Design and Performance Evaluation of SRP-6a as a Replay-Resistant Authentication Scheme in Web Applications

Darren Mansyl - 18223001
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: darrenmansyl05@gmail.com , 18223001@std.stei.itb.ac.id

***Abstract*— Traditional web authentication mechanisms often require passwords to be transmitted over a network, creating potential exposure to interception and replay attacks despite the use of Transport Layer Security (TLS). This paper presents the implementation of the Secure Remote Password (SRP-6a) protocol as a secure and replay-resistant authentication solution. SRP-6a is an augmented Password-Authenticated Key Agreement (PAKE) protocol that enables mutual authentication without transmitting passwords on the server. The proposed system integrates SRP-6a into a modern web application architecture and evaluates its computational performance against conventional hash-based authentication methods. Experimental results from 1,000 authentication sessions show that SRP-6a achieves a total computational latency of approximately 40 ms, outperforming bcrypt at around 50 ms. Moreover, its server-side processing cost is approximately 17 ms, representing a 66% reduction compared to bcrypt. A qualitative security analysis further demonstrates that SRP-6a effectively mitigates password disclosure, replay attacks, and server-side credential compromise while maintaining strong mutual authentication guarantees. These findings indicate that SRP-6a provides a practical balance between security and performance for modern web authentication systems.**

Keywords—SRP-6a, Authentication, Replay Attack, Web Security, Cryptography, PAKE

I. INTRODUCTION

Web-based services have become essential to modern commerce, communication, healthcare, and critical infrastructure, making secure user authentication a fundamental security requirement. Despite significant advances in authentication technologies, most web applications continue to rely on the traditional client–server model in which a password or password-derived value is transmitted to a remote server for verification [1]. While passwords are typically protected using cryptographic hash functions and transmitted over Transport Layer Security (TLS), this architecture still exposes users and service providers to several well-known security risks.

One major concern is credential database compromise. Even when passwords are stored using computationally expensive hashing algorithms such as bcrypt, attackers who obtain the database can perform offline dictionary or brute-force attacks against the stored hashes [1]. In addition, traditional password-based authentication remains vulnerable to phishing attacks because users ultimately disclose their credentials to the server. TLS protects credentials from passive network interception but cannot prevent users from submitting passwords to a malicious server that successfully impersonates a legitimate service. Furthermore, systems that rely on reusable authentication artifacts may be susceptible to replay attacks, in which previously captured authentication messages are reused to gain unauthorized access [2].

Password-Authenticated Key Agreement (PAKE) protocols were developed to address these limitations. PAKE protocols enable two parties to authenticate using a shared password without transmitting the password itself and without requiring a public-key infrastructure or trusted third party. Among the various PAKE schemes proposed in the literature, the Secure Remote Password (SRP) protocol is one of the most widely deployed. The SRP-6a revision introduces improvements that eliminate weaknesses present in earlier versions and is standardized in RFC 5054 for use in TLS-based authentication systems [3], [4].

SRP-6a provides mutual authentication and session key establishment while ensuring that intercepted protocol messages do not enable offline password guessing attacks. Because servers store only password-derived verifiers rather than passwords themselves, database compromise does not immediately reveal user credentials. Despite these advantages, SRP-6a is often perceived as computationally expensive due to its reliance on modular exponentiation operations, limiting its adoption in modern web application frameworks.

This paper presents the design, implementation, and evaluation of an SRP-6a-based authentication system for modern web applications. A Node.js prototype implementing the RFC 5054 specification is developed and evaluated through both qualitative and quantitative experiments. Security is assessed using four attack scenarios that examine replay resistance, verifier compromise, server impersonation, and normal authentication behavior. Performance is evaluated through a benchmark of 1,000 authentication sessions and compared against a bcrypt-based authentication model.

II. THEORETICAL BACKGROUND

A. Password-Authenticated Key Agreement (PAKE)

Password-Authenticated Key Agreement (PAKE) is a class of cryptographic protocols that enables two parties to establish a secure and authenticated communication channel using only a shared low-entropy secret without relying on a Public Key Infrastructure (PKI) or a trusted third party. Unlike conventional password-based authentication systems, in which a password or its hash is transmitted to a server for verification, PAKE protocols are designed so that the password itself is never sent over the network and cannot be verified through offline attacks using intercepted messages.

PAKE protocols are generally divided into two categories: balanced PAKE and augmented PAKE. In balanced PAKE, both parties store the same password and participate symmetrically in the authentication process. In augmented PAKE, the server stores only a verifier derived from the password rather than the password itself. As a result, even if the server database is compromised, an attacker cannot immediately recover user credentials or impersonate the client without performing a computationally expensive attack.

