

VIETNAM NATIONAL UNIVERSITY HOCHIMINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY
FACULTY OF COMPUTER NETWORKS AND
COMMUNICATIONS

TRẦN THỊ MỸ HUYỀN - TRƯƠNG LONG HÙNG - NGÔ TUẤN KIẾT

CRYPTOGRAPHY PROJECT REPORT
A MULTI – FACTOR AUTHENTICATION SCHEME
FOR CLOUD APPLICATIONS

HO CHI MINH CITY, 2023

VIETNAM NATIONAL UNIVERSITY HOCHIMINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY
FACULTY OF COMPUTER NETWORKS AND
COMMUNICATIONS

TRẦN THỊ MỸ HUYỀN - 21520269

TRƯỜNG LONG HÙNG - 21520903

NGÔ TUẤN KIẾT - 21521034

CRYPTOGRAPHY PROJECT REPORT
A MULTI – FACTOR AUTHENTICATION SCHEME
FOR CLOUD APPLICATIONS

PROJECT ADVISOR

PhD. NGUYỄN NGỌC TỰ

HO CHI MINH CITY, 2023

TABLE OF CONTENTS

TABLE OF CONTENTS.....	1
LIST OF FIGURES	3
LIST OF TABLES	5
LIST OF ABBREVIATIONS.....	6
Chapter 1. PROBLEM CONTEXT	7
1.1. System analysis	7
1.1.1. Client – Server architecture	7
1.1.2. Authentication	8
1.2. Related parties	8
1.2. Security features	9
Chapter 2. SOLUTIONS	10
2.1. Solutions overview	10
2.2. How users verify server	10
2.3. How server verify users	11
2.3.1. User registration	12
2.3.1. User grant access (login and privilege)	14
2.4. Protect data	19
2.4.1. Protect store data on database	19
2.4.2. Protect root key	21
2.5. Secure session.....	21
Chapter 3. IMPLEMENTATION	23
3.1. Overview implementation scenarios	23
3.2. First implementation plan.....	23

3.2.1.	Hardware and software overview	23
3.2.2.	System configurations	24
3.2.3.	Implementation Details	25
3.2.4.	Demonstration Results	35
3.3.	Second Implementation Plan	47
3.3.1.	Hardware and Software Overview	47
3.3.2.	System configurations	48
3.3.3.	Demonstration result	48
3.4.	Third Implementation Plan	49
3.4.1	Hardware and Software Overview	49
3.4.2.	System configurations	49
3.3.4.	Demonstration result	50
Chapter 4.	SUMMARY	52
4.1	Evalute result.....	52
4.2	Tasks.....	52
REFERENCES	54

LIST OF FIGURES

Figure 1: Client – Server architecture	7
Figure 2: Related parties and vulnerabilities	8
Figure 3: Proposed scheme	12
Figure 4: User registration.....	13
Figure 5: User factor creation.....	13
Figure 6: Verify code creation	14
Figure 7: OTP creation (Time-based OTP).....	15
Figure 8: Login intrusion detection	16
Figure 9: Grant privilege	18
Figure 10: Recover password process.....	18
Figure 11: TLS version 1.3	22
Figure 12: Implementation system.....	24
Figure 13: Code snap of Register function	26
Figure 14: Code snap of Login function	31
Figure 15: Code snap of Forget function	33
Figure 16: Code snap of generate user factor function	35
Figure 17: Code snap of generate TOTP function	35
Figure 18: User Interface when run application	36
Figure 19: Register Form	37
Figure 20: Register successfully announcement.....	37
Figure 21: Login form.....	38
Figure 22: OTP form.....	39
Figure 23: OTP sent announcement.....	39
Figure 24: Confirm login successfully notification	40
Figure 25: Confirm user privilege notification	40
Figure 26: Forget password form.....	41
Figure 27: Create new password form	41
Figure 28: Password change notification	42

Figure 29: Verify code form	42
Figure 30: Change stored IP address	43
Figure 31: Change user privileges form.....	44
Figure 32: Change user privileges successfully notification	44
Figure 33: Database after change user privileges	45
Figure 34: Delete user form	45
Figure 35: Delete user successfully notification	45
Figure 36: Database after delete user ‘tlhung’	46
Figure 37: Unlock user form.....	46
Figure 38: Unlock user ‘hehe’ successfully notification	47
Figure 39: Database before unlocking user ‘hehe’	47
Figure 40: Database after unlocking user ‘hehe’	47
Figure 41: Second Implementation system.....	48
Figure 42: Third Implementation system.....	49
Figure 43: Sign up and sign in UI	50
Figure 44: OTP user interface	51

LIST OF TABLES

Table 1: Solutions overview	10
Table 2: User permissions	13
Table 3: Suspicious table structure.....	19
Table 4: Algorithms.....	19
Table 5: Database structure	20
Table 6: Result overview.....	52
Table 7: Tasks table.....	53

LIST OF ABBREVIATIONS

ABBREVIATIONS	MEANING
AES	Advanced Encryption Standard
CA	Certificate Authority
ECC	Elliptic-curve Cryptography
HMAC	Hash-based Message Authentication Code
MFA	Multi-factor Authentication
NIST	National Institute of Standards and Technology
OTP	One-Time Password
PCR	Platform Configuration Register
SHA	Secure Hash Algorithms
SSL	Secure Socket Layer
TLS	Transport Layer Security
TOTP	Time-based One-time Password
TPM	Trusted Platform Module
UI	User Interface
VPC	Virtual Private Cloud

Chapter 1. PROBLEM CONTEXT

1.1. System analysis

With a higher degree of processing provided by the client-server network paradigm, workstation power, workgroup empowerment, remote network management, market-driven business, and the preservation of existing investments are all made more effective, Client - Server architecture offers the precise structure that today's businesses require to handle the demands of a continuously changing IT world.

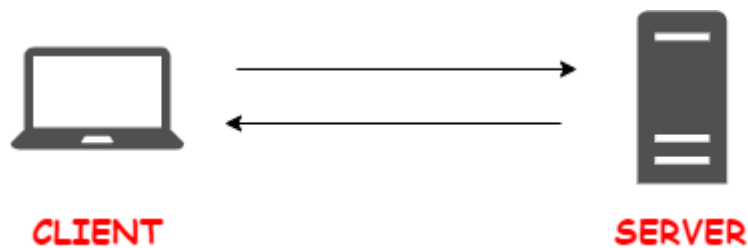


Figure 1: Client – Server architecture

1.1.1. Client – Server architecture

The client-server architecture refers to a system that hosts, delivers, and manages most of the resources and services that the client requests. In this model, all requests and services are delivered over a network, and it is also referred to as the networking computing model or client server network.

Given the sensitivity of the data they guard, it doesn't take much imagination to realize that local devices might not be as safe as they ought to be. Numerous risk concerns, such as natural disasters or other external events, can affect local storage that is kept on-site. The cloud migration of your services is one technique to secure your data from such risk factors. Such as hosting your services on a cloud server.

Cloud servers work just like physical servers, and they perform similar functions like storing data and running applications.

1.1.2. Authentication

As organizations need to control which individual users have access to their resources, authentication enables organizations to keep their networks secure by permitting only authenticated users or processes to gain access to their protected resources. This may include computer systems, networks, databases, websites and other network-based applications or services.

The classic authentication scheme is clients entered their login information. The username and password are sent to the cloud server by the client application. Then the server accesses the user database and verifies that the password and username are accurate. If so, the server gives the client application the go-ahead to permit the user to log in.

1.2. Related parties

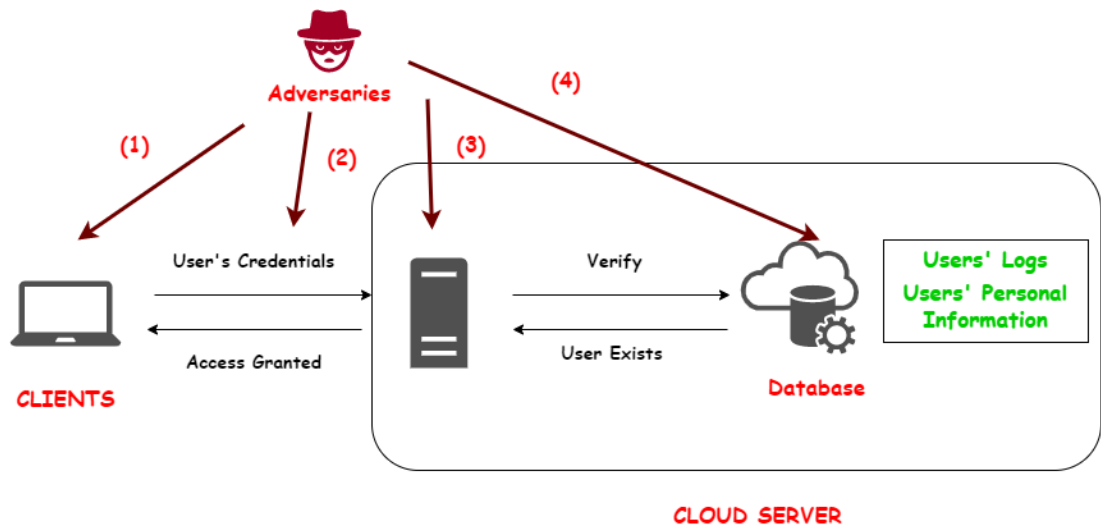


Figure 2: Related parties and vulnerabilities

Because the user's credentials are the most important factor to use for an authentication procedure, credential protection — the process of securing usernames, passwords, and other credential data against unauthorized access and misuse — is required.

Nowadays, the classic authentication scheme is no longer sufficiently safe. Hackers have created various tried-and-true techniques for stealing credentials and getting illegal access to personal accounts, ranging from straightforward relaying and spraying attacks to the more sophisticated dangers of spear-phishing and pharming.

As presented in Fig 1.2, the (1) arrow represents an *Impersonation attack*, in which involves a hacker pretending to be a valid user on the system. The (2) arrow indicated a *Man-in-the-middle attack*, in which the attacker surreptitiously transmits and maybe modifies communications between two parties who believe they are speaking directly with each other as the attackers have put themselves between the two parties. The (3) arrow indicates that attackers can also impersonate the server. The (4) arrow represents the vulnerability of outsiders who obtain credentials by social engineering or other means, or gain access to the database's credentials.

1.2. Security features

To identify the best solution to all the vulnerabilities that traditional authentication can cause, several security features must be included:

1. Client can validate the Server.
2. Server can validate the Client.
3. Secure client-server communication.
4. Protect sensitive data.

The detailed solutions are provided in the following chapter.

Chapter 2. SOLUTIONS

2.1. Solutions overview

As presented before, our scheme is proposed for maintaining the identity of users as well as server, providing a secure session for client and server to communicate, eliminating malicious participants' threats, and obtaining the privacy and confidentiality of data.

The framework authenticates, authorizes, controls and audit user's behavior with the interaction of instruction detection mechanisms. The scheme used TLS (version 1.3) to set up a secure connection and AES (Advanced encryption standard), hash algorithm to protect the data during transmission to overcome adversaries' attack.

Goals	Solutions
Users verify servers	Verify server's CA certificate (TLS handshake)
Server verify users	Using proposed scheme
Protect data	vTPM, AES - GCM, AES – CTR, Argon2id Hash
Secure session	Using TLS version 1.3

Table 1: Solutions overview

2.2. How users verify server

To ensure the validity of the server's certificate, the following steps are followed, employing TLS version 1.3:

The client initiates a TLS handshake with server by sending a ClientHello message to server. By the time server receives that message, it will respond with a ServerHello message, which includes the server's selected cipher suite and its digital certificate.

The client receives and checks its validity with three requirements:

- The certificate's signature must be ensured that it was issued by a trusted certificated authority (CA)
- The certificate is not expired and is within its validity period.
- The certificate's chain of trust must be valid by checking if the issuer's certificate is trusted.

