

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

PHAN THỊ HỒNG NHUNG ĐOÀN HẢI ĐĂNG LÊ THANH TUẤN

CRYPTOGRAPHY PROJECT REPORT

ONE TIME PASSWORD (OTP) BASED ON
ADVANCED ENCRYPTED STANDARD (AES) AND
LINEAR CONGRUENTIAL GENERATOR (LCG)

HO CHI MINH CITY, 2023

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

PHAN THỊ HỒNG NHUNG - 21521250

ĐOÀN HẢI ĐĂNG - 21520679

LÊ THANH TUẤN - 21520518

CRYPTOGRAPHY PROJECT REPORT
ONE TIME PASSWORD (OTP) BASED ON
ADVANCED ENCRYPTED STANDARD (AES) AND
LINEAR CONGRUENTIAL GENERATOR (LCG)

PROJECT ADVISOR

PhD. NGUYỄN NGỌC TỰ

HO CHI MINH CITY, 2023

ACKNOWLEDGEMENT

First and foremost, we would like to express our gratitude to our advisor, PhD Nguyễn Ngọc Tự, for his guidance and consistent supervision, as well as for providing crucial project information and assisting us in completing the research.

Our gratitude and appreciation also extend to our colleagues and lecturers who assisted us in the development of the project, as well as to those who have volunteered their time and skills to assist us.

Group 5

TABLE OF CONTENTS

Chapter 1. INTRODUCTION	1
1.1 Topic overview.....	1
1.2 Scenario.....	2
1.3 Related - Parties	3
1.4 Research Goals.....	4
Chapter 2. PROPOSED SCHEME	4
2.1 Secure Channel.....	4
2.2 Server – Client Authentication.....	5
2.3 Key Exchange	6
2.4 Database	7
2.5 Algorithm	7
Chapter 3. IMPLEMENT AND TEST	8
3.1 Overview	8
3.2 Tool and Resources	8
3.3 System Design.....	8
3.4 Demo Application	10
REFERENCES	15
TASK CHART.....	16

TABLE OF FIGURES

Figure 1 Log in using static password.....	2
Figure 2 Related Parties	3
Figure 3 Wireshark capture for TLS 1.3	4
Figure 4 Server's certificate infomation	5
Figure 5 Algorithm diagram.....	7
Figure 6 System Model	9
Figure 7 OTP Mechanism	9
Figure 8 Connect with Linux Virtual Machine in the Azure cloud platform	10
Figure 9 Example of sign up with existed username	11
Figure 10 Example of sign up	11
Figure 11 Example of how MongoDB stores user's account	12
Figure 12 Example of sign in	12
Figure 13 Example of enter OTP within valid time	13
Figure 14 Example of enter expired OTP	13
Figure 15 Example of using resend.....	14
Figure 16 Examle of blocking multiple logins to account	14
Figure 17 Packet captured from Wireshark	15

LIST OF ABBREVIATIONS

OTP	One Time Password
ECDH	Elliptic Curve Diffie-Hellman
AES	Advanced Encryption Standard
LCG	Linear Congruential Generator
TLS	Transport Layer Security
TPM	Trusted Platform Module

Chapter 1. INTRODUCTION

1.1 Topic overview

In today's modern era, the importance of safeguarding user accounts through the implementation of double-factor authentication has reached a critical level due to the escalating number of cyber-attacks. One specific method of double-factor authentication that has gained significant popularity is the use of One Time Password (OTP). Unlike traditional passwords, OTP are only valid for a single login session, thereby providing an enhanced level of security for user accounts.

This study focuses on the generation of OTP by employing a combination of the user's login credentials, phone number, and the time of access. To ensure a robust level of security, the plaintext data was encrypted using the Advanced Encryption Standard (AES), which is widely recognized as a highly secure encryption algorithm. Subsequently, the encrypted data was subjected to processing by a Linear Congruential Generator (LCG), a method that randomly selects 6 characters to generate the OTP.

The research findings clearly demonstrate that the OTP generated through the utilization of the LCG method are both unique and unpredictable, thereby effectively mitigating the risks associated with repeated usage and sniffing attacks. This proposed approach to OTP generation offers an additional layer of protection to user accounts, guaranteeing a more secure means of authenticating users.

