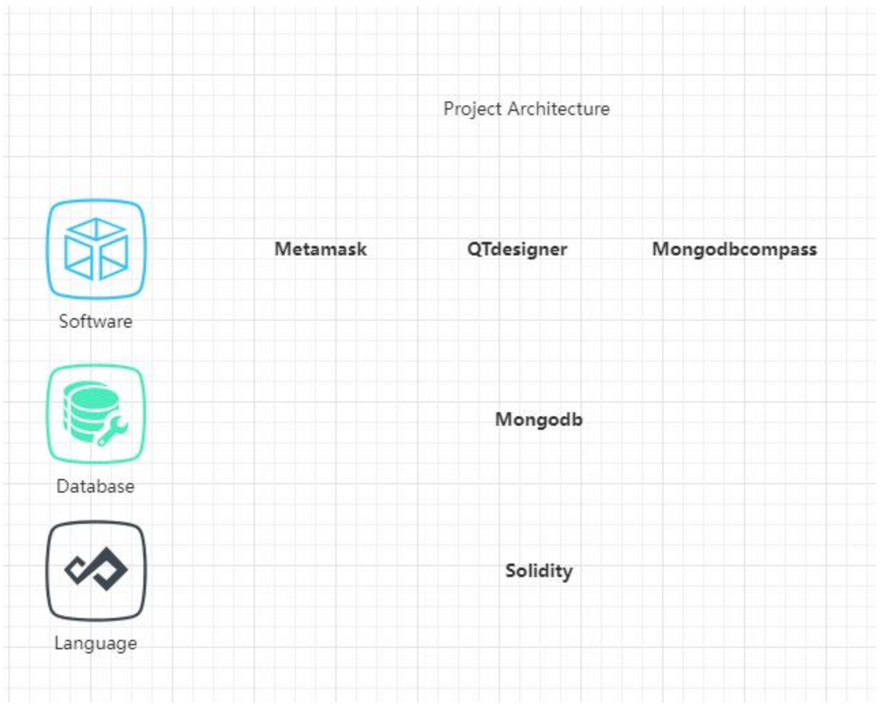


Figure 5. Project architecture

The architecture in the **Figure 5** illustrates the technologies utilized for the project.

4 TECHNICAL FRAMEWORK

5.1 Python: As you can see **in Figure 5** above, the main language of the project is Python. The first reason we chose the language is that everyone is quite confident in using the language to implement a program. The second reason is that few team members are familiar with React and JavaScript, so we would better choose the language for both the backend and frontend.

5.2 Visual Studio code: The finest interoperability between Visual Studio code and Python makes it the default IDE. Because Solidity is compatible with Visual Studio code as well as other languages, it was the best choice for us.

5.3 QTdesigner: QTdesigner is used for developing frontend user interfaces as it is the perfect software for users had no experience in design. We took up the QTdesigner while deciding QTdesigner and Tkinter because, as we mentioned before, it is such a useful tool for beginners.

5.4 MongoDB: In terms of databases, SQLite is used as a local database to test the key features in the beginning. However, we eventually decided to use the MongoDB for the final deliverable because of the limitation of SQLite. Unfortunately, SQLite does not support multi-user connections simultaneously, so the function for checking messages and files in real-time cannot be implemented with it.

5.5 Ethereum: In response to the client's requirement, we are required to develop an online decentralised blockchain application. Ether is used in a decentralised blockchain system. There are numerous nodes connected to one another that make up the Ethernet network. In each full Ether node, the entire blockchain's data is kept. Additionally, we save users' public key on each block of blockchain.

5.6 Solidity: Solidity is a language for creating contracts, or business logic and application code, the Contract, which is subsequently translated into Ethereum bytecode and deployed on the blockchain.

5.7 Brownie: Python-based Brownie is a relatively easy-to-use framework that can be used to deploy and handle smart contracts without a lot of the headaches that come with JavaScript.

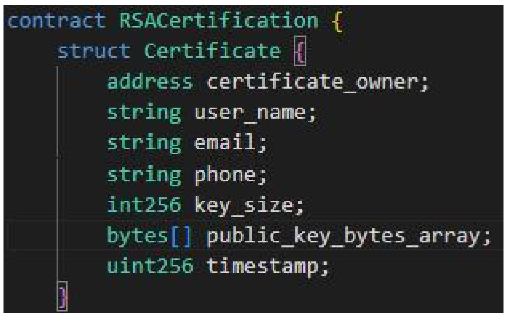
5.8 MetaMask: An open source Ethereum wallet called MetaMask makes it simple for users to handle their digital assets. It can also be used as a browser plug-in wallet; no client downloads or installation is required; simply add it to the browser extension.

5.9 Ganache: To test our smart contracts, Ganache provides a local blockchain environment. Everything we accomplish on this small blockchain stays on the computer.

5 IMPLEMENTATION

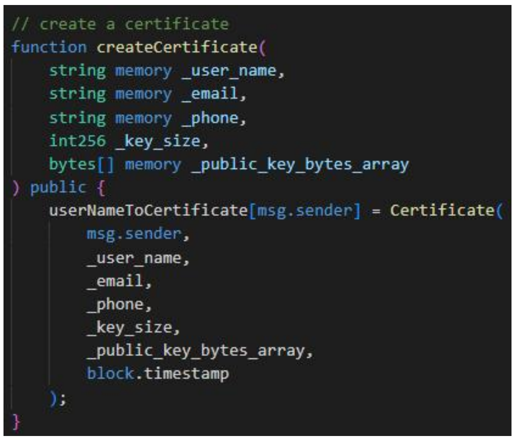
6.1 RSA certificate

Solidity is utilised to implement the RSA certificate, which is used as a smart contract. The certificate contains a user's public key, username, and address of the certificate owner, but is broken up into 9 parts due to binary format's size restriction and the time it was created. The certificate provides getter methods for the username, public key, and time in addition to the function Object() { [native code] } (the constructor function).



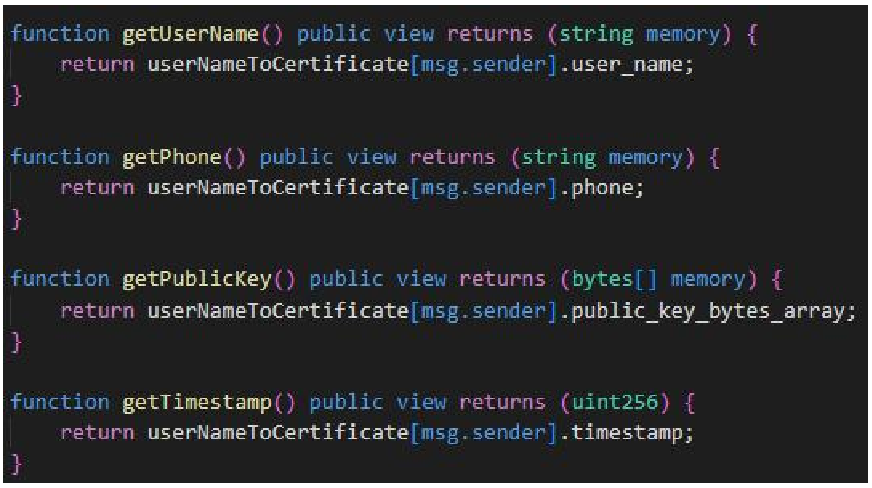
```
contract RSACertification {
    struct Certificate {
        address certificate_owner;
        string user_name;
        string email;
        string phone;
        int256 key_size;
        bytes[] public_key_bytes_array;
        uint256 timestamp;
    }
```

Figure 6. Struct of RSA certificate



```
// create a certificate
function createCertificate(
    string memory _user_name,
    string memory _email,
    string memory _phone,
    int256 _key_size,
    bytes[] memory _public_key_bytes_array
) public {
    userNameToCertificate[msg.sender] = Certificate(
        msg.sender,
        _user_name,
        _email,
        _phone,
        _key_size,
        _public_key_bytes_array,
        block.timestamp
    );
}
```

Figure 7. Constructor of RSA certificate

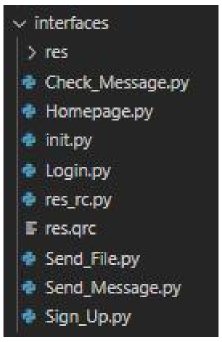


```
function getUserName() public view returns (string memory) {
    return userNameToCertificate[msg.sender].user_name;
}

function getPhone() public view returns (string memory) {
    return userNameToCertificate[msg.sender].phone;
}

function getPublicKey() public view returns (bytes[] memory) {
    return userNameToCertificate[msg.sender].public_key_bytes_array;
}

function getTimestamp() public view returns (uint256) {
    return userNameToCertificate[msg.sender].timestamp;
}
```

Figure 8. Getters of RSA certificate

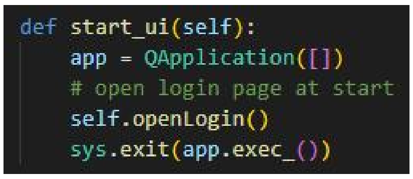
6.2 Frontend UI

All fronted UI files are made by QTdesigner. *.ui files are used as a starting point because once we design a page with QTdesigner, it converts the ui format to Python file, then we can use the Python file. We implemented a Sign up page, Login page, Main page, Message board page, Send a message page, File board page, Send a file page. Key pairs page was used for some sprints, but it ended up not using as we let the system download a key pair to the local machine straightaway. Each page has own buttons and labels, so they have own event handler to proceed a different function based on the user's requirements.



```
interfaces
  > res
  Check_Message.py
  Homepage.py
  init.py
  Login.py
  res_rc.py
  res.qrc
  Send_File.py
  Send_Message.py
  Sign_Up.py
```

Figure 9. List of frontend UI files

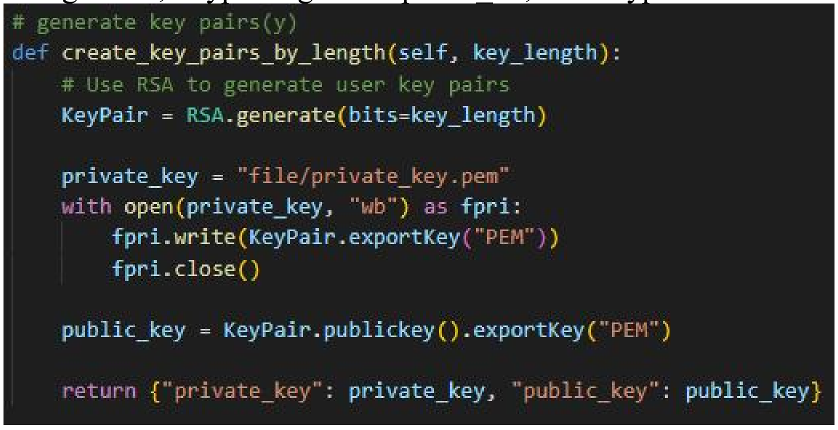


```
def start_ui(self):
    app = QApplication([])
    # open login page at start
    self.openLogin()
    sys.exit(app.exec_())
```

Figure 10. Main method calling widget

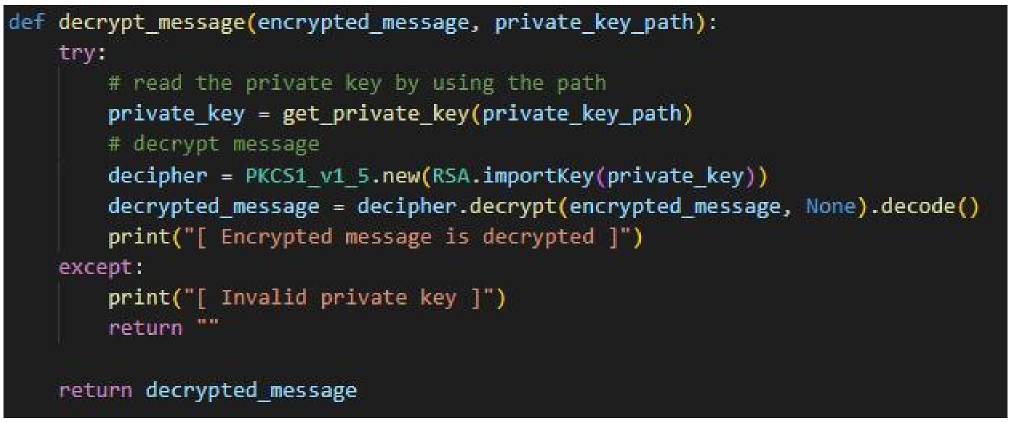
6.3 Crypto

As a core feature of the project Crypto.py has been created for implementation of encryption, decryption, signature, and validation. As the created messages and files are saved as a binary file in the MongoDB, we implemented functions in terms of crypto such as encrypting a binary format of message. As our client's requirement we used OpenSSL API. For instance, Crypto.PublicKey, Crypto.Cipher, Crypto.Signature, Crypto.Signature.pkcs1_15, and Crypto.Hash.



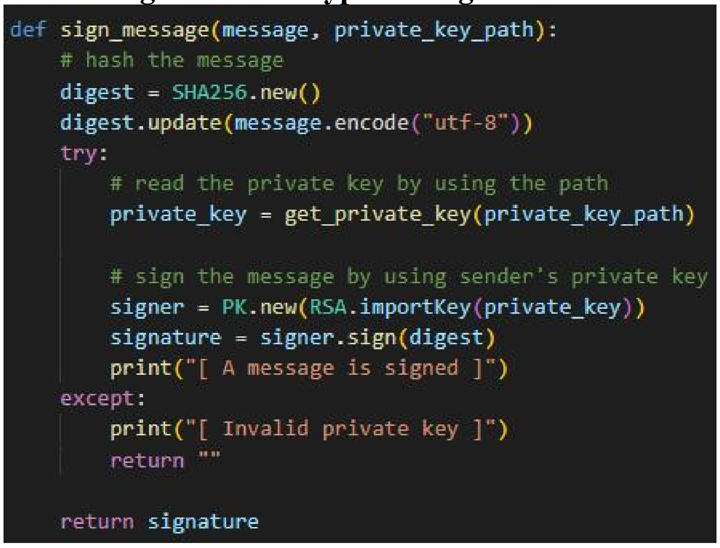
```
# generate key pairs(y)
def create_key_pairs_by_length(self, key_length):
    # Use RSA to generate user key pairs
    KeyPair = RSA.generate(bits=key_length)

    private_key = "file/private_key.pem"
    with open(private_key, "wb") as fpri:
        fpri.write(KeyPair.exportKey("PEM"))
        fpri.close()

    public_key = KeyPair.publickey().exportKey("PEM")

    return {"private_key": private_key, "public_key": public_key}
```

Figure 11. Encrypt message function



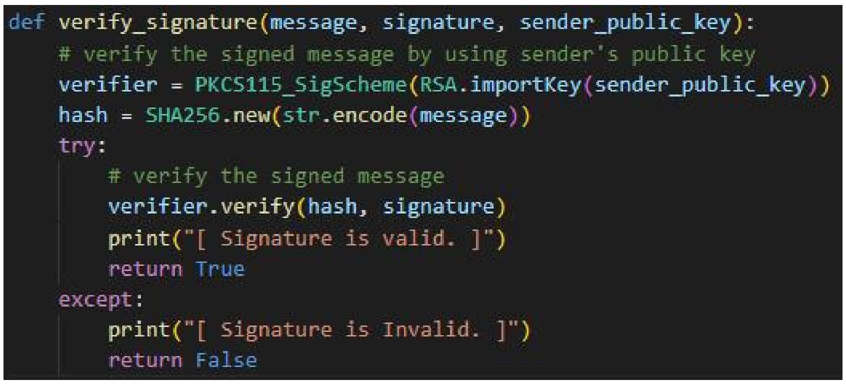
```python
def decrypt_message(encrypted_message, private_key_path):
    try:
        # read the private key by using the path
        private_key = get_private_key(private_key_path)
        # decrypt message
        decipher = PKCS1_v1_5.new(RSA.importKey(private_key))
        decrypted_message = decipher.decrypt(encrypted_message, None).decode()
        print("[ Encrypted message is decrypted ]")
    except:
        print("[ Invalid private key ]")
        return ""

    return decrypted_message
```

Figure 12. Decrypt message function



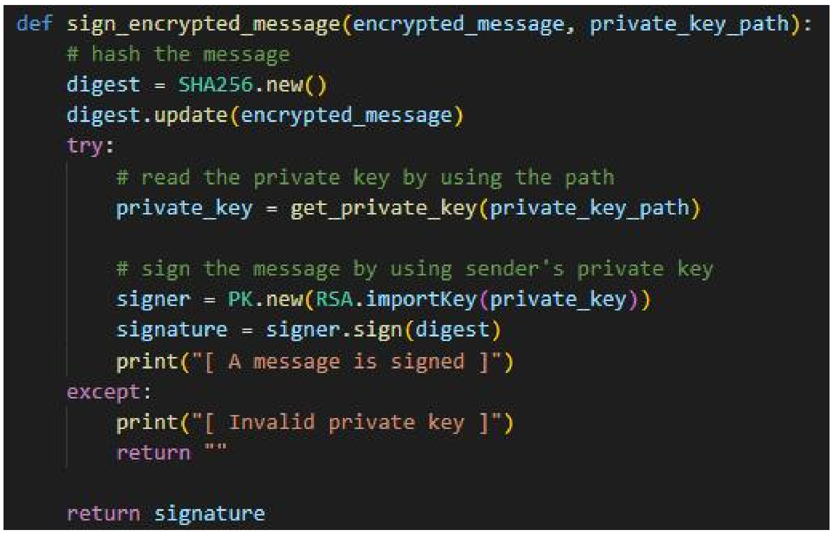
```python
def sign_message(message, private_key_path):
    # hash the message
    digest = SHA256.new()
    digest.update(message.encode("utf-8"))
    try:
        # read the private key by using the path
        private_key = get_private_key(private_key_path)

        # sign the message by using sender's private key
        signer = PK.new(RSA.importKey(private_key))
        signature = signer.sign(digest)
        print("[ A message is signed ]")
    except:
        print("[ Invalid private key ]")
        return ""

    return signature
```