Verifying the server is an essential step in establishing a secure connection between a client and a server. The server verification process serves several important purposes:

- **Authentication:** By verifying the server's identity, the client ensures that it is connecting to the intended server and not an imposter or an attacker attempting a man-in-the-middle attack.
- **Trust:** Verifying the server's certificate establishes trust in the server's identity. The certificate is issued by a trusted certificate (CA), which attests to the server's authenticity.

Server verification is crucial for establishing a secure and trusted communication channel between clients and server. It helps ensure the authenticity of the data exchanged, protecting against unauthorized access, attack, and potential security threats.

2.3. How server verify users

The aim of the proposed scheme is to ensure secure user registration and access to the application while maintaining the integrity and confidentiality of user accounts and data. Furthermore, establishing a robust and secure access control mechanism, protecting user accounts from unauthorized access, and ensuring the overall security of the application.

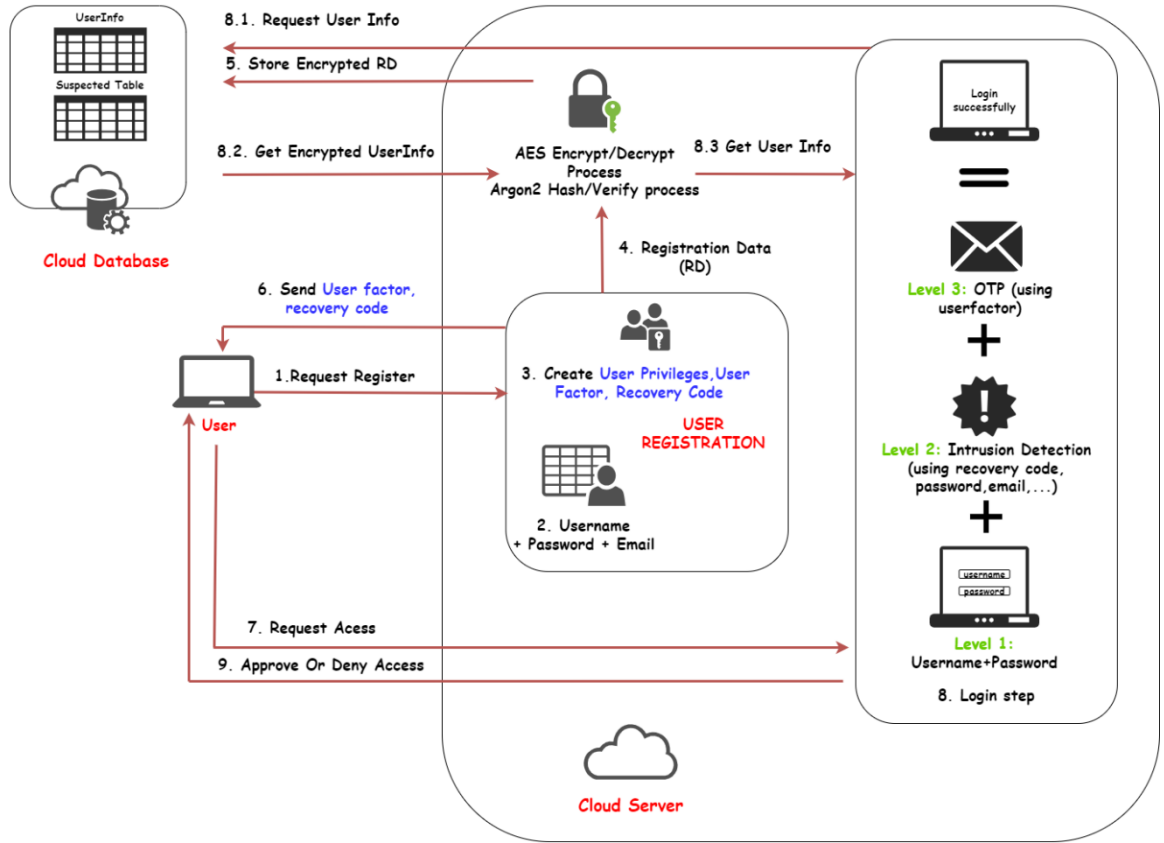


Figure 3: Proposed scheme

2.3.1. User registration

In this phase the necessary procedures for user creation are initiated and the necessary attributes needed for a user to access the application are created. It is divided into three main stages:

- **User creation:** The user initiates the registration process by providing their personal attributes and creating a user account.
- **User factor:** A user factor is generated based on predefined user privileges and random attributes. This factor contributes to the user's overall security profile.
- **Users verify code:** A verification code is generated based on the user factor. This code serves as an additional layer of verification during authentication and helps ensure the user's identity.

2.3.1.1. User creation:

As indicated, users request access to the server to use cloud resources.

It can be performed step by step like this:

After the user's request, the application will bring you to the register panel that requires you enter your username, password and email and check their validity. In case all attributes are correct. User factor and recovery code will be created (will be described in next part)

Every user will be anchored with a privilege that allow or refuse user to do something (admin and normal)

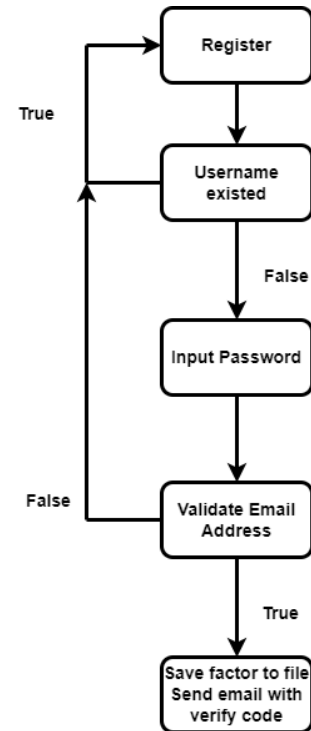


Figure 4: User registration

id	role	delete_user	search_data	insert_data	update_data	delete_data
1	admin	1	1	1	1	1
2	normal	0	1	1	1	1

Table 2: User permissions

2.3.1.2. User factor

During the registration phase, a user factor will be created. It is unique for each user in a session.

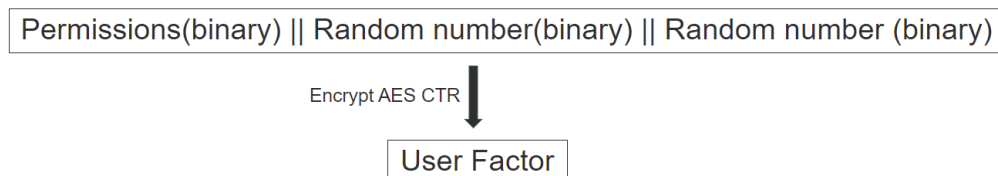


Figure 5: User factor creation

It converts the values of the permissions dictionary into binary strings and concatenates them together, along with two random 8-bit values. After that, it encrypted using AES-CTR mode and returned as a hexadecimal string.

2.3.1.2. Verify code:

Verify code is easily randomly generated from User factor using its 6 random characters.

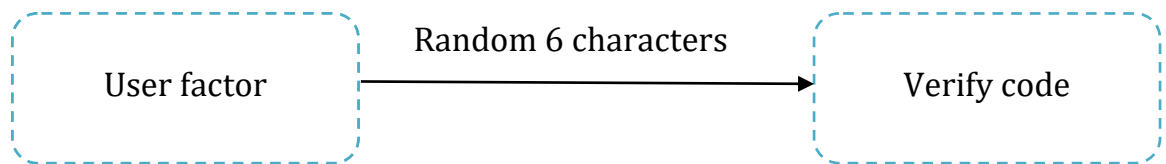


Figure 6: Verify code creation

This recovery code is important because it is used every time you login from unknow locations, forget password, ... and regenerate every time it is used.

2.3.1. User grant access (login and privilege)

In this phase the necessary procedures for granting a user to path throw and access the application are determined. It depends on four main components. These are One-Time Password (OTP) mechanism, intrusion detection and suspected table.

- **OTP mechanism:** By granting the user to access the application, the user must pass throw OTP step to enter the privilege that user registered before.
- **Intrusion detection:** To pass throw login step, user must match all requirement of intrusion detection to enter to registered privilege.
- **Suspected table:** Record all suspicious access to server.

2.3.1.1. OTP mechanism

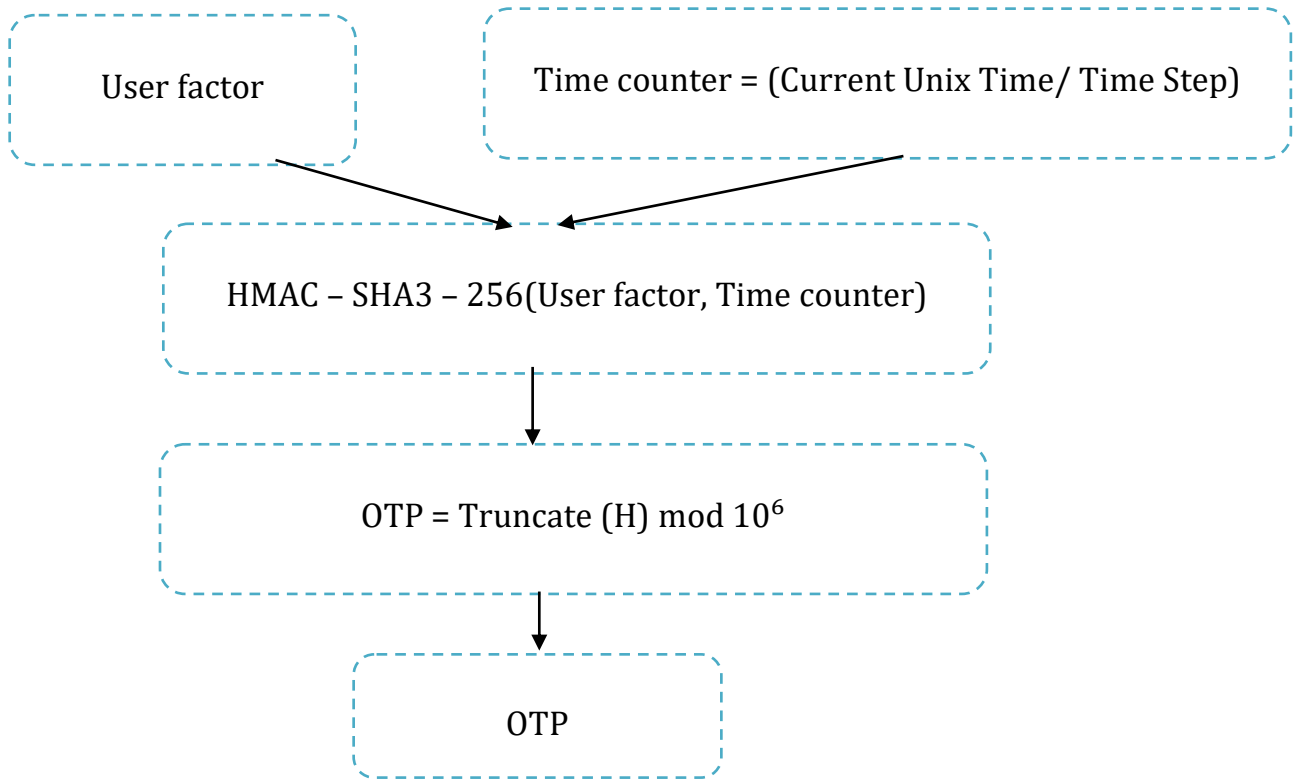


Figure 7: OTP creation (Time-based OTP)

The scheme provides an OTP mechanism that was agreed between client and server based on secret (User factor). One-Time Password (OTP) using the Time-Based One-Time Password (TOTP) algorithm.

