Experimental Analysis of Time-Based One-Time Password (TOTP) Systems: Quantifying Performance, Clock Skew Resilience, and Usability-Security Trade-offs

Lydia Gracia - 18222035
Information System & Technology Study Program
School of Electrical Engineering and Informatics
Institut Teknologi Bandung, Ganesha Street No. 10 Bandung
E-mail: ly.gracies@gmail.com , 18222035@std.stei.itb.ac.id

***Abstract*—This research provides an experimental investigation of Time-Based One-Time Password (TOTP) systems, a commonly utilized multi-factor authentication mechanism. The study includes quantitative assessment of the operational performance of TOTP through direct implementation of the RFC 6238 standard, focusing on the generation and verification speeds for several HMAC algorithms. Moreover, this study provides a thorough empirical analysis of how network client-server clock skew and network latency affect authentication success rates. Our findings show that although TOTP operations are computationally trivial and take place in microseconds, network latency significantly reduces the effective OTP validity window and its resilience to clock desynchronization is directly proportional to specified tolerance levels. Different HMAC algorithms exhibit subtle performance variations, with SHA-1 being the fastest but cryptographically weakest, and SHA-512 being the strongest but slightly slower. The result can provide insights to help developers and system administrators to create, configure, and implement reliable and easy-to-use authentication solutions.**

Keywords—component; TOTP, One-Time Password, Authentication, Clock Skew, Usability, Security, HMAC

I. INTRODUCTION

Time-Based One-Time Password (TOTP) stands as a cornerstone of modern multi-factor authentication (MFA), offering a robust defense against common cyber threats like credential stuffing and phishing, which often bypass traditional password-only or SMS-based OTP systems. Defined by RFC 6238, TOTP leverages a shared secret key and the current time to generate a unique, short-lived password, significantly enhancing authentication security. Despite its widespread adoption, a comprehensive empirical understanding of TOTP's real-world performance characteristics and its resilience to operational challenges, particularly client-server clock desynchronization and network latency, remains vital for optimal deployment.

Time-Based One-Time Passwords (TOTP), which are frequently used in modern multi-factor authentication (MFA), offer a robust defense against prevalent cyberthreats including phishing and credential stuffing. By creating a unique, short-lived password using a shared secret key and the current time, TOTP significantly increases authentication security. However, optimal deployment of TOTP still requires a thorough empirical understanding of its real-world performance characteristics and its ability to withstand operational difficulties, especially client-server clock desynchronization and network latency.

This report addresses these areas by quantitatively evaluating TOTP's operational efficiency, empirically analyzing its tolerance to clock skew and network latency. This paper specific objectives are to (1) measure the average execution times for TOTP generation and verification operations; (2) empirically examine the effects of network client-server clock skew and network latency on TOTP authentication success rates; and (3) examine the performance characteristics of various HMAC algorithms (SHA-1, SHA-256, SHA-512); and (4) discuss the practical usability-security trade-offs related to clock skew tolerance configuration and the impact of network conditions on TOTP systems.

II. BACKGROUNDS AND RELATED WORKS

A. Fundamentals of OTPs and Cryptography

Multi-Factor Authentication (MFA) schemes enhance security by requiring users to present two or more pieces of evidence (factors) to verify their identity [1]. These factors typically fall into three categories: something the user knows (e.g., a password), something the user has (e.g., a smartphone, a hardware token), or something the user are (e.g., a fingerprint, facial recognition). One-Time Passwords (OTPs) primarily leverage the "something you have" factor, where a dynamically generated code is sent to or produced by a trusted device in the user's possession. The ephemeral nature of OTPs makes them highly resistant to replay attacks, brute-force

attacks, and credential stuffing, which are common vulnerabilities of static passwords [2].

B. Time-Based One-Time Password (TOTP) - RFC 6238

According to RFC 6238 [3], the Time-Based One-Time Password (TOTP) technique is a commonly used standard for creating temporary, time-synchronized authentication credentials. Building upon the principles of HMAC, TOTP offers a stateless alternative that simplifies server-side management by eliminating the need to synchronize the event counters (as seen in HOTP, HMAC-Based One-Time Password [4]). It does this by substituting an event-based counter for a time-based factor.

