

CAYENNE+: HMAC-Derived Peppers for Compartmentalized Password Hashing

Nathaniel Kofi Adjadeh

s236y001@s.iwate-pu.ac.jp

Iwate Prefectural University

Okky Dicky Ardiansyah Prima

Iwate Prefectural University

Masaki Narita

Iwate Prefectural University

Research Article

Keywords:

Posted Date: October 29th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11227406/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

CAYENNE+: HMAC-Derived Peppers for Compartmentalized Password Hashing

Nathaniel Kofi Adjadeh · Oky Dicky Ardiansyah Prima · Masaki Narita

Abstract Passwords remain widely used because they are simple, accessible, inexpensive to deploy, and compatible with existing authentication systems. To strengthen stored password hashes, many systems add a verifier-side secret known as a pepper. However, conventional peppering commonly uses one static pepper across many accounts, creating a broad compromise domain if that secret is exposed. This study proposes CAYENNE+, a deterministic verifier-side framework that derives an account-specific pepper from two independently protected components: a master secret and a per-user derivation state. CAYENNE+ operates as an additional layer above established password-hashing algorithms and uses HMAC-SHA-256 for deterministic pepper derivation. The framework was evaluated through bcrypt-based attack simulation, executable SHA-256 workflow validation, and authentication-performance experiments under no-pepper, known-static-pepper, full-compromise, and partial-compromise conditions. Under the evaluated partial-compromise condition, where the master secret is exposed but the per-user derivation state remains unavailable, the missing state increases the estimated work required for offline password recovery. Non-rotating CAYENNE+ introduces little routine authentication overhead, while state rotation periodically renews verifier material at the cost of higher tail latency. With $k_{\max} = 100,000$, direct state enumeration gives a naive 100,000-state search bound, whereas the evaluated four-component modulo-10 mapping produces at most 10,000 distinguishable mapped outputs. Overall, CAYENNE+ demonstrates how account-specific pepper derivation and independently protected verifier components can

reduce the broad compromise domain associated with globally shared static peppers.

Keywords Authentication · Credential compartmentalization · HMAC · Password hashing · Password peppering · Password security

1 Introduction

Passwords remain deeply embedded in modern authentication systems despite the growing availability of passwordless alternatives such as biometrics and passkeys. FIDO-based authentication has expanded support for phishing-resistant sign-in, but passwords remain widely used because they are familiar, accessible, inexpensive to deploy, and compatible with existing web, enterprise, cloud, mobile, and embedded systems [1, 2, 3]. At the same time, the continued expansion of online services has increased the number of credentials generated, transmitted, and stored across distributed systems. Global internet usage reached an estimated 5.56 billion users in 2025, with annual growth of approximately 2.5% [4]. This expansion has made password databases attractive targets for attackers, while password-related compromises continue to contribute to recorded data breaches and growing cyberattack-related financial losses [5, 6, 7].

Modern password-storage mechanisms are therefore designed to limit the damage caused by database compromise. Password-hashing schemes such as bcrypt and Argon2id increase attacker workload through deliberately expensive or memory-hard computation, while unique salts prevent identical passwords from producing identical stored hashes and reduce the effectiveness of precomputed attacks [8]. Peppering, which is the focus of this paper, adds a verifier-side secret that is stored separately from the password database, often

Graduate School of Software and Information Science, Iwate Prefectural University, Takizawa, Iwate, Japan
E-mail: s236y001@s.iwate-pu.ac.jp
Corresponding author: Nathaniel Kofi Adjadeh

within protected environments such as Hardware Security Modules (HSMs) or Trusted Platform Modules (TPMs) [9]. Unlike salts, peppers are intended to remain unavailable to an attacker who obtains only the password database, thereby increasing the difficulty of offline password cracking.

However, conventional peppering commonly relies on one static secret shared across many password records [10]. Although this approach is operationally simple and provides a uniform additional layer of protection, it offers limited compartmentalization. If the shared pepper is exposed, the additional protection it provides may be lost for every account that depends on it. The security of the protected password database therefore remains heavily dependent on the continued secrecy of one globally reused value.

To address the limitation above, this paper introduces CAYENNE+, a deterministic verifier-side pepper-derivation framework that generates an account-specific pepper for each user. Rather than storing these peppers directly, CAYENNE+ reconstructs them when required from two independently protected components: a hardware-protected master secret and a separately protected per-user derivation state. These components are assumed to be maintained in separate HSM-backed security domains so that compromise of one does not automatically expose the other. Direct reconstruction of an account's pepper therefore requires access to both verifier components, while compromise of only one component does not immediately reveal the information required to reconstruct all account-specific peppers.

CAYENNE+ does not replace bcrypt, Argon2id, or other established password-hashing schemes. Instead, it operates as an additional verifier-side protection layer that can be integrated with existing password-hashing workflows. The framework uses standard Hash-based Message Authentication Code (HMAC) derivation and deterministic transformation logic to generate user- and state-dependent peppers. In the non-rotating configuration, the same per-user derivation state is retained across successive authentications. In the rotating configuration, the per-user state is replaced after a predefined number of successful logins, causing the account-specific pepper to be re-derived and the password verifier to be regenerated. The framework therefore aims to reduce the broad compromise domain associated with one globally shared static pepper, increase attacker workload when only part of the verifier-side state is exposed, and support periodic renewal of account-specific verifier material.

This study makes four main contributions. First, it defines an HMAC-based architecture for reconstructing account-specific peppers without storing the final pepper

values directly. Second, it introduces an explicit threat model covering four conditions: no pepper; exposure of a conventional static pepper; full CAYENNE+ compromise, in which both protected derivation components are available to the attacker; and partial CAYENNE+ compromise, in which one component remains protected. Third, it evaluates offline password-recovery cost through bcrypt-based simulation and executable SHA-256 workflow validation, while separately measuring registration, login, throughput, and state-rotation overhead across account scales of 10,000, 50,000, and 200,000 users. Fourth, it analyses how the configured per-user state range and mapped-pepper cardinality affect collision-aware attack cost.

The results show that CAYENNE+ provides its principal benefit under partial compromise, where one independently protected verifier component remains unavailable to the attacker. Under the evaluated condition in which the master secret is exposed but the per-user derivation state remains protected, the missing state increases the estimated work required for offline password recovery. In the non-rotating configuration, CAYENNE+ introduces little routine authentication overhead because the underlying password-hashing operation remains the dominant computational cost. In the rotating configuration, renewal of the per-user state and password verifier increases tail latency because credential rehashing is required. These findings show how account-specific pepper derivation and verifier-state compartmentalization can be integrated with established password-hashing workflows while reducing the broad compromise domain associated with a single shared static pepper.

2 Related Work

Password storage has evolved from simple one-way hashing toward adaptive and memory-hard schemes designed to resist offline guessing attacks. The UNIX password scheme introduced by Morris and Thompson established foundational principles of password hashing and salting that continue to influence modern authentication systems [11]. General-purpose cryptographic hash functions such as MD5, SHA-1, and SHA-256 have also played important roles in integrity and authentication systems. However, their high computational efficiency makes them unsuitable for password storage because it enables large-scale offline guessing. Several older constructions have also been weakened by known cryptanalytic attacks [12, 13, 14, 15, 16].

As attacker hardware improved, password-storage research shifted toward deliberately expensive password-hashing algorithms. Bcrypt combines automatic salting with an adjustable cost factor, allowing verifica-

tion cost to be increased as computational capabilities improve [17]. Argon2 extends this approach through memory-hard computation designed to reduce the advantage of GPU- and ASIC-assisted attacks [18]. RFC 9106 identifies Argon2id as the primary Argon2 variant and provides parameter-selection guidance for both general-purpose and memory-constrained environments [19]. More recent work on cost-asymmetric memory-hard password hashing has further examined how incorrect offline guesses can be made more expensive while maintaining practical verification costs for legitimate users [20]. These approaches reinforce the role of the underlying password-hashing function as the primary computational defence following database compromise.

Salts and peppers provide complementary protection within password-storage workflows. A salt is a unique, non-secret value stored alongside the password verifier. It prevents identical passwords from producing identical stored hashes and reduces the effectiveness of pre-computed attacks. A pepper, by contrast, is a secret verifier-side value intended to remain unavailable to an attacker who obtains only the password database. Manber [21] formalized peppering as a method for increasing password-cracking difficulty, while contemporary guidance recommends protecting verifier-side secrets separately from the password database, including within environments such as Hardware Security Modules (HSMs) or Trusted Platform Modules (TPMs) [9].

In many conventional peppering designs, one static pepper is shared across a large set of accounts. This approach is straightforward to manage and provides a uniform additional layer of protection, but it creates a broad compromise domain. Exposure of the shared pepper removes its added protection from every account that depends on it. Replacing a shared pepper may also require access to users' passwords during successful authentication or the use of a forced password-reset process. These limitations have motivated research into dynamic, contextual, and periodically refreshed verifier-side values.

Hash-based Message Authentication Code (HMAC) provides a suitable basis for deterministic secret derivation. HMAC is a standardized keyed-hashing construction that combines a secret key with an underlying hash function to provide message authentication and integrity protection [22]. Chen [23] offers a description of pseudorandom-function-based key-derivation mechanisms, including constructions based on HMAC, for deriving additional keying material from an existing secret key. This supports the use of standard HMAC as the deterministic derivation primitive in CAYENNE+, while input encoding, domain separation, versioning, and key-management procedures remain implementa-

tion considerations. Najjar's d-HMAC studies investigated dynamic HMAC-based mechanisms for password protection. The original d-HMAC proposal replaced the fixed inner and outer padding values used in conventional HMAC with dynamically generated values, while a later application considered dynamic padding and user-dependent variation in password-storage workflows [24, 25]. CAYENNE+ differs from d-HMAC because it does not modify the internal HMAC construction. Instead, it uses standard HMAC-SHA-256 as a deterministic derivation primitive and remains compatible with established password-hashing algorithms.