1. First, it obtains the current time, represented as the number of seconds since the Unix epoch. The function then defines a time interval of 30 seconds and calculates the number of time steps based on the current time divided by the interval. The time steps are packed into a byte string as a 64-bit big-endian unsigned integer, and the secret (User factor) is encoded into bytes (ASCII encoding).
2. Next, an HMAC-SHA3-256 hash is generated by combining the secret bytes and time steps bytes. Then calculating an offset by extracting the least significant 4 bits from the last byte of the hash. The offset is used to determine a 4-byte code within the hash.

3. To obtain the actual OTP, the code bytes are unpacked as an unsigned integer. This integer is then masked to 31 bits, discarding the most significant bit, and finally, modulo 1,000,000 is applied to ensure a 6-digit OTP. The resulting OTP is formatted as a string. OTP is regenerated after every 60 seconds.

OTP is generated by considering the current time, time steps, and a secret (User factor). It uses HMAC-SHA3-256 to generate a hash, extract a code based on an offset, apply masking and modulo operations to obtain a 6-digit OTP, and return it as a string. This OTP generation process ensures that the generated passwords are time-based and unique for every time step, providing an additional layer of security for authentication systems.

2.3.1.2. Intrusion detection:

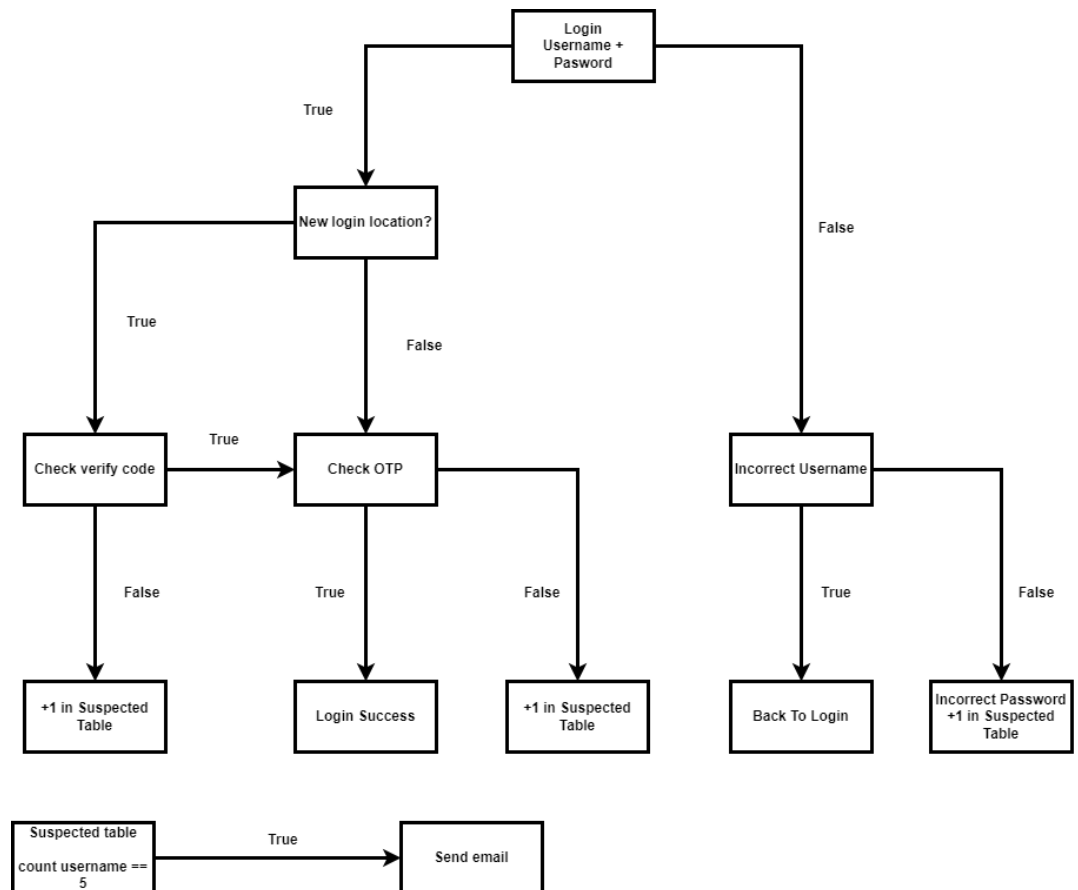


Figure 8: Login intrusion detection

Intrusion detection provides a scheme to prevent threats accessing to application.

- When a user requests a login to server, server requires user to provide username and password. First it checks the username. If it exists in database or not. In case username exist but the provided password didn't match, it will record this activity as malicious and save it into suspected table.
- When username and password all matched, server verifies client login's location by checking its IP address. In this case, if user login at a new location, make sure provided true verify code to make sure that's user which was granted access. If not, the user will go directly to OTP step.
- The last step is checking OTP which is agreed between server and client.
- During these steps, servers also check how many times the user recorded as malicious. If it's over 5 times, the user will be locked and receive an inform email.

Overall, except username, every time user mismatches a required information (verify code, password, OTP), it will be recorded as malicious access. Intrusion detection serves as an early warning system, alerting security teams or administrators about potential security breaches or vulnerabilities. Timely detection allows for prompt response and mitigation measures, minimizing the potential impact of an intrusion or attack.

When a login session success, application will check User factor to extract privilege (first part extract from user factor – Figure 5) and provide you features corresponding to user permission.

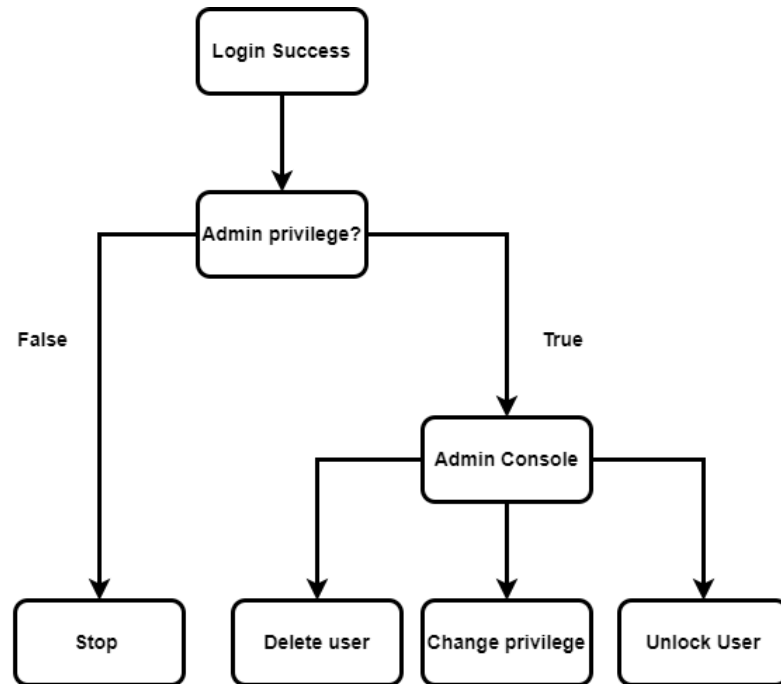


Figure 9: Grant privilege

The proposed scheme also provides a mechanism for users to recover password if it is forgotten by using verify code.

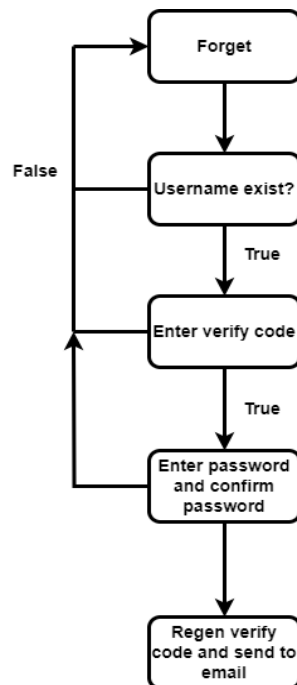


Figure 10: Recover password process

The suspected table is a record of malicious request to server (wrong password, OTP, verify code). It will pretend application from threats and some attack like brute-force.

Username	Email	Ip address	Log time
User name record	Email record	Ip record	Time record

Table 3: Suspicious table structure

2.4. Protect data

As illustrated in Figure 3, the proposed scheme encompasses robust measures to ensure the security of user data generated during the registration and access granting phases. Additionally, it places great emphasis on safeguarding the root key stored on the server, a crucial component of the overall security infrastructure.

By employing a combination of encryption and access control mechanisms, the scheme aims to protect the confidentiality, integrity, and availability of user data. During the registration phase, sensitive user attributes, such as passwords and personal information, are securely stored using encryption and hash algorithms. This ensures that even if unauthorized access to the server occurs, the data remains unintelligible and inaccessible to potential attackers.

2.4.1. Protect store data on database

There are two choices in protecting store data that we choose is hash and encrypt.

Hash (Argon2id)	Password, verify code
Encrypt (AES-GCM)	Email, user factor

Table 4: Algorithms

Why hashing password and verify code?

- When it comes to passwords and verify codes, it is generally preferred to use hashing rather than encryption.

- Hashing is designed to be irreversible, meaning that once the data is hashed, it is computationally infeasible to retrieve the original input. This property is desirable for passwords and verify codes because it adds an extra layer of security. Even if an attacker gains access to the hashed passwords or verify codes, they cannot easily reverse-engineer them to obtain the original values.
- When a user registers or sets a password, the password is hashed using a secure hashing algorithm argon2id. During the login or verification process, the entered password or verify code is hashed again using the same algorithm, and the resulting hash is compared with the stored hash. If the hashes match, it indicates that the entered password or verify code is correct.

Why encrypting email and user factor?

- Unlike hashing, encryption is a reversible process. It makes encryption suitable for scenarios where the original data needs to be retrieved in a usable format for further use case.
- Email addresses and user factors often contain sensitive information that needs to be stored and transmitted securely. By encrypting this data using a strong encryption algorithm like AES-GCM, unauthorized parties are unable to read or modify the encrypted information even if they gain access to the stored data.

Username	Password	Email	Role	Factor	Ip address	Recovery code	Status
Raw	Encrypted	Encrypted	Raw	Encrypted	Raw	Hashed	Raw

Table 5: Database structure

By using a combination of hashing for passwords and verify codes (irreversible and secure) and encryption (AES-GCM) for email and user factor

(reversible and secure), the chosen approach provides an effective balance between security and usability, ensuring that sensitive data is properly protected in the database.

2.4.2. Protect root key

Proposed scheme using Trusted Platform Module (TPM) to establish a trusted platform by measuring and storing platform integrity measurements in PCR registers. These measurements include information about the system's firmware, bootloader, and other critical components. The root key can be bound to these measurements, ensuring that it can only be accessed when the system's integrity is verified. This protects the root key from being used in compromised or tampered environments.

TPM helps protect the root key against various attacks, such as unauthorized extraction, tampering, or brute-force attacks. The root key can be securely sealed within the TPM, and its use can be restricted through policies, making it challenging for attackers to access or misuse the key.

2.5. Secure session

To achieve a secure session, proposed scheme using TLSv1.3 with certificate that signed by CA (ZeroSSL) and configured on server side. This typically involves associating the private key (NIST P-256 ECC) generated during the CSR generation with the issued certificate.



Figure 11: TLS version 1.3

When the user requests a connection to server, the TLS handshake process begins. The client and server exchange a series of messages to establish a secure connection. During the handshake, the server presents the SSL/TLS certificate to the client as proof of its identity (2.1)

The client and server perform the key share process to establish session keys for secure communication. This involves encrypting and exchanging cryptographic material to generate the shared secret key.

With the session keys established, the client and server can securely communicate using encryption and integrity protection.

Chapter 3. IMPLEMENTATION

3.1. Overview implementation scenarios

The demo scenario for this multi-factor authentication scheme involves a user attempting to log in to an application using their username and password, followed by a one-time password (OTP) generated.

The user will initially log in to the application by entering their username and password as they would typically do for a standard login process. The user will then be prompted by the application to enter a one-time password, which they will locate on their device.

The application will send the OTP to the cloud server for verification after the user enters the OTP into it. The user will subsequently be given access to the application if the OTP is legitimate after the cloud server has compared it to the user's credentials that are saved in the database.

