UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

INTRODUCTION TO CRYPTOGRAPHY

Applying TOTP in Authenticator Application

Group Members

Le Anh Quang	22BI13380
Do Minh Quang	22BI13379
Nguyen Thu Trang	22BI13426
Tran Bao Nguyen	22BI13342
Nguyen Hoang Phat	22BI13356
Ho Thanh Thuy Tien	22BI13419
Vu Ha Vy	22BI13485

Lecturer

Dr. Nguyen Minh Huong

ICT Lab

January, 2025

Table of Contents

Chapter 1

Introduction

1.1 Context and Motivation

In today's modern society, securing sensitive information has become more and more essential. With the exponential growth of online services, Internet users are under serious threat of phishing, brute-force attacks, and data breaches. Traditional single-factor authentication, typically password-based, is often not enough to address these challenges. The increasing demand for high-security applications has led to a growing interest in protecting confidential data using passwords, tokens, biometrics, etc. As a result, Two-Factor Authentication (2FA) [1] has emerged as a robust mechanism to enhance security by adding another layer of verification.

This project studies the implementation of 2FA using Time-Based One-Time Passwords (TOTP) [2], a widely used cryptographic technique that applies algorithms like SHA-1 [3] and Hash-based Message Authentication Code (HMAC) [3]. The project's goal is to understand the underlying principles of these cryptographic methods and to demonstrate their practical application in securing user authentication.

The complete project code is available on Github[1].

1.2 Objectives

The project studies the principles and functioning of 2FA, focusing on TOTP. Furthermore, we want to implement a TOTP-based authentication system from scratch using SHA-1 and HMAC, simulate a server-client TOTP system, by generating and sharing a secret key via QR code, and comparing the user's token with the server's token. Finally, we evaluate the security and efficiency of the implemented system.

1.3 Expected Outcomes

The project's goal is to create a user-friendly, fast and efficient system. The computer server and client server have the ability to generate a 6-digit token sharing high similarity with existing applications such as Google Authenticator Application or Microsoft Authenticator Application. The generated token should change every 30 seconds.

[1] https://github.com/LAQ27504/MiniAuthenticatorProject

Chapter 2

Theoretical Background

2.1 Overview of Multi-Factor Authenticator

Two-factor authentication (2FA) is a security method that requires two different forms of identification to access a resource or data. The first factor is something you know, such as a username and password. The second factor is something you have, such as a code sent to your smartphone or biometrics like fingerprint, face, or retina recognition.

2FA is a type of multi-factor authentication (MFA) that protects against a variety of security threats, including phishing, social engineering, and password brute-force attacks. It is an effective way to protect sensitive data and prevent data breaches. Compared to single-factor authentication, which relies solely on a password, 2FA provides an additional layer of security that significantly reduces the likelihood of unauthorized access. This method is widely used in areas such as banking, enterprise systems, and online platforms to safeguard accounts and sensitive information.

2.2 TOTP

Time-based one-time password (TOTP) is a dynamic password generation algorithm that produces a unique, time-sensitive code based on a shared secret key and the current time. It is widely adopted in modern 2FA implementations, providing an extra layer of security against unauthorized access. Every fixed interval (usually 30 seconds), a new password is generated. This addresses several issues with traditional passwords such as being forgotten, stolen, or guessed. The 30-second interval is chosen to strike a balance between usability and security, ensuring that the code remains valid for a short period without frequent disruptions. TOTP is commonly used in authentication apps like Google Authenticator and Authy, making it a popular choice for securing online accounts and sensitive operations.

The process of generating a TOTP involves converting the current timestamp into a time counter based on the defined time step (e.g., 30 seconds). Next, the HMAC algorithm is applied using the shared secret key and the time counter as inputs. Finally, a portion of the resulting hash is extracted to create the OTP, typically a 6 or 8-digit code (figure 2.1).