Research has also explored periodic renewal of password verifiers. Obiri-Tetteh et al. [26] proposed periodically refreshing bcrypt verifiers through an adaptive timer. Such approaches highlight the potential value of credential renewal, but password-dependent rehashing must define when renewal occurs and how the verifier is updated without retaining the plaintext password. In practice, verifier renewal can be performed following successful authentication, when the submitted password is temporarily available to the verifier.

CAYENNE+ also extends earlier work on rotating pepper mechanisms by moving beyond a single time-dependent pepper-generation process [27]. The earlier approach demonstrated the feasibility of changing verifier-side peppers over time, whereas CAYENNE+ introduces account-specific derivation based on a protected master secret and an independently protected per-user state. These components are maintained in separate HSM-backed security domains so that compromise of one does not automatically expose the other.

CAYENNE+ therefore does not eliminate the need for a global root secret. Instead, it changes the compromise condition by requiring an attacker to obtain both the master secret and the relevant per-user state for direct reconstruction of an account-specific pepper. Conventional static peppering depends on the secrecy of one globally reused value, whereas CAYENNE+ distributes the information required for pepper reconstruction across independently protected components.

The principal distinction of CAYENNE+ is its emphasis on account-specific pepper derivation and verifier-state compartmentalization. Its security benefit depends on preserving separation between the protected domains and on the output diversity provided by the selected derivation profile. The present study evaluates this design under explicit full- and partial-compromise conditions and examines both offline password-recovery cost and authentication-performance overhead.

3 Methodology

CAYENNE+ was implemented in Python as a verifier-side pepper-derivation framework and compared with two conventional password-storage configurations: password hashing with one known static global pepper and password hashing without a pepper. All bcrypt-based experiments used a cost factor of 12 so that the computational cost of the underlying password-hashing algorithm remained consistent across the evaluated configurations. The Hardware Security Modules (HSMs) and their protected storage environments were simulated in software. The implementation therefore evaluates the logical separation of protected verifier components but does not reproduce the physical tamper resistance or operational guarantees of production HSM hardware.

The methodology evaluates three aspects of the framework. First, offline password-recovery experiments examine how different levels of attacker knowledge affect cracking cost. Second, authentication-performance experiments measure registration, login, throughput, and state-rotation overhead across multiple account scales. Third, an analytical evaluation examines how the configured per-user state range and the mapped-output space affect the resulting attack-space bounds.

3.1 Experimental Configurations

Three password-storage configurations were evaluated:

1. **CAYENNE+ derived peppering:** Each password was preprocessed using HMAC-SHA-256 with an account-specific pepper reconstructed through the CAYENNE+ derivation process. The resulting fixed-length digest was then hashed using bcrypt.
2. **Static global peppering:** Each password was preprocessed using HMAC-SHA-256 with one static secret shared across all accounts. The resulting fixed-length digest was then hashed using bcrypt.
3. **No pepper:** Each password was passed directly to bcrypt without additional verifier-side secret preprocessing.

The static-pepper and CAYENNE+ configurations used the same HMAC-based password-preprocessing structure. Their principal difference was how the pepper was obtained. The static configuration reused one globally shared secret, whereas CAYENNE+ reconstructed an account-specific pepper from independently protected verifier-side components. The no-pepper configuration served as the baseline for evaluating the security and performance effects of verifier-side peppering.

The configurations were evaluated under attacker-knowledge conditions representing no pepper, exposure

of a conventional static pepper, full CAYENNE+ compromise, and partial CAYENNE+ compromise. The assumptions associated with each scenario are defined in the threat-model subsection.

3.2 Password Dataset Generation

A synthetic password corpus was generated from a controlled wordlist, `pop.txt`, containing 100 common English words. Each password was constructed by selecting three distinct words from the list and concatenating them in the sampled order. Generated passwords were required to be unique within each experimental dataset, so duplicate outputs were discarded and sampling continued until the required number of passwords had been obtained. A fixed seed of 42 was used for the password-sampling random number generator.

Because each password contains three distinct words and word order is significant, the complete candidate space contains

$$100 \times 99 \times 98 = 970,200$$

possible three-word passwords. This bounded space was intentionally used to provide a controlled and repeatable environment for comparing the evaluated protection schemes and for defining a complete candidate order during the password-recovery experiments.

The synthetic corpus was not intended to reproduce the full distribution of real-world password choices. User-generated passwords may contain names, dates, symbols, substitutions, multilingual terms, personal information, and password-reuse patterns that are not represented by the controlled wordlist. The dataset should therefore be interpreted as an experimental corpus for comparing the evaluated configurations rather than as a representative model of real-world password selection.

User accounts were assigned sequential identifiers such as `user0001`, `user0002`, and `user0003`. Depending on the experiment, the implementation generated the required number of accounts from the same controlled password-construction process. The resulting experimental data included:

- the account-password dataset;
- verifier databases for the CAYENNE+, static-pepper, and no-pepper configurations;
- protected verifier-state records for CAYENNE+; and
- ground-truth credential mappings used to evaluate password recovery.

3.3 CAYENNE+ Architecture and Protected Components

CAYENNE+ does not store or reuse one final pepper across all accounts. Instead, it reconstructs an account-specific verifier-side pepper from two independently protected components:

- a master secret, K_{secret} ; and
- a per-user derivation state, k .

The two components are assumed to be maintained in separate HSM-backed security domains. One protected domain stores K_{secret} , while the other stores each user's k value together with the configuration and version metadata required for pepper reconstruction. The purpose of this separation is to ensure that compromise of one protected domain does not automatically reveal all information required to reconstruct an account-specific pepper.

The master secret remains a global root of trust shared across the derivation process. CAYENNE+ therefore does not eliminate the use of a global secret. Instead, it changes the compromise condition by requiring both K_{secret} and the relevant user-specific state k for direct reconstruction of an account's pepper. If only one component is exposed, the attacker lacks the complete input required to reproduce the account-specific pepper.

For each registered user, the derivation state k was sampled from the bounded integer range

$$0 \leq k < k_{\text{max}}.$$

In the evaluated implementation, $k_{\text{max}} = 100,000$. The value of k is account-specific and is not stored in the password-verifier database. Different accounts may therefore share the same master secret while retaining independently assigned derivation states.

The verifier-side derivation process combines the user identifier U , the user-specific state k , and the master secret K_{secret} . A deterministic HMAC-SHA-256 digest D is computed as

$$D = \text{HMAC}_{\text{SHA-256}}(K_{\text{secret}}, U \parallel k). \quad (1)$$

In the experimental implementation, U and k were serialized by concatenating their textual representations before HMAC evaluation. A deployment implementation should instead use an unambiguous encoding, such as fixed-width, length-prefixed, or explicitly delimited fields.

Because the derivation is deterministic, the same user identifier, per-user state, and master secret reproduce the same digest during registration and subsequent

authentication. Changing the per-user state generally produces a different HMAC digest and may consequently produce a different reconstructed pepper, although distinct states can converge to the same pepper under the bounded experimental mapping.

The HSM-backed environments used in the evaluation were simulated in software. The experiments therefore examine the logical effect of independently protected verifier components rather than the physical tamper resistance, administrative controls, latency, or operational guarantees of production HSM hardware.

3.4 Pepper Derivation and Experimental Mapping

The HMAC-SHA-256 digest D , produced from the user identifier U , the per-user state k , and the master secret K_{secret} , is transformed into a readable account-specific pepper in the evaluated implementation. The word-based representation was introduced to make changes in the reconstructed pepper easy to observe during registration, authentication, and state-rotation testing. It should therefore be understood as an experimental representation of the derivation output, rather than as a requirement of the CAYENNE+ architecture.

The first four bytes of D are mapped into four bounded values using modulo-10 reduction:

$$S_i = D_i \bmod 10, \quad i \in \{1, 2, 3, 4\}. \quad (2)$$

Each S_i therefore takes a value from 0 to 9. Because 256 is not divisible by 10, direct modulo reduction introduces a small bias in the distribution of the mapped values. The uniform-mapping assumption used later for analytical estimation should therefore be interpreted as an approximation. A deployment-oriented mapping could avoid this bias through rejection sampling or another suitably designed derivation procedure.

In the evaluated implementation, each mapped value S_i selects one entry from a predefined pepper word pool, represented by

$$\mathcal{W} = (w_1, w_2, \dots, w_L).$$

For each mapped value S_i , the selected word is

$$W_i = w_{L-S_i}, \quad i \in \{1, 2, 3, 4\}. \quad (3)$$

The four mapped values are also summed to produce an additional numeric component,

$$T = \sum_{i=1}^4 S_i. \quad (4)$$

Let the configured special-character pool be

$$\mathcal{C} = (c_0, c_1, \dots, c_{m-1}).$$

The special-character component is selected deterministically as

$$S_c = c_{T \bmod m}. \quad (5)$$

The final experimental pepper P is then constructed as

$$P = W_1 \parallel W_2 \parallel T \parallel W_3 \parallel W_4 \parallel S_c. \quad (6)$$

where W_1 to W_4 are the words selected by the mapped values, T is the sum of S_1 through S_4 , and S_c is the deterministically selected special character.

For example, consider an illustrative digest whose first four bytes produce the mapped sequence

$$(S_1, S_2, S_3, S_4) = (0, 1, 2, 3).$$

Under the word and character pools used in the evaluated implementation, these values select Photosynthesis, Overexaggeration, Misinterpretation, and Irresponsibility, respectively. Since $T = 6$, the corresponding special character is $\&$, producing the readable pepper

PhotosynthesisOverexaggeration6MisinterpretationIrresponsibility&

A second mapped sequence,

$$(S_1, S_2, S_3, S_4) = (5, 8, 1, 4),$$

produces $T = 18$ and selects Circumnavigation, Counterproductive, Overexaggeration, and Discombobulated. The corresponding readable pepper is

CircumnavigationCounterproductive18OverexaggerationDiscombobulated!

These examples illustrate the purpose of the readable representation: changes in the HMAC-derived state produce visibly different verifier-side pepper values without requiring inspection of raw binary output. The examples do not represent manually selected peppers; the words, numeric component, and special character are deterministic consequences of the mapped HMAC output.

CAYENNE+ does not store the reconstructed pepper in the password-verifier database. Instead, P is reconstructed when required during registration, authentication, and, where enabled, state rotation. Given the same user identifier, master secret, per-user state, derivation configuration, and version information, the same pepper is reproduced deterministically.