Overall, the demo scenario shows how the multi-factor authentication scheme with a client, cloud server, and database functions in real life.

3.2. First implementation plan

3.2.1. Hardware and software overview

The proposed multi-factor authentication (MFA) scheme involves three nodes: the client, the cloud server, and the database. The cloud server is an Azure Virtual Machine running Ubuntu 20.04 in a virtual private cloud (VPC) environment, and the client is a laptop running Windows 10. The database is hosted on ElephantSQL. The MFA scheme is implemented using Python programming language and the following libraries: OpenSSL and Tpm2tools.

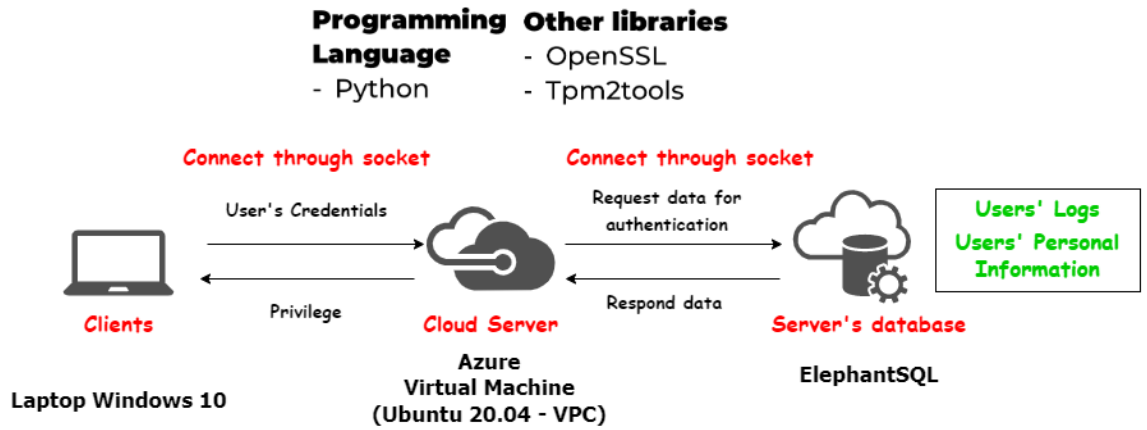


Figure 12: Implementation system

3.2.2. System configurations

3.2.2.1. Cloud Server configurations

The cloud server is configured with Ubuntu 20.04 as the operating system and is allocated sufficient resources to run the necessary software and handle incoming requests from the client.

3.2.2.2. Database configurations

To implement our system, we designed a database to store user information and permissions. Because of PostgreSQL's dependability, scalability, and support for sophisticated queries, we decided to adopt it as our database management system.

Using **ElephantSQL** to host the database, which is set up to transmit data via secure communication methods. A username and password are used to authenticate access to the database, and access control policies are put up to limit access to only approved users.

Three tables make up our database: **suspiciousTable**, **userInfo**, and **rolePermissions**. User roles and the related permissions are kept in the **rolePermissions** table. User account details, such as usernames, passwords, and email addresses, as well as the user's role and status, are kept in the **userInfo** table.

The suspiciousTable table logs suspicious login attempts and includes information about the user's IP address and login time.

3.2.2.3. Client configurations

To use our multi-factor authentication (MFA) system, clients will need to install Python and other required libraries on their Windows 10 laptop.

3.2.3. Implementation Details

3.2.3.1. Register Function

The register function is responsible for registering new users into the system. A user must enter their username, password, and email while registering.

First, the register function verifies the user's entry to make sure their username and email are distinct. The function asks the user to try again if the input is not original. The user's password is then hashed to prevent password compromise.

The register function also generates a unique authentication factor for each user, which is used to verify their identity during login. Together with the user's other data, this authentication factor is safely saved in the database.

Finally, for each user, the register functions a recovery code that can be used to regain access to their account in the event that they misplace their authentication factors. The user receives a recovery code through email, which must be kept confidential.

```
def register(client_socket):
    ph = PasswordHasher()
    client_socket.send("Username: ".encode())
    username = client_socket.recv(1024).decode()

    client_socket.send("Password: ".encode())
    password = client_socket.recv(1024).decode()

    client_socket.send("Email: ".encode())
    email = client_socket.recv(1024).decode()
    try:
        cur = conn.cursor()
        cur.execute("SELECT * FROM userInfo WHERE username = %s", [username])
        data = cur.fetchall()[0]
        if(len(data) > 0):
```

```

        client_socket.send("Username already existed! Try
again.".encode())
        register(client_socket)
    except:
        pass

    #check valid email format
    if(re.fullmatch(regex, email)):
        #check existed email:
        cur = conn.cursor()
        cur.execute("SELECT email FROM userInfo")
        data = cur.fetchall()
        for mail in data:
            tup = mail[0]
            if(getDecryptData(ast.literal_eval(tup)) == email):
                client_socket.send("Your email already been used. Try
again".encode())
                register(client_socket)

        hashpass = ph.hash(password)
        cur.execute("INSERT INTO userInfo (username,password,email,role)
VALUES (%s, %s, %s, %s)", [username,hashpass,str(encrypt(email)),"admin"])
        conn.commit()

    #factor, recoverycode, ip of client

    factor = generateFactor(username)
    encfactor = str(encrypt(str(factor)))
    recoveryCode = getRecoveryCode(factor[0])
    print(recoveryCode)
    hashRecoveryCode = ph.hash(recoveryCode)
    client_ip = client_socket.getpeername()[0]
    print("CURRENT USER FACTOR: ",factor[0])

    cur.execute("UPDATE userInfo SET factor = %s , recoverycode = %s,
ipaddress = %s WHERE username = %s",[encfactor, hashRecoveryCode, client_ip,
username])
    conn.commit()
    cur.close()
    client_socket.send(("FACTOR:" + username + '/'+'(' +
str(factor[0])+')').encode())
    client_socket.send(("Register successfully! Please remember this
recovery code:" + recoveryCode).encode())
    sendRecoveryCode(email, recoveryCode)
    else:
        client_socket.send("Your email is not valid. Try again".encode())
    return

```

Figure 13: Code snap of Register function

3.2.3.2. Login Function

The login function is in charge of verifying the identity of users who want to access the system. A user must enter their username, password, and a one-time

password (OTP) produced using the time-based one-time password (TOTP) algorithm to log in.

The user is first prompted to input their username and password by the login function. The function returns an error message and asks the user to try again if the username or password are invalid. If the user's account has been locked because of suspicious conduct, the function returns an error message and advises the user to get in touch with the administration.

The login function then confirms that the user's IP address matches the IP address that is recorded in the database. The function asks the user to input a recovery code if the IP address does not match before allowing them to proceed. If the recovery code is invalid, the function returns an error message and adds the user to the suspicious table.

The login function then prompts the user to enter an OTP code after the IP address check is successful. After that, the function verifies that the OTP matches the TOTP produced by the user's authentication factor. If the OTP is invalid, the function returns an error message and adds the user to the suspicious table.

After successfully verifying the user's OTP, the login function examines the user's factor and role to determine their permissions. The function also checks if the user role has been modified by the third party through user's factor.

```
def login(client_socket):
    ph = PasswordHasher()
    client_socket.send("Username: ".encode())
    username = client_socket.recv(1024).decode()

    client_socket.send("Password: ".encode())
    password = client_socket.recv(1024).decode()
    #check Invalid Username
    if (checkUsername(username) == False): # doesn't exist
        client_socket.send("Invalid username. Please try again.".encode())
        return login(client_socket)

    #Check incorrect Password

    cur = conn.cursor()
    cur.execute("SELECT status FROM userInfo WHERE username = %s",
[username])
    data = cur.fetchall()[0][0]
```

```

    if (int(data) == 1):
        client_socket.send("Your account has been locked. Please contact
adminstrators for more information.".encode())
        cur.close()
        return

    cur.execute("SELECT password FROM userInfo WHERE username = %s",
[username])
    data = cur.fetchall()[0][0]
    try:
        verifyValid = ph.verify(str(data),password)
    except:
        cur.execute("INSERT INTO suspiciousTable
(usernameSussy,ipaddress,logtime) VALUES (%s,%s,%s)",
[username,client_socket.getpeername()[0],datetime.now()])
        conn.commit()
        cur.execute("SELECT COUNT(*) FROM suspiciousTable WHERE usernameSussy
= %s", [username])
        count = cur.fetchone()[0]
        cur.execute("SELECT email FROM userInfo WHERE username = %s",
[username])
        data = ast.literal_eval(cur.fetchone()[0])
        gmailofSussy = getDecryptData(data)
        client_socket.send("Invalid Password, you are added into suspicious
table".encode())
        client_socket.send(("User added to suspiciousTable count: " +
str(count)).encode())
        sendEmail(count,gmailofSussy)
        if (count >= 5):
            client_socket.send("Your account has been locked. Please contact
adminstrators for more information.".encode())
            try:
                cur = conn.cursor()
                cur.execute("""UPDATE userInfo
SET status = 1
WHERE username = %s""", [username])
                conn.commit()
                cur.close()
                return
            except:
                return
        else:
            return

    #check valid ip, if new have to auth
    client_ip = client_socket.getpeername()[0]
    cur.execute("SELECT ipaddress, recoverycode FROM userInfo WHERE username
= %s", [username])
    ans=cur.fetchone()

    ipofusername =ans[0]
    storedRecoveryCode = ans[1]

    if ipofusername and ipofusername != client_ip:
        client_socket.send("Login IP does not match the stored IP address.
Please enter recoverycode to continue.".encode())
        recoveryCode = client_socket.recv(1024).decode()
        try:
            verifyValid = ph.verify(str(storedRecoveryCode),recoveryCode)

```

```

        client_socket.send("Change login location".encode())
    except:
        client_socket.send("Invalid recovery code. Please try login
again.".encode())
        cur.execute("INSERT INTO suspiciousTable
(usernameSussy,ipaddress,logtime) VALUES (%s,%s,%s)",
[username,client_ip,datetime.now()])
        conn.commit()
        cur.execute("SELECT COUNT(*) FROM suspiciousTable WHERE
usernameSussy = %s", [username])
        count = cur.fetchone()[0]
        cur.execute("SELECT email FROM userInfo WHERE username = %s",
[username])
        data = ast.literal_eval(cur.fetchone()[0])
        gmailofSussy = getDecryptData(data)
        client_socket.send(("User added to suspiciousTable count: " +
str(count)).encode())
        sendEmail(count,gmailofSussy)
        cur.close()
        return

    # Retrieve the updated user information

    factor = generateFactor(username)
    storedRecoveryCode = getRecoveryCode(factor[0])
    encRecvCode = ph.hash(storedRecoveryCode)
    encFactor = str(encrypt(str(factor)))

    cur.execute("UPDATE userInfo SET recoverycode = %s, factor =
%s,ipaddress = %s WHERE username = %s",
[encRecvCode,encFactor,client_ip,username])
    conn.commit()

    client_socket.send(("FACTOR:" + username + '/'+'(' +
str(factor[0])+')').encode())
    cur.execute("SELECT email FROM userInfo WHERE username = %s",
[username])
    data = ast.literal_eval(cur.fetchone()[0])
    email = getDecryptData(data)
    sendRecoveryCode(email, storedRecoveryCode)
    #start generating OTP as username + password are valid

    if (verifyValid == True):
        client_socket.send(("Start"+username).encode())
        client_socket.send("Input your OTP: ".encode())
        otp_code = client_socket.recv(1024).decode()
        factor = None
        try:
            cur = conn.cursor()
            cur.execute("SELECT factor FROM userInfo WHERE username = %s",
[username])
            data = ast.literal_eval(cur.fetchone()[0])
            factor = ast.literal_eval(getDecryptData(data))
            print("CURRENT USER FACTOR: ",factor[0])

        except Exception as e:
            print(e)
            return
        #user name exists or not