The TOTP algorithm's core components are a dynamically changing time value (T) and a shared secret key (K), which are only known by the client (authenticator app) and the server. The time value T is derived by dividing the current Unix time (seconds since the Unix epoch, January 1, 1970, 00:00:00 UTC) by a predefined time step (X), commonly set to 30 or 60 seconds [5]. This efficiently divides time into distinct, sequential periods. The formula for calculating T is given by

$$T = \lfloor (\text{Current Unix Time} - T0)/X \rfloor \quad (1)$$

where T0 is the Unix time epoch (typically 0, unless specified otherwise for synchronization purposes), and X is the time step in seconds.

Once the time value T is computed, it is used as the "message" input for the HMAC calculation. The following formula is then used to generate the complete TOTP value:

$$TOTP = TRUNCATE(HMAC\text{-}SHA(K,T)) \quad (2)$$

The HMAC digest produced with the shared secret key K, the time value T, and a selected SHA cryptographic hash function (such as SHA-1, SHA-256, or SHA-512) is denoted here by the notation HMAC-SHA(K,T). This HMAC digest is then transformed into a human-readable, fixed-digit OTP using the TRUNCATE function. The following phases make up the truncation procedure, which was taken from RFC 4226 Section 5.3 [3]:

1) Offset Calculation: As an offset, the last four bits of the last byte of the HMAC digest are utilized. For example, if the last byte is 0x3A, the last 4 bits are 0xA (decimal 10), so the offset is 10.

2) Dynamic Truncation: Four bytes are extracted from the HMAC digest starting from the calculated offset.

3) MSB Clearing: To generate the final, fixed-length decimal OTP, this 31-bit integer is then modulo 10^{digits}, where digits is the required OTP length, usually 6 or 8.

4) Modulo Operation: This 31-bit integer is then taken modulo 10^{digits} (where digits is the desired OTP length, typically 6 or 8) to produce the final, fixed-length decimal OTP.

This time-dependency is a key strength of TOTP since it limits the validity of codes to a relatively short time, making them naturally resistant to replay attacks beyond that time window. However, this also creates a crucial reliance on precise time synchronization between the server that verifies the OTP and the client that generates it.

C. Related Work

Various aspects of OTP systems have been thoroughly examined in previous studies, including their general security against relay attacks and brute-force attempts [6]. Studies on the performance of cryptographic primitives and authentication mechanisms [7] generally indicate that the underlying cryptographic operations are efficient. However, specific empirical benchmarks that focus on combined speeds of TOTP creation and verification, especially in Python implementations, are less often described.

Moreover, the issue of clock synchronization in distributed systems and its effect on time-sensitive protocols such as TOTP has been recognized [8], alongside some practical concern on the *allowed_skew* parameter [9]. The impact of network latency on TOTP authentication, while intuitively understood, has not been rigorously quantified in many studies. Similarly, while the security and performance characteristics of different HMAC algorithms are well-studied in other contexts, their specific impact within the TOTP framework warrants empirical investigation.

This report extends existing knowledge by providing direct empirical quantification of TOTP performance, conducting a detailed analysis of its resilience to various client-server clock offsets and network latencies, and comparing the performance of different HMAC algorithms, offering concrete data to inform the critical usability-security trade-off in real-world TOTP deployments.

$$\alpha + \beta = \chi. \quad (1) \quad (1)$$

III. METHODOLOGY

A. TOTP Algorithm Implementation Details

RFC 6238's requirements were carefully followed in the implementation of the Time-Based One-Time Password (TOTP). It generates 6-digit OTPs by default using SHA-256 as the underlying HMAC (Hash-based Message Authentication Code) method.

The OTP generation process fundamentally involves computing an HMAC-SHA256 digest. This digest is derived from a time-dependent integer (specifically, the current Unix timestamp divided by a predefined time step, typically 30 seconds, to yield the current "time window") and a shared secret key. Following this, a dynamic truncation algorithm is applied to a specific portion of the resulting digest to extract and produce the final N-digit OTP. The cryptographically secure pseudo-random number generator (CSPRNG) in the operating system is used by Python's secrets module [9] to produce the shared secret itself, guaranteeing robust randomness.