The natural-language representation and the four-component modulo-10 mapping are properties of the evaluated experimental profile and are not requirements of the CAYENNE+ architecture. Because each of the four mapped components can take one of ten values, this profile produces at most

$$10^4 = 10,000$$

distinct mapped selection vectors before the deterministic numeric and special-character components are added. Those additional components are functions of the same four mapped values and therefore do not increase the independent mapped-output space.

A deployment-oriented implementation should preserve substantially more of the HMAC-derived output space. This could be achieved through a higher-cardinality mapping or through direct derivation of fixed-length cryptographic pepper material using an appropriately domain-separated construction. Merely replacing the readable words with cryptographic tokens while retaining the same four-component modulo-10 mapping would not increase the 10,000-element mapped-output bound. Stronger deployment-oriented profiles were not experimentally evaluated in the present study and remain future work.

3.5 Password Registration, Verification, and State Rotation

After reconstructing the current account-specific pepper P , CAYENNE+ uses it as the HMAC key for preprocessing the user's password:

$$H_f = \text{HMAC}_{\text{SHA-256}}(P, \text{Password}). \quad (7)$$

In the evaluated implementation, H_f is the raw 32-byte HMAC-SHA-256 digest returned by the cryptographic library. This binary digest is passed directly to bcrypt without hexadecimal, Base64, or other textual encoding. Bcrypt then generates an independent random salt s_b and applies cost factor $c = 12$:

$$V = \text{bcrypt}(H_f, s_b, c), \quad c = 12. \quad (8)$$

The resulting verifier V contains the bcrypt parameters and salt required for subsequent verification and is stored in the authentication database. The reconstructed pepper P and the intermediate digest H_f are not stored with the verifier.

This layered process combines deterministic account-specific pepper derivation, keyed password preprocessing,

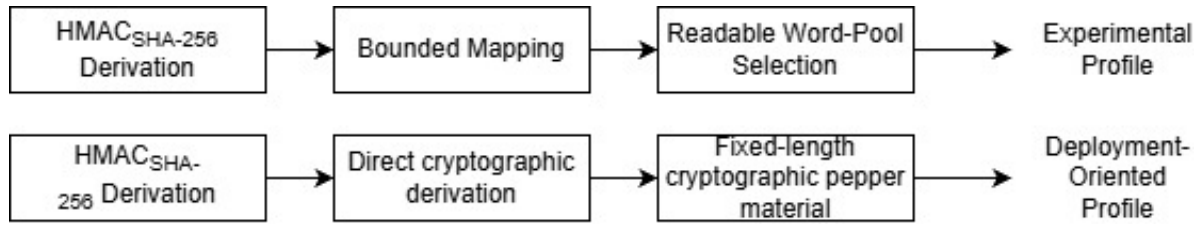


Fig. 1 Experimental and deployment-oriented CAYENNE+ derivation profiles. The evaluated experimental profile uses a four-component modulo-10 mapping and a readable word-pool representation for observability, whereas a deployment-oriented profile should preserve substantially more of the HMAC-derived output space through either a higher-cardinality mapping or direct derivation of fixed-length cryptographic pepper material.

and computationally expensive password hashing. The pepper required for a particular account is reconstructed when needed from the protected master secret, user identifier, current per-user derivation state, and associated derivation configuration.

During registration, the system receives the user identifier and password and assigns the account a derivation state k . The corresponding pepper P is reconstructed and used as the HMAC-SHA-256 key for preprocessing the submitted password. The resulting raw 32-byte digest H_f is then passed directly to bcrypt, which generates an independent salt and produces the stored verifier V . The verifier and its public bcrypt metadata are stored in the authentication database, while the protected CAYENNE+ derivation components remain in their respective HSM-backed security domains.

During authentication, the system retrieves the user’s current protected derivation state and reconstructs the corresponding account-specific pepper. The submitted password is processed using the same HMAC-SHA-256 procedure, and the resulting raw 32-byte digest is checked against the stored bcrypt verifier. Authentication succeeds when the bcrypt verification is valid.

CAYENNE+ was evaluated in both non-rotating and rotating configurations. In the non-rotating configuration, the same per-user derivation state is retained across successful authentications, allowing the same account-specific pepper to be reconstructed whenever required.

In the evaluated rotating configuration, an account is marked for state renewal after five successful authentications using its current state. The next successful authentication is first completed using the existing k , pepper, and verifier. After that verification succeeds, a fresh k value is selected, the protected per-user state is updated, a new account-specific pepper is reconstructed, and a new bcrypt verifier is generated from the password supplied during that same authentication. Under the 24-login experimental workload, this procedure results in verifier renewal on logins 6, 12, 18, and 24.

Password-dependent verifier regeneration therefore occurs only following successful authentication, when

the submitted plaintext password is already transiently available to the verifier. The plaintext password is used only for the current authentication and regeneration procedure and is not retained for future state rotation.

The static-pepper configuration applies HMAC-SHA-256 preprocessing followed by bcrypt. It uses one globally shared secret in place of the account-specific pepper P . The resulting raw 32-byte HMAC digest is likewise passed directly to bcrypt. In the no-pepper configuration, the password is passed directly to bcrypt without HMAC-based pepper preprocessing.

Fig. 2 summarizes the registration and authentication workflow. The system receives the user identifier and password, retrieves the protected derivation components, reconstructs the account-specific pepper, preprocesses the password using HMAC-SHA-256, and generates or verifies the bcrypt verifier. In the rotating configuration, a successful authentication may additionally trigger state replacement and verifier regeneration.

3.6 Threat Model and Attacker Scenarios

The security evaluation assumes an offline database-compromise scenario in which the attacker obtains the password-verifier database and any authentication metadata stored alongside it. The attacker is also assumed to know the design of CAYENNE+, including the derivation procedure, mapping logic, and public configuration parameters. Security therefore does not depend on secrecy of the algorithm.

Four attacker scenarios were evaluated:

1. **N1 – No Pepper:** The attacker obtains bcrypt verifiers generated without any verifier-side pepper. Password candidates can therefore be tested directly against the stored bcrypt verifiers.
2. **S1 – Known Static Global Pepper:** The attacker obtains both the password-verifier database and the shared static pepper. Because the pepper is known, each password candidate can be preprocessed using the same HMAC-based procedure as the legitimate

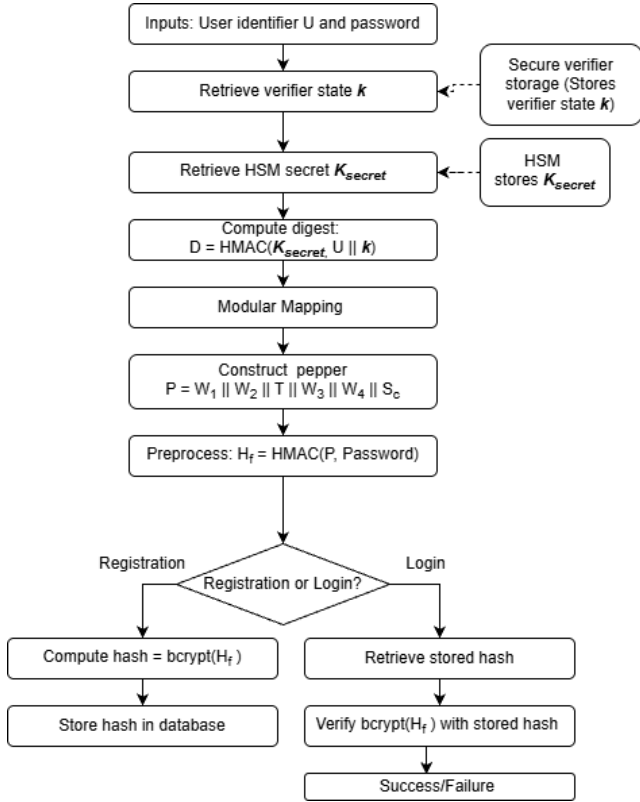


Fig. 2 CAYENNE+ password registration and authentication workflow.

verifier and tested directly against the stored bcrypt verifier.

3. **R1 – Full CAYENNE+ Compromise:** The attacker obtains the password-verifier database, the master secret K_{secret} , and the per-user derivation state k associated with each account. The attacker can therefore reconstruct the correct account-specific pepper and test password candidates directly.
4. **R2 – Partial CAYENNE+ Compromise:** The attacker obtains the password-verifier database and the master secret K_{secret} , but the independently protected per-user derivation states remain unavailable. The attacker must therefore combine password guessing with additional uncertainty over the missing account-specific state k . Under direct state enumeration, candidate values are drawn from

$$0 \leq k < k_{\text{max}}.$$

The S1 scenario intentionally models compromise of the conventional global pepper. A database-only compromise in which the static pepper remains secret is not represented as an executable offline-cracking condition because the attacker lacks the secret required to reproduce the password-preprocessing function. The comparison therefore does not imply that CAYENNE+

provides greater offline resistance than an uncompromised conventional pepper. Instead, it compares the consequences of verifier-side secret exposure under the evaluated compromise conditions.

N1, S1, and R1 all permit direct testing of password candidates because the attacker possesses all information required to reproduce the corresponding verification procedure. Their attack costs are therefore determined primarily by the underlying password-hashing operation and the additional preprocessing required by each configuration.

R2 isolates the effect of verifier-state compartmentalization. In the evaluated R2 condition, the master-secret domain is compromised while the separate per-user state domain remains protected. The attacker can compute the CAYENNE+ derivation function because K_{secret} is available, but the correct account-specific state k is unknown. Password guessing must therefore be combined with state or mapped-pepper enumeration before the correct account-specific preprocessing value can be reproduced.

The converse partial-compromise condition, in which the per-user k values are exposed while K_{secret} remains protected, was not experimentally evaluated. Under the stated offline model, possession of k values alone does not provide the HMAC key required to derive candidate account-specific peppers. This condition is part of the broader architectural threat model but lies outside the experimental comparison reported in this study.