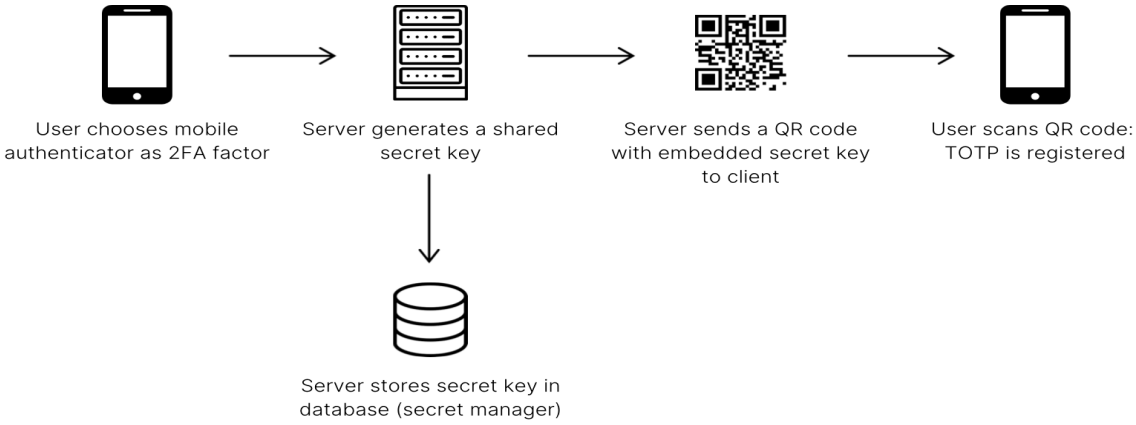

Figure 2.1: Workflow diagram of TOTP Registration

2.3 HMAC - SHA1

HMAC-SHA1 is a keyed hash algorithm that combines a secret key with the SHA-1 hash function to create a secure message authentication code. It works by mixing a secret key with the message, hashing the result, and reapplying the process to generate a 160-bit hash value. This ensures the integrity and authenticity of messages transmitted over insecure channels, as both the sender and receiver must share the secret key.

Although SHA-1 has known vulnerabilities, it remains suitable for HMAC due to the added security of the secret key. However, transitioning to stronger algorithms like HMAC-SHA256 is recommended. HMAC-SHA1 is commonly used in systems requiring secure authentication, such as generating TOTP codes.

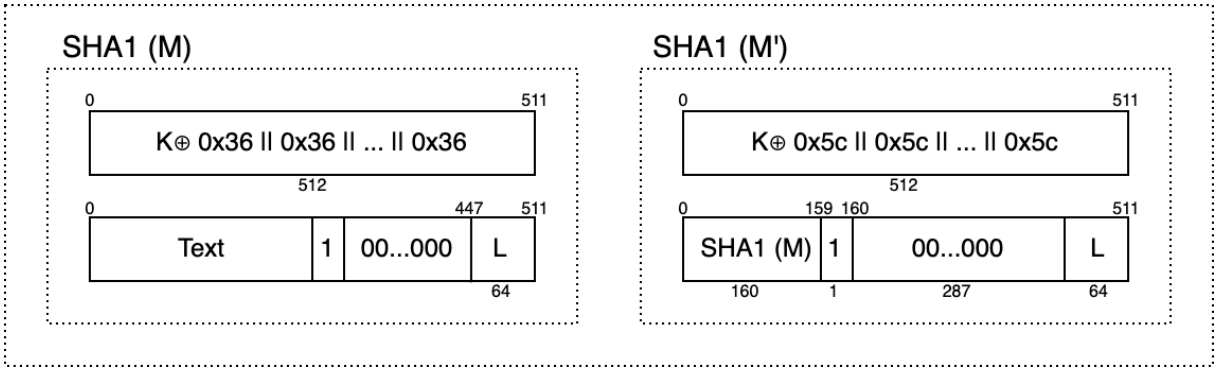

Figure 2.2: A graphical representation of HMAC-SHA1

The HMAC process [4] involves two stages: the inner and outer hash. The secret key (K) is XORed with constants $0x36$ (inner padding) and $0x5C$ (outer padding) to create distinct keys. The inner hash processes $K \oplus 0x36$ concatenated with the message, and the outer hash processes $K \oplus 0x5C$ concatenated with the inner hash output.