The system's goal for OTP verification is to verify an OTP by comparing it to a variety of potentially legitimate OTPs. Potentially legitimate OTPs are defined by the *allowed_skew* parameter, consisting of the server's current time

window, as well as an adjustable number of adjacent time windows (both past and future). This multi-window checking mechanism is crucial for accommodating minor clock desynchronization between the client and server, and for mitigating the impact of network latency on authentication success [3]. The underlying OTP computation within the verification process was specifically designed to dynamically generate and compare OTPs for each of these potential time windows *(T±i×time_step)*. According to the RFC, this method made sure that the experimental setup properly replicated client-side time offsets and network-induced delays, and that the server-side verification mechanism evaluated these against its defined tolerance windows.

B. Experimental Design

Three main studies were carried out: network latency impact testing, clock skew resilience testing, and performance benchmarking.

1) Performance Benchmarking: Performance was evaluated by measuring the average execution time for TOTP generation and verification operations. Each operation was executed 100,000 times, utilizing Python's time.perf_counter() for high-resolution timing. Tests were conducted with SHA-1, SHA-256, and SHA-512 as the HMAC algorithms and generated 6-digit OTPs to compare their performance characteristics. Figure 1 provides a detailed flowchart illustrating the experiment.

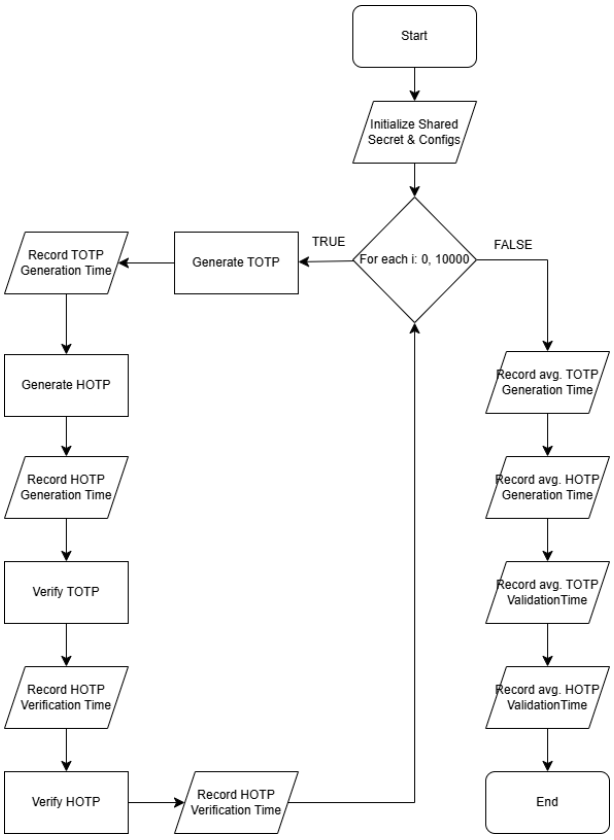

Fig. 1. Performance Benchmarking Flowchart

2) Clock Skew Resilience Testing: Client-server clock skew resilience was assessed by simulating various client clock offsets ranging from −60 seconds to +60 seconds, in 1-second increments. For each client offset, 100 OTPs were precisely generated on the client side using the corresponding skewed time. These generated client OTPs were then submitted for verification against a simulated server. The server's OTP validation process was configured with *allowed_skew* settings of 0, 1, and 2 time steps. Each time step represents 30 seconds of tolerance in either direction (i.e., a 1-step skew allows for checking current, previous, and next 30-second windows). The success rate for each client offset and server skew configuration was recorded. Figure 2 provides a detailed flowchart outlining the experiment.