The threat model does not assume that CAYENNE+ resists simultaneous compromise of all verifier-side components. If both protected domains are exposed, as in R1, the framework falls back to the protection provided by the underlying password-hashing algorithm. The intended benefit of CAYENNE+ is therefore compartmentalization: direct reconstruction of an account-specific pepper requires access to both protected derivation components.

3.7 Analytical Bounds on State and Pepper Uncertainty

The partial-compromise scenario introduces uncertainty because the attacker does not know the per-user derivation state k . If k is sampled from the bounded range

$$0 \leq k < k_{\text{max}},$$

then the maximum uncertainty associated with the state variable is

$$H(k) = \log_2(k_{\text{max}}). \quad (9)$$

For the evaluated configuration, $k_{\max} = 100,000$, giving

$$H(k) = \log_2(100,000) \approx 16.61 \text{ bits.}$$

The value $k_{\max} = 100,000$ is an experimental parameter that provides a finite state-enumeration range for the present evaluation. It should not be interpreted as a recommended deployment value or as cryptographic key strength comparable to a 128-bit or 256-bit secret. A deployment-oriented implementation should use a substantially larger randomly selected per-user state according to the required security margin and the capabilities of the protected state-management environment.

The configured state space and the mapped-output space are distinct properties of the evaluated profile. The state range contains 100,000 possible values of k . However, the experimental four-component modulo-10 mapping transforms the HMAC-derived state into a vector

$$(S_1, S_2, S_3, S_4),$$

where each component has ten possible values. The maximum number of distinct mapped vectors is therefore

$$M = b^n = 10^4 = 10,000, \quad (10)$$

where $b = 10$ is the number of possible values per mapped component and $n = 4$ is the number of mapped components.

The numeric component T and special-character component S_c are deterministic functions of the same four mapped values. They therefore do not add independent mapped-output diversity. Assuming unique word-pool entries and unambiguous reconstruction, the experimental mapping is bounded by

$$H(P) \leq \log_2(M) \approx 13.29 \text{ bits.} \quad (11)$$

Consequently, different values of k may reconstruct the same mapped pepper. The number of distinct outcomes therefore satisfies

$$N_{\text{distinct}} \leq \min(k_{\max}, M). \quad (12)$$

Under an idealized uniform random-mapping assumption, the expected number of distinct mapped outcomes after evaluating all $k_{\max}$ state values is

$$E[N_{\text{distinct}}] = M \left[1 - \left(1 - \frac{1}{M} \right)^{k_{\max}} \right]. \quad (13)$$

For $k_{\max} = 100,000$ and $M = 10,000$,

$$E[N_{\text{distinct}}] \approx 9,999.55.$$

Thus, in the idealized mapping model, the 100,000 possible state values correspond on average to approximately ten state values per distinct mapped output. This does not imply that each mapped output has exactly ten preimages, and the modulo-10 reduction described earlier introduces a small departure from uniformity.

The R2 attack cost is therefore reported using two analytical interpretations:

- **R2-naive**: direct state enumeration in which up to $k_{\max} = 100,000$ candidate state values are considered for each password candidate without deduplicating states that reconstruct the same mapped pepper.
- **R2-map**: mapping-aware enumeration in which duplicate mapped outputs are removed before the expensive password-verification stage. For the evaluated profile, the number of distinct mapped candidates is bounded above by $M = 10,000$.

These two interpretations are descriptions of different attacker procedures rather than different definitions of the CAYENNE+ state space. R2-naive reflects direct traversal of the configured k range, whereas R2-map accounts for collisions introduced by the bounded experimental mapping. Because bcrypt verification dominates the cost of the evaluated attack, deduplicating repeated mapped peppers can substantially reduce the number of expensive downstream bcrypt checks.

The R2-map value of 10,000 should be interpreted as an upper bound on the number of distinct mapped outputs in the evaluated profile, not as a guaranteed number of bcrypt checks required for every account or password candidate. An attacker may terminate earlier when the correct candidate is encountered, and candidate ordering, mapped-output frequencies, precomputation, and reuse of derived values may further affect practical work.

Neither R2 interpretation constitutes a formal lower bound on real-world attack cost. The estimates assume the candidate-testing procedures defined for the experiments and do not model every possible attacker optimization, distributed strategy, side channel, or access to a live derivation oracle. They are used to provide transparent analytical bounds for interpreting the experimental results.

The 10,000-output bound is a consequence of the evaluated four-component modulo-10 mapping rather

than the use of natural-language words. Replacing the readable word entries with cryptographic tokens while retaining the same mapping would leave $M = 10,000$ unchanged. A larger mapped-output space requires a higher-cardinality mapping, a greater number of independently mapped components, or more direct use of the HMAC-derived output. The R2-map analysis should therefore be understood as specific to the experimental mapping profile rather than as an inherent limit of the CAYENNE+ architecture.

3.8 Cracking and Authentication-Performance Evaluation

The evaluation was conducted in two stages to examine both password-recovery cost and authentication overhead. The offline password-recovery experiments and the authentication-performance experiments were evaluated separately because they measure different properties of the framework.

First, a bcrypt-based simulation was used to estimate password-recovery cost under slow-hashing conditions. Exhaustive bcrypt execution across the complete account and candidate space would require impractical runtime. The simulation therefore combined an observed bcrypt verification rate with the known password-candidate ordering and the recovery position of each account.

Passwords were generated from ordered permutations of three distinct words from the 100-word experimental dictionary, producing

$$100 \times 99 \times 98 = 970,200$$

candidate passwords. Candidates were considered in a fixed order, and each candidate was evaluated across the account population.

For the N1, S1, and R1 scenarios, the attacker possesses all verifier-side information required to test a password candidate directly. Let r denote the one-based candidate rank of the target password, p the zero-based position of the target account in the stored user order, and N the total number of accounts. The simulated number of bcrypt verification checks required before recovery of that account is

$$C_{\text{known}} = (r - 1)N + (p + 1). \quad (14)$$

This expression follows the candidate-major attack schedule used in the simulation: all accounts are checked for each preceding password candidate before the target candidate is reached.

For the direct R2 state-enumeration interpretation, the attacker knows the master secret but not the target account's per-user state k . Each candidate-password and account pair may therefore require consideration of up to k_{max} state values. Under the simulated enumeration order, the estimated number of bcrypt checks for a target account is

$$C_{\text{R2-naive}} = (r - 1)Nk_{\text{max}} + pk_{\text{max}} + (k + 1). \quad (15)$$

The first term represents complete state enumeration for all users under all preceding password candidates. The second represents complete state enumeration for users preceding the target account under the matching password candidate. The final term represents testing state values from zero through the target account's correct k .

Estimated recovery time for each account was then obtained by dividing the corresponding check count by the observed bcrypt verification rate,

$$t_i = \frac{C_i}{R_{\text{bcrypt}}}, \quad (16)$$

where R_{bcrypt} is the measured number of bcrypt verification checks per second on the experimental platform. The resulting account-recovery times were ordered to obtain T_{50} and T_{90} , defined as the times by which 50% and 90% of the evaluated accounts were recovered, respectively. The reported average estimated check count is the arithmetic mean of the per-account check counts.

The R2-map results were not produced by exhaustively executing a separate bcrypt attack. Instead, they were derived analytically from the mapping-aware bound introduced in the preceding subsection. Because the experimental four-component modulo-10 mapping produces at most 10,000 distinct mapped outputs, duplicate mapped peppers may be removed before the expensive bcrypt-verification stage. The resulting R2-map values should therefore be interpreted as mapping-aware extrapolations rather than directly executed attack times.

Second, executable SHA-256 cracking experiments were used to validate the candidate-generation, preprocessing, and recovery workflow under a fast-hashing condition. SHA-256 was not treated as a secure password-storage algorithm and was not evaluated as a recommended alternative to bcrypt. Its purpose was to make complete traversal of the controlled candidate space feasible and to confirm that the cracking framework reproduced the expected behavior of the no-pepper, known-static-pepper, and full-compromise scenarios.

The R2 condition was not executed exhaustively with SHA-256 because each password candidate additionally requires consideration of the unknown per-user

state. R2-naive and R2-map values were therefore extrapolated from the corresponding known-state baseline using the state-space and mapped-output bounds defined previously.

To distinguish experimental evidence from analytical estimates, results were classified as follows:

- **Measured:** obtained directly from executable cracking or authentication experiments;
- **Simulated:** calculated using the observed bcrypt verification rate together with the implemented candidate ordering and account-specific recovery position; and
- **Extrapolated:** derived from measured or simulated baselines using the stated R2 state-space or mapping-aware bounds.

Authentication performance was evaluated independently across account populations of 10,000, 50,000, and 200,000 users. Four configurations were compared: no pepper, static global pepper, non-rotating CAYENNE+, and rotating CAYENNE+. Each account performed one authentication per simulated period over 24 periods.

The reported authentication metrics included mean login latency, median login latency, 95th-percentile login latency, authentication throughput, registration latency, and rotation-related verifier-regeneration overhead. The non-rotating configuration measured the routine cost of pepper reconstruction, HMAC preprocessing, and bcrypt verification. The rotating configuration additionally measured verifier regeneration when a pending state rotation was completed following successful authentication.

For each configuration and account scale, hourly results were retained as 24 repeated observations for statistical analysis. These repeated periods were treated as matched blocks when comparing the four configurations. Because the 24 periods belong to a single benchmark sequence rather than 24 independently repeated full experiments, the resulting inferential statistics are interpreted as exploratory repeated-period comparisons rather than independent experimental replications.

3.9 Implementation and Reproducibility Details

CAYENNE+ was implemented in Python 3.11.9. The password recovery and authentication performance experiments were run on a Windows 11 system equipped with an Intel Core i7-14700KF processor, 64 GB of DDR4 memory, and an NVIDIA GeForce RTX 5060 Ti GPU. The implementation used the Python `bcrypt` package version 4.3.0, SciPy version 1.14.1 for statistical analysis, and `psutil` version 6.0.0 for system-utilization

measurements. All bcrypt-based configurations used cost factor $c = 12$, ensuring that the computational cost of the underlying password-hashing operation remained consistent across the no-pepper, static-pepper, and CAYENNE+ configurations.