Figure 13. Sign message function



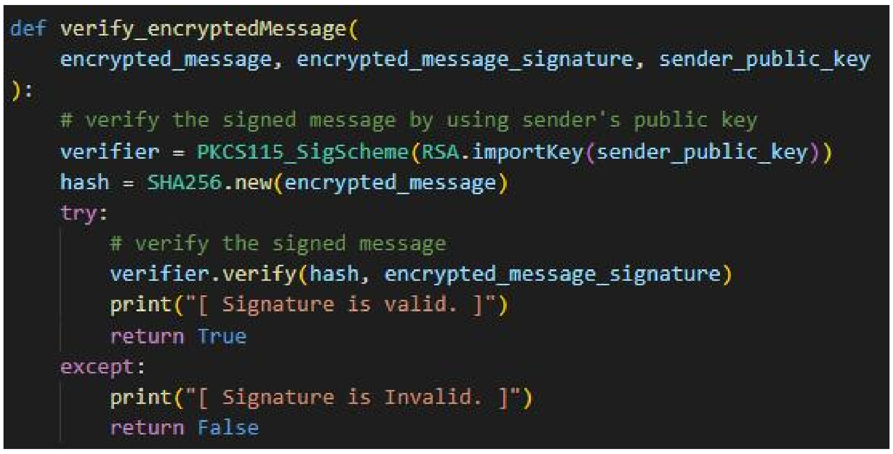
```python
def verify_signature(message, signature, sender_public_key):
    # verify the signed message by using sender's public key
    verifier = PKCS115_SigScheme(RSA.importKey(sender_public_key))
    hash = SHA256.new(str.encode(message))
    try:
        # verify the signed message
        verifier.verify(hash, signature)
        print("[ Signature is valid. ]")
        return True
    except:
        print("[ Signature is Invalid. ]")
        return False
```

Figure 14. Verify message function



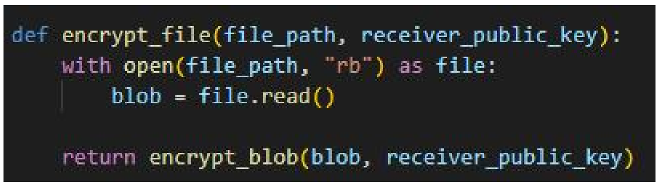
```python
def sign_encrypted_message(encrypted_message, private_key_path):
    # hash the message
    digest = SHA256.new()
    digest.update(encrypted_message)
    try:
        # read the private key by using the path
        private_key = get_private_key(private_key_path)

        # sign the message by using sender's private key
        signer = PK.new(RSA.importKey(private_key))
        signature = signer.sign(digest)
        print("[ A message is signed ]")
    except:
        print("[ Invalid private key ]")
        return ""

    return signature
```

Figure 15. Sign encrypted message function



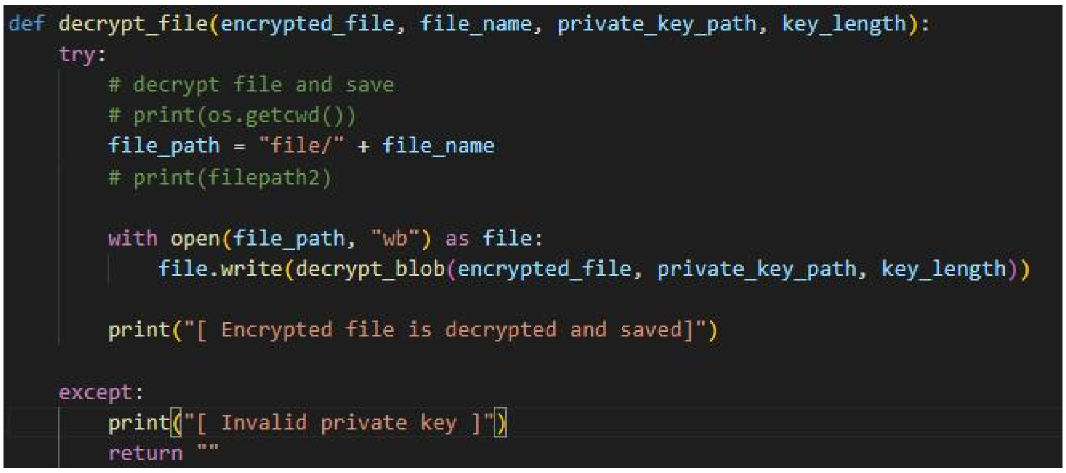
```python
def verify_encryptedMessage(
    encrypted_message, encrypted_message_signature, sender_public_key
):
    # verify the signed message by using sender's public key
    verifier = PKCS115_SigScheme(RSA.importKey(sender_public_key))
    hash = SHA256.new(encrypted_message)
    try:
        # verify the signed message
        verifier.verify(hash, encrypted_message_signature)
        print("[ Signature is valid. ]")
        return True
    except:
        print("[ Signature is Invalid. ]")
        return False
```

Figure 16. Verify encrypted message function



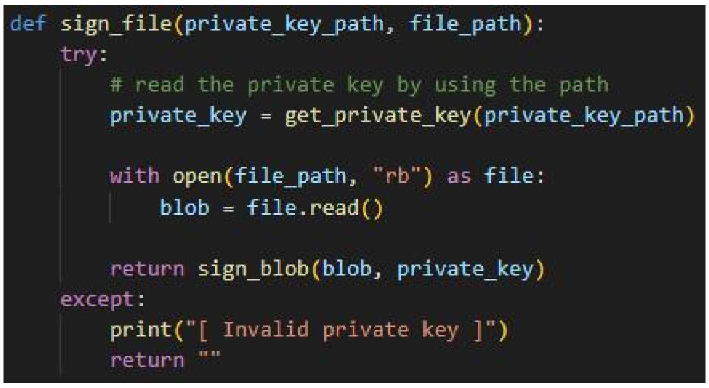
```python
def encrypt_file(file_path, receiver_public_key):
    with open(file_path, "rb") as file:
        blob = file.read()

    return encrypt_blob(blob, receiver_public_key)
```

Figure 17. Encrypt file function



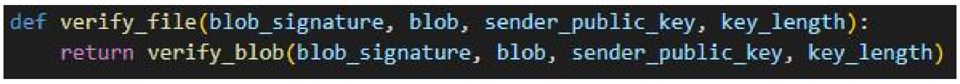
```python
def decrypt_file(encrypted_file, file_name, private_key_path, key_length):
    try:
        # decrypt file and save
        # print(os.getcwd())
        file_path = "file/" + file_name
        # print(filepath2)

        with open(file_path, "wb") as file:
            file.write(decrypt_blob(encrypted_file, private_key_path, key_length))

        print("[ Encrypted file is decrypted and saved]")

    except:
        print("[ Invalid private key ]")
        return ""
```

Figure 18. Decrypt file function



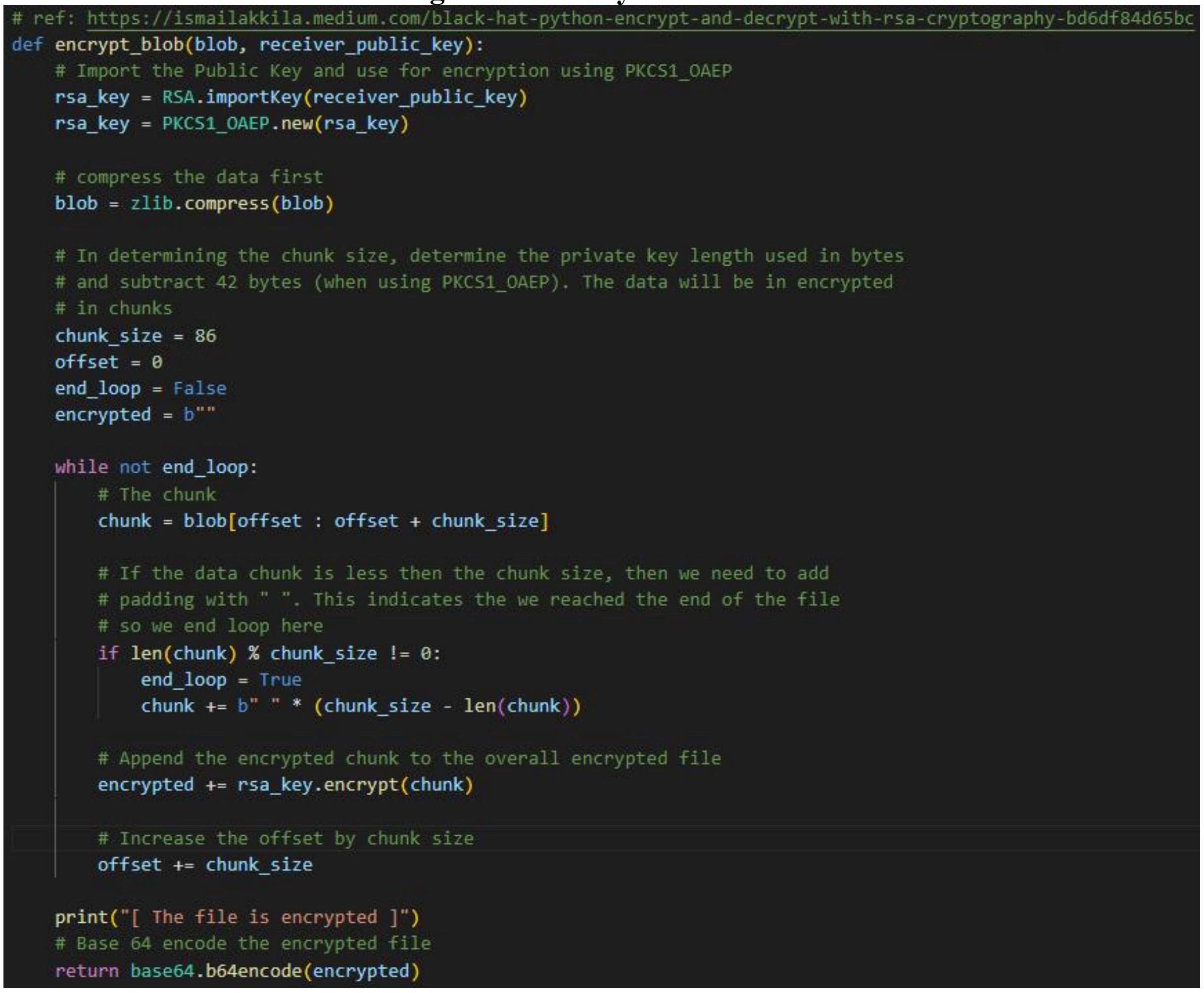
```python
def sign_file(private_key_path, file_path):
    try:
        # read the private key by using the path
        private_key = get_private_key(private_key_path)

        with open(file_path, "rb") as file:
            blob = file.read()

        return sign_blob(blob, private_key)
    except:
        print("[ Invalid private key ]")
        return ""
```

Figure 19. Sign file function



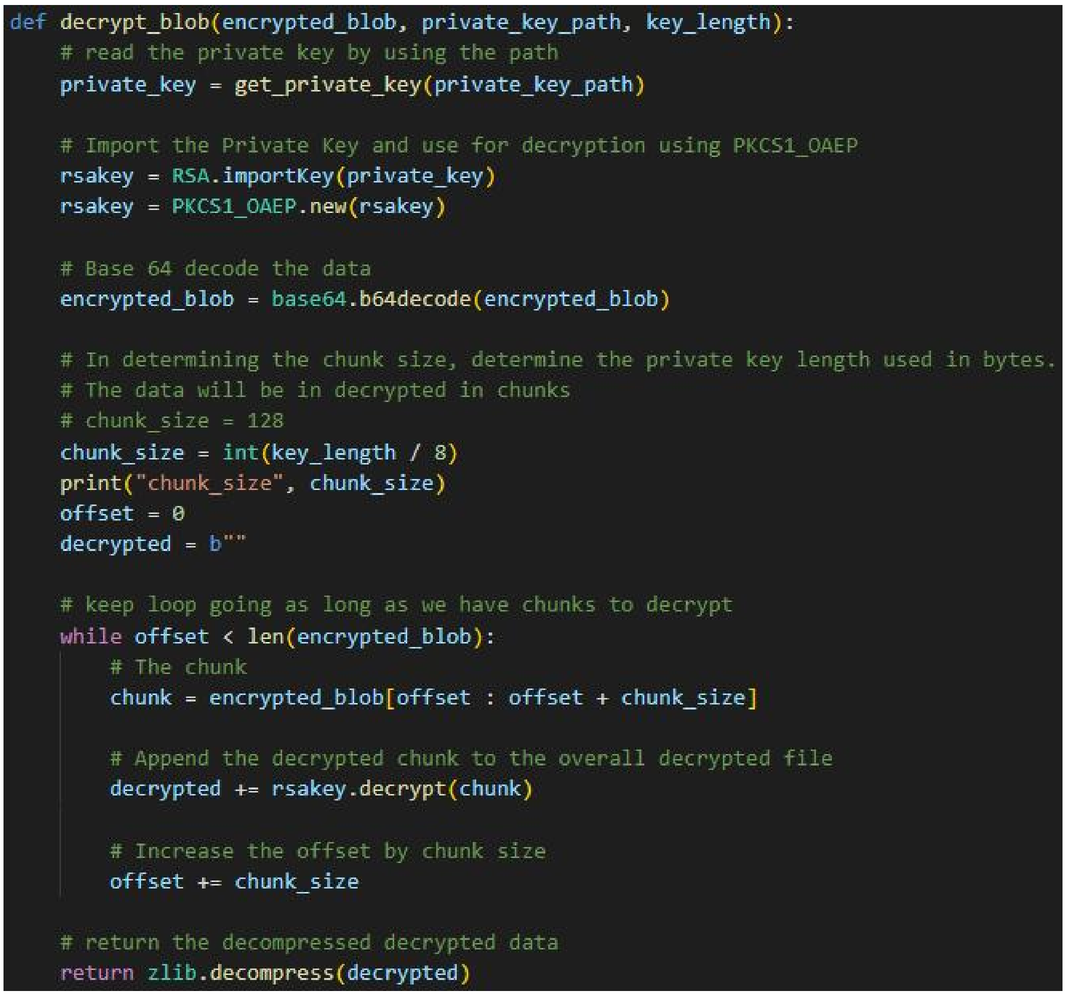
```python
def verify_file(blob_signature, blob, sender_public_key, key_length):
    return verify_blob(blob_signature, blob, sender_public_key, key_length)
```

Figure 20. Verify file function



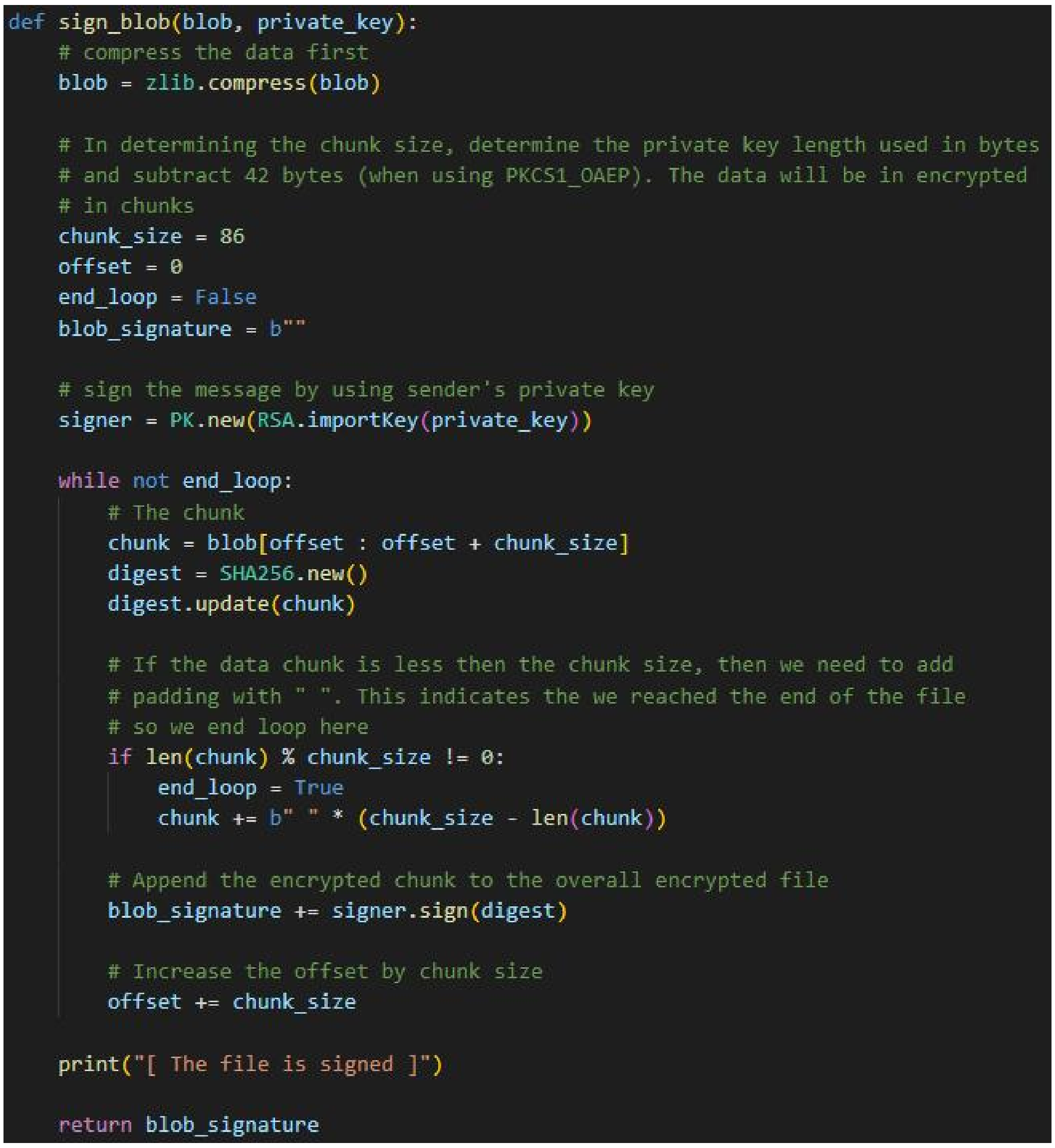
```python
# ref: https://ismailakkila.medium.com/black-hat-python-encrypt-and-decrypt-with-rsa-cryptography-bd6df84d65bc
def encrypt_blob(blob, receiver_public_key):
    # Import the Public Key and use for encryption using PKCS1_OAEP
    rsa_key = RSA.importKey(receiver_public_key)
    rsa_key = PKCS1_OAEP.new(rsa_key)

    # compress the data first
    blob = zlib.compress(blob)

    # In determining the chunk size, determine the private key length used in bytes
    # and subtract 42 bytes (when using PKCS1_OAEP). The data will be in encrypted
    # in chunks
    chunk_size = 86
    offset = 0
    end_loop = False
    encrypted = b""

    while not end_loop:
        # The chunk
        chunk = blob[offset : offset + chunk_size]

        # If the data chunk is less then the chunk size, then we need to add
        # padding with " ". This indicates the we reached the end of the file
        # so we end loop here
        if len(chunk) % chunk_size != 0:
            end_loop = True
            chunk += b" " * (chunk_size - len(chunk))

        # Append the encrypted chunk to the overall encrypted file
        encrypted += rsa_key.encrypt(chunk)

        # Increase the offset by chunk size
        offset += chunk_size

    print("[ The file is encrypted ]")
    # Base 64 encode the encrypted file
    return base64.b64encode(encrypted)
```