```

```

        if(otp_code == generate_totp(factor[0]) or otp_code
==generate_totp(factor[0],-1)):
            client_socket.send("Login complete!".encode())

            #get "database role"
            cur.execute("SELECT role FROM userInfo Where username = %s",
[username])
            role = cur.fetchone()[0]
            permission = getPermission(role)

            #check role using factor
            print(factor[1])
            if ((decryptFactor(factor[0],factor[1])[0:5]) == permission and
role == "admin"):
                factor = generateFactor(username)
                client_socket.send(("FACTOR:" + username + '/'+'('+
str(factor[0])+')').encode())
                encfactor = str(encrypt(str(factor)))
                #UPDATE NEW FACTOR INTO DB
                cur.execute("UPDATE userInfo SET factor = %s WHERE username =
%s", [encfactor,username])
                conn.commit()
                client_socket.send("You are recognized as admin
privilege".encode())

                return adminConsole(client_socket)

            elif (decryptFactor(factor[0],factor[1])[0:5] == permission):
                factor = generateFactor(username)
                client_socket.send(("FACTOR:" + username + '/'+'('+
str(factor[0])+')').encode())
                encfactor = str(encrypt(str(factor)))
                #UPDATE NEW FACTOR INTO DB
                cur = conn.cursor()
                cur.execute("UPDATE userInfo SET factor = %s WHERE username =
%s", [encfactor,username])
                conn.commit()
                client_socket.send("You are recognized as user
privilege".encode())
                return

            else:
                client_socket.send("Your role was changed by the third party.
Please contact admin of the service.")
                return

        else:
            cur.execute("INSERT INTO suspiciousTable
(usernameSUSPY,ipaddress,logtime) VALUES (%s,%s,%s)",
[username,client_ip,datetime.now()])
            conn.commit()
            cur.execute("SELECT COUNT(*) FROM suspiciousTable WHERE
usernameSUSPY = %s", [username])
            count = cur.fetchone()[0]
            cur.execute("SELECT email FROM userInfo WHERE username = %s",
[username])
            data = ast.literal_eval(cur.fetchone()[0])
            gmailofSussy = getDecryptData(data)
            client_socket.send("Invalid OTP code, you are added into
suspicious table".encode())

```



```

        client_socket.send(("User added to suspiciousTable count: " +
str(count)).encode())
        sendEmail(count,gmailofSussy)
        if (count >= 5):
            client_socket.send("Your account has been locked. Please
contact adminstrators for more information.".encode())
            try:
                cur = conn.cursor()
                cur.execute("""UPDATE userInfo
SET status = 1
WHERE username = %s""", [username])
                conn.commit()
                cur.close()
                return
            except:
                return
        else:
            return
    return

```

Figure 14: Code snap of Login function

3.2.3.3. Forget Function

When a user forgets their password, the forget function is in charge of resetting it. A user must enter their username and the recovery code they received when they first created their account to request a password reset.

First, the forget function prompts the user to enter their username and recovery code. If the recovery code is incorrect, the function gives an error message and adds the user to the suspect table. If the user's account has been locked because of suspicious conduct, the function returns an error message and prompts the user to contact the administrators.

The forget function then retrieves the user's stored recovery code, encrypted factor, and email from the userInfo table in the database. The PasswordHasher module is then used to check that the user-entered recovery code matches the recovery code that has been previously stored. If the recovery code is invalid, the function returns an error message and adds the user to the suspicious table.

After successfully verifying the user's recovery code, the forget function retrieves the user's database role from the userInfo table and checks whether the user's factor matches the appropriate permission level. The function will produces an

error message and advise the user to get in touch with the administrators if the user's role has been altered by a third party.

Then the forget function asks the user to input a new password and confirm it. After successfully verifying the new password, the forget function generates a new factor and recovery code using the generateFactor and getRecoveryCode functions. The function modifies the user's password, recovery code, and factor in the userInfo table after encrypting the new factor. Finally, the sendRecoveryCode function is used to send the new recovery code to the user's email address.

```
def forget(client_socket):
    ph = PasswordHasher()
    client_socket.send("Enter your username to recover: ".encode())
    username = client_socket.recv(1024).decode()

    client_socket.send("Enter your recovery code: ".encode())
    recoverycode = client_socket.recv(1024).decode()

    #get neccessary data
    cur = conn.cursor()
    cur.execute("SELECT recoverycode, factor, email FROM userInfo WHERE
username = %s", [username])
    result = cur.fetchone()
    storedRecoveryCode = result[0]
    print(storedRecoveryCode)

    #verify RecoveryCode
    try:
        verifyValid = ph.verify(storedRecoveryCode, recoverycode)
        cur = conn.cursor()
        cur.execute("SELECT role FROM userInfo Where username = %s",
[username])
        #check role from factor

        role = cur.fetchone()[0]
        permission = getPermission(role)
        data = ast.literal_eval(result[1])
        factor = ast.literal_eval(getDecryptData(data))
        print(factor)

        if ((decryptFactor(factor[0],factor[1])[0:5]) == permission):
            client_socket.send("Enter your new password: ".encode())
            newpassword1 = client_socket.recv(1024).decode()
            client_socket.send("Please confirm your password: ".encode())
            newpassword2 = client_socket.recv(1024).decode()
        else:
            client_socket.send("Your role was changed by the third party.
Please contact admin of the service.")
            return

        if newpassword1 != newpassword2:
```

```

        client_socket.send("Your password did not match. Please try
again".encode())
        return
    else:
        #extract factor and email when have verify success only
        email = getDecryptData(ast.literal_eval(result[2])).strip()

        #REGEN recv code + factor
        newfactor = generateFactor(username)
        client_socket.send(("FACTOR:" + username + '/'+'(' +
str(newfactor[0])+')').encode())
        encfactor = str(encrypt(str(newfactor)))
        newRecoveryCode = getRecoveryCode(newfactor[0])
        cur = conn.cursor()
        cur.execute("UPDATE userInfo SET password = %s, recoverycode =
%s,factor = %s WHERE username = %s", [ph.hash(newpassword1),
ph.hash(newRecoveryCode),encfactor, username])
        conn.commit()
        sendRecoveryCode(email, newRecoveryCode)
        client_socket.send("Password changed successfully".encode())
        cur.close()
        return

#Verify Fail
except Exception as e:
    print(e)
    client_ip = client_socket.getpeername()[0]
    client_socket.send("Wrong recovery code! Please try again".encode())
    cur.execute("INSERT INTO suspiciousTable
(username,SUSSY,ipaddress,logtime) VALUES (%s,%s,%s)",
[username,client_ip,datetime.now()])
    conn.commit()
    cur.execute("SELECT COUNT(*) FROM suspiciousTable WHERE usernameSUSSY
= %s", [username])
    count = cur.fetchone()[0]
    cur.execute("SELECT email FROM userInfo WHERE username = %s",
[username])
    data = ast.literal_eval(cur.fetchone()[0])
    gmailofSussy = getDecryptData(data)
    client_socket.send(("User added to suspiciousTable count: " +
str(count)).encode())
    sendEmail(count,gmailofSussy)
    cur.close()
    return

```

Figure 15: Code snap of Forget function

3.2.3.4. Admin Functions

After successfully authenticate, the admin user will then enter admin menu with multiple admin functions that they can use:

- **Change User Privilege:**

The `ChangeUserPrivilege` function is responsible for changing a user's privilege level in the system. An administrator must specify the username of the user whose privileges they desire to modify as well as the new privilege level (either "admin" or "normal") when initiating a privilege change.

The `userInfo` table's user role is then updated by the function using a SQL UPDATE query.

- **Delete User:**

The `DeleteUser` function is responsible for deleting a user's account from the system. Then the admin must provide the username of the user whose account they wish to delete.

The user's account is then deleted from the `userInfo` table by the function using a SQL DELETE query.

- **Unlock User:**

The `UnlockUser` function is responsible for unlocking a user's account in the system. When an administrator initiates an account unlock, they must provide the username of the user whose account they wish to unlock. The user's status is then updated to 0 in the `userInfo` table by the function's execution of a SQL UPDATE statement.

The `UnlockUser` function then looks for any entries connected to the unlocked user in the `suspiciousTable`. The function runs a SQL DELETE query to remove any items from the `suspiciousTable` if any are found.

3.2.3.5. Generate user factor function

```
def generateFactor(username):
    role = getRoles(username)
    try:

        cur = conn.cursor()
        cur.execute("SELECT delete_user, search_data, insert_data,
update_data, delete_data FROM rolePermissions WHERE role = %s", [role])
        result = cur.fetchall()
    except Exception as e:
        print(e)
```

```

if result:
    permissions = {
        'delete_user': result[0][0],
        'search_data': result[0][1],
        'insert_data': result[0][2],
        'update_data': result[0][3],
        'delete_data': result[0][4]
    }
    permissions_bin = ''.join([bin(value)[2:].zfill(1) for value in
permissions.values()])
    factor = permissions_bin + bin(getrandbits(8))[2:] +
bin(getrandbits(8))[2:]
    enc = AES.new(unhexlify(getAES_KEY()), AES.MODE_CTR)
    ciphertext = enc.encrypt(factor.encode())
    nonce = enc.nonce
    return ciphertext.hex(), nonce.hex()
else:
    print(f"No permissions found for role: {role}")
    return

```

Figure 16: Code snap of generate user factor function

3.2.3.6. Generate TOTP function

```

def generate_totp(secret_key, state=0):
    current_time = int(time.time())
    time_interval = 30
    time_steps = (current_time // time_interval) + state
    time_steps_bytes = struct.pack(">Q", time_steps)
    secret_key_bytes = secret_key.encode("ascii")

    # Generate an HMAC-SHA1 hash of the time steps using the secret key
    hmac_hash = hmac.new(secret_key_bytes, time_steps_bytes,
hashlib.sha3_256).digest()

    # Calculate the offset and take last 4-byte for the TOTP code
    offset = hmac_hash[-1] & 0x0F
    code_bytes = hmac_hash[offset:offset+4]
    code = struct.unpack(">I", code_bytes)[0]
    totp_code= '{0:06d}'.format((code & 0x7FFFFFFF) % 1000000)

    return totp_code

```

Figure 17: Code snap of generate TOTP function

3.2.4. Demonstration Results

3.2.4.1. Register session

An individual might create a brand-new account in the system. The user is given a main menu with three buttons to start the registration process: "register,"

"login," and "forget password." To start the registration procedure, the user clicks the "register" button.

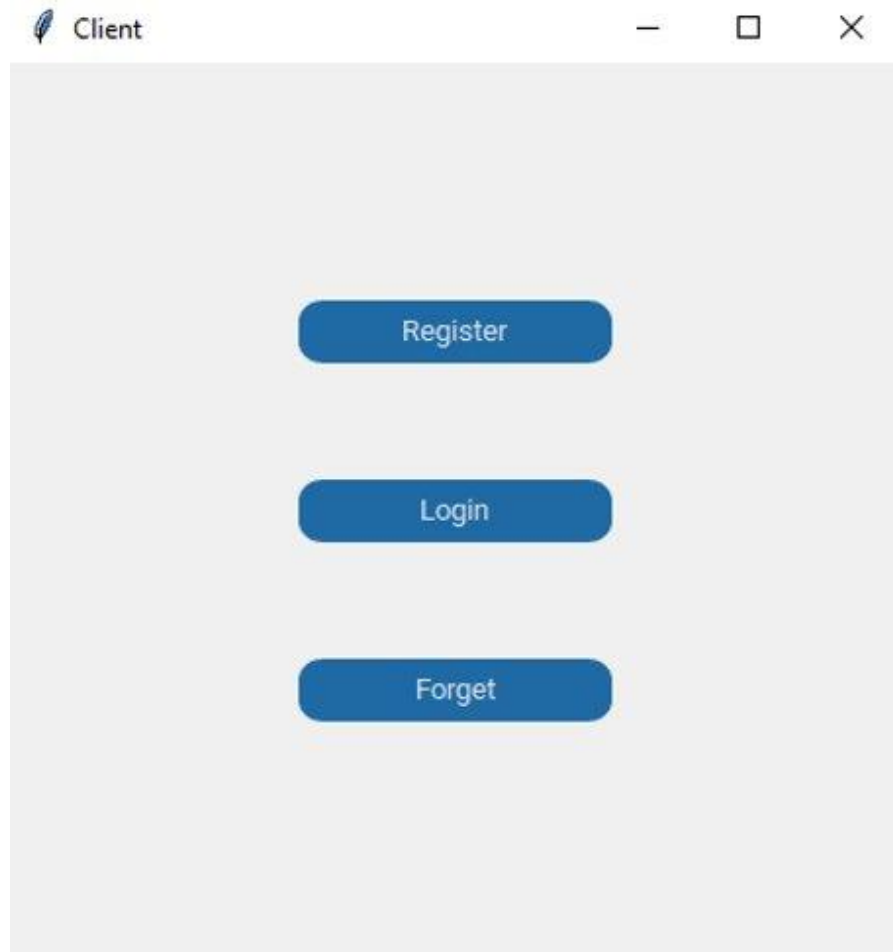


Figure 18: User Interface when run application

After that, a registration form asking for the user's username, password and email address is displayed. The user fills out the form with their data and presses the "submit" button.