HMAC-SHA-256 was used for both deterministic pepper derivation and password preprocessing in the peppered configurations. As described previously, the HMAC password-preprocessing output was passed to bcrypt as the raw 32-byte SHA-256 digest rather than as a hexadecimal or Base64 string.

The HSM-backed domains used to protect K_{secret} and the per-user derivation states were simulated in software. The implementation therefore reproduced the logical separation, retrieval, and update of protected verifier components, but did not reproduce the physical tamper resistance, hardware access controls, latency, or availability characteristics of a production HSM deployment.

For generation of the controlled password corpus, a dedicated pseudorandom number generator with seed 42 was used to select passwords reproducibly from the experimental wordlist. This seed controlled the synthetic password-generation process only. The master secret was generated using a cryptographically secure random source, while the per-user k values in the evaluated implementation were generated separately from the password-sampling random number generator.

The bcrypt attack simulation first measured local bcrypt verification throughput on the experimental system. A bcrypt verifier at the configured cost factor was generated for a fixed benchmark password, and parallel calls to `bcrypt.checkpw` were timed using the `time.perf_counter()`. The benchmark performed ten verification checks per worker and calculated the observed verification rate as the number of successful checks divided by the elapsed wall-clock time. The resulting measured rate was then used with the candidate-order and per-account check-count equations defined in the preceding subsection to estimate bcrypt recovery times.

The authentication-performance benchmark used 15 worker processes for all evaluated configurations. Account populations of 10,000, 50,000, and 200,000 users were evaluated over 24 authentication periods, with one login per account in each period. The same initial account population and CAYENNE+ state were restored before each configuration so that the no-pepper, static-pepper, non-rotating CAYENNE+, and rotating CAYENNE+ tests began from comparable account sets. Timing measurements were based on high-resolution wall-clock measurements using `time.perf_counter()`.

For each authentication configuration, the implementation recorded login latency, verification latency, verifier-regeneration latency where applicable, throughput, registration latency, successful and failed login counts, rotation counts, CPU utilization, and memory utilization. The benchmark also retained 24 period-level observations per configuration and account scale for subsequent statistical analysis.

Executed cracking and authentication runs were classified as *measured*. Bcrypt recovery times calculated from the observed verification rate and implemented candidate schedule were classified as *simulated*. R2 values derived using the state-space or mapped-output bounds were classified as *extrapolated*. This distinction was maintained throughout the results so that analytically derived values were not presented as directly measured runtimes.

4 Experiments and Results

The security experiments compared four attacker scenarios: N1, representing password hashing without a pepper; S1, representing exposure of a conventional static global pepper; R1, representing full CAYENNE+ compromise; and R2, representing partial CAYENNE+ compromise in which the master secret is exposed while the independently protected per-user derivation state remains unavailable. The comparison was designed to examine how the availability of verifier-side secret material affects offline password-recovery cost.

Exhaustive bcrypt execution across the complete account and password-candidate space was impractical. Bcrypt recovery times were therefore simulated using the locally measured verification rate together with the implemented candidate ordering and account-specific recovery positions described in Sect. 3.8. Executable SHA-256 experiments were conducted separately to validate the complete candidate-generation, preprocessing, and recovery workflow under a fast-hash condition. SHA-256 was used only for workflow validation and is not proposed as a password-storage alternative.

Results are classified according to how they were obtained. *Measured* results were produced directly by executable cracking or authentication experiments. *Simulated* results were calculated from measured bcrypt throughput and the implemented attack schedule. *Extrapolated* results were derived from measured or simulated baselines using the previously defined state-space or mapped-output bounds. This distinction is maintained throughout this section so that analytical estimates are not interpreted as directly measured attack times.

The metrics T_{50} and T_{90} denote the times by which 50% and 90% of the evaluated accounts were recovered, respectively. Because T_{50} corresponds to the median account-recovery time in the ordered evaluation, a separate median cracking-time metric is not reported.

4.1 Bcrypt Attack Simulation

The bcrypt simulation used 200,000 accounts, a controlled password-candidate space of 970,200 ordered three-word permutations, and bcrypt cost factor $c = 12$. The local benchmark produced an observed verification rate of 68.9632 bcrypt checks per second.

For N1, S1, and R1, all verifier-side information required for direct password testing is available to the attacker. As defined in Sect. 3.8, the simulated check count for an account whose password has one-based candidate rank r and zero-based account position p within a population of N accounts is

$$C_{\text{known}} = (r - 1)N + (p + 1). \quad (17)$$

The corresponding estimated recovery time is

$$t_i = \frac{C_i}{68.9632}. \quad (18)$$

The attack schedule is candidate-major: each password candidate is considered across the account population before the next candidate is reached. The reported average estimated check count is therefore the arithmetic mean of the per-account check counts rather than the aggregate number of checks required to recover the entire database.

Table 1 summarizes the simulated and extrapolated bcrypt results.

4.1.1 N1: No Pepper

N1 served as the baseline condition. No verifier-side pepper was applied, and the attacker was assumed to possess the bcrypt password-verifier database. Under this condition, attack cost depends on the bcrypt verification rate and the position of each password within the controlled candidate ordering.

The simulation produced a T_{50} of 44.62 years and a T_{90} of 80.22 years at the observed rate of 68.9632 checks per second. These values are specific to the experimental candidate space, account population, search order, bcrypt cost, and measured local verification rate and should not be interpreted as universal bcrypt cracking times.

Table 1 Simulated and extrapolated bcrypt password-recovery results across attacker scenarios

Scenario	Attacker condition	T_{50}	T_{90}	Avg. estimated checks/account
N1	No pepper	44.62 years	80.22 years	9.70×10^{10}
S1	Known static global pepper	44.62 years	80.22 years	9.70×10^{10}
R1	Full CAYENNE+ compromise	44.62 years	80.22 years	9.70×10^{10}
R2-naive	Partial CAYENNE+ compromise; direct state enumeration	4.46×10^6 years	8.02×10^6 years	9.70×10^{15}
R2-map	Partial CAYENNE+ compromise; mapping-aware extrapolation	4.46×10^5 years	8.02×10^5 years	9.70×10^{14}

4.1.2 S1: Known Static Global Pepper

S1 used one static global pepper that was assumed to be available to the attacker together with the password-verifier database. The attacker could therefore apply the known HMAC-based preprocessing operation to every password candidate before bcrypt verification.

Under the check-count model used for the bcrypt simulation, S1 has the same candidate and account traversal schedule as N1 and therefore produces the same simulated bcrypt check counts: $T_{50} = 44.62$ years and $T_{90} = 80.22$ years. The small computational cost of HMAC preprocessing was not separately added to the simulated bcrypt time; the estimate is based on bcrypt verification throughput, which dominates the evaluated password-verification cost.

This result should not be interpreted as showing that an uncompromised static pepper provides no security benefit. S1 deliberately models the condition in which the shared static pepper has already been exposed. Under that condition, the additional verifier-side secret no longer prevents offline candidate testing.

4.1.3 R1: Full CAYENNE+ Compromise

R1 represents full CAYENNE+ compromise. The attacker possesses the password-verifier database, master secret K_{secret} , and the corresponding per-user derivation state k for each account. Consequently, the attacker can reconstruct the correct account-specific pepper and directly test password candidates.

Because no unknown derivation component remains, R1 follows the same candidate-major bcrypt check schedule as N1 and S1. It therefore produces the same simulated check counts, with $T_{50} = 44.62$ years and $T_{90} = 80.22$ years.

As with S1, the CAYENNE+ derivation and HMAC preprocessing cost is not separately incorporated into the bcrypt simulation. The values in Table 1 represent bcrypt-check-equivalent estimates using the locally measured verification rate.

The R1 result establishes an important boundary of the proposed framework. CAYENNE+ is not intended

to provide additional protection after simultaneous compromise of all verifier-side components. Its design objective is to compartmentalize the information required for pepper reconstruction so that compromise of one protected domain does not automatically expose all required derivation material.

4.1.4 R2: Partial Compromise with Unknown Per-User State

R2 assumes that the attacker obtains the password-verifier database and K_{secret} , while the independently protected per-user state k remains unavailable.

For direct state enumeration, the simulated check-count expression is

$$C_{\text{R2-naive}} = (r - 1)Nk_{\text{max}} + pk_{\text{max}} + (k + 1). \quad (19)$$

The first term represents complete state enumeration for all accounts under preceding password candidates. The second represents state enumeration for accounts preceding the target under the matching password candidate, while the final term represents traversal from $k = 0$ through the target account's correct state.

With $k_{\text{max}} = 100,000$, the R2-naive interpretation gives $T_{50} \approx 4.46 \times 10^6$ years and $T_{90} \approx 8.02 \times 10^6$ years. These values represent the direct state-enumeration procedure used by the analytical model rather than an exhaustively executed bcrypt attack.

The experimental mapping, however, produces at most 10,000 distinct four-component mapped outputs. Consequently, different values of k may result in the same reconstructed experimental pepper. A mapping-aware attacker could derive candidate mapped peppers in advance, deduplicate collisions, and avoid repeating an expensive bcrypt check for identical mapped outputs.

For comparison, R2-map therefore applies the $M = 10,000$ mapped-output bound to the R1 simulated baseline. This produces $T_{50} \approx 4.46 \times 10^5$ years and $T_{90} \approx 8.02 \times 10^5$ years.

The R2-map values should be interpreted as mapping-aware extrapolations, not as a lower bound on attacker

effort or as an exact prediction of an optimized attack. Candidate ordering, early stopping, mapped-output frequency, precomputation, and reuse of derived values can affect the number of operations required in practice. Similarly, R2-naive represents direct traversal of the full configured state range rather than an optimal attack strategy. The two results are therefore useful for characterizing the evaluated experimental profile, but neither is an exhaustively measured bcrypt attack.

4.2 SHA-256 Workflow-Validation Experiment

The SHA-256 experiment provided an executable validation of the complete offline password-recovery workflow. Unlike the bcrypt simulation, candidate generation, verifier preprocessing, hash computation, and comparison were executed directly.

Under N1, S1, and R1, all 200,000 account passwords were recovered from the controlled candidate space of 970,200 combinations. The results are shown in Table 2.