It is important to note, however, that while OTP can certainly bolster the security of user accounts, they are not impervious to attacks. For instance, if an attacker manages to gain access to a user's phone or email account, they can intercept the OTP and exploit it to gain unauthorized entry to the user's account. Therefore, it is crucial to combine OTP with other robust security measures such as password managers, firewalls, and antivirus software. This multi-faceted approach ensures comprehensive protection against the ever-evolving landscape of cyber-attacks.

1.2 Scenario

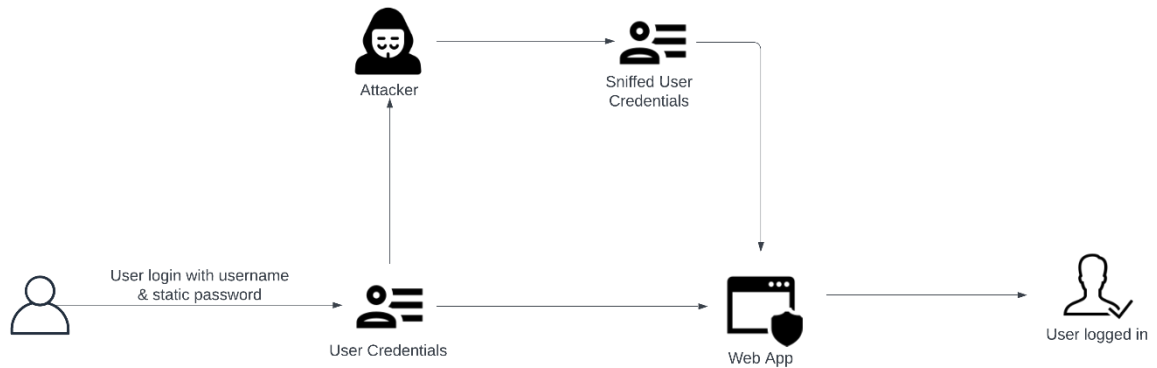


Figure 1 Log in using static password

In this case, when users rely solely on static passwords for login without any additional security layers, they are vulnerable to attacks that can compromise their login information and expose their personal data. This can lead to fraudulent activities and have negative consequences for the users.

Nowadays, attackers employ various methods to steal user information. They can impersonate users or act as "man in the middle" to manipulate data during transmission. Therefore, it is essential to establish mutual authentication between the client and server, ensure secure channels for data exchange, and protect the confidentiality and integrity of the data.