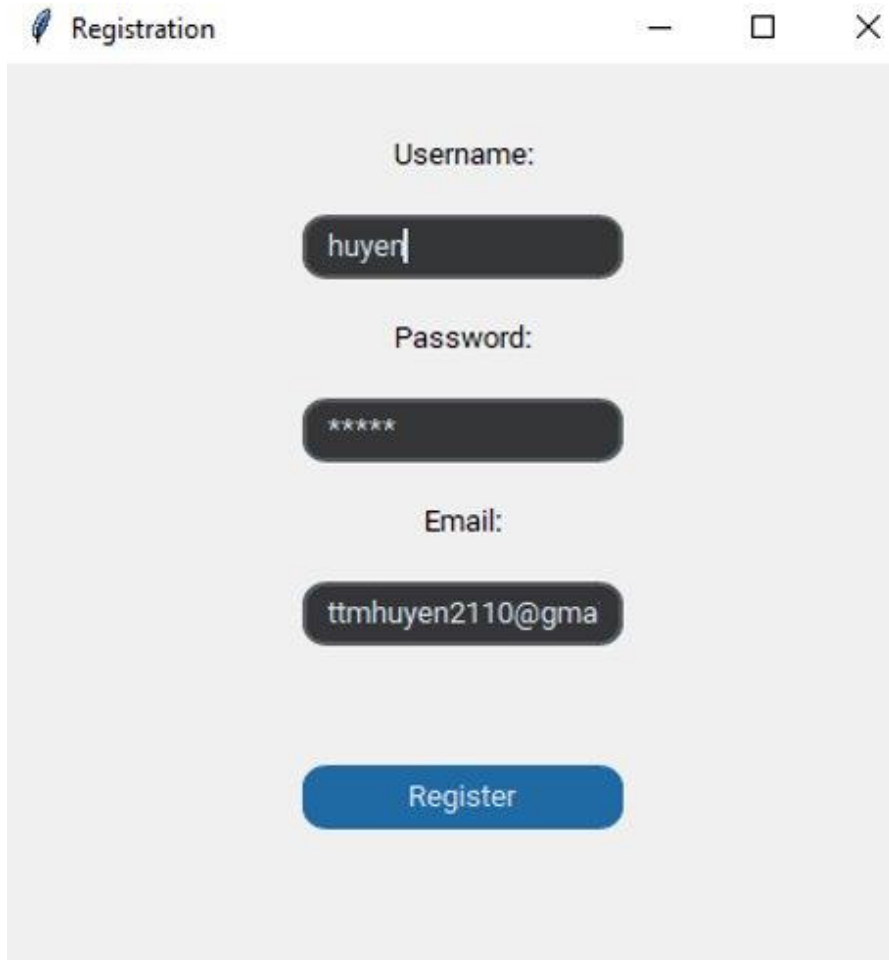
A screenshot of a web application window titled "Registration". The window has a light gray background and a dark gray header bar with the title and standard window controls (minimize, maximize, close). The form contains three input fields: "Username:" with the text "huyen", "Password:" with six asterisks, and "Email:" with the text "ttmhuyen2110@gma". Below the input fields is a blue "Register" button.

Figure 19: Register Form

The system then goes through a number of processes to create the user's account and add their data to the database's userInfo table. A new factor and recovery code gets generated by the server and sent to the user's client. Additionally, the server also emails the recovery code to the user's email address.

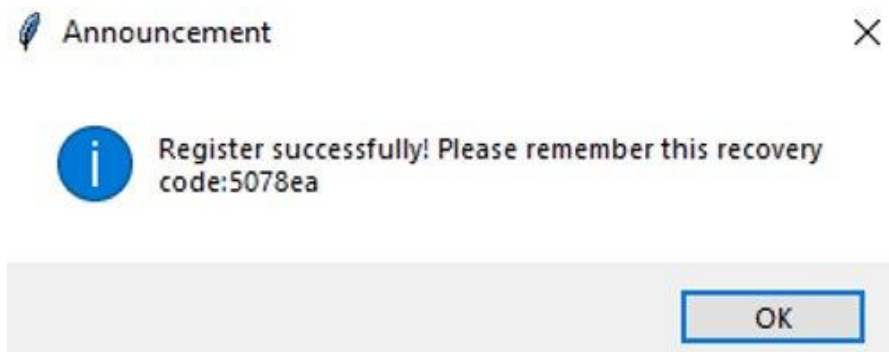


Figure 20: Register successfully announcement

3.2.4.2. Login session

To begin the login session, the user clicks the “Login” button. The user is then presented with a login form that prompts them to enter their username and password.

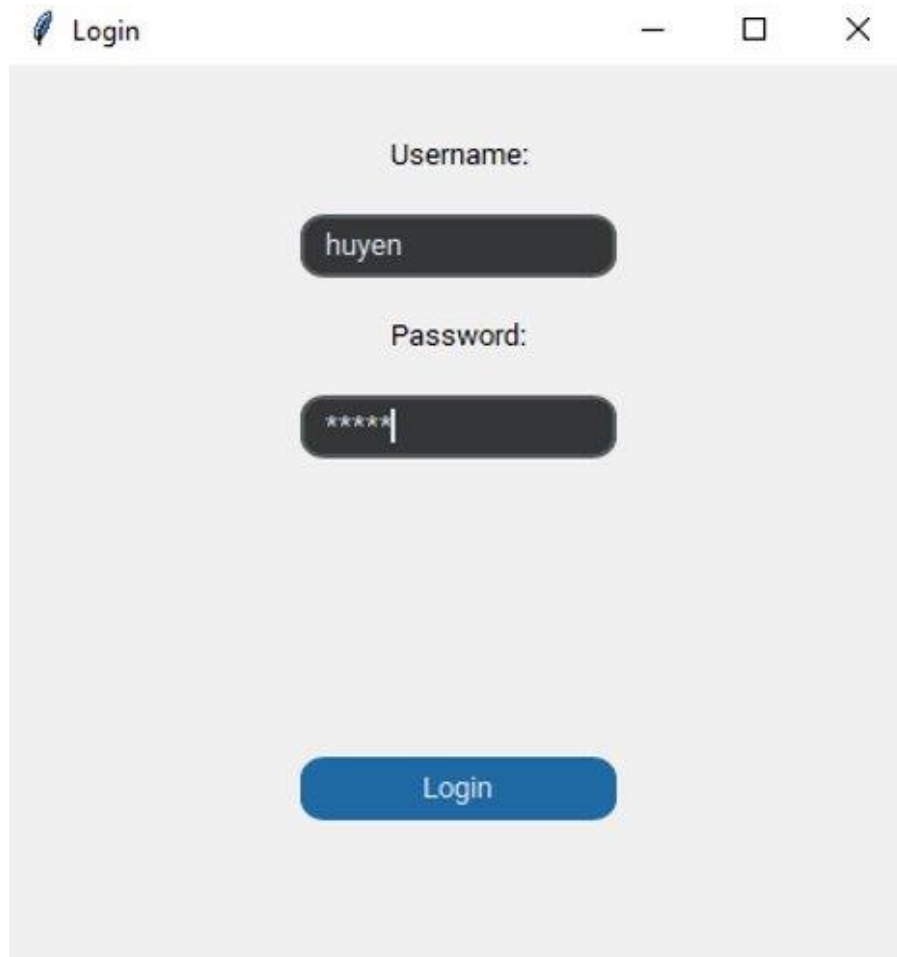
The image shows a web browser window with the title "Login". Inside the window, there is a light gray background. At the top, there is a "Username:" label followed by a dark gray rounded rectangular input field containing the text "huyen". Below this is a "Password:" label followed by a similar dark gray rounded rectangular input field containing six asterisks "*****". At the bottom of the form is a blue rounded rectangular button with the text "Login" in white. The browser window has standard OS controls (minimize, maximize, close) in the top right corner.

Figure 21: Login form

After entering the required information and clicking “Login”, user then enters their OTP code into the form and click “OTP” to send it

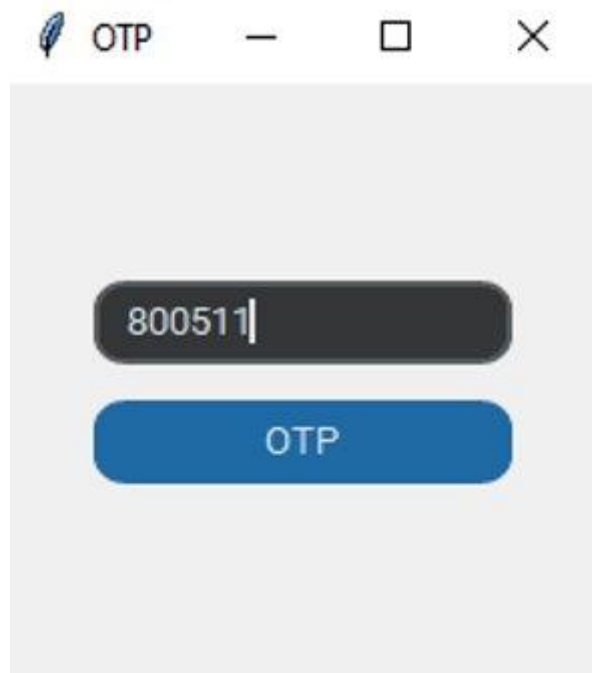


Figure 22: OTP form

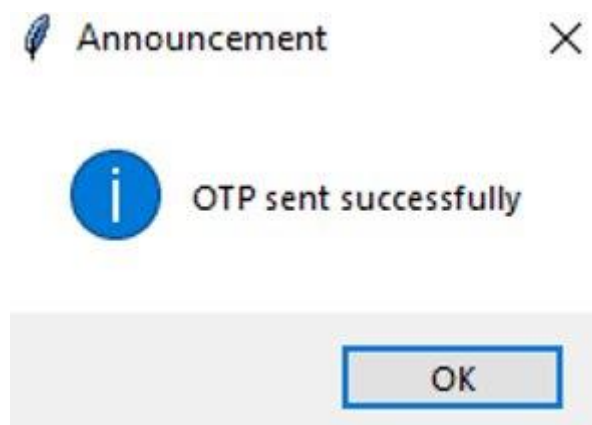


Figure 23: OTP sent announcement

The user's identity is then confirmed by the server, and they are given access to their account.



Figure 24: Confirm login successfully notification

Users will also be granted privileges corresponding to their role.