N1 completed fastest, requiring 3.76 hours for recovery of all 200,000 passwords. The experiment reached T_{50} after 1.89 hours and T_{90} after 3.39 hours.

S1 required 13.99 hours for complete recovery, with $T_{50} = 7.00$ hours and $T_{90} = 12.61$ hours. Although the static pepper was known, each password candidate required HMAC preprocessing before the SHA-256 comparison, reducing the measured candidate-processing rate relative to N1.

R1 produced the longest directly measured runtime among the known-state conditions. Complete recovery required 1.47 days, with $T_{50} = 18.07$ hours and $T_{90} = 1.33$ days. In addition to password preprocessing, each candidate required reconstruction of the account-specific CAYENNE+ pepper. The additional derivation work reduced the measured candidate-processing rate to 7.6281 candidates per second.

The complete recovery observed under N1, S1, and R1 reinforces that SHA-256 is unsuitable as a password-storage algorithm. The differences among these scenarios reflect preprocessing and derivation overhead rather than sufficient resistance to offline guessing.

R2 was not exhaustively executed. Under the direct state-enumeration interpretation, applying the $k_{\max} = 100,000$ analytical multiplier to the measured R1 values gives an estimated complete recovery time of approximately 403.31 years, $T_{50} \approx 206.25$ years, and $T_{90} \approx 364.70$ years.

Using the mapping-aware 10,000-output bound gives an estimated complete recovery time of approximately 40.33 years, $T_{50} \approx 20.63$ years, and $T_{90} \approx 36.47$ years.

As with the bcrypt R2 results, these SHA-256 values are analytical extrapolations from the measured

R1 baseline rather than directly executed attack times. They illustrate the effect of the specified state-space and mapped-output assumptions within the experimental attack model.

4.3 Cracking-Time Comparison

Fig. 3 compares the T_{50} password-recovery times for SHA-256 and bcrypt across the evaluated attacker scenarios using a logarithmic year scale. The figure combines measured SHA-256 known-state results, extrapolated SHA-256 R2 values, simulated bcrypt known-state results, and extrapolated bcrypt R2 values.

A logarithmic scale is used because the results span several orders of magnitude, ranging from hours in the executable SHA-256 experiments to millions of years in the bcrypt R2 extrapolations.

Within the bcrypt simulation, N1, S1, and R1 produce the same estimated bcrypt check counts because all verifier-side information required for candidate testing is available and the same candidate-major traversal is used. The executable SHA-256 experiment, in contrast, records different wall-clock times for N1, S1, and R1 because the additional HMAC and CAYENNE+ derivation operations are directly executed.

Both R2-naive and R2-map increase substantially relative to R1 because the per-user verifier state remains unavailable. R2-naive represents direct enumeration of the configured 100,000-state range, whereas R2-map shows the supplementary mapping-aware extrapolation based on at most 10,000 distinct mapped outputs in the evaluated experimental profile.

These values should be interpreted within their respective measurement categories. Fig. 3 therefore illustrates the relative effect of the stated compromise assumptions rather than providing universal password-cracking times.

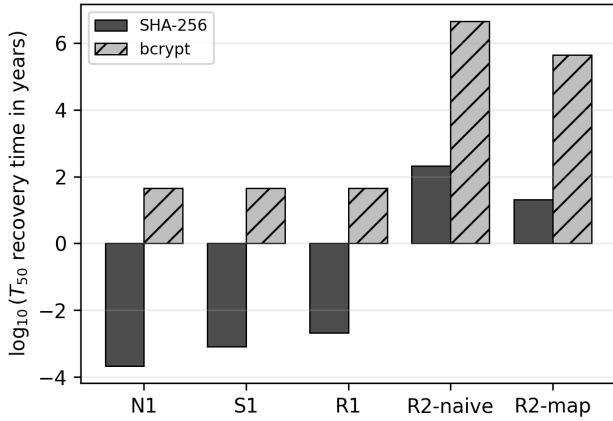
4.4 Access Evaluation

An authentication-performance experiment evaluated the operational overhead introduced by CAYENNE+ during legitimate registration and authentication. Three account populations were evaluated: 10,000, 50,000, and 200,000 users. Each account registered once and subsequently performed one executed authentication operation during each of 24 simulated login periods.

All configurations used bcrypt cost factor $c = 12$ and 15 worker processes. Four configurations were compared: No Pepper, Static Global Pepper, Non-Rotating CAYENNE+, and Rotating CAYENNE+.

Table 2 Measured and extrapolated SHA-256 password-recovery results across attacker scenarios

Scenario	Method	Result type	Cracked	Runtime	T_{50}	T_{90}	Candidates/s
N1	SHA-256 crack	Measured	200,000/200,000	3.76 h	1.89 h	3.39 h	71.6380
S1	SHA-256 with known static pepper	Measured	200,000/200,000	13.99 h	7.00 h	12.61 h	19.2575
R1	SHA-256 with CAYENNE+ full compromise	Measured	200,000/200,000	1.47 days	18.07 h	1.33 days	7.6281
R2-naive	R1 $\times k_{\max}$ state-space bound	Extrapolated	200,000/200,000	403.31 years	206.25 years	364.70 years	0.0000763
R2-map	R1 $\times 10,000$ mapped-output bound	Extrapolated	200,000/200,000	40.33 years	20.63 years	36.47 years	0.0007628

**Fig. 3** Comparison of T_{50} password-recovery times for measured or extrapolated SHA-256 results and simulated or extrapolated bcrypt results across attacker scenarios.

In the Non-Rotating CAYENNE+ configuration, the account retained the same per-user derivation state across successive authentications, however, in the Rotating CAYENNE+ configuration, an account was marked for renewal after five successful authentications using the current state. At the next successful authentication, the submitted password was first verified using the existing state and verifier. A fresh k was then selected, the corresponding account-specific pepper was reconstructed, and a new bcrypt verifier was generated using the password submitted during that same authentication. The plaintext password was not retained between authentication events.

This procedure resulted in state and verifier renewal during logins 6, 12, 18, and 24, producing four rotations per account during the 24-period experiment.

Table 3 reports the performance metrics calculated by the final access benchmark. Mean, median, and P95 login latency are calculated from the complete set of individual authentication latencies pooled across all 24 periods for each configuration and account scale. Average throughput is the arithmetic mean of the 24 period-level throughput measurements. This distinction is important for Rotating CAYENNE+, because the four renewal

periods account for approximately one-sixth of all authentication events and therefore substantially affect the pooled tail-latency distribution.

The three non-rotating configurations had closely grouped authentication performance across all evaluated account scales. At 200,000 users, mean login latency was 173.33 ms for No Pepper, 172.76 ms for Static Global Pepper, and 172.80 ms for Non-Rotating CAYENNE+. Their pooled P95 values were similarly close at 181.57, 181.60, and 181.65 ms, respectively.

At the same scale, average throughput was 83.24 authentications per second for No Pepper and 83.51 for Static Global Pepper. The corresponding value for Non-Rotating CAYENNE+ was 83.44 authentications per second. Under the evaluated conditions, deterministic pepper reconstruction and HMAC preprocessing contributed little cost relative to bcrypt verification.

Rotating CAYENNE+ produced higher mean and tail latency because four of the 24 login periods additionally performed state renewal and bcrypt verifier regeneration. At 200,000 users, the pooled mean login latency increased to 201.54 ms, while the pooled P95 increased to 351.45 ms. Average throughput decreased to 76.55 authentications per second.

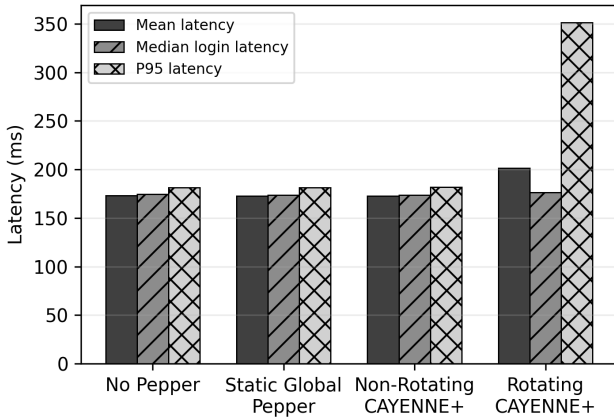
The increased latency was concentrated in the scheduled renewal periods rather than distributed uniformly throughout the workload. At 200,000 users, mean login latency during renewal periods 6, 12, 18, and 24 was approximately 345.53, 344.60, 345.70, and 346.00 ms, respectively. The corresponding period-level P95 values were approximately 362.05, 360.42, 361.74, and 361.61 ms. During ordinary non-rotation periods, mean login latency remained close to approximately 172–173 ms.

The periodic behavior was consistent across the three account scales. Four rotations per account produced 40,000 total verifier renewals at 10,000 users, 200,000 at 50,000 users, and 800,000 at 200,000 users. All authentication attempts completed successfully across all evaluated configurations.

These results indicate that routine Non-Rotating CAYENNE+ authentication introduced little measurable overhead relative to the no-pepper and static-

Table 3 Measured authentication performance across CAYENNE+ configurations and account scales

Scale	Scenario	Rotations	Mean (ms)	Median (ms)	P95 (ms)	Throughput (auth/s)
10,000	No Pepper	0	173.39	174.51	181.55	86.25
10,000	Static Global Pepper	0	173.26	174.31	181.58	86.30
10,000	Non-Rotating CAYENNE+	0	173.37	174.39	181.60	86.27
10,000	Rotating CAYENNE+	40,000	202.16	176.56	352.39	79.10
50,000	No Pepper	0	173.37	174.32	181.59	85.76
50,000	Static Global Pepper	0	173.33	174.24	181.61	85.76
50,000	Non-Rotating CAYENNE+	0	173.40	174.39	181.61	85.77
50,000	Rotating CAYENNE+	200,000	202.15	176.61	352.53	78.66
200,000	No Pepper	0	173.33	174.41	181.57	83.24
200,000	Static Global Pepper	0	172.76	173.50	181.60	83.51
200,000	Non-Rotating CAYENNE+	0	172.80	173.45	181.65	83.44
200,000	Rotating CAYENNE+	800,000	201.54	176.17	351.45	76.55

**Fig. 4** Authentication-latency comparison at 200,000 users. Mean, median, and P95 values are calculated from the complete pooled login latency distribution for each configuration.

pepper configurations in the evaluated environment. Rotating CAYENNE+ introduced additional cost primarily during renewal events, when the existing verifier was first checked and a new bcrypt verifier was subsequently generated.