1.3 Related - Parties

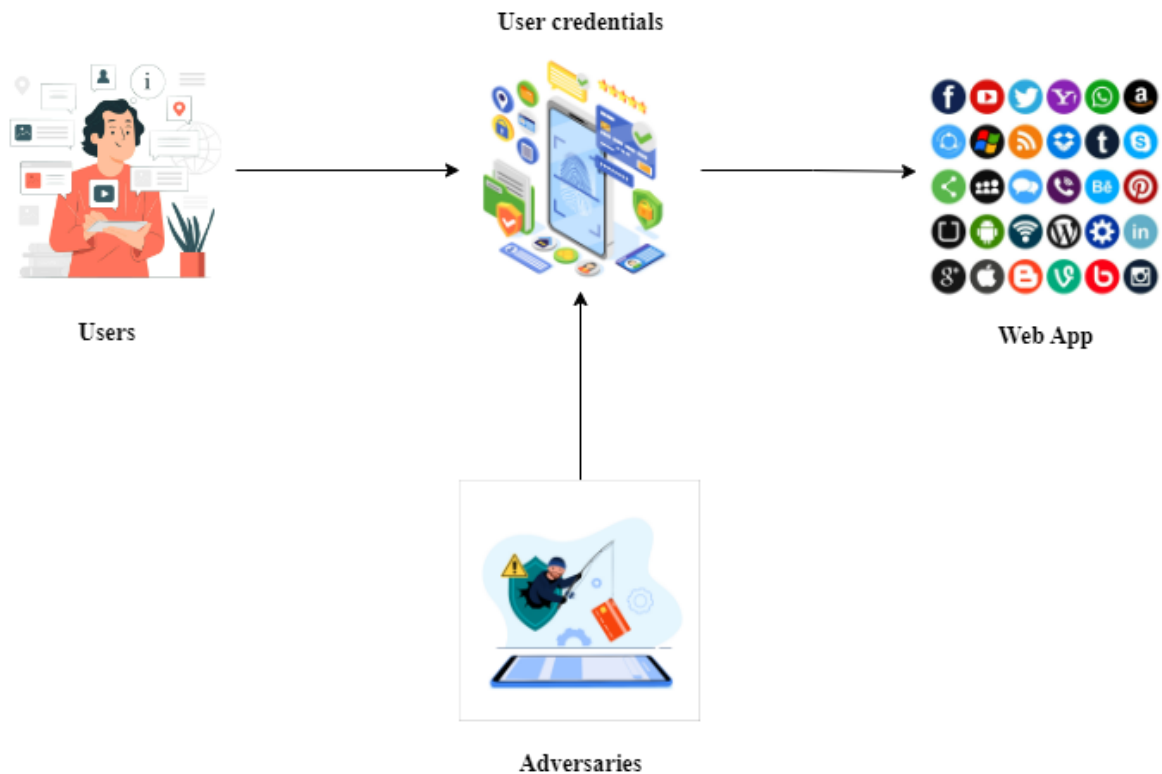


Figure 2 Related Parties

- **Users**: They act as clients of the web app. After successful registration, they will log in using a username and password.
- **Web App**: It acts as a server for user authentication. The Web App provides the interface and functionality for users to register and log in to their accounts. Once users provide their login information, the web app verifies the validity of this information.
- **Adversaries**: They act as attackers within the system. The adversaries' goal may be to cause harm to your web app by engaging in destructive actions or stealing users' login information.

1.4 Research Goals

- **High security:** Server and client exchange keys using ECDH over a secure channel TLS 1.3.
- **Reliability:** Both parties independently generate OTPs based on agreed-upon inputs, ensuring that the OTPs remain confidential and not susceptible to leakage.
- **Authentication:** Server authenticates users using static password and OTP, while users authenticate the server using a certificate.

Chapter 2. PROPOSED SCHEME

OTP is a security measure that adds an extra layer of protection. It can be used only once and has a short time limit. If not used immediately, it expires and becomes unusable. This is why OTP is widely used in Web and Android systems.

In this project, due to limited capabilities, we have implemented it on a PC. However, we plan for further enhance our skills in mobile development and deploy the application on smartphones.

2.1 Secure Channel

Transport Layer Security (TLS) 1.3 is the latest version of this protocol, offering advantages such as faster speeds and higher security compared to previous versions. It has an architecture that comprises three main protocols: the Handshake protocol, the Record protocol, and the Alert protocol.

Capture by Wireshark to check whether a connection is using TLS 1.3:

No.	Time	Source	Destination	Protocol	Length	Info
378	15.948089	40.81.29.50	172.16.0.178	TLSv1.3	101	Application Data
376	15.916736	172.16.0.178	40.81.29.50	TLSv1.3	100	Application Data
302	10.445522	40.81.29.50	172.16.0.178	TLSv1.3	274	Application Data
298	10.185877	172.16.0.178	40.81.29.50	TLSv1.3	348	Application Data
188	5.516401	40.81.29.50	172.16.0.178	TLSv1.3	321	Application Data
185	5.432903	40.81.29.50	172.16.0.178	TLSv1.3	321	Application Data
183	5.394385	172.16.0.178	40.81.29.50	TLSv1.3	146	Change Cipher Spec, Application Data
181	5.390319	40.81.29.50	172.16.0.178	TLSv1.3	834	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data
178	5.356467	172.16.0.178	40.81.29.50	TLSv1.3	583	Client Hello

Figure 3 Wireshark capture for TLS 1.3

2.2 Server – Client Authentication

In the authentication process implemented in this project, the client and server utilize a combination of technologies to securely verify each other's identities. The server presents its ECDSA certificate based on the NIST P-256 curve to the client, establishing the server's authenticity and allowing the client to establish a trusted connection.

To verify the client's identity, the server follows a 2-factor authentication. Initially, the client provides a username and password, which are securely transmitted to the server. The server then validates these credentials against its stored user database, ensuring that the provided information matches the records of an authorized user.

Upon successful authentication of the client's username and password, an additional layer of security is added through the use of OTP. The OTP is generated using the AES algorithm, which encrypts the password and provides a unique and time-limited authentication code. To enhance the unpredictability of the OTP, a LCG algorithm is employed.