Figure 21. Encrypt BLOB function



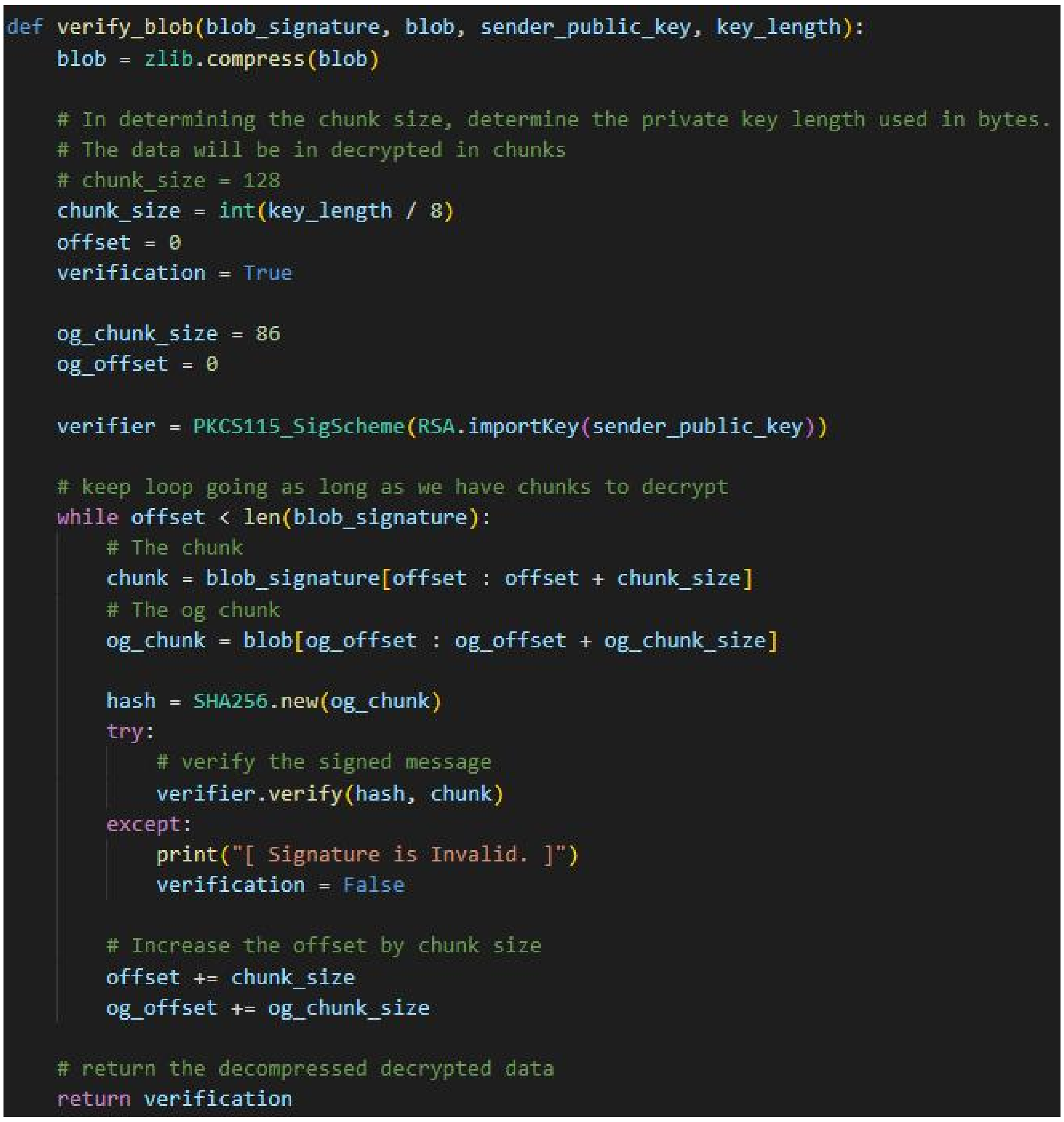
```python
def decrypt_blob(encrypted_blob, private_key_path, key_length):
    # read the private key by using the path
    private_key = get_private_key(private_key_path)

    # Import the Private Key and use for decryption using PKCS1_OAEP
    rsakey = RSA.importKey(private_key)
    rsakey = PKCS1_OAEP.new(rsakey)

    # Base 64 decode the data
    encrypted_blob = base64.b64decode(encrypted_blob)

    # In determining the chunk size, determine the private key length used in bytes.
    # The data will be in decrypted in chunks
    # chunk_size = 128
    chunk_size = int(key_length / 8)
    print("chunk_size", chunk_size)
    offset = 0
    decrypted = b""

    # keep loop going as long as we have chunks to decrypt
    while offset < len(encrypted_blob):
        # The chunk
        chunk = encrypted_blob[offset : offset + chunk_size]

        # Append the decrypted chunk to the overall decrypted file
        decrypted += rsakey.decrypt(chunk)

        # Increase the offset by chunk size
        offset += chunk_size

    # return the decompressed decrypted data
    return zlib.decompress(decrypted)
```

Figure 22. Decrypt BLOB function



```
def sign_blob(blob, private_key):
    # compress the data first
    blob = zlib.compress(blob)

    # In determining the chunk size, determine the private key length used in bytes
    # and subtract 42 bytes (when using PKCS1_OAEP). The data will be in encrypted
    # in chunks
    chunk_size = 86
    offset = 0
    end_loop = False
    blob_signature = b""

    # sign the message by using sender's private key
    signer = PK.new(RSA.importKey(private_key))

    while not end_loop:
        # The chunk
        chunk = blob[offset : offset + chunk_size]
        digest = SHA256.new()
        digest.update(chunk)

        # If the data chunk is less then the chunk size, then we need to add
        # padding with " ". This indicates the we reached the end of the file
        # so we end loop here
        if len(chunk) % chunk_size != 0:
            end_loop = True
            chunk += b" " * (chunk_size - len(chunk))

        # Append the encrypted chunk to the overall encrypted file
        blob_signature += signer.sign(digest)

        # Increase the offset by chunk size
        offset += chunk_size

    print("[ The file is signed ]")

    return blob_signature
```

Figure 23. Sign BLOB function



```python
def verify_blob(blob_signature, blob, sender_public_key, key_length):
    blob = zlib.compress(blob)

    # In determining the chunk size, determine the private key length used in bytes.
    # The data will be in decrypted in chunks
    # chunk_size = 128
    chunk_size = int(key_length / 8)
    offset = 0
    verification = True

    og_chunk_size = 86
    og_offset = 0

    verifier = PKCS115_SigScheme(RSA.importKey(sender_public_key))

    # keep loop going as long as we have chunks to decrypt
    while offset < len(blob_signature):
        # The chunk
        chunk = blob_signature[offset : offset + chunk_size]
        # The og chunk
        og_chunk = blob[og_offset : og_offset + og_chunk_size]

        hash = SHA256.new(og_chunk)
        try:
            # verify the signed message
            verifier.verify(hash, chunk)
        except:
            print("[ Signature is Invalid. ]")
            verification = False

        # Increase the offset by chunk size
        offset += chunk_size
        og_offset += og_chunk_size

    # return the decompressed decrypted data
    return verification
```

Figure 24. Verify BLOB function

6.4 Database

As we mentioned before, we used two different databases. The first database we used is SQLite, SQLite has high compatibility with Python, so no doubt to use it for the project. However, we realised that SQLite does not support multi-user connection we had to adapt a cloud-based database which is the MongoDB. The MongoDB provides multi connections, so we could implement real-time receiving messages and files.

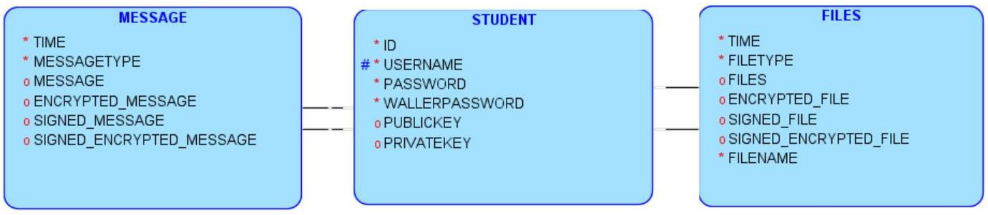
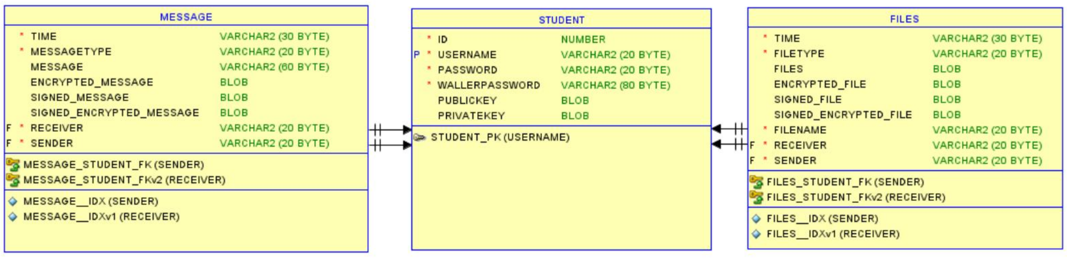
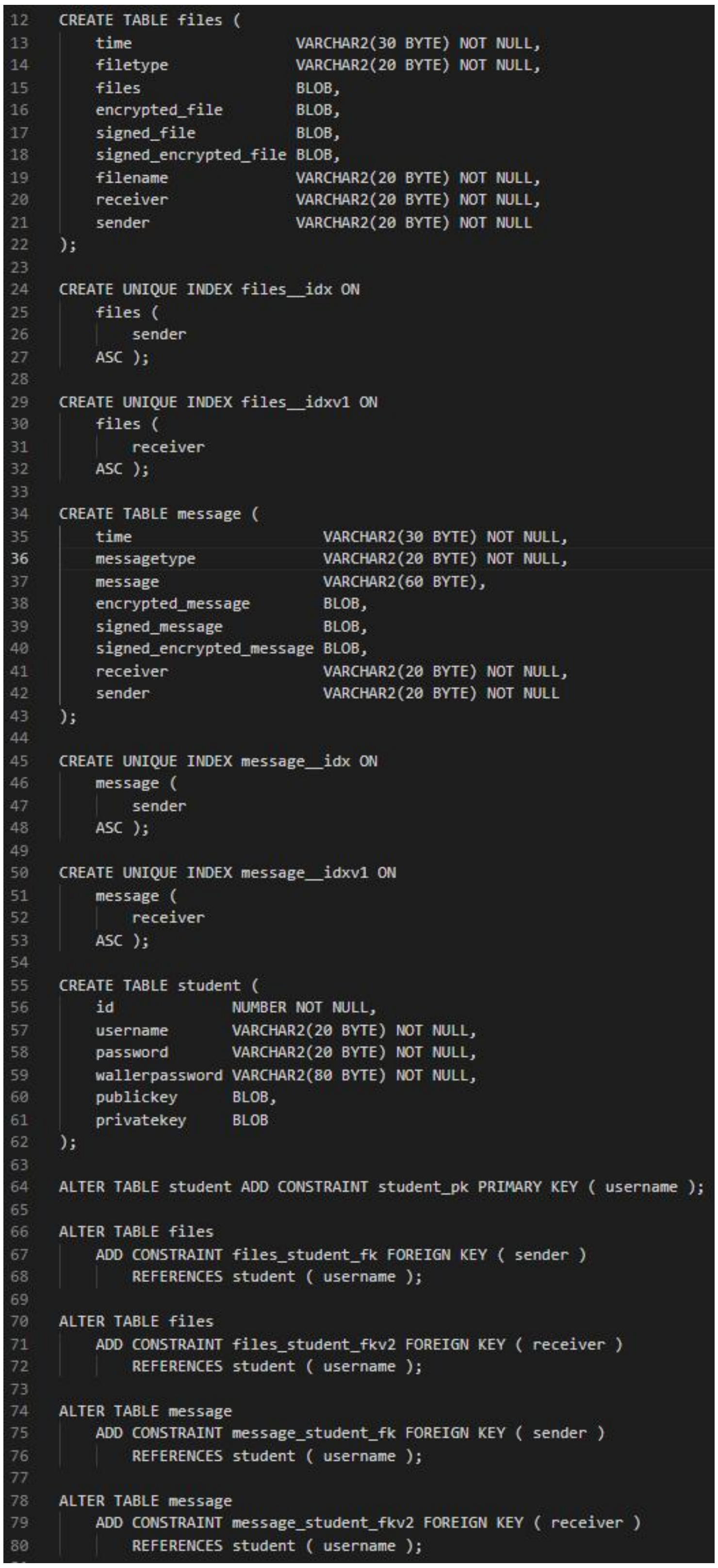


Figure 25. Logical Database Design for Both SQLite


MESSAGE
* TIME VARCHAR2 (30 BYTE)
* MESSAGETYPE VARCHAR2 (20 BYTE)
MESSAGE VARCHAR2 (60 BYTE)
ENCRYPTED_MESSAGE BLOB
SIGNED_MESSAGE BLOB
SIGNED_ENCRYPTED_MESSAGE BLOB
F * RECEIVER VARCHAR2 (20 BYTE)
F * SENDER VARCHAR2 (20 BYTE)
MESSAGE_STUDENT_FK (SENDER)
MESSAGE_STUDENT_FKv2 (RECEIVER)
MESSAGE__IDX (SENDER)
MESSAGE__IDXv1 (RECEIVER)
STUDENT
* ID NUMBER
P * USERNAME VARCHAR2 (20 BYTE)
* PASSWORD VARCHAR2 (20 BYTE)
* WALLERPASSWORD VARCHAR2 (80 BYTE)
PUBLICKEY BLOB
PRIVATEKEY BLOB
STUDENT_PK (USERNAME)
FILES
* TIME VARCHAR2 (30 BYTE)
* FILETYPE VARCHAR2 (20 BYTE)
FILES BLOB
ENCRYPTED_FILE BLOB
SIGNED_FILE BLOB
SIGNED_ENCRYPTED_FILE BLOB
* FILENAME VARCHAR2 (20 BYTE)
F * RECEIVER VARCHAR2 (20 BYTE)
F * SENDER VARCHAR2 (20 BYTE)
FILES_STUDENT_FK (SENDER)
FILES_STUDENT_FKv2 (RECEIVER)
FILES__IDX (SENDER)
FILES__IDXv1 (RECEIVER)
Figure 26. Relational Database Design (ER Diagram)


```
CREATE TABLE files (
    time                  VARCHAR2(30 BYTE) NOT NULL,
    filetype              VARCHAR2(20 BYTE) NOT NULL,
    files                 BLOB,
    encrypted_file        BLOB,
    signed_file           BLOB,
    signed_encrypted_file BLOB,
    filename              VARCHAR2(20 BYTE) NOT NULL,
    receiver              VARCHAR2(20 BYTE) NOT NULL,
    sender                VARCHAR2(20 BYTE) NOT NULL
);

CREATE UNIQUE INDEX files__idx ON
    files (
        sender
    ASC );

CREATE UNIQUE INDEX files__idxv1 ON
    files (
        receiver
    ASC );

CREATE TABLE message (
    time                     VARCHAR2(30 BYTE) NOT NULL,
    messagetype              VARCHAR2(20 BYTE) NOT NULL,
    message                  VARCHAR2(60 BYTE),
    encrypted_message        BLOB,
    signed_message           BLOB,
    signed_encrypted_message BLOB,
    receiver                 VARCHAR2(20 BYTE) NOT NULL,
    sender                   VARCHAR2(20 BYTE) NOT NULL
);

CREATE UNIQUE INDEX message__idx ON
    message (
        sender
    ASC );

CREATE UNIQUE INDEX message__idxv1 ON
    message (
        receiver
    ASC );

CREATE TABLE student (
    id             NUMBER NOT NULL,
    username       VARCHAR2(20 BYTE) NOT NULL,
    password       VARCHAR2(20 BYTE) NOT NULL,
    wallerpassword VARCHAR2(80 BYTE) NOT NULL,
    publickey      BLOB,
    privatekey     BLOB
);

ALTER TABLE student ADD CONSTRAINT student_pk PRIMARY KEY ( username );

ALTER TABLE files
    ADD CONSTRAINT files_student_fk FOREIGN KEY ( sender )
        REFERENCES student ( username );

ALTER TABLE files
    ADD CONSTRAINT files_student_fkv2 FOREIGN KEY ( receiver )
        REFERENCES student ( username );

ALTER TABLE message
    ADD CONSTRAINT message_student_fk FOREIGN KEY ( sender )
        REFERENCES student ( username );

ALTER TABLE message
    ADD CONSTRAINT message_student_fkv2 FOREIGN KEY ( receiver )
        REFERENCES student ( username );
```
Figure 27. DDL Script for SQLite

6.4.1. SQLite
SQLite is a lightweight, ACID-compliant relational database management system.

6.4.1.1. The Reasons for Using SQLite

6.4.1.1.1. Less System Resources Occupation
SQLite is a typical embedded database, one of its most obvious advantages is that it occupies incredibly low system resources, and the system hardware configuration requirements are friendly to most devices.