The security of PAKE protocols has been formally analyzed using models such as the Bellare–Pointcheval–Rogaway (BPR) model [5] and the Universal Composability (UC) framework. These models evaluate resistance against active adversaries capable of controlling network communications and initiating concurrent protocol sessions. A secure PAKE protocol is expected to provide mutual authentication, session key confidentiality, and protection against dictionary attacks, while limiting attackers to a single password guess per server interaction [5].

Several PAKE protocols have been proposed over the years, including SPEKE, the SRP family, CPace, and the more recent OPAQUE protocol. Among these, SRP-6a remains one of the most widely deployed in practice due to its maturity, interoperability with existing web technologies, and relatively straightforward implementation over finite cyclic groups.

B. The SRP-6a Protocol

The Secure Remote Password protocol is an augmented PAKE scheme designed to provide secure password-based authentication without exposing user credentials during transmission. This study focuses on SRP-6a, an improved revision of the original protocol introduced by Wu [3]. SRP-6a operates over a large cyclic group defined by a safe prime N and a generator g, with its security primarily relying on the computational hardness of the discrete logarithm problem.

During registration (Registration Phase), the client generates a random salt s and computes a private value x by hashing the salt together with the user's password p:

$$x = H(s \parallel p) \tag{1}$$

A password verifier v is then derived as the group element:

$$v = g^x \ (\boldsymbol{mod\ N}) \tag{2}$$

The server stores the user's identifier, salt, and verifier. Because the plaintext password is discarded after registration and never transmitted to the server, a compromise of the verifier database does not directly reveal user credentials.

To initiate authentication (Authentication Phase), the client and server each generate fresh ephemeral secrets. The client selects a random value a and computes:

$$A = g^a \ (\boldsymbol{mod\ N}) \tag{3}$$

Similarly, the server selects a random value b and computes:

$$B = kv + g^b \ (\boldsymbol{mod\ N}) \tag{4}$$

where:

$$k = H(N, g) \tag{5}$$

is a multiplier parameter introduced in SRP-6a to address weaknesses identified in earlier protocol versions. Both parties then derive a hashing scalar:

$$u = H(A, B) \tag{6}$$

Using these values, the client and server then independently derive a shared premaster secret S. The client computes:

$$S = (B - kg^x)^{\,a+ux} \ (\boldsymbol{mod\ N}) \tag{7}$$

while the server computes:

$$S = (A \cdot v^u)^b \ (\boldsymbol{mod\ N}) \tag{8}$$

If and only if the client knows the correct password, both computations result in the same shared secret. A session key:

$$K = H(S) \tag{9}$$

is subsequently derived and used to generate authentication proofs M_1 and M_2, allowing both parties to verify each other's identity.

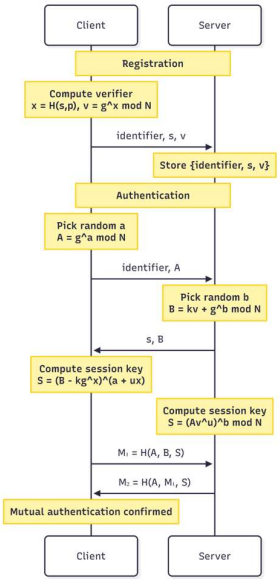

Fig. 1. Sequence Diagram of the SRP-6a Protocol

C. Replay Attacks

A replay attack is a form of network layer attack in which a valid authentication message previously captured by an adversary is retransmitted later with the intent of gaining unauthorized access [2]. Such attacks are particularly effective against systems that rely on static authentication credentials or reusable authentication tokens. If intercepted data remains valid across multiple sessions, an attacker may be able to authenticate successfully without knowing the user's password.

SRP-6a mitigates replay attacks by using fresh ephemeral values in every authentication session. At the beginning of each login attempt, the client generates a new random value a, while the server generates a new random value b. These values produce unique public parameters A and B, which in turn lead to a unique session key K for every authentication exchange.

Since the authentication proofs M_1 and M_2 are derived from session-specific parameters, messages captured from a previous session cannot be reused successfully in a future session. Even if an attacker records a complete and valid authentication exchange, replaying the captured messages will fail because the server generates a different ephemeral value for each new session, resulting in a different expected proof.