By implementing this combined approach, both the client and server can mutually verify each other's identities. The server verifies the client's identity through the username, password, and OTP, ensuring that only authorized individuals can access its services. Similarly, the client verifies the server's authenticity by validating the server's ECDSA certificate with the NIST P-256 curve. This multi-layered authentication process establishes a secure and trusted connection between the client and server, mitigating the risk of unauthorized access and potential security threats.

```
Subject Public Key Info:
  Public Key Algorithm: id-ecPublicKey
    Public-Key: (256 bit)
      pub:
        04:54:28:37:b3:7a:d8:86:ce:11:39:5a:fc:46:1d:
        7f:27:3a:87:c5:19:14:32:81:dd:63:56:a9:ff:d4:
        d4:47:60:90:30:71:be:2c:73:4b:86:b5:7e:44:3e:
        1f:b5:3d:b6:13:ab:5a:14:47:27:62:4c:be:d6:1a:
        52:b1:59:3b:4f
      ASN1 OID: prime256v1
      NIST CURVE: P-256
```

Figure 4 Server's certificate information

2.3 Key Exchange

Elliptic Curve Diffie-Hellman (ECDH) is a key exchange algorithm used in cryptography to establish a shared secret key between two parties over an insecure channel. It ensures that even if an attacker intercepts the public keys exchanged during the process, they cannot easily compute the shared secret without the private keys. Besides that, ECDH offers efficient key exchange with smaller key sizes and lower computational costs.

In this project, we use ECDH Key Exchange with Secp256r1 – one of the secure and widely curve in ECDH implementations.

Regarding key storage, we have two main approaches.

- The first approach is to use fixed keys that are periodically changed. After users register, they will exchange keys and store them in the TPM (Trusted Platform Module). The server will store the keys in the available vTPM on Azure. Storing keys in TPM offers several benefits, including hardware-backed trust and key protection. If a device is changed, a recovery code will be issued and the key will be sent again.
- The second approach is to use session keys, which eliminates the need for key storage. This is similar to the approach used in the demo application. We will manage it based on a dictionary array {username: socket}. When a client logs in, it checks if the username exists in the dictionary. If it doesn't, the username is added to the dictionary. If it already exists, the client is not allowed to log in. This is similar to the restriction of only allowing one device per account, as seen in applications like Zalo. After the initial check is done, the key exchange process begins. Therefore, only the user who registered the account will have the key because the system keeps track of the first login session. If a user changes devices, the system will kick out the user currently in session and allow a new user to join with the condition of having the display code from the old device.

2.4 Database

MongoDB is an open-source NoSQL database software that supports multiple platforms, making it suitable for implementation in this project. We store username, hashed password, and email on it, and can easily perform database operations.