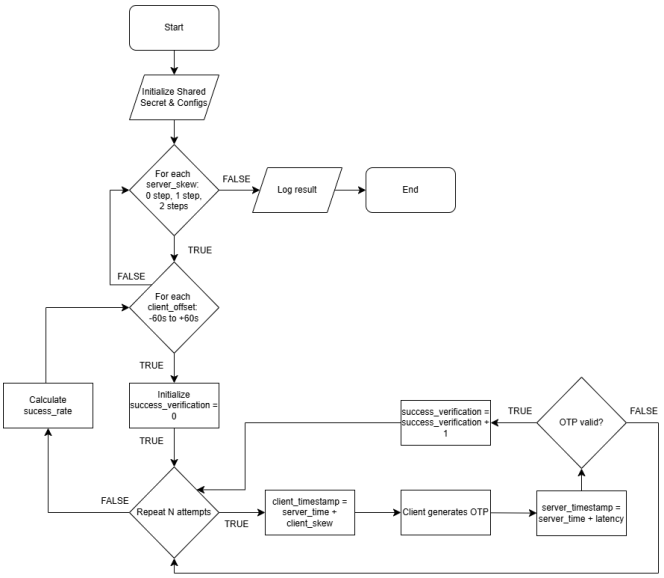

Fig. 2. Clock Skew Resilience Flowchart

3) Network Latency Impact Testing: This experiment builds upon previous analysis by varying both client-side clock skew and network latency. The latency values range from 0 milliseconds to 60,000 milliseconds (1 minute). For each latency value, and for server allowed_skew settings of 0, 1, and 2 time steps, 30 OTP verification attempts were made. These tests were run at clock drift of 0s, -10s, and 10s. The success rate for each latency, skew, and offset combination was recorded. Figure 3 provides a detaled flowchart outlining the experiment.

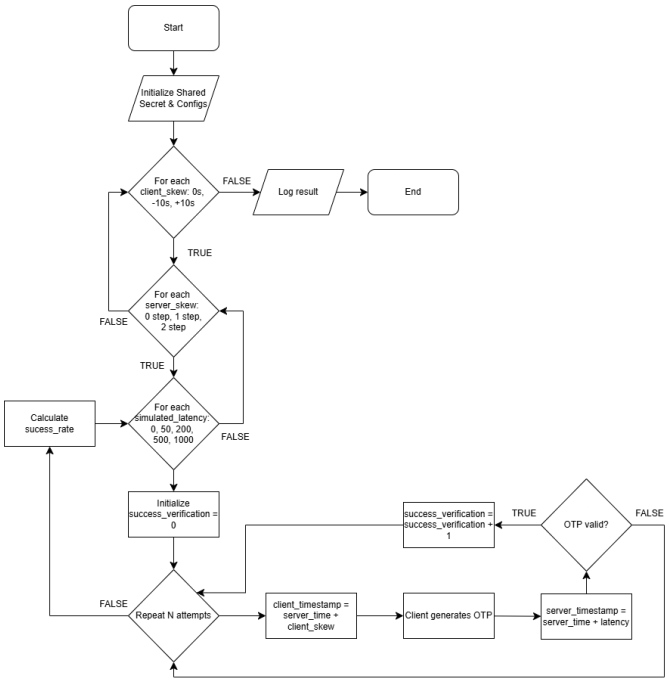

Fig. 3. Clock Skew Resilience Flowchart

IV. RESULTS AND ANALYSIS

A. *Performance Analysis*

The empirical analysis of TOTP operations yielded the following average execution times.

TABLE I. RESULT OF TOTP VS. HOTP PERFORMANCE ANALYSIS

Operation	SHA-1 (seconds)	SHA-256 (seconds)	SHA-512 (seconds)
TOTP Generation (6-digit)	0.000016	0.000017	0.000019
HOTP Generation (6-digit)	0.000017	0.000016	0.000019
TOTP Verification (6-digit, Skew: 1)	0.000034	0.000035	0.000038
HOTP Verification (6-digit, Window: 10)	0.000018	0.000017	0.000019

As described in the table, TOTP generation and verification are both computationally lightweight operations that take only a few microseconds to complete. It is also observed that performance differences between the HMAC algorithms only spans around 0,003ms to 0,004ms, with SHA-1 being the fastest performance and SHA-512 is slightly slower, likely due to the larger block size and more complex computations involved. The performance differences, however, are negligible and are unlikely to be noticeable in typical use cases.

B. *Clock Skew Resilience*

The clock skew analysis reveals a direct and quantifiable correlation between the server's *allowed_skew* configuration and its tolerance to client clock desynchronization. Figure 4 below shows the observed success rates for various client offsets.