Although SRP-6a provides replay protection at the protocol level, transport-layer security mechanisms such as TLS remain important in practice. TLS protects protocol messages from passive observation and modification, providing an additional layer of defense against network-based attacks and complementing the security guarantees offered by SRP-6a.

III. System Design and Implementation

A. System Architecture

The proposed system follows a two-tier client–server architecture. The frontend is implemented as a Node.js simulation module representing the web client, and the backend is a Node.js server module that manages the user database and cryptographic verification logic. In a production deployment, the frontend would correspond to a browser-based React application and the backend to a RESTful API server, communicating over HTTPS protocol.

The implementation is divided into five main modules:

- **utils/utils.js**, provides shared cryptographic utilities and SRP group parameters,
- **client/SRPClient.js,** implements the client-side SRP-6a operations,
- **server/SRPServer.js**, handles server-side SRP-6a validation and mock database,
- **benchmark.js**, measures & evaluates performance,
- **security_tests.js**, implements qualitative attack scenario evaluation

The goal of the proposed system is to prevent plaintext passwords from being transmitted, stored, or disclosed at any stage of the authentication process. Instead of storing user passwords, the server stores only a verifier-based credential record consisting of the tuple *{username, s, v}*, where s is the salt and v is the password verifier. This approach minimizes the impact of credential database breaches, as attackers cannot directly obtain user passwords.

B. Cryptographic Parameter Selection

The implementation uses the 2048-bit SRP group defined in Appendix A of RFC 5054 [4]. This group is based on a 2048-bit safe prime N, where:

$$N = 2q + 1 \tag{10}$$

with both N and q being prime. Using a safe prime ensures that the multiplicative group modulo N contains a large prime-order subgroup, eliminating the risk of small-subgroup attacks in which an adversary could exploit low-order elements to recover ephemeral exponents. The corresponding generator for this group is $g = 2$, as specified in RFC 5054 [4]. Adopting a standardized group rather than generating custom parameters at runtime eliminates the risk of generating custom groups.

SHA-256 is used as the hash function H throughout the protocol. The implementation provides a generic hashing utility that accepts multiple inputs sequentially. Integer values represented as BigInt are encoded as their minimum even-length hexadecimal byte representation before hashing, while string inputs are encoded using UTF-8. Unlike RFC 5054, which recommends padding all integer inputs to the byte length of (N), this implementation uses compact encodings for efficiency. Since the encoding procedure is applied by both the client and server, the resulting protocol remains interoperable within the implementation and preserves the required cryptographic properties.

The private value x is derived according to the convention introduced in RFC 2945 [6] rather than the simplified SRP formulation. Specifically, x is computed as:

$$x = H(s, H(username : password)) \tag{11}$$

where the string “username” is first encoded in UTF-8 and hashed before being combined with the salt s. Combining the username into the derivation process ensures that identical passwords registered under different usernames yield distinct verifiers, which prevents cross-account verifier collisions within the same server.

The multiplier k is computed once at module initialization. Ephemeral secrets a and b are each generated as 256-bit random values using Node.js *crypto.randomBytes(32)* function, providing sufficient entropy level for the 2048-bit group.

C. Implementation Details

The authentication process follows the standard three-phase SRP-6a handshake. All large-integer computations are implemented using JavaScript native BigInt type.

During the first phase (initiation phase), the client begins the protocol by invoking *startAuthentication()* function, which generates a 256-bit ephemeral secret a and computes the public value A. The *handleLoginInitiation(username, A)* function in the server then retrieves the stored *{s,v}*, verifies that $A \neq 0 \pmod N$,

and generates a fresh 256-bit ephemeral secret b. The server then computes its public value B.

During the second phase (client authentication proof), the client calls the *processServerChallenge(s, B)* function, which verifies that $B \neq 0 \pmod N$ and computes x as shown in (11), u as shown in (6), and S as shown in (7).

The exponent $(a + ux)$ involves the sum of 256-bit quantities (a from randomBytes, u and x from SHA-256), resulting in an effective size of approximately 512 bits. This modular exponentiation represents the most computationally intensive operation performed by the client. The client then derives the session key K as shown in the (9) and computes the client proof:

$$M_1 = H(A, B, K) \tag{12}$$

The client then sends $\{A, M_1\}$ to the server.

During the third phase (mutual verification phase), the function *verifyClientProof*(M_1) in the server computes u as shown in (6), S as shown in (8), K as shown in (9), and derives its expected value of M_1. Authentication succeeds only if the received and computed proofs are identical. After successful verification, the server then generates the server proof:

$$M_2 = H(A, M_1, K) \tag{13}$$

and return it to the client. The client then calls *verifyServerProof*(M_2) function that verifies M_2 by computing the expected value locally and comparing it with the received proof. Any mismatch results in immediate session termination and indicates a potential man-in-the-middle or phishing attack. Mutual authentication is completed only when both M_1 *and* M_2 are successfully validated.