6.4.1.1.2. Support for a Variety of Mainstream Operating Systems
SQLite can support mainstream operating systems such as Windows, Linux, and Unix.

6.4.1.1.3. Support for a Variety of Programming Languages
SQLite can support a variety of programming languages, including Python used by the project backend and other common programming languages such as Tcl, C#, PHP, and Java.

6.4.1.1.4. Faster Query Speed
In terms of query time consumption, SQLite performs even better than MySQL and PostgreSQL, the two most famous open-source database management systems.

6.4.1.2. Limitations of SQLite

6.4.1.2.1. Additional Work Required to Implement SQLite Cloud
Although it is possible to use SQLite in the cloud, SQLite does not include any server and requires additional work to implement this functionality, such as wrapping a web service. However, the front-end and back-end of the project are implemented based on a web-free Python program and using web content may not be compatible with previous implementations.

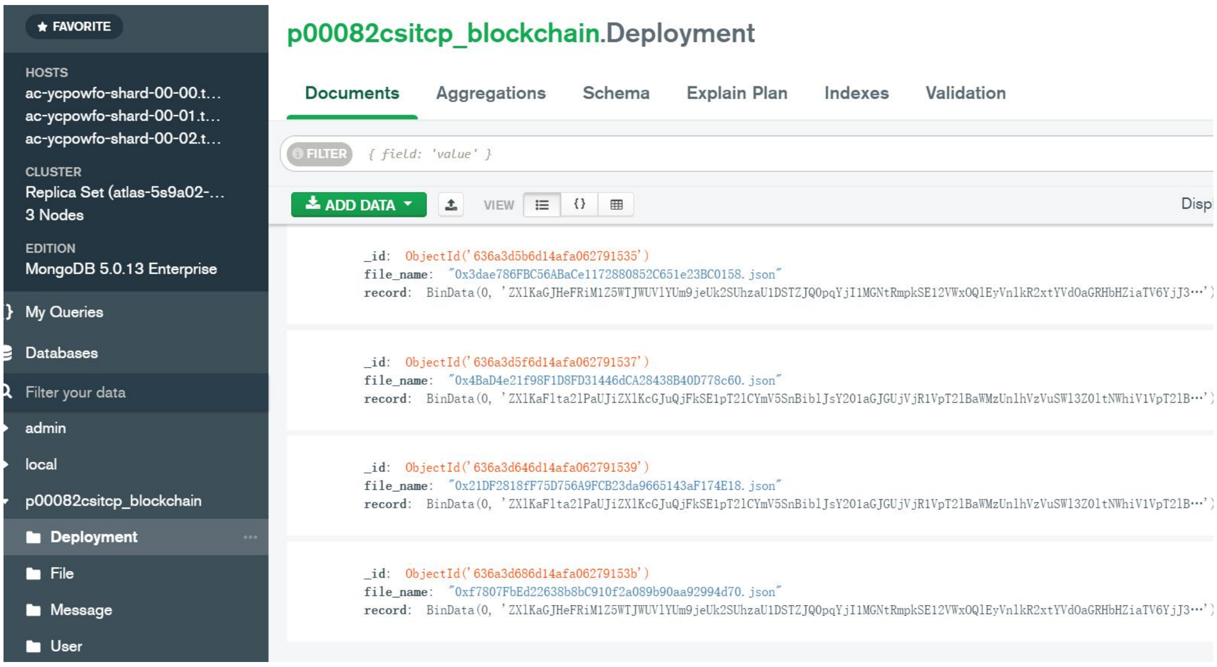
6.4.1.2.2. SQLite Does Not Support Multi-User Functionality
SQLite does not support simultaneous concurrent writes, for which it does not support real multi-user functionality. Although read lock and write lock may help, operations need to be queued when applications are concurrently writing to the database at the same time, resulting in delays or even timeouts.

6.4.2. MongoDB
MongoDB is one of the most popular open source non-relational databases today.

6.4.2.1. The Reasons for Switching to MongoDB
6.4.2.1.1. Available Cloud
MongoDB provides open source and free cloud extensions with excellent stability.

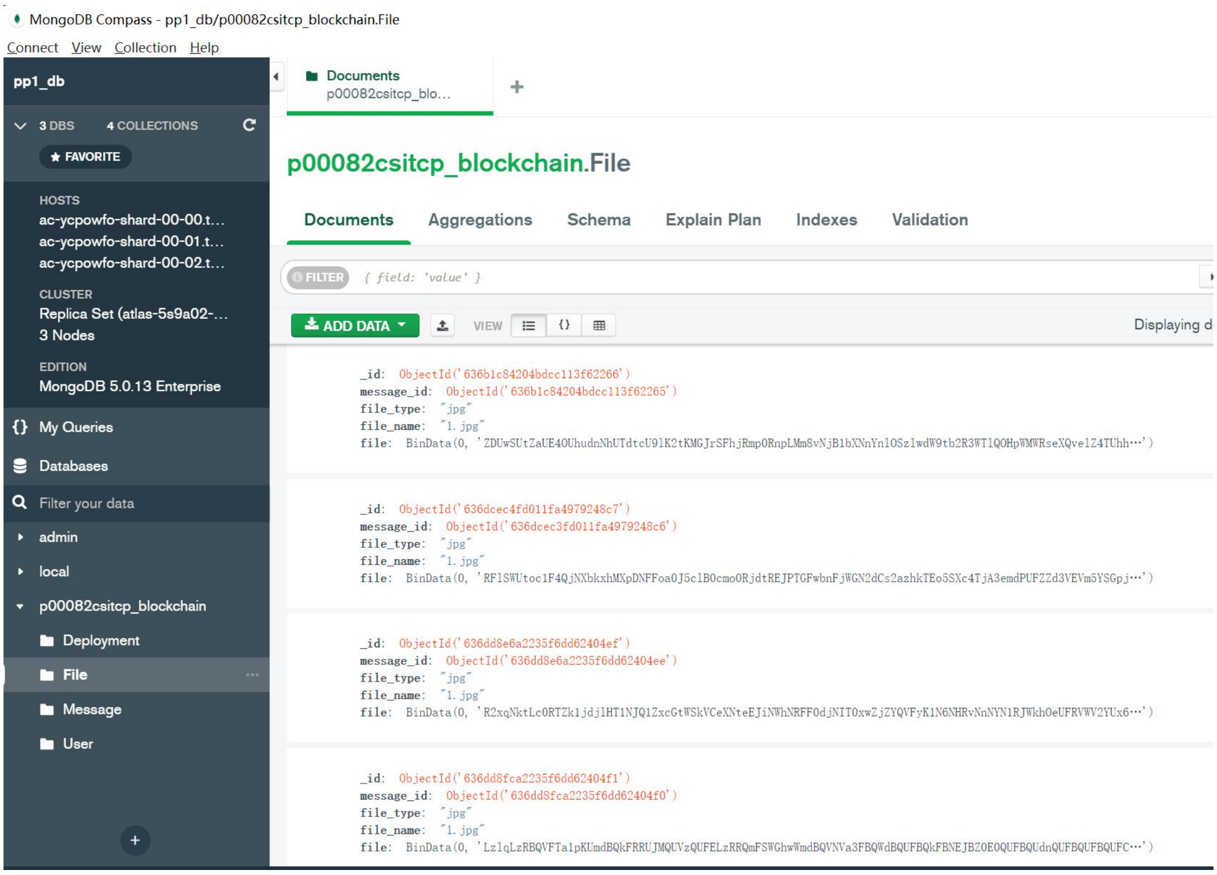
6.4.2.1.2. High Compatibility
MongoDB is highly compatible with most platforms and operating systems.

6.4.2.1.3. High Flexibility
MongoDB is a NoSQL database, which means it does not need to store data in a defined schema. The document-oriented design of MongoDB makes it simple to query and update data in documents.



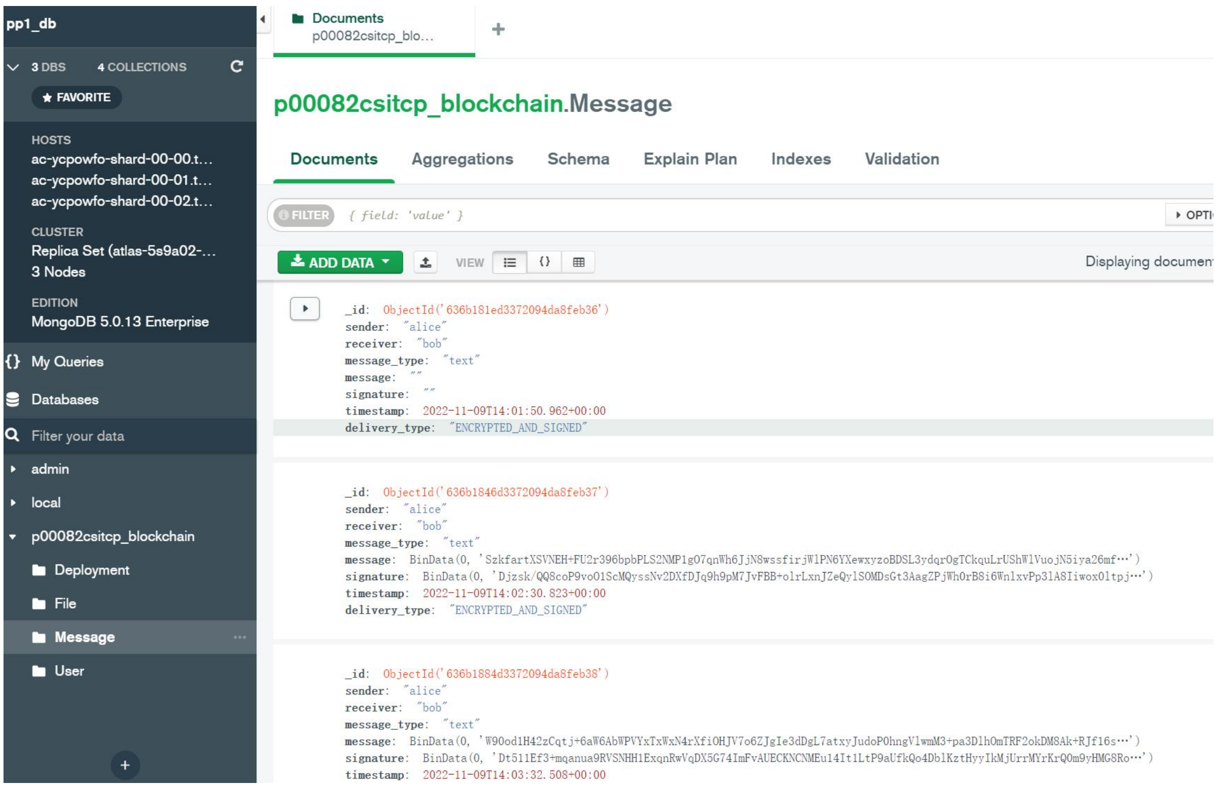
```
_id:  ObjectId('636a3d5b6d14afa062791535')
file_name:  "0x3dae786FBC56ABaCe1172880852C651e23BC0158.json"
record:  BinData(0, 'ZX1KaGJHeFRiM1Z5WTJWUV1YUm9jeUk2SUhzaU1DSTZJQ0pqYjI1MGNtRmpkSE12VWxOQ1EyVnlkR2xtYVd0aGRHbHZiaTV6YjJ3…')

_id:  ObjectId('636a3d5f6d14afa062791537')
file_name:  "0x4BaD4e21f98F1D8FD31446dCA28438B40D778c60.json"
record:  BinData(0, 'ZX1KaF1ta21PaUJiZX1KcGJuQjFkSE1pT21CYmV5SnBib1JsY201aGJGUjVjR1VpT21BaWMzUnlhVzVuSW13Z01tNWhiV1VpT21B…')

_id:  ObjectId('636a3d646d14afa062791539')
file_name:  "0x21DF2818fF75D756A9FCB23da9665143aF174E18.json"
record:  BinData(0, 'ZX1KaF1ta21PaUJiZX1KcGJuQjFkSE1pT21CYmV5SnBib1JsY201aGJGUjVjR1VpT21BaWMzUnlhVzVuSW13Z01tNWhiV1VpT21B…')

_id:  ObjectId('636a3d686d14afa06279153b')
file_name:  "0xf7807FbEd22638b8bC910f2a089b90aa92994d70.json"
record:  BinData(0, 'ZX1KaGJHeFRiM1Z5WTJWUV1YUm9jeUk2SUhzaU1DSTZJQ0pqYjI1MGNtRmpkSE12VWxOQ1EyVnlkR2xtYVd0aGRHbHZiaTV6YjJ3…')
```

Figure 28. MongoDB Deployment Documents



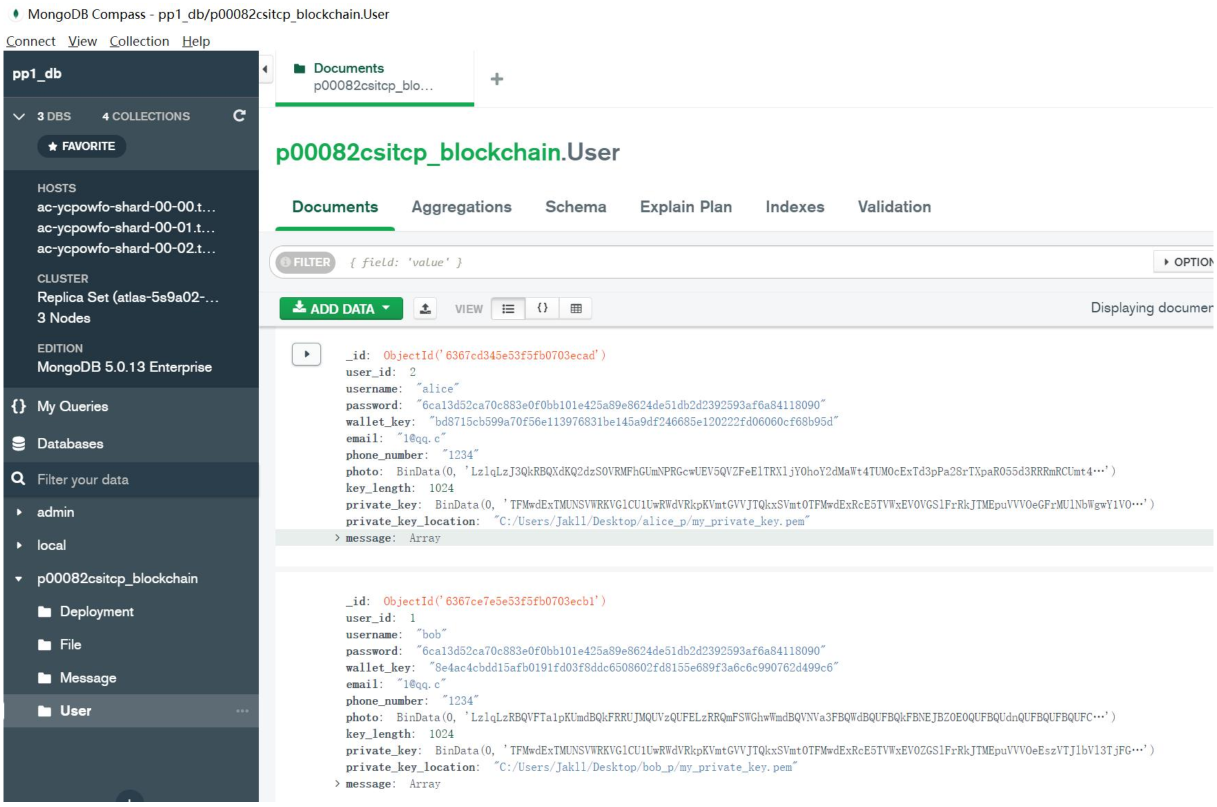
```
_id:  ObjectId('636b1c84204bdcc113f62266')
message_id:  ObjectId('636b1c84204bdcc113f62265')
file_type:  "jpg"
file_name:  "1.jpg"
file:  BinData(0, 'ZDUwSUtZaUE4OUhudnNhUTdtcU91K2tKMGJrSFhjRmp0RnpLMm8vNjB1bXNnYnl0Sz1wdW9tb2R3WT1QOHpWMWRseXQvelZ4TUhh…')

_id:  ObjectId('636dcec4fd011fa4979248c7')
message_id:  ObjectId('636dcec3fd011fa4979248c6')
file_type:  "jpg"
file_name:  "1.jpg"
file:  BinData(0, 'RF1SWUtoc1F4QjNXbkxhMXpDNFFoa0J5c1B0cmo0RjdtREJPTGFwbnFjWGN2dCs2azhkTEo5SXc4TjA3emdPUFZZd3VEVm5YSGpj…')

_id:  ObjectId('636dd8e6a2235f6dd62404ef')
message_id:  ObjectId('636dd8e6a2235f6dd62404ee')
file_type:  "jpg"
file_name:  "1.jpg"
file:  BinData(0, 'R2xqNktLc0RTZk1jdjlHT1NJQ1ZxcGtWSkVCeXNteEJiNWhNRFF0djNIT0xwZjZYQVFyK1N6NHRvNnNYN1RJWkh0eUFRVWV2YUx6…')

_id:  ObjectId('636dd8fca2235f6dd62404f1')
message_id:  ObjectId('636dd8fca2235f6dd62404f0')
file_type:  "jpg"
file_name:  "1.jpg"
file:  BinData(0, 'LzlqLzRBQVFTa1pKUmdBQkFRRUJMQUVzQUFELzRRQmFSWGhwWmdBQVNVa3FBQWdBQUFBQkFBNEJBZ0E0QUFBQUdnQUFBQUFBQUFC…')
```