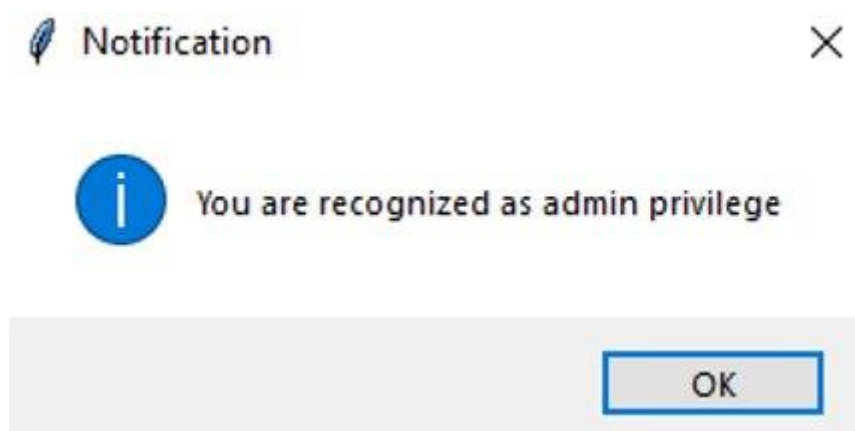


Figure 25: Confirm user privilege notification

To begin the forget session, the user clicks the “Forget” button. The user is then presented with a form that prompts them to enter their username and recovery code.

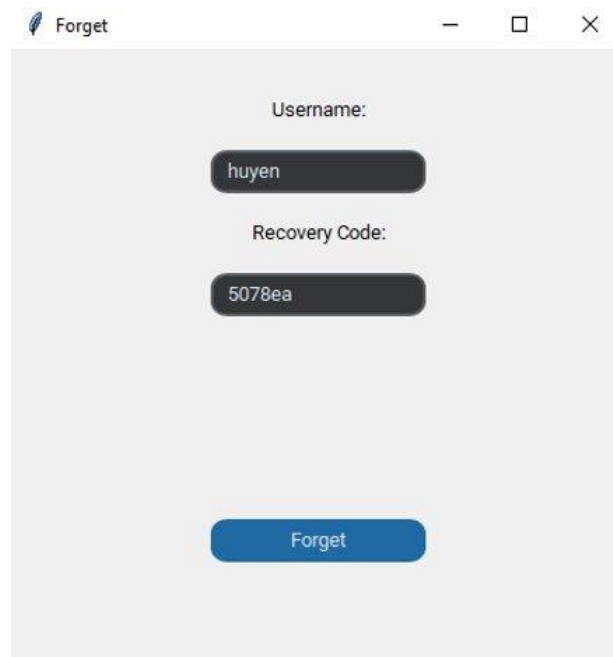
A screenshot of a web application window titled "Forget". The window has a light gray background and a dark gray header bar with the title "Forget" and standard window controls (minimize, maximize, close). The form contains two input fields: "Username:" with the value "huyen" and "Recovery Code:" with the value "5078ea". Both fields have dark gray backgrounds and rounded corners. Below the input fields is a blue button with the text "Forget".

Figure 26: Forget password form

After entering the required information and clicking “Forget”, user then enters their new password into the form and click “Change” to send it.

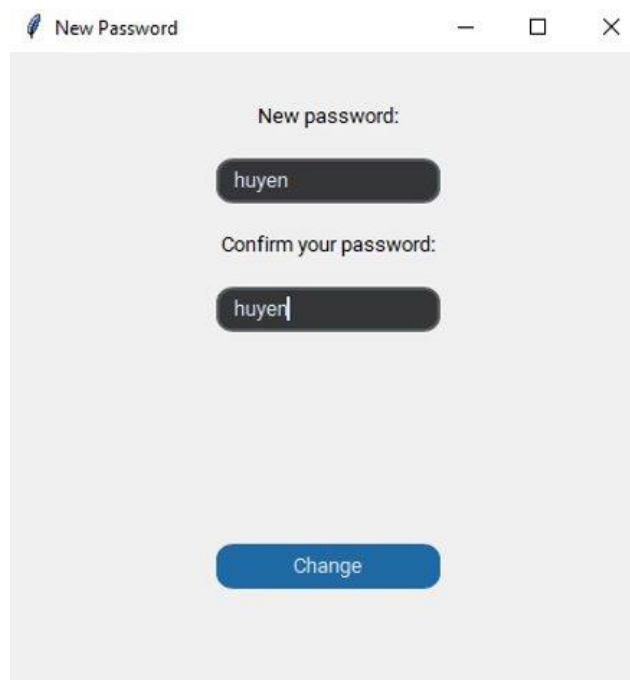
A screenshot of a web application window titled "New Password". The window has a light gray background and a dark gray header bar with the title "New Password" and standard window controls (minimize, maximize, close). The form contains two input fields: "New password:" with the value "huyen" and "Confirm your password:" with the value "huyen". Both fields have dark gray backgrounds and rounded corners. Below the input fields is a blue button with the text "Change".

Figure 27: Create new password form

The server then sends a success message to the user's client indicating that their password has been successfully reset.

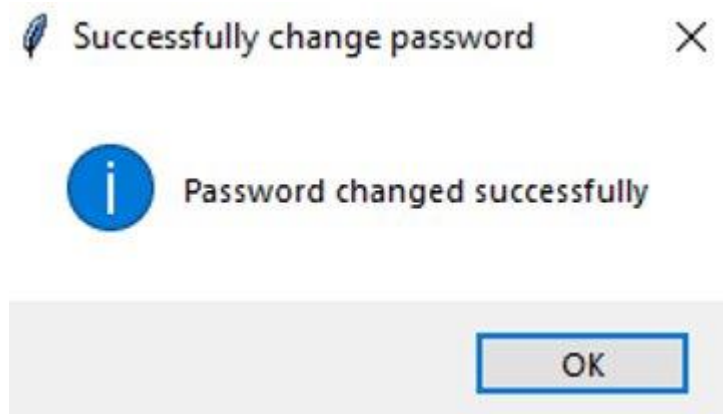


Figure 28: Password change notification

3.2.4.3. Login with different IP session

For this session, the user clicks the “Login” button. The user is then presented with a login form that prompts them to enter their username and password.

After entering the required information and clicking “Login”, the server checks whether the IP address matches with the one in the database then prompts a notification. The user is then prompted to enter a verification code sent over email to confirm their identity.

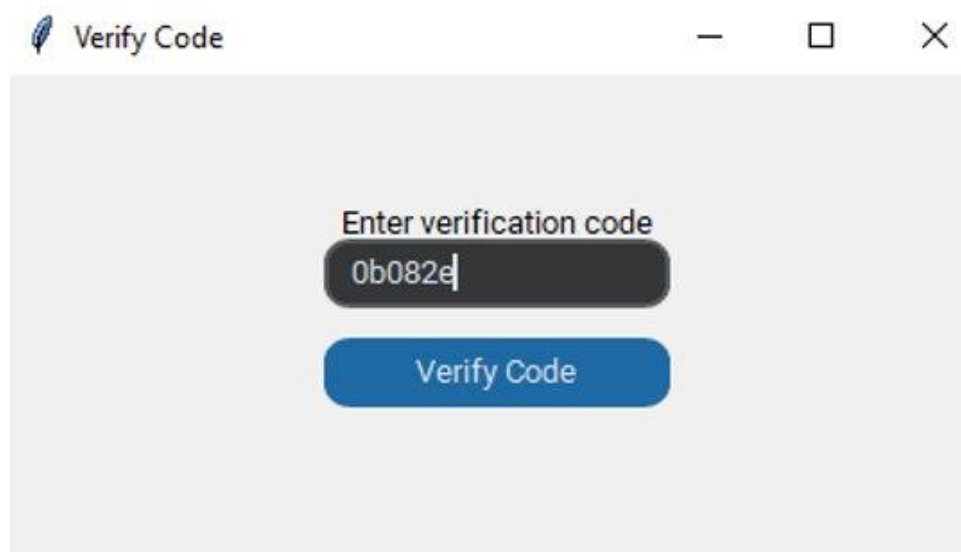


Figure 29: Verify code form

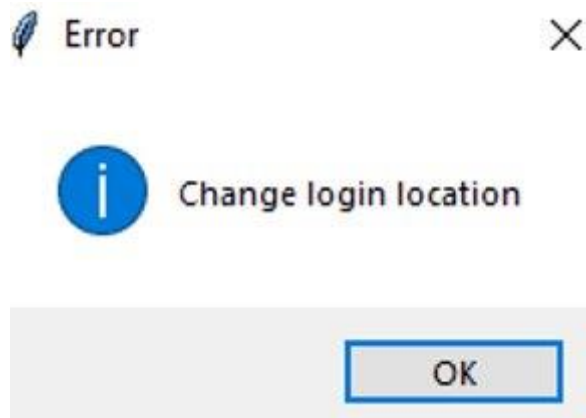


Figure 30: Change stored IP address

When the verification code has been checked by the server, the user will receive a new factor and recovery code. This new factor then be used to generate their OTP code required later. And the OTP process will be the same as Login session.

3.2.4.4. Admin privileges

After Login and recognized as an admin, the user will interact with a new form with three buttons (Change user privileges, Delete user, Unlock user). Pressing the button will forward them to the corresponding function.

Change user privileges:

In the user privileges change form, admin must specify the username of the user whose privileges they desire to modify as well as the new privilege level ("admin" or "normal") and press "Change" to send the data to the server.

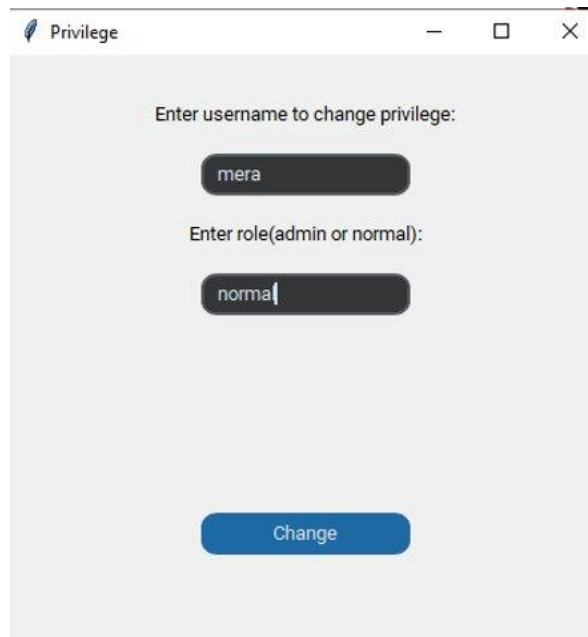


Figure 31: Change user privileges form

After the server has modified the corresponding privileges in the database, client will receive a Success notification.



Figure 32: Change user privileges successfully notification

And when inspecting the table, the user privileges has been altered.

SELECT username, role FROM "public"."userinfo" LIMIT 100

Table queries ▾

Previous queries ▾

username	role
kizme	admin
tlhung	admin
huyen	admin
mera	normal

Figure 33: Database after change user privileges

Delete user:

In the user delete form, admin must identify the username of the user whose privileges they desire to delete and press “Delete” to send the data to the server.

Delete User

Enter username to delete:

tlhung

Delete

Figure 34: Delete user form

After the server has removed the stated user from the database, client will receive a Success notification.

Success

i

User tlhung deleted

OK

Figure 35: Delete user successfully notification

And when inspecting the table, the user has been deleted.

id	username	password	email
1	mera	\$argon2id\$v=19\$m=65536,t=3,p=4\$2ZZOphPuk8wVdT1/ad46Jg\$SrWhc9GafQp3s3mBMBeeGQ+JKCp9YAAx2haHd0biBevw	('274737c3f63ae08d328764372a6f073,'a5065285d40c5fd0ea7de40271ce54d,'316b999a84f53d6a3d11c6d105f0bf4c
2	kizme	\$argon2id\$v=19\$m=65536,t=3,p=4\$KNc58bL11Y8UFoCEPrTpXg\$UrUTY14wwaxKw5Gglvt/yJIJ7EA6bMTi6fXYyCV5FB3M	('3ed4c81270f70567e651d70d15dcbe,'87044e7ebe42e772e463c16ab984492,'ae4443cab83cd2a693d2014d67e231d
4	huyen	\$argon2id\$v=19\$m=65536,t=3,p=4\$4vTJnTq0bW+fxvFTcHeA\$vOXOr/90BII82xCcLmW3fTk/DivhepmQNYD8+pDgPoY	('14c96e9b40d874d94570a71e139651,'801936f65aa0bf4075e121ccf5f1b4f2','0bbbc7397a5bcbbf3e13741cb06dfc27

Figure 36: Database after delete user 'tlhung'

Unlock user:

In the user unlock form, admin must state the username of the user who they want to unlock and press “Unlock” to send the data to the server.