IV. Performance Evaluation

A. Experimental Setup

The evaluation consists of two independent experiments. First, *security_tests.js* performs four qualitative security scenarios to validate the correctness of the implementation and its resistance to common attack. Second, *benchmark.js* evaluates computational performance by comparing the latency of SRP-6a authentication against a conventional bcrypt-based authentication model over 1,000 iterations.

All experiments were conducted using Node.js v22 on a system equipped with an Intel Core i5 processor and 8 GB of RAM. The SRP-6a implementation utilizes the 2048-bit RFC 5054 group and is implemented entirely in JavaScript using the native BigInt type, without WebAssembly acceleration or native cryptographic bindings. This configuration is for a realistic deployment scenario for JavaScript-based web applications. For comparison, bcrypt was configured with a cost factor of 10.

Execution times were measured using the high-resolution *performance.now()* timer, with network latency excluded to isolate computational overhead. For SRP-6a, client-side timing includes the generation of the public value A during Phase 1 and the computation of the shared secret S and proof M_1 during Phase 2. Server-side timing includes the generation of (B) during Phase 1 and the verification of M_1 together with the generation of M_2 during Phase 3. In the bcrypt model, the client-side operation is represented by credential preparation only, while the server-side operation consists of bcrypt password verification. For each benchmark, the mean, median, 95th percentile (p95), and 99th percentile (p99) execution times were recorded across 1,000 iterations to capture both average performance and latency variability.

B. Security Scenario Evaluation

Four attack scenarios were implemented in *security_tests.js* to validate the protocol's security properties. The results are summarized in Table 1.

TABLE I. Security Scenario Evaluation Results

Scenario	Attack Type	Description	Outcome
S1	Legitimate Authentication	Correct credentials, full three-phase handshake	ACCEPTED
S2	Replay Attack	Correct credentials, full three-phase handshake	REJECTED
S3	Stolen Verifier	Stolen verifier v submitted directly as the client's password	REJECTED
S4	Rogue Server	Client receives a forged B and a random M_2	REJECTED by Client

S1 (Legitimate Authentication). A client using valid credentials successfully completed all three phases of the SRP-6a handshake. Both client and server proofs (M_1 *and* M_2) were verified successfully, confirming the correctness of the implementation.

S2 (Replay Attack). Previously captured authentication data $\{A, M_1\}$ was replayed against a new server session. Because the server generated a fresh ephemeral secret b, the resulting value B and scrambling parameter u differed from the original session. Consequently, the server derived a different session key and expected proof value, causing verification to fail. The replay attempt was therefore rejected.

S3 (Stolen Verifier Attack). An attacker who obtained the stored verifier v attempted to authenticate by using it directly as a password. Since the client computes x as shown in (11), substituting v produces a different private key and session key from those computed by the legitimate server. As a result, proof verification failed, demonstrating that having the verifier alone is insufficient for authentication.

S4 (Rogue Server Attack). A malicious server generated a forged challenge value B and returned a randomly generated server proof M_2. The client independently computed the expected M_2 from its derived session key and detected the mismatch, causing the authentication process to terminate. This confirms the protocol's mutual authentication property and its ability to detect server impersonation attempts.

C. Computation Time Analysis

TABLE II. PERFORMANCE RESULTS (IN MILLISECONDS, 1000 ITERATIONS)

Scheme	Component	Mean	Median	p95	p99
Bcrypt (10)	Client-side	0.00	0.00	0.01	0.01
Bcrypt (10)	Server-side	50.18	49.86	52.47	54.35
SRP-6a	Client-side	23.12	22.79	25.73	29.29
SRP-6a	Server-side	17.34	17.09	19.44	22.29

The bcrypt client-side latency was negligible because the benchmark modeled the client operation as simple credential preparation without local cryptographic computation. The primary cost was incurred on the server, where bcrypt verification required an average of approximately 50.18 ms due to the intentionally expensive cost factor of 10 hashing process.

For SRP-6a, the average client-side latency was approximately 23.12 ms. This overhead is dominated by the modular exponentiation operations required to compute the public value A and the shared secret S. The server-side latency averaged 17.34 ms, reflecting the corresponding exponentiation operations involved in generating B and verifying the client proof.