2.5 Algorithm

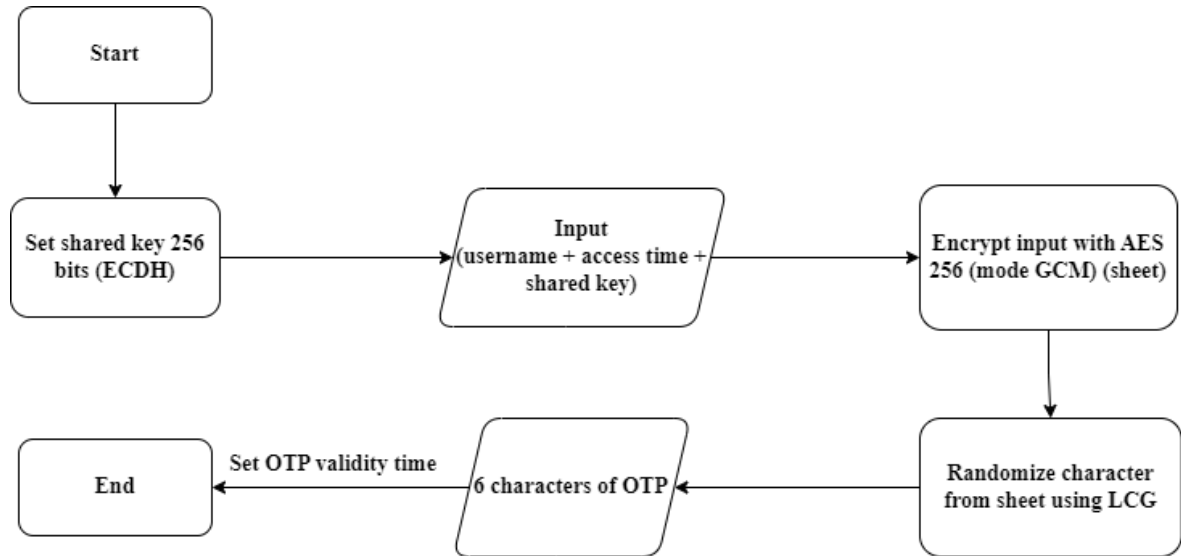


Figure 5 Algorithm diagram

In this flowchart, OTP is generated based on two main algorithms: AES and LCG. The AES algorithm utilizes the GCM mode, which is one of the most advanced and secure symmetric encryption modes available today. The input is encrypted using AES with the ECDH exchanged key, and the result is stored in a value table (sheet). As mentioned, the shared key is the result of the ECDH key exchange between the client and server. This key is only known to the account owner. Therefore, it is considered safe to include inputs such as username, email, and the shared key.

The seed0 of the LCG algorithm is calculated using the formula $\text{Seed0 of LCG} = \text{Timestamp} \% \text{length(sheet)}$. Then we have 6 seeds representing 6 positions pointing to the sheet. From that, we have a 6-digit OTP. Each OTP has a specific valid time, and once it expires, it cannot be reused.

Chapter 3. IMPLEMENT AND TEST

3.1 Overview

In this chapter, we will provide detailed information about our demo scenario and the results of our experiments. We will deploy OTP using AES algorithm, LCG, and a secure system for user authentication.

3.2 Tool and Resources

- Model: Client – Server
- Cloud Service: Microsoft Azure
- Library: Cryptography, SSL
- Language Programming: Python
- Database Management System: MongoDB
- Hardware Resources: Laptop, PC

3.3 System Design

- Server: Linux Virtual Machine in the Azure cloud platform.
This allows users to connect to your server remotely and interact with your applications or access resources hosted on the server.
- Client: Linux Laptop.

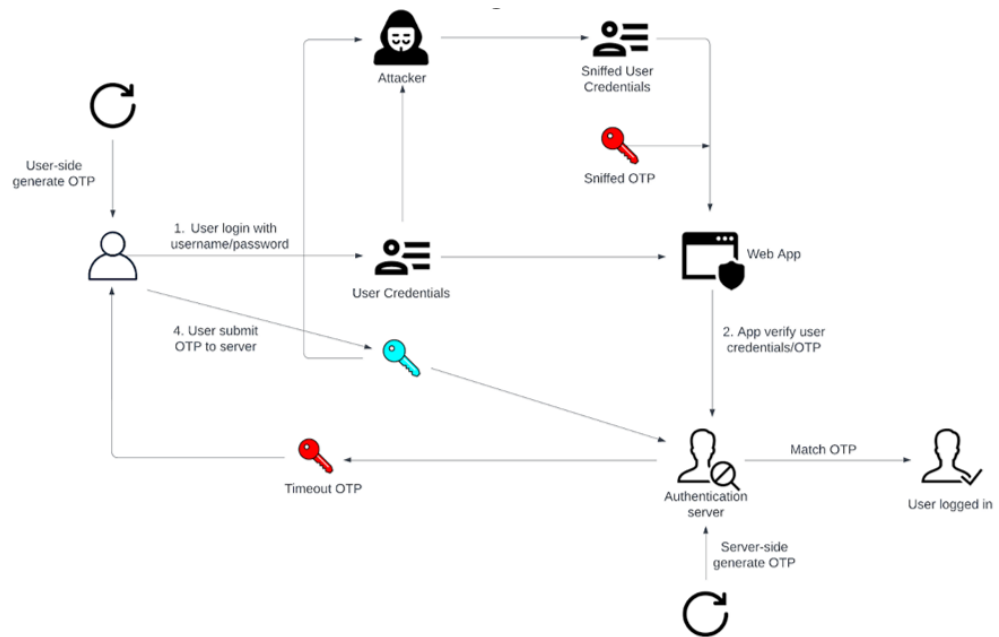


Figure 6 System Model

The user information, after registration, will be hashed and stored in the server-side database (MongoDB). When a user logs in, the system will authenticate whether the account and password match the stored information in the database, and then request an OTP. If the credentials do not match, the system will prompt for re-login. We will use Wireshark to capture information during transmission and check for any potential leakage.