The measured authentication latencies remained below one second, including during state-renewal periods. Nielsen’s response-time guidelines identify approximately one second as an important threshold for maintaining a user’s flow of thought [28]. The evaluated latencies therefore suggest that verifier renewal would not introduce a large individual delay under the tested hardware and workload conditions, although production latency would additionally depend on HSM access, storage operations, network conditions, concurrency, and deployment architecture.

Fig. 4 compares authentication latency at the largest evaluated account scale. The figure shows the close performance of the three non-rotating configurations and the higher tail latency of Rotating CAYENNE+ caused by the periodic verifier-renewal operations.

4.5 Friedman Test Results for Authentication Performance

A Friedman test was used as an exploratory repeated-period comparison of authentication performance across the four configurations. The test was performed separately for each account scale and performance metric using the 24 period-level observations as matched blocks. The four configurations were No Pepper, Static Global Pepper, Non-Rotating CAYENNE+, and Rotating CAYENNE+.

The evaluated metrics were period-level mean login latency, median login latency, P95 login latency, and authentication throughput. The null hypothesis for each test was that the rank distributions of the four configurations were equal across the 24 matched periods.

A significance level of $\alpha = 0.05$ was used for the individual omnibus tests. Because 12 Friedman tests were conducted, a conservative Bonferroni threshold of

$$\alpha_B = \frac{0.05}{12} \approx 0.0042$$

was also considered when assessing multiplicity. All results identified as statistically significant below remain below this conservative threshold.

Kendall’s W was reported as an effect-size measure and was calculated as

$$W = \frac{\chi^2}{n(k-1)}, \quad (20)$$

where $n = 24$ is the number of matched periods and $k = 4$ is the number of authentication configurations.

Table 4 summarizes the results.

At 10,000 users, the Friedman test did not identify an overall configuration effect for mean login latency ($\chi^2 = 4.10$, $p = 0.2509$, $W = 0.057$), median login latency ($\chi^2 = 0.75$, $p = 0.8614$, $W = 0.010$), or authentication throughput ($\chi^2 = 3.75$, $p = 0.2898$, $W = 0.052$). P95 login latency showed an overall difference ($\chi^2 = 33.35$, $p < 0.001$, $W = 0.463$).

Table 4 Friedman test results for authentication performance across account scales

Scale	Metric	Friedman χ^2	p -value	Kendall's W
10,000	Mean login latency	4.10	0.2509	0.057
10,000	Median login latency	0.75	0.8614	0.010
10,000	P95 login latency	33.35	< 0.001	0.463
10,000	Authentication throughput	3.75	0.2898	0.052
50,000	Mean login latency	6.75	0.0803	0.094
50,000	Median login latency	6.50	0.0897	0.090
50,000	P95 login latency	15.30	0.0016	0.213
50,000	Authentication throughput	2.10	0.5519	0.029
200,000	Mean login latency	26.55	< 0.001	0.369
200,000	Median login latency	27.15	< 0.001	0.377
200,000	P95 login latency	56.95	< 0.001	0.791
200,000	Authentication throughput	21.65	< 0.001	0.301

At 50,000 users, no overall difference was identified for mean login latency ($p = 0.0803$), median login latency ($p = 0.0897$), or authentication throughput ($p = 0.5519$). P95 login latency again showed an overall difference ($\chi^2 = 15.30$, $p = 0.0016$, $W = 0.213$).

At 200,000 users, overall configuration differences were detected for all four metrics: mean login latency ($\chi^2 = 26.55$, $p < 0.001$, $W = 0.369$); median login latency ($\chi^2 = 27.15$, $p < 0.001$, $W = 0.377$); P95 login latency ($\chi^2 = 56.95$, $p < 0.001$, $W = 0.791$); and authentication throughput ($\chi^2 = 21.65$, $p < 0.001$, $W = 0.301$).

The particularly pronounced period-level P95 effect at 200,000 users is consistent with the descriptive results, which show substantial tail-latency increases during the four deterministic verifier-renewal periods. The three non-rotating configurations, by contrast, remain closely grouped in the descriptive measurements.

The Friedman tests are omnibus tests and do not identify which specific pairs of configurations differ. No post-hoc pairwise significance tests or equivalence tests were used; consequently, the descriptive similarity among the three non-rotating configurations should not be interpreted as a formal statistical demonstration of equivalence.

In addition, the 24 matched periods originate from a single benchmark sequence rather than 24 independently repeated full experiments. Successive periods may therefore contain temporal dependence, and the rotation periods are deterministic by design. The Friedman results should consequently be interpreted as exploratory repeated-period comparisons that complement the descriptive latency and throughput measurements rather than as inference from 24 independent experimental replications.

Taken together, the performance results show that account-specific pepper reconstruction in Non-Rotating CAYENNE+ remained close to the no-pepper and static-

pepper configurations under the evaluated conditions. Optional state rotation introduced a larger but temporally concentrated cost because renewal requires an additional bcrypt verifier generation after successful authentication.

5 Discussion

The results from the experiments indicate that the security benefit of CAYENNE+ is conditional on the separation of the verifier-side components rather than being an inherent increase in the strength of the underlying password-hashing algorithm. Under the R1 full-compromise condition, the attacker possesses the password-verifier database, the master secret K_{secret} , and the corresponding per-user derivation state k . The account-specific pepper can therefore be reconstructed directly, and CAYENNE+ provides no additional offline resistance beyond that of the underlying password-hashing process. This result defines an important security boundary of the framework: CAYENNE+ is intended to provide compartmentalization under partial compromise rather than protection against simultaneous compromise of all verifier-side components.

The distinction from conventional static global peppering follows from this compromise model. A conventional global pepper depends on the secrecy of a single shared value; once that value becomes available to an attacker, the additional protection associated with pepper secrecy is lost across all accounts protected by it. CAYENNE+ retains a global master secret but combines it with independently protected per-user state when reconstructing each account-specific pepper. Consequently, compromise of K_{secret} alone does not immediately provide the complete derivation information required for every account. The security objective is therefore not the elimination of a global root secret, but the distribution

of the information required for pepper reconstruction across independently protected security domains.

The R2 evaluation illustrates the additional work introduced when the per-user derivation state remains unavailable. Under the evaluated attack model, an attacker possessing K_{secret} must additionally search possible state or mapped-pepper values before a password candidate can be verified. The R2-naïve estimate represents direct enumeration of the configured $k_{\text{max}} = 100,000$ state range, whereas the R2-map estimate accounts for the maximum 10^4 distinguishable selection vectors produced by the experimental four-component modulo-10 mapping. These values should be interpreted as analytical bounds under the specified enumeration procedures rather than as guaranteed multiplicative security factors or lower bounds on the cost of an optimized attack.

This distinction is particularly important because the readable pepper representation used in the experiments was designed primarily to support implementation transparency and validation. The four-component modulo-10 construction limits the number of distinguishable mapped outputs independently of the larger configured state range. The resulting 10^4 -output bound is therefore a property of the experimental mapping rather than a fundamental limitation of the CAYENNE+ architecture. A deployment-oriented implementation should preserve substantially more of the HMAC-derived output space, for example by using higher-cardinality mapping components or by deriving fixed-length cryptographic pepper material more directly from the HMAC output.

The authentication experiments further show that the security mechanism can be introduced without substantial routine computational overhead under the evaluated conditions. Non-Rotating CAYENNE+ remained practically close to the No Pepper and Static Global Pepper configurations because deterministic HMAC-based pepper reconstruction contributes relatively little computation compared with bcrypt verification. The statistical analysis should be interpreted consistently with these descriptive measurements: although statistically detectable differences appeared for some metrics, particularly at the largest account scale, the absolute differences among the non-rotating configurations remained small.

Rotating CAYENNE+ exhibited a different performance profile because state renewal requires regeneration of the bcrypt verifier after successful authentication. The resulting overhead was concentrated in scheduled rotation periods rather than being imposed uniformly on every authentication attempt. This behavior explains the larger differences observed in tail latency, including the stronger P95 effects at higher account scales. Rotation therefore represents an explicit security–performance

trade-off: additional computation is incurred periodically in exchange for renewal of the per-user derivation state, while routine authentication remains close to the non-rotating configuration.

State rotation should nevertheless not be interpreted as retroactively protecting previously compromised verifier records. If an attacker obtains a valid verifier before state renewal, subsequent rotation does not invalidate the attacker’s stored copy or prevent continued offline analysis of that verifier. Rotation instead renews the state associated with future authentication records and limits continued dependence on an unchanged verifier-side derivation state. Its value therefore complements, rather than replaces, the compartmentalization provided by independently protected derivation components.

Taken together, the results support CAYENNE+ as a verifier-side compartmentalization framework rather than as a replacement for established password-hashing algorithms. Its practical security benefit depends on maintaining genuine independence between the master-secret and per-user-state protection domains, while its performance cost is dominated by the frequency and computational requirements of optional verifier-state rotation.

6 Security Interpretation and Design Recommendations

CAYENNE+ should be deployed as a verifier-side compartmentalization mechanism in conjunction with an established password-hashing algorithm. Its security benefit depends on maintaining genuine independence between the master secret K_{secret} and the per-user derivation state k . The two components should therefore be maintained in separate protected security domains such that compromise of one domain does not automatically provide access to the other.

This separation should extend beyond the logical storage of the two values. A production deployment should prevent compromise of a single authentication host, administrative credential, service account, or management interface from providing unrestricted access to both derivation components. Separate authorization policies, administrative roles, audit paths, access-control mechanisms, and rate limits should therefore be applied to the master-secret and per-user-state services. Otherwise, a compromised application with unrestricted access to both services could potentially use them as on-line derivation oracles without directly extracting either protected value.