Figure 29. MongoDB File Documents



```
_id:  ObjectId('636b181ed3372094da8feb36')
sender:  "alice"
receiver:  "bob"
message_type:  "text"
message:  ""
signature:  ""
timestamp:  2022-11-09T14:01:50.962+00:00
delivery_type:  "ENCRYPTED_AND_SIGNED"

_id:  ObjectId('636b1846d3372094da8feb37')
sender:  "alice"
receiver:  "bob"
message_type:  "text"
message:  BinData(0, 'SzkfartXSVNEH+FU2r396bpbPLS2NMP1g07qnWh6JjN8wssfirjW1PN6YXewxyzoBDSL3ydqr0gTCkquLrUShW1VuojN5iya26mf…')
signature:  BinData(0, 'Djzsk/QQ8coP9vo01ScMQyssNv2DXfDJq9h9pM7JvFBB+olrLxnJZeQy1SOMDsGt3AagZPjWh0rB8i6WnlxvPp31A8Iiwox01tpj…')
timestamp:  2022-11-09T14:02:30.823+00:00
delivery_type:  "ENCRYPTED_AND_SIGNED"

_id:  ObjectId('636b1884d3372094da8feb38')
sender:  "alice"
receiver:  "bob"
message_type:  "text"
message:  BinData(0, 'W90od1H42zCqtj+6aW6AbWPVYxTxWxN4rXfiOHJV7o6ZJgIe3dDgL7atxyJudoP0hngV1wmM3+pa3D1hOmTRF2okDM8Ak+RJf16s…')
signature:  BinData(0, 'Dt511Ef3+mqanua9RVSNHH1ExqnRwVqDX5G74ImFvAUECKNCNMEu14It1LtP9aUfkQo4DblKztHyyIkMjUrrMYrKrQ0m9yHMG8Ro…')
timestamp:  2022-11-09T14:03:32.508+00:00
```

Figure 30. MongoDB Message Documents



```
_id:  ObjectId('6367cd345e53f5fb0703ecad')
user_id:  2
username:  "alice"
password:  "6ca13d52ca70c883e0f0bb101e425a89e8624de51db2d2392593af6a84118090"
wallet_key:  "bd8715cb599a70f56e113976831be145a9df246685e120222fd06060cf68b95d"
email:  "1@qq.c"
phone_number:  "1234"
photo:  BinData(0, 'LzlqLzJ3QkRBQXdKQ2dzS0VRMFhGUmNPRGcwUEV5QVZFeElTRX1jY0hoY2dMaWt4TUM0cExTd3pPa28rTXpaR055d3RRRmRCUmt4…')
key_length:  1024
private_key:  BinData(0, 'TFMwdExTMUNSVWRKVGlCU1UwRWdVRkpKVmtGVVJTQkxSVmt0TFMwdExRcE5TVWxEV0VGS1FrRkJTMEpuVVV0eGFrMU1NbWgwY1V0…')
private_key_location:  "C:/Users/Jakl1/Desktop/alice_p/my_private_key.pem"
> message:  Array

_id:  ObjectId('6367ce7e5e53f5fb0703ecb1')
user_id:  1
username:  "bob"
password:  "6ca13d52ca70c883e0f0bb101e425a89e8624de51db2d2392593af6a84118090"
wallet_key:  "8e4ac4cbdd15afb0191fd03f8ddc6508602fd8155e689f3a6c6c990762d499c6"
email:  "1@qq.c"
phone_number:  "1234"
photo:  BinData(0, 'LzlqLzRBQVFTa1pKUmdBQkFRRUJMQUVzQUFELzRRQmFSWGhwWmdBQVNVa3FBQWdBQUFBQkFBNEJBZ0E0QUFBQUdnQUFBQUFBQUFC…')
key_length:  1024
private_key:  BinData(0, 'TFMwdExTMUNSVWRKVGlCU1UwRWdVRkpKVmtGVVJTQkxSVmt0TFMwdExRcE5TVWxEV0ZGS1FrRkJTMEpuVVV0eEszVTJlbV13TjFG…')
private_key_location:  "C:/Users/Jakl1/Desktop/bob_p/my_private_key.pem"
> message:  Array
```

Figure 31. MongoDB User Documents

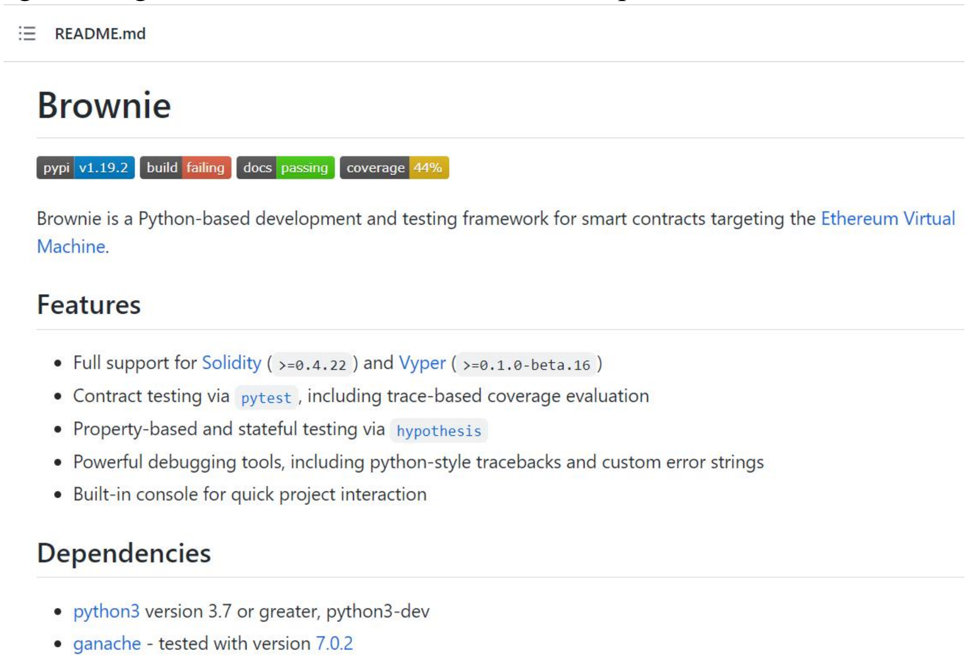
6.5 Blockchain

To ensure decentralization, the Ethereum distributed blockchain network is used in this project.

6.5.1. Eth-Brownie

6.5.1.1. The Reason for Using Eth-Brownie

Eth-Brownie is a Python-based development and testing framework for smart contracts targeting the Ethereum Virtual Machine. Since the front-end and back-end of the project system are developed in Python language, Python-based Eth-Brownie programming can work well with non-blockchain parts.



README.md

Brownie

pypi v1.19.2 | build failing | docs passing | coverage 44%

Brownie is a Python-based development and testing framework for smart contracts targeting the Ethereum Virtual Machine.

Features

- Full support for Solidity (`>=0.4.22`) and Vyper (`>=0.1.0-beta.16`)
- Contract testing via `pytest`, including trace-based coverage evaluation
- Property-based and stateful testing via `hypothesis`
- Powerful debugging tools, including python-style tracebacks and custom error strings
- Built-in console for quick project interaction

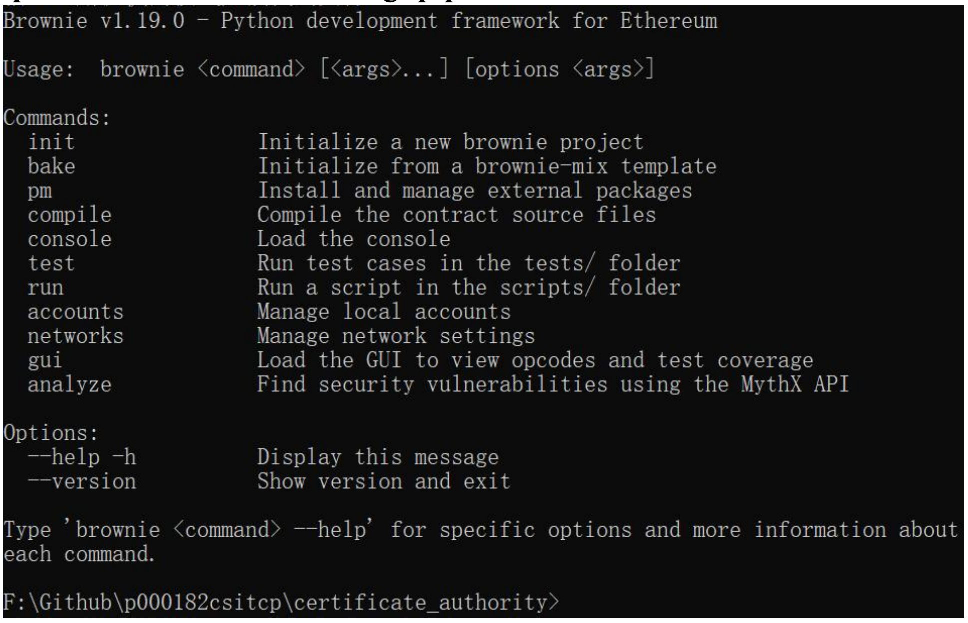
Dependencies

- python3 version 3.7 or greater, python3-dev
- ganache - tested with version 7.0.2

Figure32. Eth-Brownie Introduction

6.5.1.2. The Use of Eth-Brownie

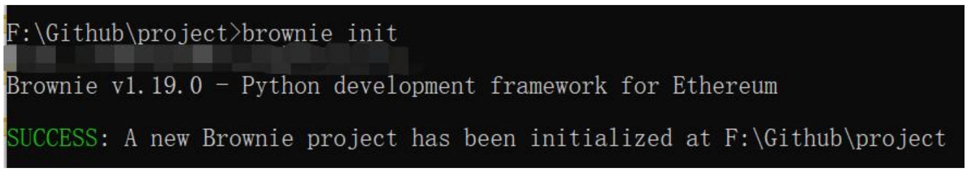
To install Eth-Brownie, in command line, run **‘pip install eth-brownie’** using **‘pip’** or **‘pipx install eth-brownie**’ using **‘pipx**’.



```
Brownie v1.19.0 - Python development framework for Ethereum

Usage:  brownie <command> [<args>...] [options <args>]

Commands:
  init               Initialize a new brownie project
  bake               Initialize from a brownie-mix template
  pm                 Install and manage external packages
  compile            Compile the contract source files
  console            Load the console
  test               Run test cases in the tests/ folder
  run                Run a script in the scripts/ folder
  accounts           Manage local accounts
  networks           Manage network settings
  gui                Load the GUI to view opcodes and test coverage
  analyze            Find security vulnerabilities using the MythX API

Options:
  --help -h          Display this message
  --version          Show version and exit

Type 'brownie <command> --help' for specific options and more information about
each command.

F:\Github\p000182csitcp\certificate_authority>
```

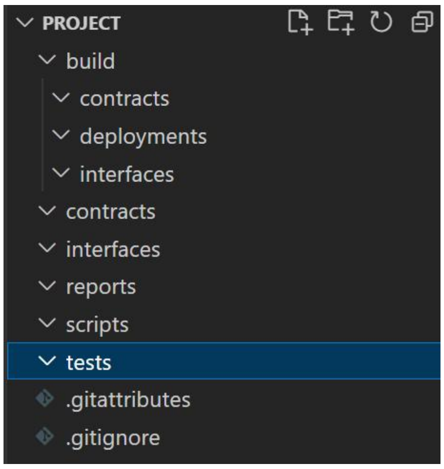
Figure 33. Eth-Brownie in Command Line

When the project started from scratch, run **'brownie init'** in command line on the empty project file path. This step is not required when cloning the system project from Github.



```
F:\Github\project>brownie init
[redacted]
Brownie v1.19.0 - Python development framework for Ethereum

SUCCESS: A new Brownie project has been initialized at F:\Github\project
```

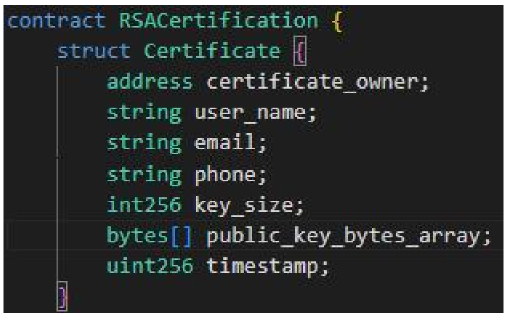
Figure34. Creating Project Structure at an Empty Project Folder Using Brownie



```
PROJECT
  build
    contracts
    deployments
    interfaces
  contracts
  interfaces
  reports
  scripts
  tests
  .gitattributes
  .gitignore
```

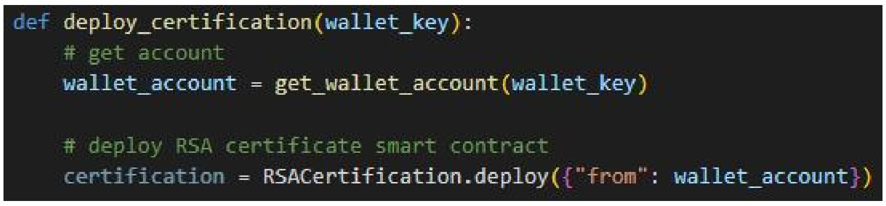
Figure35. Project Folder Structure Created by Brownie

The smart contract is developed in Solidity language, and the deployment script of smart contract is developed in Python language and Eth-Brownie Programming Syntax.



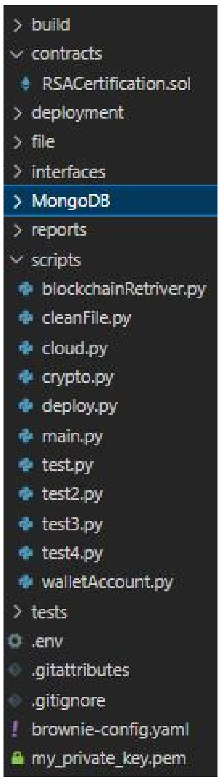
```
contract RSACertification {
    struct Certificate {
        address certificate_owner;
        string user_name;
        string email;
        string phone;
        int256 key_size;
        bytes[] public_key_bytes_array;
        uint256 timestamp;
    }
```

Figure 36. Smart Contract (Including Certificate Constructor and Certificate Functions) in Solidity Language



```
def deploy_certification(wallet_key):
    # get account
    wallet_account = get_wallet_account(wallet_key)

    # deploy RSA certificate smart contract
    certification = RSACertification.deploy({"from": wallet_account})
```

Figure 37. Smart Contract Deployment in Eth-Brownie Programming Syntax



```
> build
v contracts
    RSACertification.sol
> deployment
> file
> interfaces
> MongoDB
> reports
v scripts
    blockchainRetriver.py
    cleanFile.py
    cloud.py
    crypto.py
    deploy.py
    main.py
    test.py
    test2.py
    test3.py
    test4.py
    walletAccount.py
> tests
  .env
  .gitattributes
  .gitignore
  brownie-config.yaml
  my_private_key.pem
```

Figure 38. The Project Folder Structure in Eth-Brownie Framework

6.5.2. Ganache-CLI

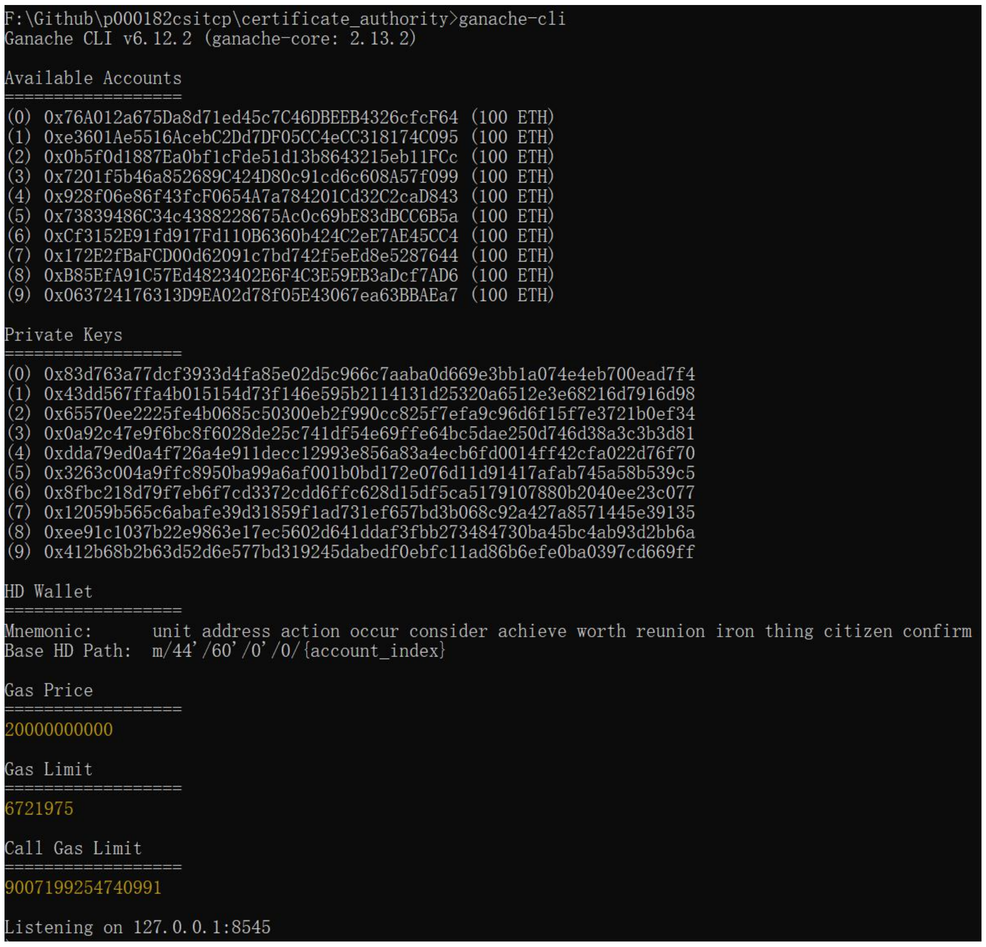
6.5.2.1. The Reason for Using Ganache-CLI

Although Ethereum provides developers with free blockchain test nets such as the Rinkeby Test Network and Goerli Test Network, like the Ethereum mainnet, on-chain transactions require gas fees in virtual currency. However, a developer account can only obtain 0.1 test coins per day for on-chain transactions on the blockchain test network, which means that although the test network is free, the number of on-chain transactions that can be performed per day is limited.