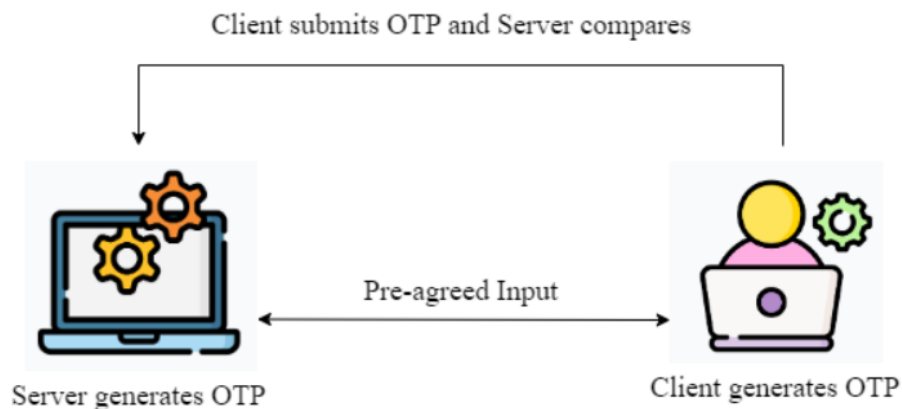


Figure 7 OTP Mechanism

With the mechanism of generating OTP from both the client and server based on a pre-agreed input, the user will obtain the OTP from a third-party app.

Each OTP has a specific validity period, and if it is submitted after the expiration or if an incorrect OTP is provided, access to the account will be denied. The server relies on the submitted OTP to authenticate the client.

3.4 Demo Application

We build an application where users can sign up and sign in from screen. When a user logs in, the system will check whether the username and password match with the stored information in the database, and then request an OTP.

```
root@DESKTOP-ORH3LTL:/mnt# ssh rosy@40.81.29.50
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-1040-azure x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/advantage

System information as of Sat Jun 24 14:57:41 UTC 2023

System load:  0.0                Processes:            139
Usage of /:   14.3% of 28.89GB   Users logged in:     0
Memory usage: 10%               IPv4 address for eth0: 10.0.0.4
Swap usage:   0%

* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
  just raised the bar for easy, resilient and secure K8s cluster deployment.

https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

1 update can be applied immediately.
To see these additional updates run: apt list --upgradable

6 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

New release '22.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sat Jun 24 12:10:51 2023 from 115.79.139.92
```

Figure 8 Connect with Linux Virtual Machine in the Azure cloud platform

Sign up:

Users sign up by giving their username, password and email address


```
rosy@MMH: ~/Cryptography_F X root@DESKTOP-ORH3LTL: /mi X + v
root@DESKTOP-ORH3LTL:/mnt/d/Clone/New/Cryptography_Project/Console# python3 client.py
/mnt/d/Clone/New/Cryptography_Project/Console/client.py:20: DeprecationWarning: ssl.wrap_socket() is deprecated
  client_socket = ssl.wrap_socket(client_socket, ca_certs="cert.crt")
Connected to server!

+-----+
| One Time Password (OTP) Based on Advanced Encrypted Standard |
| (AES) and Linear Congruential Generator(LCG)                  |
+-----+

+-----+
| Welcome to the Menu                                           |
|                                                                |
| /signup <username> <password> <email> : signup               |
| /signin <username> <password> : signin                       |
| /auth <OTP> : OTP                                             |
| /resend : resend OTP                                         |
+-----+

$ /signup test test123 test@gmail.com
$ [POST]: You have signed up, let's sign in.
```

Figure 10 Example of sign up

```
rosy@MMH: ~/Cryptography_F X root@DESKTOP-ORH3LTL: /mn X root@DESKTOP-ORH3LTL: /mi X + v
root@DESKTOP-ORH3LTL:/mnt/d/Clone/New/Cryptography_Project/Console# python3 client.py
/mnt/d/Clone/New/Cryptography_Project/Console/client.py:20: DeprecationWarning: ssl.wrap_socket() is deprecated
  client_socket = ssl.wrap_socket(client_socket, ca_certs="cert.crt")
Connected to server!

+-----+
| One Time Password (OTP) Based on Advanced Encrypted Standard |
| (AES) and Linear Congruential Generator(LCG)                  |
+-----+

+-----+
| Welcome to the Menu                                           |
|                                                                |
| /signup <username> <password> <email> : signup               |
| /signin <username> <password> : signin                       |
| /auth <OTP> : OTP                                             |
| /resend : resend OTP                                         |
+-----+

$ /signup test test2 test@gmail.com
$ [POST]: Username already existed!
```