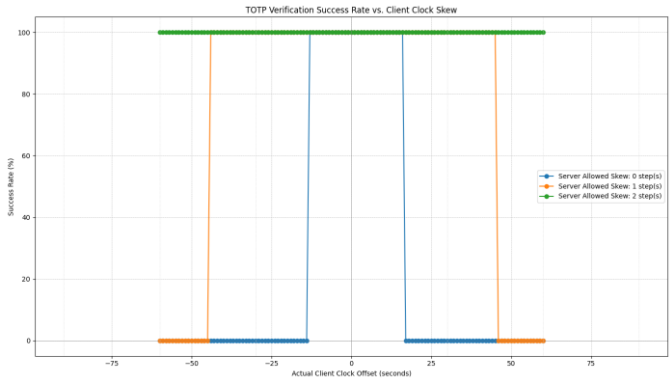

Fig. 4. TOTP Verification Success Rate vs.Client Clock Skew

As illustrated in Figure 4, the observed success rates for various client offsets are:

1) allowed_skew = 0: With no tolerance for adjacent time windows, successful verification was highly restrictive. Empirical results show a 100% success rate only for client offsets between approximately −15 seconds and +5 seconds relative to the server's time window center. Outside of this small window, all tested offsets had 0% success. This shows that the precise current time step must be met without any buffer.

2) allowed_skew = 1: Allowing for one adjacent time window in each direction significantly extended the successful verification window, achieving a 100% success rate for client offsets ranging from approximately −45 seconds to +45 seconds (90 seconds timeframe). This illustrates how well one-step skew can account for common minor clock drifts.

3) allowed_skew = 2: This configuration achieves a 100% success rate for client offsets ranging from approximately −60 seconds to +60 seconds (120 seconds time frame). This wider window offers greater resilience against more significant clock desynchronization.

These findings represent the intrinsic usability-security trade-off in TOTP's *allowed_skew* parameter. For instance, a 2-step skew allows an OTP to be valid for up to 2 minutes, compared to the standard 30-second window for a 0-step skew. The higher *allowed_skew* value will directly enhance user experience by forgiving greater clock discrepancies, which is crucial in real-world scenarios where perfect clock synchronization is challenging. However, with significantly more valid time, the 2-step skew conformation is more prone to brute-force attack. Needless to say, system administrators must carefully weigh the security consequences of a longer OTP validity window against the goal for better usability.

C. *Network Latency-Clock-Skew Impact*

This section presents the empirical findings on how various combinations of client-side clock skew, network latency, and server-side *allowed_skew* impacts the success rate of the TOTP

authentication. The visual representation of both variables' complex relationship can be seen in Figure 5, 6, and 7.

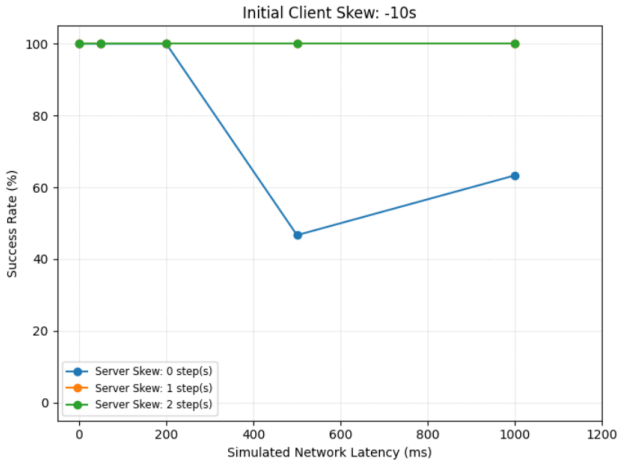

Fig. 5. Network Latency-Clock-Skew Variance vs. Success Rates in Clients with Initial Client Skew -10s