Ganache-CLI, part of the Ethereum Development Kit, is the command-line version of Ganache for local blockchain testing in Ethereum development. By using the Ganache-CLI, transaction creation and functional testing of the blockchain becomes locally available, solving the problem of the limited number of on-chain transactions that can be performed per day.

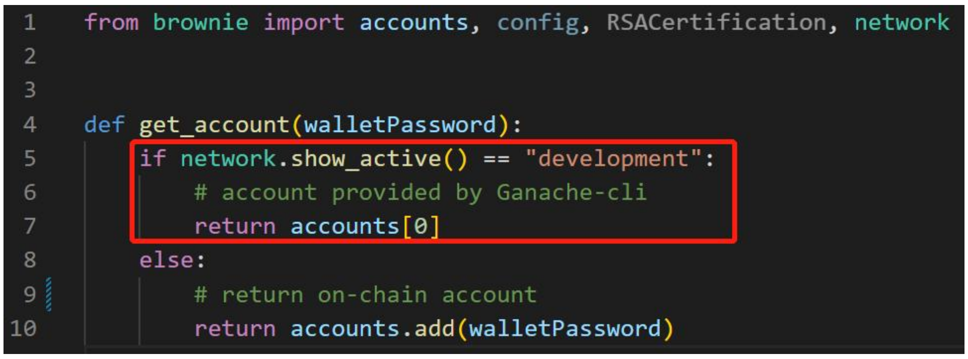
6.5.2.2. The Use of Ganache-CLI

To install Ganache-CLI, in command line, run **‘npm install -g ganache-cli’** using **‘npm’** or **‘yarn global add ganache-cli’** using **‘yarn’**.
To use Ganache-CLI in command line, run **‘$ ganache-cli <options>’**.



```
F:\Github\p000182csitcp\certificate_authority>ganache-cli
Ganache CLI v6.12.2 (ganache-core: 2.13.2)

Available Accounts
==================
(0) 0x76A012a675Da8d71ed45c7C46DBEEB4326cfcF64 (100 ETH)
(1) 0xe3601Ae5516AcebC2Dd7DF05CC4eCC318174C095 (100 ETH)
(2) 0x0b5f0d1887Ea0bf1cFde51d13b8643215eb11FCc (100 ETH)
(3) 0x7201f5b46a852689C424D80c91cd6c608A57f099 (100 ETH)
(4) 0x928f06e86f43fcF0654A7a784201Cd32C2caD843 (100 ETH)
(5) 0x73839486C34c4388228675Ac0c69bE83dBCC6B5a (100 ETH)
(6) 0xCf3152E91fd917Fd110B6360b424C2eE7AE45CC4 (100 ETH)
(7) 0x172E2fBaFCD00d62091c7bd742f5eEd8e5287644 (100 ETH)
(8) 0xB85EfA91C57Ed4823402E6F4C3E59EB3aDcf7AD6 (100 ETH)
(9) 0x063724176313D9EA02d78f05E43067ea63BBAEa7 (100 ETH)

Private Keys
==================
(0) 0x83d763a77dcf3933d4fa85e02d5c966c7aaba0d669e3bb1a074e4eb700ead7f4
(1) 0x43dd567ffa4b015154d73f146e595b2114131d25320a6512e3e68216d7916d98
(2) 0x65570ee2225fe4b0685c50300eb2f990cc825f7efa9c96d6f15f7e3721b0ef34
(3) 0x0a92c47e9f6bc8f6028de25c741df54e69ffe64bc5dae250d746d38a3c3b3d81
(4) 0xdda79ed0a4f726a4e911decc12993e856a83a4ecb6fd0014ff42cfa022d76f70
(5) 0x3263c004a9ffc8950ba99a6af001b0bd172e076d11d91417afab745a58b539c5
(6) 0x8fbc218d79f7eb6f7cd3372cdd6ffc628d15df5ca5179107880b2040ee23c077
(7) 0x12059b565c6abafe39d31859f1ad731ef657bd3b068c92a427a8571445e39135
(8) 0xee91c1037b22e9863e17ec5602d641ddaf3fbb273484730ba45bc4ab93d2bb6a
(9) 0x412b68b2b63d52d6e577bd319245dabedf0ebfc11ad86b6efe0ba0397cd669ff

HD Wallet
==================
Mnemonic:      unit address action occur consider achieve worth reunion iron thing citizen confirm
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
==================
20000000000

Gas Limit
==================
6721975

Call Gas Limit
==================
9007199254740991

Listening on 127.0.0.1:8545
```

Figure 39. The Use of Ganache-CLI in Command Line



```python
from brownie import accounts, config, RSACertification, network


def get_account(walletPassword):
    if network.show_active() == "development":
        # account provided by Ganache-cli
        return accounts[0]
    else:
        # return on-chain account
        return accounts.add(walletPassword)
```

Figure 40. The Use of Ganache-CLI in the Code

6.5.3. Metamask

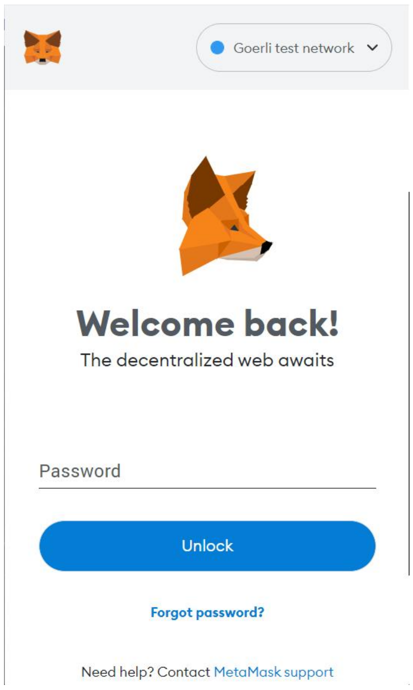
MetaMask is a software cryptocurrency wallet for interacting with the Ethereum blockchain. It allows users to access their Ethereum wallets through a browser extension or mobile app to interact with decentralized programs like the project system.

6.5.3.1. The Reasons for Using Metamask

- Metamask handles account management and connects users to the blockchain, allowing users to manage accounts and their keys in multiple ways, including hardware wallets, while isolating them from the site context.
- MetaMask is a tool to implement user interaction and experience on Web3. MetaMask supports Web3-based blockchain development tools and can work well with Web3-based Eth-Brownie.

6.5.3.2. The Use of Metamask (Browser Extension)

First, install the Metamask extension in the browser. Then, sign up for a Metamask account and log in.



Goerli test network

Welcome back!

The decentralized web awaits

Password

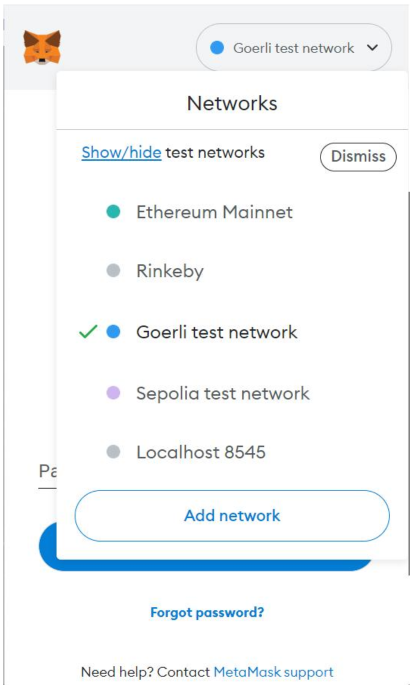
Unlock

Forgot password?

Need help? Contact MetaMask support

Figure 41. Metamask login

Second, choose the Ethereum blockchain network. In the development of this project system, the Goerli test network is used.



Goerli test network

Networks

Show/hide test networks Dismiss

Ethereum Mainnet

Rinkeby

✓ Goerli test network

Sepolia test network

Localhost 8545

Add network

Forgot password?

Need help? Contact MetaMask support

Figure 42. Ethereum Blockchain Network Selection (Goerli Test Network)

Third, get a test balance for your Metamask account via the Ethereum faucet. In the development of this project system, the Goerli test network faucet is used.

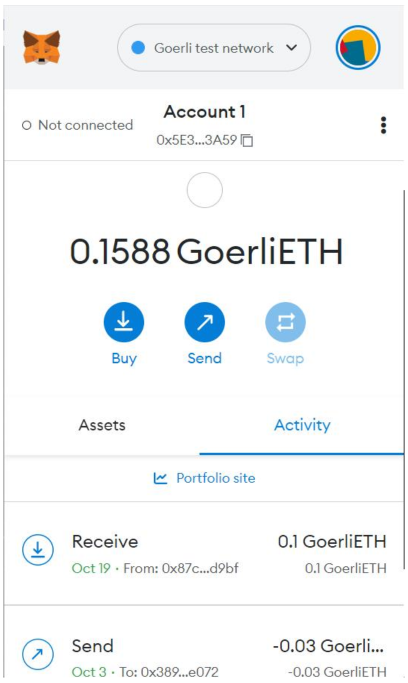


Figure 43. Viewing Account Balance

6.5.4. Infura

Infura provides high availability blockchain APIs and developer tools to help develop applications that connect to the Ethereum blockchain. Infura interacts with the Ethereum blockchain and runs nodes on behalf of its users.

6.5.4.1. The Reasons for Using Infura

- Infura provides faster access to the Ethereum blockchain.
- Infura can help Eth-Brownie and Metamask connect to the Ethereum blockchain network in the Web3 framework.

6.5.4.2. The Use of Infura

First, register and log in to your Infura account.

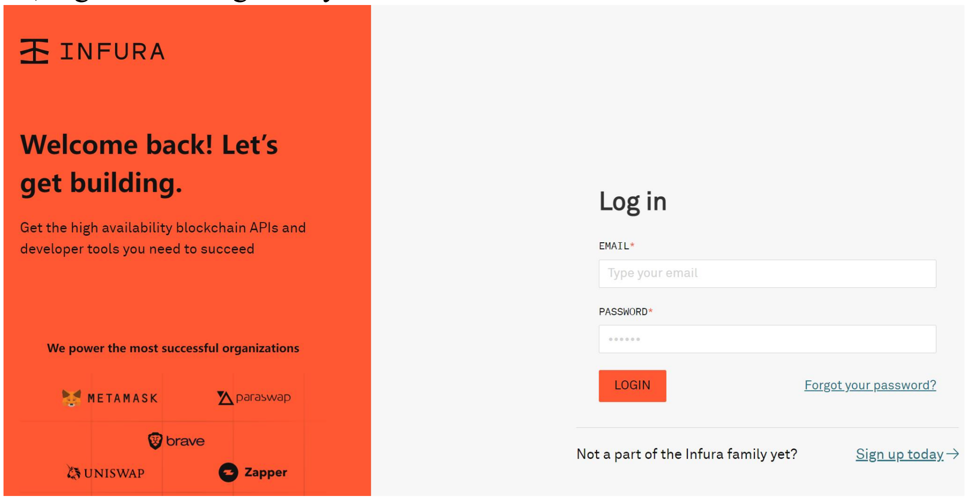
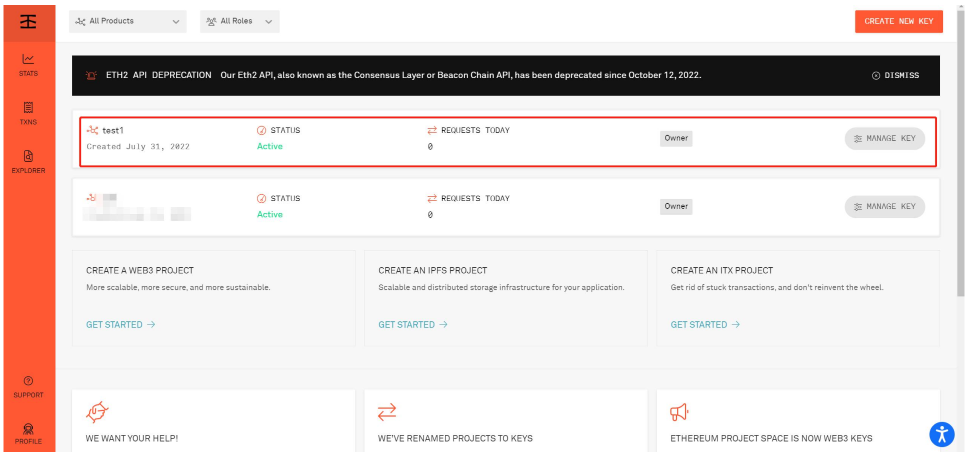

INFURA
Welcome back! Let's get building.
Get the high availability blockchain APIs and developer tools you need to succeed
We power the most successful organizations
METAMASK
paraswap
brave
UNISWAP
Zapper
Log in
EMAIL*
Type your email
PASSWORD*
······
LOGIN
Forgot your password?
Not a part of the Infura family yet?
Sign up today →

Figure 44. Infura Login

Second, create and choose the blockchain project in Infura.



ETH2 API DEPRECATION Our Eth2 API, also known as the Consensus Layer or Beacon Chain API, has been deprecated since October 12, 2022.

test1 Created July 31, 2022 | STATUS Active | REQUESTS TODAY 0 | Owner

[redacted] | STATUS Active | REQUESTS TODAY 0 | Owner

CREATE A WEB3 PROJECT
More scalable, more secure, and more sustainable.

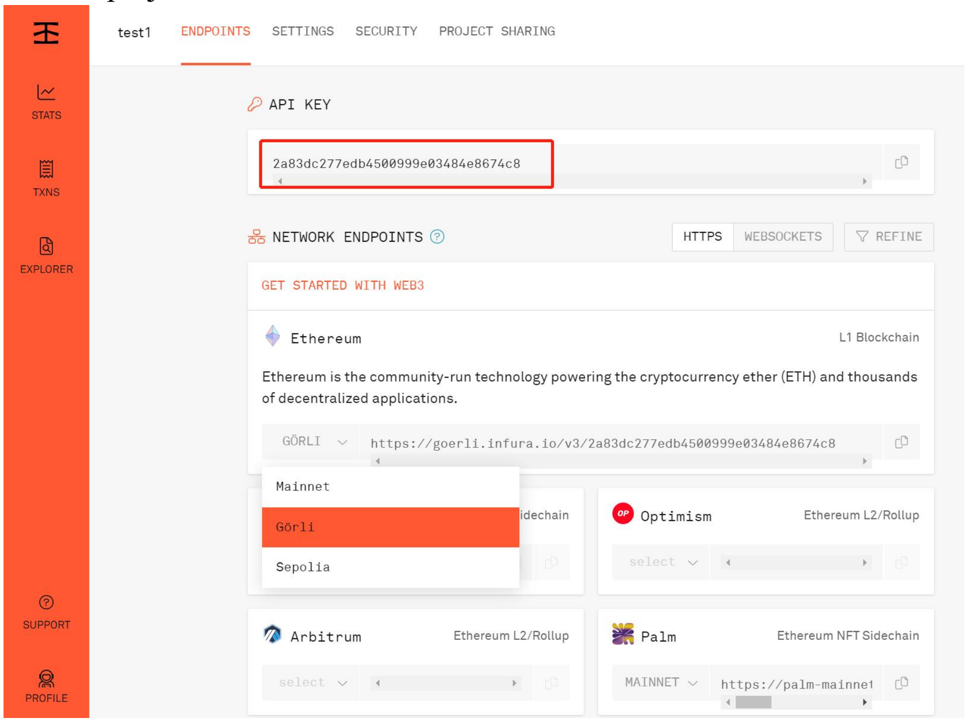
CREATE AN IPFS PROJECT
Scalable and distributed storage infrastructure for your application.

CREATE AN ITX PROJECT
Get rid of stuck transactions, and don't reinvent the wheel.

WE WANT YOUR HELP!
WE'VE RENAMED PROJECTS TO KEYS
ETHEREUM PROJECT SPACE IS NOW WEB3 KEYS

Figure 45. Choosing Infura Project

Third, copy the project API key in Infura and paste it into the environment variables file of the project.



API KEY

2a83dc277edb4500999e03484e8674c8

NETWORK ENDPOINTS

GET STARTED WITH WEB3

Ethereum L1 Blockchain

Ethereum is the community-run technology powering the cryptocurrency ether (ETH) and thousands of decentralized applications.

GÖRLI https://goerli.infura.io/v3/2a83dc277edb4500999e03484e8674c8

idechain

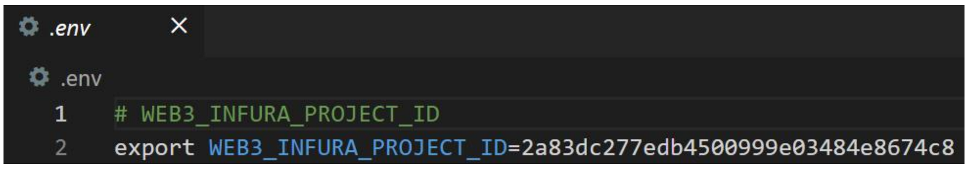
Optimism Ethereum L2/Rollup

Arbitrum Ethereum L2/Rollup

Palm Ethereum NFT Sidechain

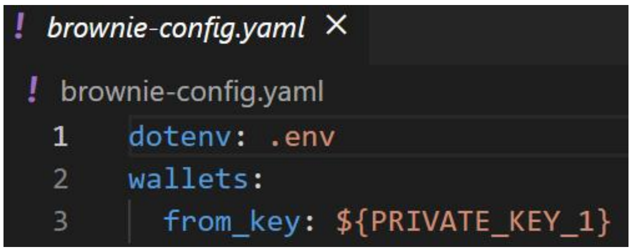
MAINNET https://palm-mainnet

Figure 46. Accessing Infura Project API Key



```
.env
# WEB3_INFURA_PROJECT_ID
export WEB3_INFURA_PROJECT_ID=2a83dc277edb4500999e03484e8674c8
```

Figure 47. Adding Infura Project API Key into the Environment Variables File



```
brownie-config.yaml
dotenv: .env
wallets:
  from_key: ${PRIVATE_KEY_1}
```

Figure 48. Configuring the yaml file to Use the env File

After that, Eth-Brownie, Metamask and Infura can work together, and the project system is able to connect to the Goerli test network of Ethereum.