Chapter 3

Materials and Methodology

3.1 System Architecture

3.1.1 Generating secret key and URL

Our secret keys are generated by Python `secrets` module and are based on the Base32 encoding system. Base32 converts binary data into a readable text format using 32 characters (A-Z and 2-7). Base32 is easy to read, safe to use in URLs or file systems, and highly compatible. To calculate the number of bytes required to generate a secret key of a given length in Base32, the following formula is used:

$$\text{Number of Bytes} = \frac{\text{Key Length} \times 5}{8}$$

Each character in Base32 represents 5 bits of data. To determine how many bits are needed for a given number of Base32 characters, multiply the key length by 5. Since 1 byte equals 8 bits, divide the total number of bits by 8 to get the number of bytes. After the bytes are encoded, they are decoded into a UTF-8 string.

The authenticator general URL link will follow the format:

otpauth://totp/{label}?secret={secret}&issuer={issuer_str}

Where `label` identifies the token, `secret` is the secret key, and `issuer` is the service or service name. With this format, most authenticator apps can recognize and process the URL to extract important information for saving the data.

3.1.2 Generating 6-digit token

The process of generating a 6-digit token relies on the standard algorithm called HMAC SHA1, combined with a time-based variable. Once the secret key has been shared between the service and the app, a 6-digit token is created through a sequence of three main steps. First, a hash value is generated using the HMAC function, where the secret key and the current time (counter) serve as input parameters. Next, 32 bits are extracted from the hash value to form a truncated version. Finally, this truncated HMAC value is converted into a 6-digit token.

In the first step, a hash value will be generated using HMAC SHA1 function. The input for this function includes: the secret key and the counter. The secret key, which is a shared

secret between the service and the application application, ensures that only authorized parties can generate the same hash value. Meanwhile, the counter is based on the current Unix timestamp and updates every 30 seconds. By relying on the Unix timestamp, the system avoids issues caused by time zone differences between the service and the user. The length of the counter must be 8 bytes for compatibility with the algorithm.

The SHA-1 hash function produces a fixed length hash, typically 20 bytes (160 bits). The HMAC algorithm will take above 2 inputs and then applies the following formula:

$$\text{HMAC-SHA1}(K, T) = \text{SHA-1}((K \oplus \text{opad}) \| \text{SHA-1}((K \oplus \text{ipad}) \| T))$$

Where:

- K: secret key
- T: time counter
- $\oplus$: the bitwise exclusive
- $||$: concatenation
- ipad: the inner padding
- opad: the outer padding
- SHA1: SHA-1 cryptographic hash function.

For the next step, as the original hash value is much larger than required, 32 bits will be extracted for easier management of bits. The truncated 4 bytes from hash value will then be converted into integer value which will serve as the foundation for the 6 - digit token. Finally, a modulo operation will be applied to the 32-bit integer to ensures that the result is always a 6-digit number. In case if the result of modulo generates fewer than 6 digits, the result will then be pad with leading zeros.

3.2 Simulating system

Based on the TOTP verification model, we build a simulation system which has 2 sides, service and app. Service side represents the service or service that users need to use 2FA, and the application side represents the authenticator application.

Starting with the service side, it will ask the user to enter the personal information for the label to identify its secret key. Then the service will generate a secret key with the following URL with the structure of label, secret key and issuer. With the Python library qrcode, the URL will convert into QR image and save a copy of it in the local system. After generating the QR, the service will simulate the authenticate step by generating 6-digit token for validation and requiring a user token for checking with the service token. If the user token matches the service token, the service-side will respond with “validation”, otherwise, the service will ask the user to enter the token again.

On the application-side, we automate the scanning QR step from the saved QR code local and get the URL from it which will provide the application with the generated URL of all the filled important information. That information will be processed then get the shared

secret key and use the HOTP function to generate a token for the user. Moreover, for every 30 seconds, the otp code will be renewed following the TOTP model. The workflow can be illustrated by the diagram below.