The combined computational cost of SRP-6a was approximately 40.46 ms, compared to 50.18 ms for bcrypt-based authentication. While both approaches provide practical authentication performance, SRP-6a reduced overall computation time by approximately 19%. More importantly, its server-side cost was approximately 66% lower than bcrypt, suggesting improved scalability under high authentication loads.

Latency variability was also low. The p99 latency reached 29.29 ms on the client side and 22.29 ms on the server side, remaining close to the corresponding mean values. This indicates predictable execution behavior and stable performance across repeated authentication sessions.

These results demonstrate that SRP-6a not only provides stronger security guarantees, such as resistance to replay attacks, verifier theft, and server impersonation, but also achieves competitive computational performance. In the evaluated environment, SRP-6a imposed no significant latency penalty and substantially reduced server-side processing costs compared with bcrypt-based authentication.

V. CONCLUSION

This paper presented the design, implementation, and evaluation of an SRP-6a-based authentication system for modern web applications. The motivation for this work stems from the limitations of conventional password-based authentication mechanisms, which require users to transmit passwords or password-derived values to a server and therefore remain vulnerable to credential theft, phishing, replay attacks, and database compromise. By adopting an augmented Password-Authenticated Key Agreement (PAKE) protocol, the proposed system aims to provide stronger authentication guarantees while maintaining practical deployment in modern web environments.

The implementation follows the SRP-6a specification and utilizes the standardized 2048-bit group parameters defined in RFC 5054 [4]. In addition, the system incorporates username binding in the derivation of the private value x, ensuring that identical passwords associated with different usernames generate distinct verifiers. By storing only password-derived verifiers rather than plaintext passwords or password-equivalent credentials, the system reduces the risks associated with server-side credential database breaches while preserving compatibility with the SRP-6a protocol.

A qualitative security evaluation was conducted through four representative authentication scenarios. The results demonstrated that legitimate authentication requests were accepted successfully, while replay attacks, stolen-verifier impersonation attempts, and rogue server attacks were consistently rejected. These findings confirm that the implementation satisfies the intended security objectives of SRP-6a, including mutual authentication, resistance to replay attacks, protection against verifier compromise, and detection of server impersonation attempts.

To evaluate practical feasibility, a performance benchmark consisting of 1,000 authentication sessions was performed and compared against a bcrypt-based authentication model. The results showed that SRP-6a achieved an average total computational cost of approximately 40 ms, compared with approximately 50 ms for bcrypt. Although SRP-6a relies on computationally intensive modular exponentiation operations, the measured latency remained within a practical range for interactive web applications.

The benchmark also revealed that SRP-6a significantly reduces server-side computational cost. The average server-side latency was approximately 17 ms, representing a reduction of roughly 66% compared with bcrypt. Because authentication workloads are often concentrated on the server, this reduction can translate directly into improved scalability and a greater capacity to handle concurrent authentication requests. These results challenge the common perception that PAKE-based authentication is prohibitively expensive for real-world deployment.

Overall, the findings indicate that SRP-6a provides a balance between security and performance. The protocol eliminates password transmission, supports mutual authentication, derives a unique session key for each authentication exchange, and demonstrates competitive computational efficiency. Consequently, SRP-6a represents a practical and recommendable authentication solution for web applications that manage sensitive user information and require stronger protection than conventional password-based authentication schemes can provide.

The source code and experimental scripts used in this study are available at:

https://github.com/usernamedarren/SRP-6a-Simulation

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to the lecturers and assistants of the II4021 Cryptography course at Institut Teknologi Bandung for their guidance and valuable lessons throughout the course.

REFERENCES

[1] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *Proc. IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, May 2012, pp. 553–567.

[2] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 8th ed. Hoboken, NJ, USA: Pearson, 2017, pp. 63–65.

[3] T. Wu, "SRP-6: Improvements and Refinements to the Secure Remote Password Protocol," 2002.

[4] D. Taylor, T. Wu, N. Mavrogiannopoulos, and T. Perrin, "Using the Secure Remote Password (SRP) protocol for TLS authentication," IETF, Fremont, CA, USA, RFC 5054, Nov. 2007. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5054

[5] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *Proc. Advances in Cryptology (CRYPTO)*, Santa Barbara, CA, USA, Aug. 1993, pp. 232–249.

[6] T. Wu, "The SRP authentication and key exchange system," IETF, RFC 2945, Sep. 2000. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2945

PERNYATAAN

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, 18 Juni 2026

Darren Mansyl
18223001