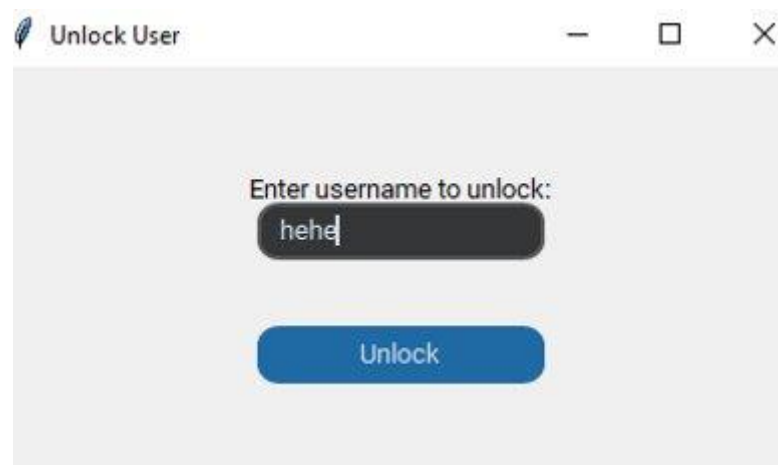


Figure 37: Unlock user form

After the server has changed the status of user from the database, user will receive a Success notification.



Figure 38: Unlock user ‘hehe’ successfully notification

When inspecting the table, the user status has been modified from 1 (locked) to 0 (unlocked).

username	role	status
mera	admin	0
kizme	admin	0
tlhung	admin	0
hehe	admin	1
huyen	admin	0

Figure 39: Database before unlocking user ‘hehe’

username	role	status
mera	admin	0
kizme	admin	0
tlhung	admin	0
huyen	admin	0
hehe	admin	0

Figure 40: Database after unlocking user ‘hehe’

3.3. Second Implementation Plan

3.3.1. Hardware and Software Overview

The proposed multi-factor authentication (MFA) scheme involves three nodes: the client, the cloud server, and the database. The cloud server is an Azure Virtual Machine running Ubuntu 20.04 in a virtual private cloud (VPC) environment, and the client is a virtual machine running Ubuntu 20.04. The database is hosted on ElephantSQL. The MFA scheme is implemented using Python programming language and the following libraries: OpenSSL and Tpm2tools.

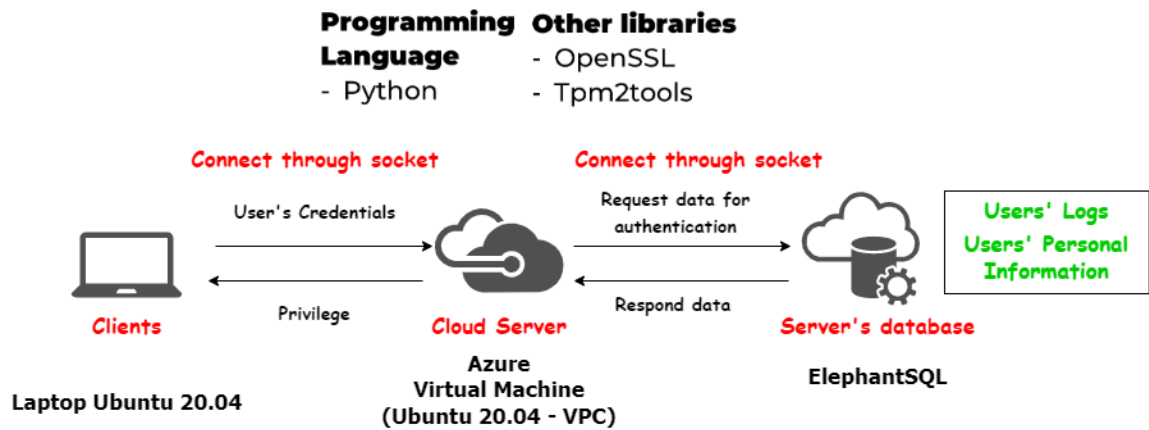


Figure 41: Second Implementation system

3.3.2. System configurations

3.3.2.1. Cloud Server configurations

The same as the first implementation plan

3.3.2.2. Database configurations

The same as the first implementation plan

3.3.2.3. Client configurations

As tpmtools cannot run on Windows, clients using Ubuntu 20.04 can store user factor (pre-shared key for TOTP) in vTPM.

This setup ensures the security of the user factor used to produce TOTP.

The disadvantage of this configuration is user does not have a User Interface (UI) to communicate with the server.

3.3.3. Demonstration result

The same as the first implementation

3.4. Third Implementation Plan

3.4.1 Hardware and Software Overview

The proposed multi-factor authentication (MFA) scheme involves three nodes: the client, the cloud server, and the database. The cloud server is an Azure Virtual Machine running Ubuntu 20.04 in a virtual private cloud (VPC) environment, and the client is a smartphone running on Android 10. The database is hosted on ElephantSQL. The MFA scheme is implemented using Python programming language and the following libraries: OpenSSL and Tpm2tools. The client side is implemented using Java.

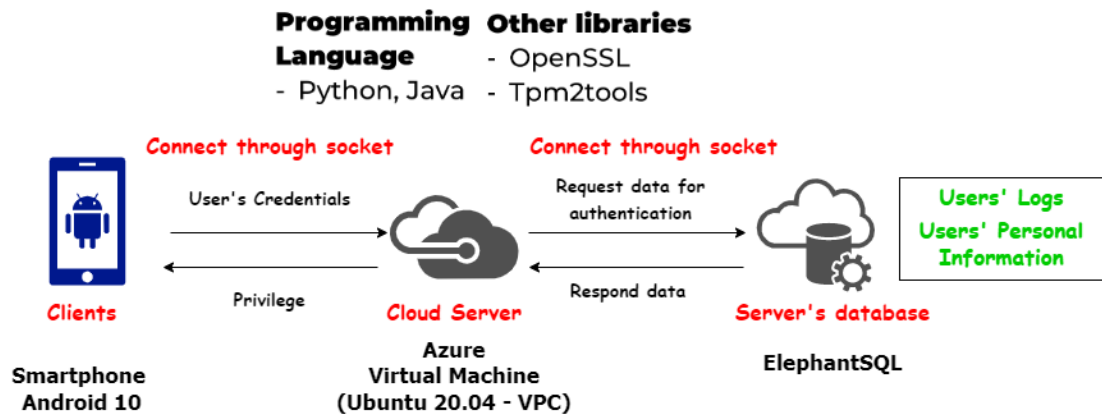


Figure 42: Third Implementation system

3.4.2. System configurations

3.4.2.1. Cloud Server configurations

The same as the first implementation plan

3.4.2.2 Database configurations

The same as the first implementation plan

3.4.2.2. Client configurations

As mobile apps offer many advantages over other forms of software applications, and they are becoming increasingly popular as more people rely on mobile devices as their primary computing platform.

We have encountered challenges in implementing the application on mobile devices due to limited internet access, and the work is still ongoing. As a result, the application will only have a User Interface on mobile devices.

3.3.4. Demonstration result

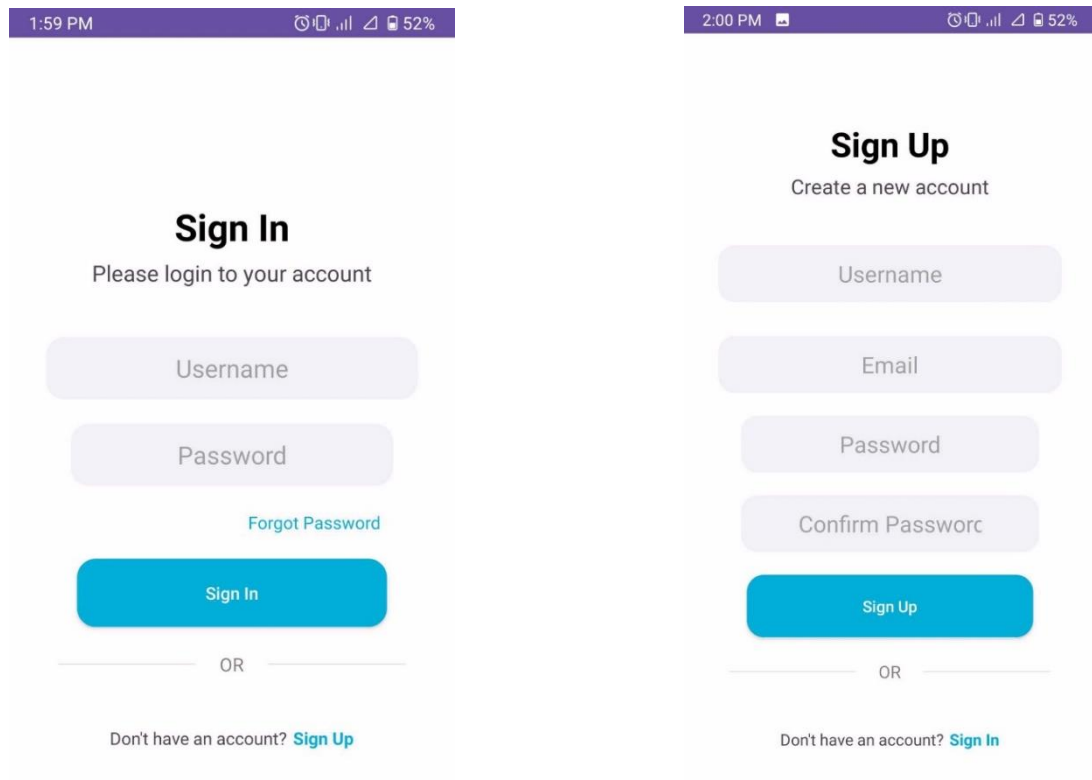








Figure 43: Sign up and sign in UI

2:00 PM      52%



Please enter OTP code

Email Address

1

2

3

4

5

6

Verify

Didn't receive OTP? [Resend Code \(56\)](#)

Figure 44: OTP user interface

Chapter 4. SUMMARY

4.1 Evalute result

	Feature name	Evaluate
1	Encrypt and decrypt process	Guaranteed
2	Client can verify Server	Guaranteed
3	Server can verify Client	Guaranteed in most situation,
4	Secure communication	Guaranteed
5	Protect data	Guaranteed
6	Network between nodes	Guaranteed, except for mobile device

Table 6: Result overview

4.2 Tasks

Task name	Huyền	Kiệt	Hưng
Research on Idea, Paper, Determine Related Parties, Scenario, Protected Assets, Solution	x	x	
Research on Algorithms, Cloud Platform, Environments, Libraries	x	x	x
Controlling implementation of the proposed scheme	x		
User Interface implement	x	x	
Implementing Database	x		
Implementing generate recovery code,	x	x	

sign certificate			
Implementing generating factor, OTP			x
Building features, Testing, and debugging the Demo model	x	x	x
Running demo model, Deploy on Cloud, Evaluate the result	x	x	x
Presenting the Project, Slide design	x	x	x

Table 7: Tasks table

REFERENCES

Said, W. *et al.* (2021) ‘A multi-factor authentication-based framework for identity management in cloud applications’, *Computers, Materials & Continua*, 71(2), pp. 3193–3209. doi:10.32604/cmc.2022.023554.