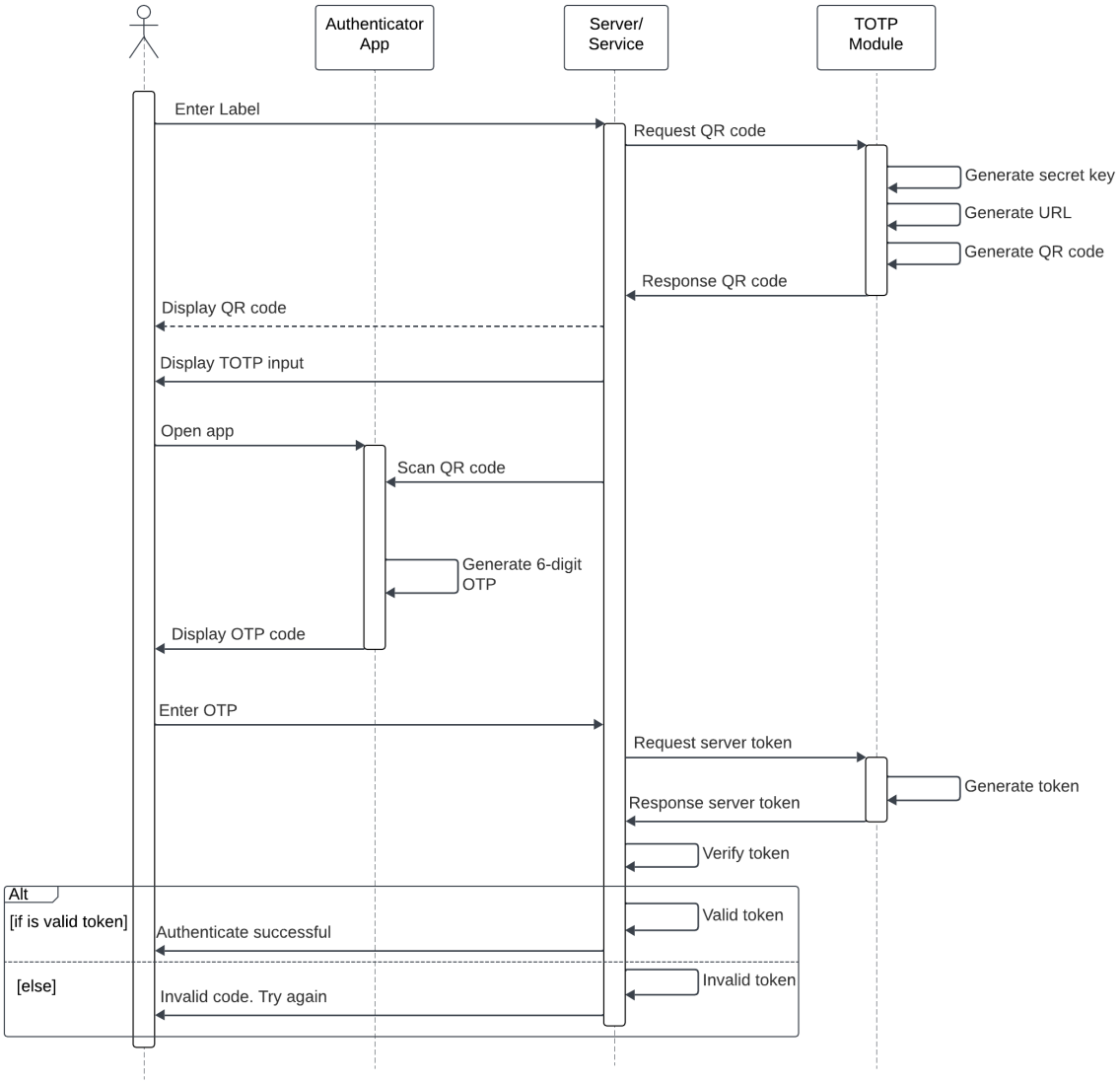

Figure 3.1: Authenticator simulation sequence diagram

3.3 Model Evaluation

3.3.1 OTP Expiration

Unlike HOTP [5], which is an event-based OTP algorithm that uses a shared secret key and an event counter, TOTP is a time-based OTP algorithm that uses a shared secret key and a time counter. This time-based characteristic makes OTPs in TOTP valid only for a short period of time. To ensure our code works correctly, we include tests for expired OTPs by deliberately using an old OTP, which should trigger an authentication failure.