Figure 9 Example of sign up with existed username

```

_id: ObjectId('649705a4a3872e89542d2df4')
name: "test"
email: "test@gmail.com"
password: "fafb77448432052ed74961c815c996c0c36694a5f1cb7ff0dfab5e11b991a167"

```

Figure 11 Example of how MongoDB stores user's account

Sign in:

After the system compares the username and password with the stored information in database, an OTP is generated and show in file with name as username (for future work, a separete application will show the OTP).

```

rosy@MMH: ~/Cryptography_F × root@DESKTOP-ORH3LTL: /mi × + ~
root@DESKTOP-ORH3LTL:/mnt/d/Clone/New/Cryptography_Project/Console# python3 client.py
/mnt/d/Clone/New/Cryptography_Project/Console/client.py:20: DeprecationWarning: ssl.wrap_socket() is deprecated
  client_socket = ssl.wrap_socket(client_socket, ca_certs="cert.crt")
Connected to server!

+-----+
| One Time Password (OTP) Based on Advanced Encrypted Standard |
| (AES) and Linear Congruential Generator(LCG)                  |
+-----+

+-----+
| Welcome to the Menu                                           |
|                                                                |
| /signup <username> <password> <email> : signup              |
| /signin <username> <password> : signin                        |
| /auth <OTP> : OTP                                             |
| /resend : resend OTP                                         |
+-----+

$ /signin test test123
$ Please enter OTP to authorize
[POST]: OTP generated, open file test to get OTP

```

Figure 12 Example of sign in

Case 1: Enter the OTP within 30 seconds (OTP valid time)

```
root@DESKTOP-ORH3LTL:/mnt/d/Clone/New/Cryptography_Project/Console# python3 client.py
/mnt/d/Clone/New/Cryptography_Project/Console/client.py:20: DeprecationWarning: ssl.wrap_socket = ssl.wrap_socket(client_socket, ca_certs="cert.crt")
Connected to server!

+-----+
| One Time Password (OTP) Based on Advanced Encrypted Standard |
| (AES) and Linear Congruential Generator(LCG) |
+-----+

+-----+
| Welcome to the Menu |
| |
| /signup <username> <password> <email> : signup |
| /signin <username> <password> : signin |
| /auth <OTP> : OTP |
| /resend : resend OTP |
| |
+-----+

$ /signin test test123
$ Please enter OTP to authorize
[POST]: OTP generated, open file test to get OTP
/auth 676868
$ [POST]: Authenticated
```

Figure 13 Example of enter OTP within valid time

Case 2: Enter the expired OTP

```
rosy@MMH: ~/Cryptography_F X root@DESKTOP-ORH3LTL: /mi X + v
root@DESKTOP-ORH3LTL:/mnt/d/Clone/New/Cryptography_Project/Console# python
/mnt/d/Clone/New/Cryptography_Project/Console/client.py:20: DeprecationWa
client_socket = ssl.wrap_socket(client_socket, ca_certs="cert.crt")
Connected to server!

+-----+
| One Time Password (OTP) Based on Advanced Encrypted Standard |
| (AES) and Linear Congruential Generator(LCG) |
+-----+

+-----+
| Welcome to the Menu |
| |
| /signup <username> <password> <email> : signup |
| /signin <username> <password> : signin |
| /auth <OTP> : OTP |
| /resend : resend OTP |
| |
+-----+

$ /signin test test123
$ Please enter OTP to authorize
[POST]: OTP generated, open file test to get OTP

$ /auth 420472
$ [POST]: OTP expired
```