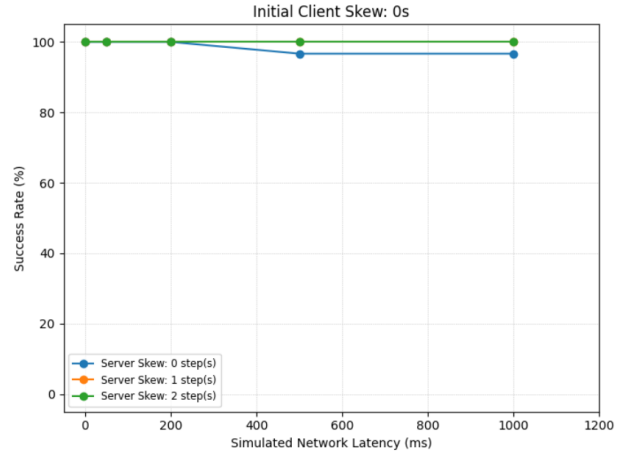

Fig. 6. Network Latency-Clock-Skew Variance vs. Success Rates in Clients with Initial Client Skew 0s

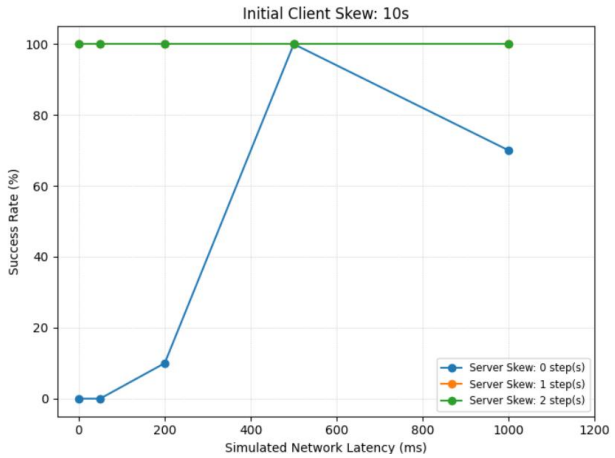

Fig. 7. Network Latency-Clock-Skew Variance vs. Success Rates in Clients with Initial Client Skew +10s

Figure 5, 6, and 7 proves the server's *allowed_skew* settings importance in determining the system's tolerance to both initial client clock desynchronization and subsequent network latency. Data with a strict *allowed_skew = 0* impact clients with perfect clock synchronization for delay at 1000ms with a slight drop to 96.67%.

Clients with an initial skew of -10 seconds, however, experience a greater impact, where success rates plummeted to 63.33% with latencies of 500ms and 1000ms. This result shows that no server-side tolerance for clock skew might make the OTP generated by the client to become invalid due to the combined effect of negative skew and network delay before it reaches the server.

Even more extreme for clients with an initial clock skew of +10 seconds, that consistently gives 0% success rate for minimal latency. The +10 seconds client offset causes the client to generate OTPs for the next time step before the server has officially entered that time step. However, with network delay, later the server's clock catches up and transitions into the time step for which the client's OTP was generated. As a result, the success rate in the range of 500ms to 1000ms finally increases to an average of 80%. This interesting finding introduces us to a possible vulnerability of TOTP, forward-replay attack [10].

On the other hand, cases with *allowed_skew = 2* shows 100% success rate in all of the test cases. The result further shows that without sufficient server-side skew tolerance, even relatively small network latencies (e.g. 500ms to 1000ms) can cause significant authentication failures, especially when coupled with existing client-server clock discrepancies. It also highlights the security issues implications on larger *allowed_skew*.

V. CONCLUSION

This study has demonstrated several key features of Time-Based One-Time Passwords, including their remarkable computational efficiency, their adaptable robustness to client-server clock desynchronization, and their susceptibility to network latency. Our findings confirm that TOTP operations pose negligible performance overhead for authentication systems and the choice of HMAC algorithm has a minor impact on performance. SHA-1 is the fastest, but SHA-256 and SHA-512 offer stronger cryptographic security.

The clock skew and network delay tolerance test reveal a usability-security trade-off. It is crucial to implement reasonable server-side's allowed clock skew to ensure high authentication success rates in environments with varying network conditions and potential client clock inaccuracies. However, the optimal value should be determined based on an assessment of expected network latency distribution and typical client device clock synchronization accuracy. Overly strict skew settings can lead to usability issues, while overly lenient settings could potentially weaken the time-based security aspect of TOTP. Thus, it is recommended to consider the following aspects before configuring and implementing reliable TOTP systems.