6.5.5. Etherscan

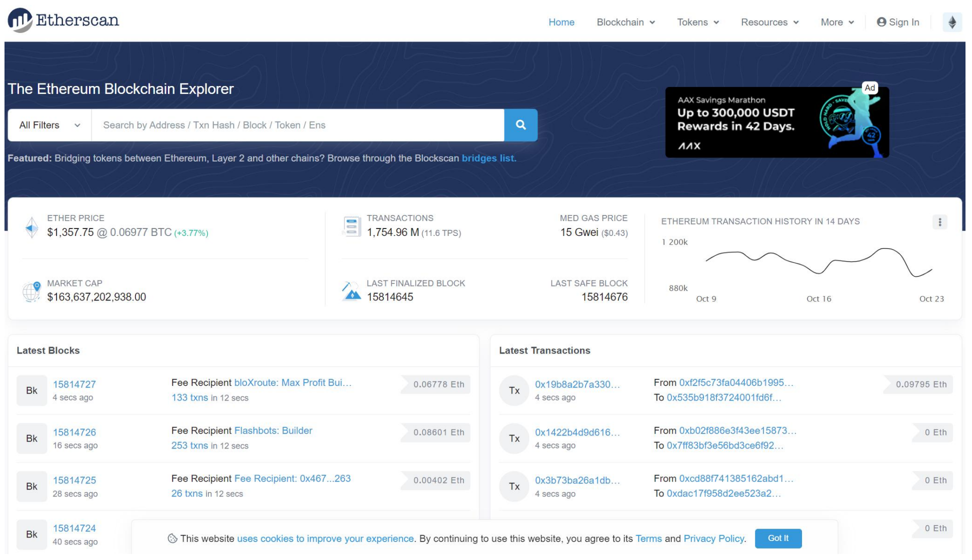
Etherscan is a block explorer and analysis platform that provides services for viewing and analyzing assets, balances, and transactions on the Ethereum network.

6.5.5.1. The Reasons for Using Etherscan

By using Etherscan, everyone can view transaction information on a specific wallet address, which can provide transparency and contribute to system decentralization. On the other hand, during the development process, Etherscan can help to test the on-chain transaction functions of the project system.

6.5.5.2. The Use of Etherscan

First browse the Etherscan website.



The Ethereum Blockchain Explorer

Featured: Bridging tokens between Ethereum, Layer 2 and other chains? Browse through the Blockscan bridges list.

ETHER PRICE $1,357.75 @ 0.06977 BTC (+3.77%)
TRANSACTIONS 1,754.96 M (11.6 TPS)
MED GAS PRICE 15 Gwei ($0.43)
ETHEREUM TRANSACTION HISTORY IN 14 DAYS
MARKET CAP $163,637,202,938.00
LAST FINALIZED BLOCK 15814645
LAST SAFE BLOCK 15814676

Latest Blocks

Bk	15814727 4 secs ago	Fee Recipient bloXroute: Max Profit Bui… 133 txns in 12 secs	0.06778 Eth
Bk	15814726 16 secs ago	Fee Recipient Flashbots: Builder 253 txns in 12 secs	0.08601 Eth
Bk	15814725 28 secs ago	Fee Recipient Fee Recipient: 0x467…263 26 txns in 12 secs	0.00402 Eth
Bk	15814724 40 secs ago		

Latest Transactions

Tx	0x19b8a2b7a330… 4 secs ago	From 0xf2f5c73fa04406b1995… To 0x535b918f3724001fd6f…	0.09795 Eth
Tx	0x1422b4d9d616… 4 secs ago	From 0xb02f886e3f43ee15873… To 0x7ff83bf3e56bd3ce6f92…	0 Eth
Tx	0x3b73ba26a1db… 4 secs ago	From 0xcd88f741385162abd1… To 0xdac17f958d2ee523a2…	0 Eth
			0 Eth

This website uses cookies to improve your experience. By continuing to use this website, you agree to its Terms and Privacy Policy.

Figure 49. Etherscan Website

Second, copy the wallet address in Metamask, paste it into Etherscan, and click the search button.

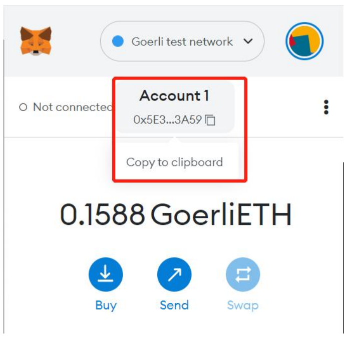
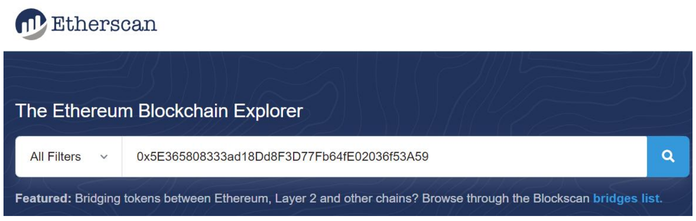


Figure 50. Copying the Wallet Address in Metamask



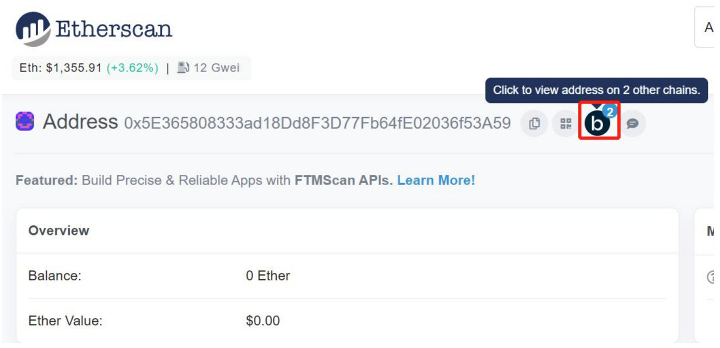
The Ethereum Blockchain Explorer

0x5E365808333ad18Dd8F3D77Fb64fE02036f53A59

Featured: Bridging tokens between Ethereum, Layer 2 and other chains? Browse through the Blockscan bridges list.

Figure 51. Searching for the Wallet Address on Etherscan

Third, click the button to view information on other chains.



Etherscan

Eth: $1,355.91 (+3.62%) | 12 Gwei

Click to view address on 2 other chains.

Address 0x5E365808333ad18Dd8F3D77Fb64fE02036f53A59

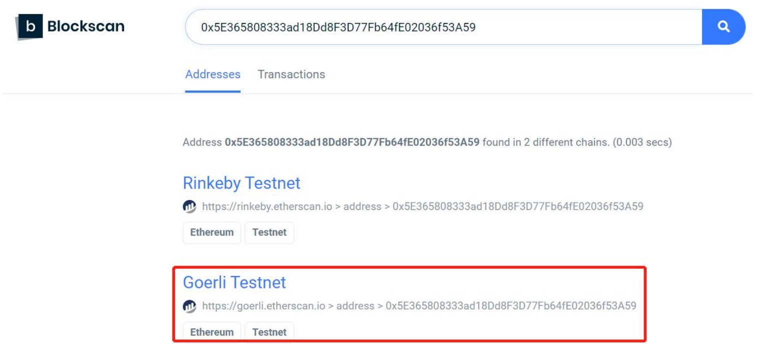
Featured: Build Precise & Reliable Apps with FTMScan APIs. Learn More!

Overview

Balance:	0 Ether
Ether Value:	$0.00

Figure 52. Clicking the Button to View Information on Other Chains

Fourth, choose to view information on the Goerli test network.



Blockscan

0x5E365808333ad18Dd8F3D77Fb64fE02036f53A59

Addresses Transactions

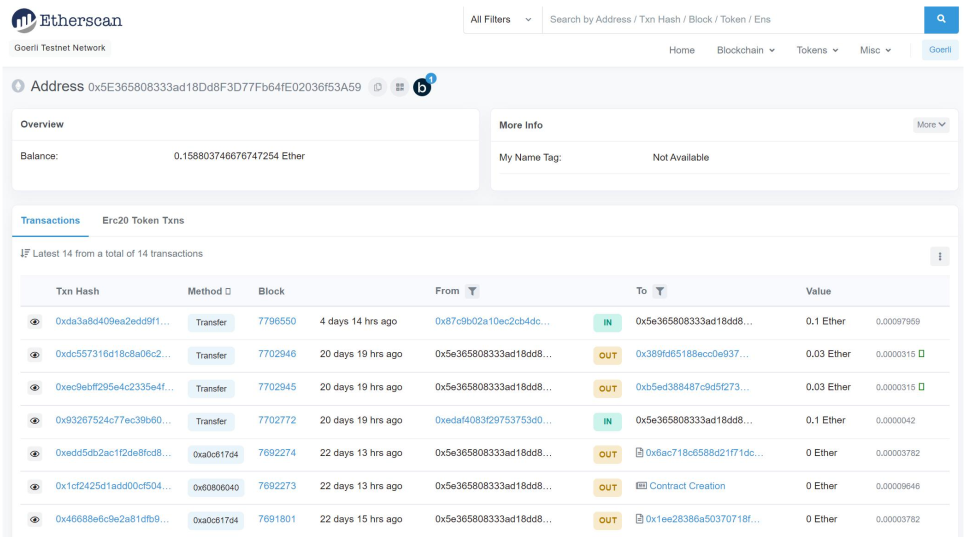
Address **0x5E365808333ad18Dd8F3D77Fb64fE02036f53A59** found in 2 different chains. (0.003 secs)

Rinkeby Testnet
https://rinkeby.etherscan.io > address > 0x5E365808333ad18Dd8F3D77Fb64fE02036f53A59
Ethereum Testnet

Goerli Testnet
https://goerli.etherscan.io > address > 0x5E365808333ad18Dd8F3D77Fb64fE02036f53A59
Ethereum Testnet

Figure 53. Choosing Goerli Testnet

Fifth, transaction information for this wallet address on the Goerli test network is available.



Etherscan
Goerli Testnet Network

All Filters | Search by Address / Txn Hash / Block / Token / Ens

Home Blockchain Tokens Misc Goerli

Address 0x5E365808333ad18Dd8F3D77Fb64fE02036f53A59

Overview

Balance: 0.158803746676747254 Ether

More Info

My Name Tag: Not Available

Transactions Erc20 Token Txns

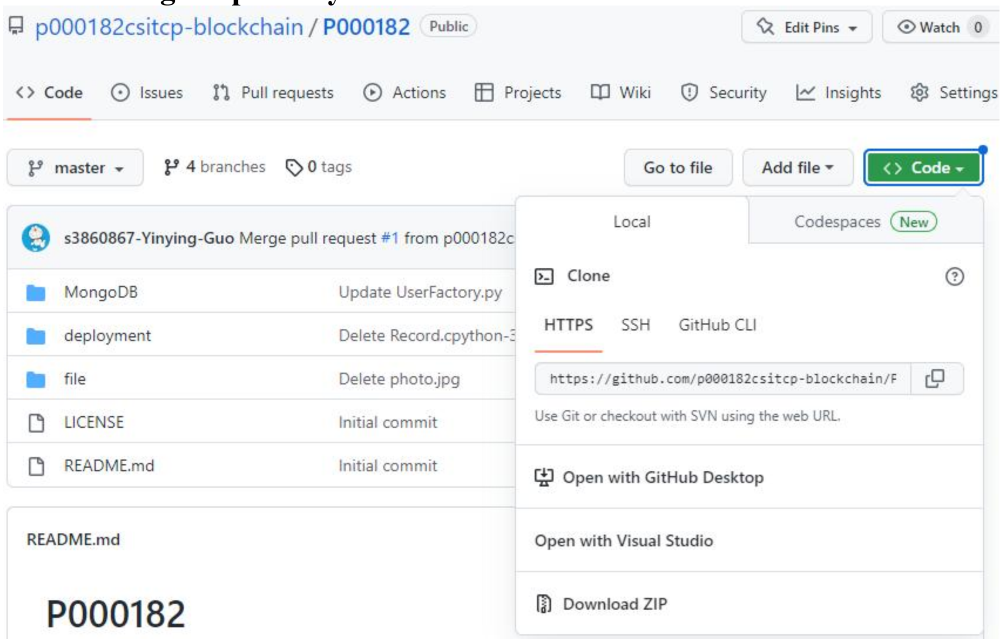
Latest 14 from a total of 14 transactions

	Txn Hash	Method	Block		From		To	Value	
	0xda3a8d409ea2edd9f1…	Transfer	7796550	4 days 14 hrs ago	0x87c9b02a10ec2cb4dc…	IN	0x5e365808333ad18dd8…	0.1 Ether	0.00097959
	0xdc557316d18c8a06c2…	Transfer	7702946	20 days 19 hrs ago	0x5e365808333ad18dd8…	OUT	0x389fd65188ecc0e937…	0.03 Ether	0.0000315
	0xec9ebff295e4c2335e4f…	Transfer	7702945	20 days 19 hrs ago	0x5e365808333ad18dd8…	OUT	0xb5ed388487c9d5f273…	0.03 Ether	0.0000315
	0x93267524c77ec39b60…	Transfer	7702772	20 days 19 hrs ago	0xedaf4083f29753753d0…	IN	0x5e365808333ad18dd8…	0.1 Ether	0.0000042
	0xedd5db2ac1f2de8fcd8…	0xa0c617d4	7692274	22 days 13 hrs ago	0x5e365808333ad18dd8…	OUT	0x6ac718c6588d21f71dc…	0 Ether	0.00003782
	0x1cf2425d1add00cf504…	0x60806040	7692273	22 days 13 hrs ago	0x5e365808333ad18dd8…	OUT	Contract Creation	0 Ether	0.00009646
	0x46688e6c9e2a81dfb9…	0xa0c617d4	7691801	22 days 15 hrs ago	0x5e365808333ad18dd8…	OUT	0x1ee28386a50370718f…	0 Ether	0.00003782

Figure 54. Viewing Transaction Information

6 DEPLOYMENT INSTRUCTIONS

7.1 Clone the git repository



s3860867-Yinying-Guo Merge pull request #1 from p000182c

MongoDB	Update UserFactory.py
deployment	Delete Record.cpython-3
file	Delete photo.jpg
LICENSE	Initial commit
README.md	Initial commit

https://github.com/p000182csitcp-blockchain/F

Use Git or checkout with SVN using the web URL.

Figure 55. GitHub repository

Run **'git clone https://github.com/p000182csitcp-blockchain/P000182.git'.**

7.2 Command

```
PS F:\pp1_workSpace\certificate_authority - 1> brownie run scripts/main.py --network goerli
```

Figure 56. Run Brownie command

As the project is using Brownie as an entry point, run **'brownie run scripts/main.py --network goerli'.**

7 TEST SPECIFICATIONS

Test case ID	Test Objective	Preconditions	Steps	Test data	Expected result
1	Sign up for a new account	1. The user account hasn't signed up	1. Click Sign up button in main page 2. Type username 3. Type password 4. Type password again 5. Type email 6. Type phone number 7. Choose a key length 8. Click Register button	User object data	The user account information is successfully saved into mongoDB
2	Log in	1. The user already has an account 2. The user knows	1. Type username in log in page 2. Type password	User object data	The user account information matched with the

		the username and password	3. Click Login button		saved on in database, so the user is moved to the home page
3	Check message	1. The user already registered an account in the system 2. The user already logged in to the system	1. Click Check message button in main page	Fronten d UI button event	The Check message page is popped up.
4	Download private key	1. The user already registered an account in the system 2. The user already logged in to the system	1. Click Download key pairs button in home page 2. Choose a download location and confirm it	User object data	The info of "Your private key file has been download." popped up. The user's private key pair is downloaded into his/her local machine
5	Send a message	1. The user already registered an account in the system 2. The user already logged in to the system	1. Click Send a message button in home page	Fronten d UI button event	The Send a message page is popped up.
6	Choose receiver	1. The user already registered an account in the system 2. The user already logged in to the system 3. There exists at least one user who can receive a message	1. Click Send a message button in home page 2. Click a dropdown box to choose a receiver	User object data	All available users' name is listed in the dropdown box.
7	Encrypt, Sign the message	1. The user already registered an account in the system 2. The user already logged in to the system 3. There exists at least one user who can receive a message 4. The user has RSA key pair in his/her local machine	1. Click Send a message button in home page 2. Click a dropdown box to choose a receiver 3. Type a message 4. Click Select private key button 5. Choose the path of private key in local machine 6. Click Encrypt and Sign button 7. Click Send button	User object data and message object data	The info of "Message sent!" popped up. The created ecrypted and signed message is successfully saved into MongoDB
8	Encrypt the	1. The user already	1. Click Send a	User	The info of

	message	registered an account in the system 2. The user already logged in to the system 3. There exists at least one user who can receive a message	message button in home page 2. Click a dropdown box to choose a receiver 3. Type a message 4. Click Encrypt button 5. Click Send button	object data and message object data	"Message sent!" popped up. The created ecrypted message is successfully saved into MongoDB
9	Sign the message	1. The user already registered an account in the system 2. The user already logged in to the system 3. There exists at least one user who can receive a message 4. The user has private key in his/her local machine	1. Click Send a message button in home page 2. Click a dropdown box to choose a receiver 3. Type a message 4. Click Select private key button 5. Choose the path of private key in local machine 6. Click Sign button 7. Click Send button	User object data and message object data	The info of "Message sent!" popped up. The created signed message is successfully saved into MongoDB
10	Decrypt the message	1. The user already registered an account in the system 2. The user already logged in to the system 3. There exists at least one encrypted received text message from another user 4. The user has private key in his/her local machine	1. Click Check message button in home page 2. Click a text message that is encrypted 3. Click Select private key button 4. Choose the private key location on local machine 5. Click Decrypt button 6. Check the Decrypted message on message box	Message object data	The original message is shown in the message box
11	Validate the message	1. The user already registered an account in the system 2. The user already logged in to the system 3. There exists at least one signed received text message from another user	1. Click Check message button in home page 2. Click a text message that is signed 3. Click Validate button 4. Check the Signed message on message box	Message object data	The info of "Verify Succssful" popped up. The original message is shown in the message box
12	Send a file	1. The user already registered an account in the	1. Click Send a file button in home page	Fronten d UI button	The Send a file page is popped up.