3.3.2 Time Drift

In addition, the time drift between two systems also has been considered. Time drift refers to the gradual difference in time between two systems that are not perfectly synchronized [2]. When a application and service are generating OTPs based on different system clocks, slight discrepancies in time can cause the application and service to use different time steps, resulting in either invalid OTPs or synchronization errors.

To tackle this challenge, the time sync software needs to be set to check a specific number of time intervals within a specific time window before the OTPs are rejected and access is denied.

In the case of a 30-second time step, the system will allow OTPs from the previous time step (up to 30 seconds before the current time), the current time step, and the next time step (up to 30 seconds after the current time). This means that the system can accept OTPs generated within the time window of ± 30 seconds from the service's time.

3.3.3 Resynchronization

This section illustrates the effect of clock skew on OTP validity. The test token shared secret uses the base64 encoded value:

```
3N6IXFJWA4HTEL7NXHIG3I2H5BTVVXQDHDZJWRJYW4PGTFWVYBDBQIZ4K5Z66GQU
```

With Time Step $X = 30$ seconds and the Unix epoch as the initial reference point for counting time steps, the TOTP algorithm produces the following values for the specified timestamps:

Application Time Offset (s)	**UTC Time**	**Application TOTP**	**Accepted**
-90	2009-02-13 23:30:00	417031	NO
-60	2009-02-13 23:30:30	049659	NO
-30	2009-02-13 23:31:00	915681	YES
0	2009-02-13 23:31:30	678030	YES
30	2009-02-13 23:32:00	711501	YES
60	2009-02-13 23:32:30	755072	NO
90	2009-02-13 23:33:00	917627	NO

Table 3.1: TOTP Validation Results with Time Offset

As seen in Table 3.1, OTPs generated within the same 30-second interval have been accepted.

Chapter 4

Conclusion and Future Work

4.1 Conclusion

This project successfully implemented a TOTP authentication system, integrating server-client simulation, QR code generation, and token validation. By employing cryptographic techniques like HMAC and SHA-1, we demonstrated the method's effectiveness in enhancing security, reducing unauthorized access, and safeguarding sensitive information. The results highlight TOTP's practicality in bolstering authentication mechanisms for modern applications. This project serves as a foundational step for integrating robust authentication techniques into various security-critical systems.

4.2 Future Work

Future efforts could focus on improving user-friendly designs to make TOTP accessible for non-technical users, enabling offline TOTP generation for greater reliability in diverse environments, and integrating with popular applications via APIs for seamless operation. Additionally, offering customizable options such as code length and lifespan would further enhance user experience and flexibility. Exploring the application of TOTP in emerging technologies like IoT and blockchain could also present innovative security solutions.

4.3 References

[1] T. Petsas, G. Tsirantonakis, E. Athanasopoulos, and S. Ioannidis, "Two-factor authentication: is the world ready? quantifying 2fa adoption," in *Proceedings of the eighth european workshop on system security*, 2015, pp. 1–7.

[2] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, "Totp: Time-based one-time password algorithm," Tech. Rep., 2011.

[3] H. Krawczyk, M. Bellare, and R. Canetti, "Hmac: Keyed-hashing for message authentication," Tech. Rep., 1997.

[4] A. Visconti and F. Gorla, "Exploiting an hmac-sha-1 optimization to speed up pbkdf2," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 4, pp. 775–781, 2018.

[5] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen, "Hotp: An hmac-based one-time password algorithm," Tech. Rep., 2005.