Figure 14 Example of enter expired OTP

Case 3: Enter the wrong OTP, use resend

```
root@DESKTOP-ORH3LTL:/mnt/d/Clone/New/Cryptography_Project/Console# python3 client.py
/mnt/d/Clone/New/Cryptography_Project/Console/client.py:20: DeprecationWarning: ssl.wrap_socket() is deprecated
  client_socket = ssl.wrap_socket(client_socket, ca_certs="cert.crt")
Connected to server!

+-----+
| One Time Password (OTP) Based on Advanced Encrypted Standard |
| (AES) and Linear Congruential Generator(LCG) |
+-----+

+-----+
| Welcome to the Menu |
| /signup <username> <password> <email> : signup |
| /signin <username> <password> : signin |
| /auth <OTP> : OTP |
| /resend : resend OTP |
+-----+

$ /signin test test123
$ Please enter OTP to authorize
[POST]: OTP generated, open file test to get OTP

$ /auth 123456
$ [POST]: Wrong OTP

$ /resend
$ [POST]: New OTP generated, open file test to get OTP

$ /auth 071897
$ [POST]: Authenticated
```

Figure 15 Example of using resend

The server will check whether the account has already logged in. If it has, block further access to the account.

```
root@DESKTOP-ORH3LTL:/mnt/d/Clone/New/Cryptography_Project/Console# python3 client.py
/mnt/d/Clone/New/Cryptography_Project/Console/client.py:20: DeprecationWarning: ssl.wrap_socket() is deprecated
  client_socket = ssl.wrap_socket(client_socket, ca_certs="cert.crt")
Connected to server!

+-----+
| One Time Password (OTP) Based on Advanced Encrypted Standard |
| (AES) and Linear Congruential Generator(LCG) |
+-----+

+-----+
| Welcome to the Menu |
| /signup <username> <password> <email> : signup |
| /signin <username> <password> : signin |
| /auth <OTP> : OTP |
| /resend : resend OTP |
+-----+

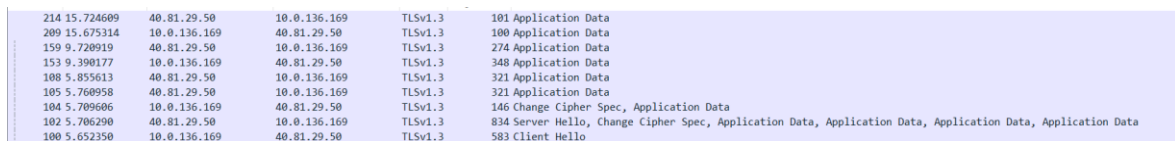
$ /signin test test123
$ [POST]: Already logged in
```

Figure 16 Examble of blocking multiple logins to account

TLS set up:

```
# Create an SSL context with TLS 1.3 support
context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
context.minimum_version = ssl.TLSVersion.TLSv1_3
context.load_cert_chain(certfile="cert.crt", keyfile="cert.key")

# Initialize server socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket = context.wrap_socket(server_socket, server_side=True)
server_socket.bind(('0.0.0.0', 1234))
server_socket.listen(10)
```



214	15.724609	40.81.29.50	10.0.136.169	TLSv1.3	101 Application Data
209	15.675314	10.0.136.169	40.81.29.50	TLSv1.3	100 Application Data
159	9.720919	40.81.29.50	10.0.136.169	TLSv1.3	274 Application Data
153	9.390177	10.0.136.169	40.81.29.50	TLSv1.3	348 Application Data
108	5.855613	40.81.29.50	10.0.136.169	TLSv1.3	321 Application Data
105	5.760958	40.81.29.50	10.0.136.169	TLSv1.3	321 Application Data
104	5.709606	10.0.136.169	40.81.29.50	TLSv1.3	146 Change Cipher Spec, Application Data
102	5.706290	40.81.29.50	10.0.136.169	TLSv1.3	834 Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data
100	5.652350	10.0.136.169	40.81.29.50	TLSv1.3	583 Client Hello

Figure 17 Packet captured from Wireshark

REFERENCES

Imamah (2018, October). One Time Password (OTP) Based on Advanced Encrypted Standard (AES) and Linear Congruential Generator(LCG). In 2018 *Electrical Power, Electronics, Communications, Controls and Informatics Seminar (EECCIS)* (pp. 391-394). IEEE.

TASK CHART

Task	Phan Thị Hồng Nhưng (21521250)	Đoàn Hải Đăng (21520679)	Lê Thanh Tuấn (21520518)
Topic research	x	x	x
Design architecture	x		
Database		x	x
Implement and Test	x	x	x
Presentation	x	x	x
Write the project report	x	x	
Efficiency	100%	100%	100%