1) Implement Server-Side Clock Skew Tolerance: Prioritize a sensible value on the server to balance security (smaller window) and usability/reliability (larger window). Based on this analysis, setting server skew to tolerate one step clock skew can provide excellent resilience. Given a standard TOTP time step of 30 seconds, this means an effective window of 90 seconds (30s past, 30s current, and 30s future). Tolerating two step clock skew is generally still acceptable and sufficient for most practical applications.

2) Monitor and Address Clock Synchronization: While server-side tolerance helps, it is still a good practice to encourage client clock synchronization. For example, via NTP. This increases security and reduces reliance on larger server-side skew allowances.

3) Consider Network Conditions: The result shows that high latency coupled with client skew can be problematic without any clock skew tolerance. For critical application, consider network performance and potencial latency spikes when determining the time step and allowed clock skew.

These findings provide developers and system administrators with practical information that helps them configure TOTP systems in a way that strikes the best possible balance between strong security, a smooth user experience, and adaptability to actual network conditions. Potential options for future research include investigating how TOTP can be integrated with other multi-factor authentication factors, comparing TOTP with new authentication standards like FIDO2/WebAuthn, or conducting user studies to assess the security and usability implications of various allowed clock skew settings and network latency conditions from the user's perspective.

SOURCE CODE REPOSITORY AT GITHUB

https://github.com/gracialy/totp-experiment

VIDEO LINK AT YOUTUBE

https://youtu.be/CjsjZdQSQOc

ACKNOWLEDGMENT

The author would like to express sincere gratitude to Dr. Ir. Rinaldi Munir, M.T. for his invaluable guidance and insightful lectures. The author also extends appreciation to II4021 Cryptography classmates for their constant support and collaborative spirit. Finally, special thanks are due to Distributed Systems 21' Lab Assistants for inspiring the author to look more into TOTP mechanism.

REFERENCES

[1] L. A. Meyer, S. Romero, G. Bertoli, T. Burt, A. Weinert, and J. L. Ferres, "How effective is multifactor authentication at deterring cyberattacks?," arXiv.org, May 01, 2023. https://arxiv.org/abs/2305.00945

[2] "CWE - CWE-308: Use of Single-factor Authentication (4.4)," cwe.mitre.org, Oct. 26, 2023. https://cwe.mitre.org/data/definitions/308.html

[3] D. M'Raihi et al., "TOTP: Time-Based One-Time Password Algorithm," datatracker.ietf.org, May 2011. https://datatracker.ietf.org/doc/html/rfc6238

[4] D. M'Raihi et al., "HOTP: An HMAC-Based One-Time Password Algorithm," Ietf.org, 2022. https://www.datatracker.ietf.org/rfc/rfc4226.txt

[5] J. Hoagland, "Multi-factor Authentication Using Time-based One-Time Password (TOTP)," Pangea. https://pangea.cloud/securebydesign/authn-using-totp/

[6] L. Lumburovska, J. Dobreva, S. Andonov, H. Trpcheska, and V. Dimitrova, "A Comparative Analysis of HOTP and TOTP Authentication Algorithms. Which one to choose?," 2021. Available: https://stumejournals.com/journals/confsec/2021/4/131.full.pdf

[7] A. Rahmatulloh, R. Gunawan, and F. M. S. Nursuwars, "Performance comparison of signed algorithms on JSON Web Token," IOP Conference Series: Materials Science and Engineering, vol. 550, p. 012023, Aug. 2019, doi: https://doi.org/10.1088/1757-899x/550/1/012023.

[8] Emin Huseynov and J.-M. Seigneur, "Hardware TOTP tokens with time synchronization," 2019 IEEE 13th International Conference on Application of Information and Communication Technologies (AICT), Oct. 2019, doi: https://doi.org/10.1109/aict47866.2019.8981762.

[9] "PEP 506 – Adding A Secrets Module To The Standard Library | peps.python.org," peps.python.org. https://peps.python.org/pep-0506/

[10] G. Bianchi and L. Valeriani, "Time Is on My Side: Forward-Replay Attacks to TOTP Authentication," Security and Privacy in Social Networks and Big Data, pp. 109–126, 2023, doi: https://doi.org/10.1007/978-981-99-5177-2_7.

STATEMENT

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Juni 2025

Lydia Gracia 18222035