		system 2. The user already logged in to the system		event	
13	Encrypt, Sign the file	1. The user already registered an account in the system 2. The user already logged in to the system 3. There exists at least one user who can receive a message	1. Click Send a message button in home page 2. Click a dropdown box to choose a receiver 3. Click Add File button 4. Choose a file and confirm 5. Click Select private key button 6. Choose the path of private key in local machine 7. Click Encrypt and Sign button 8. Click Send button	Message object data, File object data	The info of “Message sent!” popped up. The created ecrypted and signed message is successfully saved into MongoDB
14	Encrypt the file	1. The user already registered an account in the system 2. The user already logged in to the system 3. There exists at least one user who can receive a message	1. Click Send a message button in home page 2. Click a dropdown box to choose a receiver 3. Click Add File button 4. Choose a file and confirm 5. Click Encrypt button 6. Click Send button	Message object data, File object data	The info of “Message sent!” popped up. The created ecrypted message is successfully saved into MongoDB
15	Sign the file	1. The user already registered an account in the system 2. The user already logged in to the system 3. There exists at least one user who can receive a message 4. The user has private key in his/her local machine	1. Click Send a message button in home page 2. Click a dropdown box to choose a receiver 3. Click Add File button 4. Choose a file and confirm 5. Click Select private key button 6. Choose the path of private key in local machine 7. Click Sign button 8. Click Send button	Message object data, File object data	The info of “Message sent!” popped up. The created signed message is successfully saved into MongoDB
16	Decrypt the file	1. The user already registered an account in the system	1. Click Check message button in home page 2. Click a file message	Message object data,	The filename is shown in the filename box. The

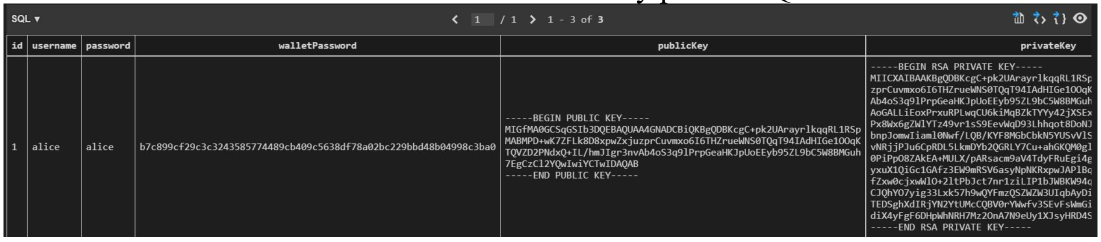
		2. The user already logged in to the system 3. There exists at least one encrypted received file message from another user 4. The user has private key in his/her local machine	that is encrypted 3. Click Select private key button 4. Choose the private key location on local machine 5. Click Decrypt button 6. Check the filename on filename box 7. Click Download the file 8. Choose a location and confirm	File object data	file is downloaded into the chosen location.
17	Validate the file	1. The user already registered an account in the system 2. The user already logged in to the system 3. There exists at least one signed received file message from another user	1. Click Check message button in home page 2. Click a file message that is signed 3. Click Validate button 4. Check the filename on filename box 5. Click Download the file 6. Choose a location and confirm	Message object data, File object data	The info of "Verify Succssful" popped up. The filename is shown in the filename box. The file is downloaded into the chosen location.

Figure 60. Test cases

8 TESTING RESULTS

Even though we didn't use any particular test frameworks, we made an effort to manually test some instances. With the SQLite database since beginning of the project, we have evaluated all the essential features. We originally experimented all test cases with an SQLite database until we started using the MongoDB.

1. Create a user account and save with RSA key pair in SQLite and retrieve them

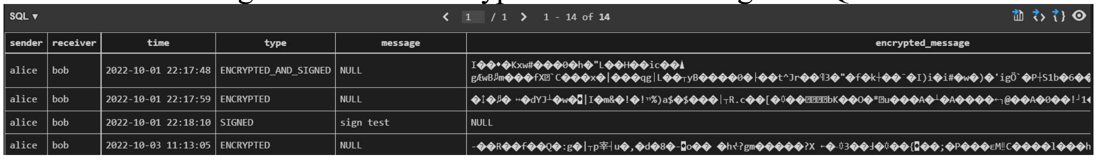


SQL ▾ ‹ 1 / 1 › 1 - 3 of 3

id	username	password	walletPassword	publicKey	privateKey
1	alice	alice	b7c899cf29c3c3243585774489cb409c5638df78a02bc229bbd48b04998c3ba0	-----BEGIN PUBLIC KEY----- MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDBKcgC+pk2UArayrlkqqRL1RSp MABMPD+wK7ZFLk8D8xpwZxjuzprCuvmxo6I6THZrueWNS0TQqT94IAdHIGe1OOqK TQVZD2PNdxQ+IL/hmJIgr3nvAb4oS3q9lPrpGeaHKJpUoEEyb95ZL9bC5W8BMGuh 7EgCzCl2YQwIwiYCTwIDAQAB -----END PUBLIC KEY-----	-----BEGIN RSA PRIVATE KEY----- MIICXAIBAAKBgQDBKcgC+pk2UArayrlkqqRL1RSp zprCuvmxo6I6THZrueWNS0TQqT94IAdHIGe1OOqK Ab4oS3q9lPrpGeaHKJpUoEEyb95ZL9bC5W8BMGuh AoGALLiEoxPrxuRPLwqCU6kiMqBZkTYYy42jXSEx Px8Wx6gZWlYTz49vr1sS9EevWqD93Lhhqot8DoNJ bnpJomwIiaml0Nwf/LQB/KYF8MGbCbkN5YUSvVlS vNRjjPJu6CpRDL5LkmDYb2QGRLY7Cu+ahGKQM0gl 0PiPpO8ZAkEA+MULX/pARsacm9aV4TdyFRuEgi4g yxuX1QiGc1GAfz3EW9mRSV6asyNpNKRxpwJAPlBq fZxw0cjxwWlO+2ltPbJct7nr1ziLIP1bJWBKW94q CJQhYO7yig33Lxk57h9wQYFmzQSZWZW3UIqbAyDi TEDSghXdIRjYN2YtUMcCQBV0rYWwfv3SEvFsWmGi diX4yFgF6DHpWhNRH7Mz2OnA7N9eUy1XJsyHRD4S -----END RSA PRIVATE KEY-----

Figure 61. SQLite 'Alice' data

2. Create messages with different types for each message in SQLite and retrieve them

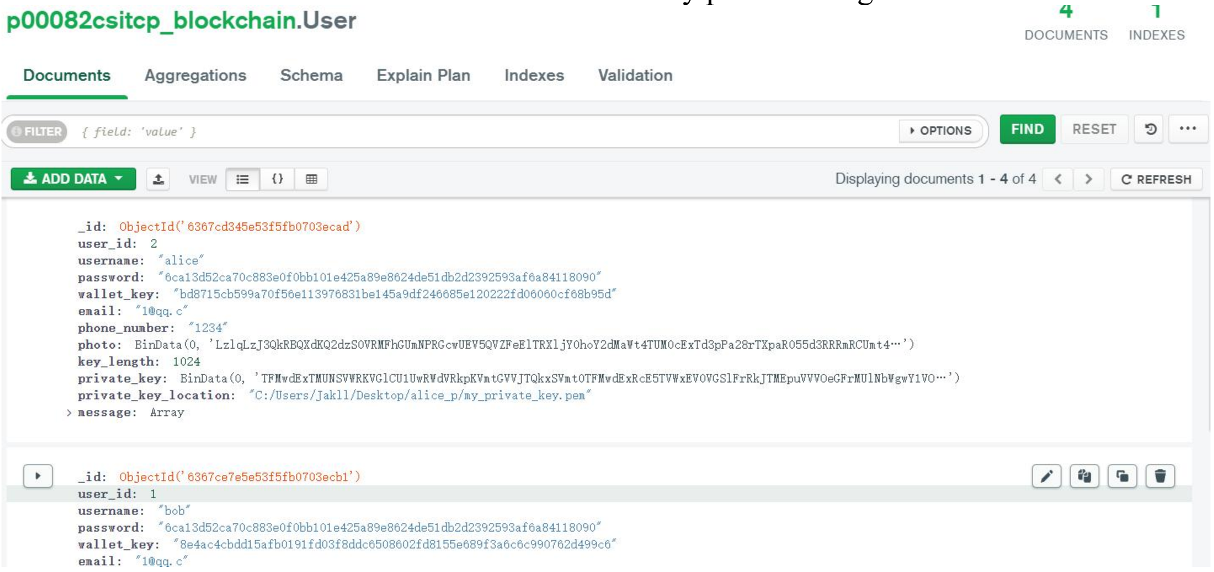


SQL ▾ ‹ 1 / 1 › 1 - 14 of 14

sender	receiver	time	type	message	encrypted_message
alice	bob	2022-10-01 22:17:48	ENCRYPTED_AND_SIGNED	NULL	I��•�Kxw#����0�h�"L��H��ic��[illegible] gÆwB[illegible]m���fX[illegible]`C���x�|���qg|L��┬yB�����0�├��t^Jr��¶3�"�f�k┼��¯�I)i�i#�w�)�'igÖ`�P┤S1b�6�[illegible]
alice	bob	2022-10-01 22:17:59	ENCRYPTED	NULL	�↕�[illegible]� ↔�dYJ┴�w�[illegible]|I�m&�!�!¬%)a$�$���|┬R.c��[�[illegible]��[illegible]bK��0�*[illegible]u���A�┴�A����←┐@��A�0��!┘1[illegible]
alice	bob	2022-10-01 22:18:10	SIGNED	sign test	NULL
alice	bob	2022-10-03 11:13:05	ENCRYPTED	NULL	-��R��f��Q�:g�|┬p[illegible]┤u�,�d�8�-[illegible]o�� �h[illegible]?gm�����?X ←�[illegible]3��┤�[illegible]��{[illegible]��;�P���εM‼C����1���h

Figure 62. SQLite 'alice' received messages data

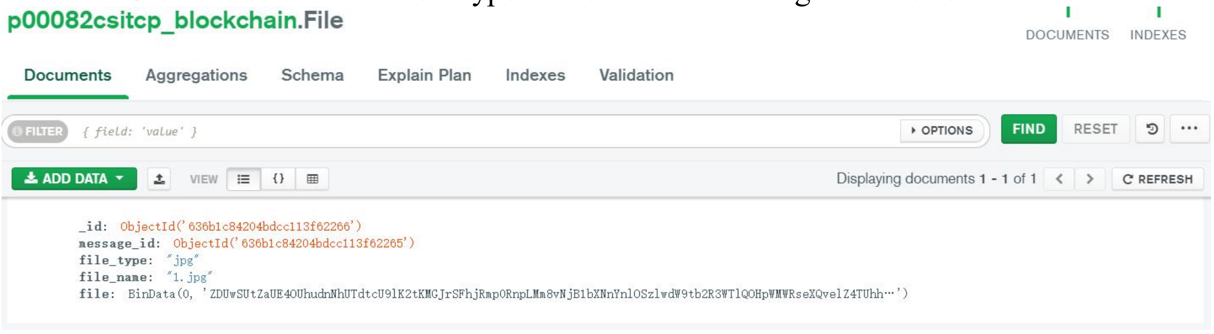
3. Create user accounts and save with RSA key pair in MongoDB and retrieve them



```
_id: ObjectId('6367cd345e53f5fb0703ecad')
user_id: 2
username: "alice"
password: "6ca13d52ca70c883e0f0bb101e425a89e8624de51db2d2392593af6a84118090"
wallet_key: "bd8715cb599a70f56e113976831be145a9df246685e120222fd06060cf68b95d"
email: "1@qq.c"
phone_number: "1234"
photo: BinData(0, 'LzlqLzJ3QkRBQXdKQ2dzS0VRMFhGUmNPRGcwUEV5QVZFeElTRXljY0hoY2dMaWt4TUM0cExTd3pPa28rTXpaR055d3RRRmRCUmt4…')
key_length: 1024
private_key: BinData(0, 'TFMwdExTMUNSVWRKVGlCU1UwRWdVRkpKVmtGVVJTQkxSVmt0TFMwdExRcE5TVWxEV0VGSlFrRkJTMEpuVVVOeGFrMUlNbWgwY1VO…')
private_key_location: "C:/Users/Jakll/Desktop/alice_p/my_private_key.pem"
> message: Array

_id: ObjectId('6367ce7e5e53f5fb0703ecb1')
user_id: 1
username: "bob"
password: "6ca13d52ca70c883e0f0bb101e425a89e8624de51db2d2392593af6a84118090"
wallet_key: "8e4ac4cbdd15afb0191fd03f8ddc6508602fd8155e689f3a6c6c990762d499c6"
email: "1@qq.c"
```

Figure 63. MongoDB user data

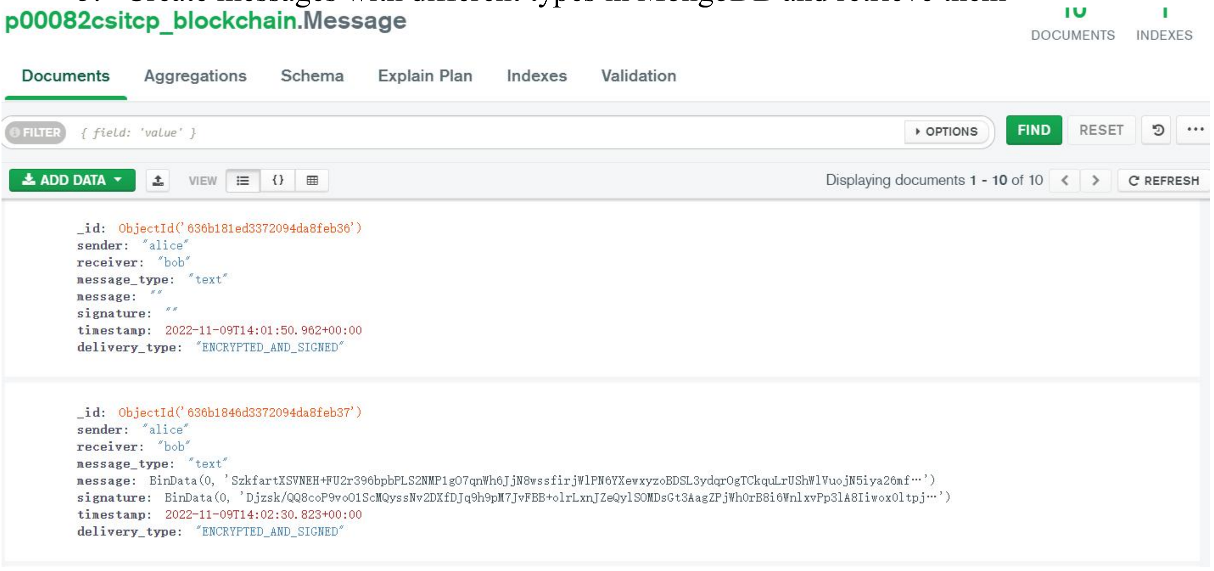
4. Create files with different types for each file in MongoDB and retrieve them



```
_id: ObjectId('636b1c84204bdcc113f62266')
message_id: ObjectId('636b1c84204bdcc113f62265')
file_type: "jpg"
file_name: "1.jpg"
file: BinData(0, 'ZDUwSUtZaUE4OUhudnNhUTdtcU9lK2tKMGJrSFhjRmp0RnpLMm8vNjB1bXNnYnl0SzlwdW9tb2R3WTlQOHpWMWRseXQvelZ4TUhh…')
```

Figure 64. MongoDB file data

5. Create messages with different types in MongoDB and retrieve them



```
_id: ObjectId('636b181ed3372094da8feb36')
sender: "alice"
receiver: "bob"
message_type: "text"
message: ""
signature: ""
timestamp: 2022-11-09T14:01:50.962+00:00
delivery_type: "ENCRYPTED_AND_SIGNED"

_id: ObjectId('636b1846d3372094da8feb37')
sender: "alice"
receiver: "bob"
message_type: "text"
message: BinData(0, 'SzkfartXSVNEH+FU2r396bpbPLS2NMP1g07qnWh6JjN8wssfirjWlPN6YXewxyzoBDSL3ydqrOgTCkquLrUShWlVuojN5iya26mf…')
signature: BinData(0, 'Djzsk/QQ8coP9vo01ScMQyssNv2DXfDJq9h9pM7JvFBB+olrLxnJZeQylSOMDsGt3AagZPjWh0rB8i6WnlxvPp3lA8Iiwox0ltpj…')
timestamp: 2022-11-09T14:02:30.823+00:00
delivery_type: "ENCRYPTED_AND_SIGNED"
```

Figure 65. MongoDB message data

9 OTHER CONSIDERATIONS

Although we completed our project on schedule, it still has certain flaws, such as an insufficiently attractive interface and an imperfect database. Our lack of time to update our projects is primarily to blame for these deficiencies. Our crew was initially tasked with completing a CA certification system, thus we decided to create a user certificate certification system. Customers then requested the addition of information encryption and transmission capabilities, necessitating the addition of encryption and interaction capabilities between students. Later, we were instructed to use cloud databases, so we experimented with other databases before selecting MongoDB to complete the project. Although it took us a considerable amount of time, we ultimately accomplished our assignment.

10 REFERENCES

1. Gamage HTM, Weerasinghe HD and Dias NGJ (2020) 'A survey on blockchain technology concepts, applications, and issues', *SN Computer Science*, *1*(2), pp.1-15.
2. Kolb J, AbdelBaky M, Katz RH and Culler DE (2020) 'Core concepts, challenges, and future directions in blockchain: A centralized tutorial', *ACM Computing Surveys (CSUR)*, *53*(1), pp.1-39.
3. Nguyen, N. 2021, *WhatsApp, Signal, Telegram and iMessage: Choosing a Private Encrypted Chat App; After a user-policy change and a social-media crackdown, independent messaging apps Signal and Telegram are experiencing a surge in downloads*, New York, N.Y.
4. Bernard, M. 2021, *Why Use Blockchain Technology?*, viewed 21 Oct 2022, < https://bernardmarr.com/why-use-blockchain-technology/>