Access to HSM or secret-management interfaces should consequently be authenticated, authorized, rate

limited, audited, and monitored for abnormal behavior. In particular, unusually large numbers of derivation requests, sequential access across many accounts, or repeated requests for different state values should be treated as potentially suspicious activity. These controls are important because compartmentalization provides meaningful protection only when the independently protected components cannot be jointly queried without restriction.

The readable natural-language pepper representation and four-component modulo-10 mapping evaluated in this study should be regarded as an experimental profile rather than a recommended production parameterization. The representation was selected primarily to make state-dependent changes observable during implementation and testing. Because the evaluated mapping provides at most 10^4 distinguishable selection vectors, simply increasing the configured state range without changing the mapping does not proportionally increase the effective mapped-pepper space.

A deployment-oriented CAYENNE+ profile should therefore preserve substantially more of the pseudo-random output produced by HMAC-SHA-256. One approach is to increase both the number and cardinality of independently mapped components. A more direct approach is to derive fixed-length cryptographic pepper material from the HMAC output without reducing it to the low-cardinality representation used in the experimental implementation. The compartmentalization architecture itself does not depend on natural-language words or on the specific modulo-10 mapping evaluated in this study.

Production implementations should also support explicit algorithm, key, and derivation-profile versioning. Version metadata allows changes to K_{secret} , derivation parameters, or pepper representations to be introduced without making existing verifier records ambiguous. Such versioning also supports controlled migration and incident response when cryptographic parameters or protected secrets must be replaced.

Because password-dependent verifier regeneration requires access to the submitted password, migration can naturally occur following successful authentication. After a user’s password has been verified using the existing state and configuration, the system can generate a new per-user state, reconstruct a new pepper under the current derivation profile, and create a replacement password verifier without storing the plaintext password beyond the authentication operation. Accounts that remain inactive, or accounts requiring immediate migration following a security incident, may instead require forced credential reset.

Optional state rotation should similarly be performed only when the user’s password is available during successful authentication. The rotation frequency should be selected according to the deployment’s performance requirements and threat model rather than treated as a mandatory operation on every login. More frequent renewal reduces the lifetime of unchanged verifier-side state but increases the frequency of password-verifier regeneration and its associated computational cost.

Finally, deployment evaluation should consider operational factors not represented by the software-simulated environment used in this study. These include HSM access latency, service concurrency limits, failover behavior, key and state backup procedures, recovery after partial service failure, replication consistency, rate-limiting behavior, and denial-of-service resilience. Evaluation on production-grade hardware-backed secret-management infrastructure is therefore necessary before drawing conclusions about the operational performance of a deployed CAYENNE+ system.

7 Limitations

Several limitations should be considered when interpreting the results of this study. First, the password corpus was generated from a controlled set of 100 common English words and was designed to provide a bounded and reproducible candidate space. Although this supports consistent comparison among the evaluated protection schemes, it does not represent the full diversity of real-world password selection. Actual user passwords may include names, dates, symbols, substitutions, multilingual content, personal information, reused credentials, and other structures that are not represented in the experimental dataset. The reported cracking times should therefore be interpreted within the controlled candidate space used in this study rather than as estimates of general real-world password-cracking performance.

Second, the experiments were conducted using a Python implementation on a single workstation, and the HSM-backed protection domains were simulated in software. Consequently, the measured authentication and attack performance reflects the evaluated hardware, software stack, and implementation environment. The study does not reproduce the latency, concurrency behavior, physical tamper resistance, access-control guarantees, or operational characteristics of production HSM or secret-management infrastructure.

Third, the bcrypt R2 results were analytically extrapolated rather than obtained through exhaustive execution of the complete partial-compromise attack space. The R2-naïve estimate applies the configured

$k_{\max} = 100,000$ state-space multiplier to the corresponding R1 baseline, whereas the R2-map estimate uses the maximum 10^4 distinguishable selection vectors produced by the evaluated mapping. These estimates describe the cost of the specified enumeration strategies and should not be interpreted as lower bounds on the cost of an optimized attacker. Alternative candidate ordering, precomputation, early stopping, or other attack optimizations may reduce the effective work required.

Fourth, the evaluated threat model focuses primarily on verifier-database compromise combined with different levels of access to the CAYENNE+ derivation components. In particular, R2 considers exposure of K_{secret} while the independently protected per-user state k remains unavailable. Other compromise paths were outside the scope of the current experiments, including simultaneous compromise of both protected domains, live access to HSM or secret-management interfaces, abuse of derivation services as online oracles, online password-guessing attacks, state-manipulation attacks, denial-of-service attacks, and administrative compromise. These cases require separate evaluation.

Fifth, the experimental CAYENNE+ profile used four modulo-10 mapped components, resulting in at most 10^4 distinguishable selection vectors. This limit arises from the configured mapping rather than from HMAC-SHA-256 or from the use of natural-language words themselves. Replacing the words with fixed-length cryptographic tokens while preserving the same four-component modulo-10 structure would alter the representation but would not increase the mapped-output cardinality. Evaluation of higher-cardinality mappings or direct derivation of cryptographic pepper material from the HMAC output is therefore required for a deployment-oriented configuration.

Finally, state rotation renews the verifier-side state used for future authentications but does not invalidate verifier records that may already have been copied by an attacker. An adversary possessing a previously stolen verifier can continue performing offline analysis against that verifier after rotation occurs. The rotation mechanism should therefore be interpreted as renewal of current verifier-side state rather than retroactive protection of previously compromised records.

These limitations constrain the generalizability of the numerical results but do not alter the architectural objective evaluated in this study: distributing the information required for account-specific pepper reconstruction across independently protected verifier-side components.

8 Conclusion

This study introduced CAYENNE+, a deterministic verifier-side pepper-derivation framework designed to reduce dependence on a single globally shared static pepper. Rather than replacing established password-hashing algorithms, CAYENNE+ operates as an additional protection layer that reconstructs account-specific peppers from a protected master secret K_{secret} and independently protected per-user derivation state k .

The security evaluation shows that the principal benefit of CAYENNE+ arises under partial verifier-side compromise. Under the R1 condition, where the attacker obtains the password-verifier database, K_{secret} , and the corresponding per-user state, CAYENNE+ provides no additional resistance beyond the underlying password-hashing process. Under R2, where the per-user state remains unavailable, the attacker must perform additional state or mapped-output enumeration before password candidates can be evaluated. The mapping-aware analysis further demonstrates that this additional search must be interpreted according to the number of distinguishable outputs produced by the implemented derivation profile rather than directly from the configured state-space size.

The authentication experiments indicate that deterministic account-specific pepper reconstruction introduces little observed routine overhead relative to bcrypt under the evaluated conditions. Non-Rotating CAYENNE+ remained practically close to the No Pepper and Static Global Pepper configurations across the evaluated account scales. Optional state rotation introduced additional computation because successful rotation requires regeneration of the bcrypt verifier, but this overhead was concentrated in scheduled rotation events rather than ordinary authentication.

Overall, CAYENNE+ demonstrates the feasibility of distributing the information required for pepper reconstruction across independently protected verifier-side components while maintaining compatibility with existing password-hashing workflows. Its security contribution is therefore best understood as verifier-side compartmentalization rather than modification of the underlying password-hashing algorithm. Future work should evaluate higher-cardinality or direct cryptographic pepper derivation, production-grade HSM-backed deployments, broader attacker models, and more diverse password datasets.

Statements and Declarations

Funding

This work was partially supported by The Telecommunications Advancement Foundation Research Grant and JSPS KAKENHI Grant Number 23K11337.

Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data Availability

The data used in this study were generated synthetically as part of the experimental evaluation. The datasets and experimental code are available from the corresponding author upon reasonable request.

Author Contributions

Nathaniel Kofi Adjadeh: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Visualization, Writing – original draft. Oky Dicky Ardiansyah Prima: Supervision, Writing – review & editing.

Masaki Narita: Supervision, Funding acquisition, Writing – review & editing.

References

1. Bonneau, J., Herley, C., van Oorschot, P.C., Stajano, F.: The quest to replace passwords: a framework for comparative evaluation of web authentication schemes. In: 2012 IEEE Symposium on Security and Privacy, pp. 553–567. IEEE Computer Society (2012). <https://doi.org/10.1109/SP.2012.44>
2. FIDO Alliance: Passkeys. <https://fidoalliance.org/passkeys/>
3. FIDO Alliance: New survey: half of people use passkeys as frustrations with passwords continue (2024). <https://fidoalliance.org/new-survey-half-of-people-use-passkeys-as-frustrations-with-passwords-continue/>
4. Kemp, S.: Digital 2025: Global Overview Report. DataReportal (2025). <https://datareportal.com/reports/digital-2025-global-overview-report>
5. IBM: Cost of a Data Breach Report 2025. IBM (2025)
6. Verizon: 2025 Data Breach Investigations Report. Verizon Business (2025)
7. Sharif, M.H.U., Mohammed, M.A.: A literature review of financial losses statistics for cyber security and future trend. World Journal of Advanced Research and Reviews 15(1), 138–156 (2022)
8. Ah Kioon, M.C., Wang, Z.S., Deb Das, S.: Security analysis of MD5 algorithm in password storage. In: Proceedings of the 2nd International Symposium on Computer, Communication, Control and Automation (ISCCCA 2013), pp. 706–709. Atlantis Press (2013). <https://doi.org/10.2991/isccca.2013.177>
9. Grassi, P., Fenton, J.L., Newton, E.M., Perlner, R.A., Regenscheid, A.R., Burr, W.E., Richer, J.P.: NIST Special Publication 800-63B: Digital Identity Guidelines. NIST (2020). <https://doi.org/10.6028/NIST.SP.800-63b>
10. OWASP Foundation: Password Storage Cheat Sheet (2024). https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
11. Morris, R., Thompson, K.: Password security: a case history. Communications of the ACM 22(11), 594–597 (1979). <https://doi.org/10.1145/359168.359172>
12. Rivest, R.: RFC 1321: The MD5 Message-Digest Algorithm. RFC Editor (1992)
13. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Advances in Cryptology – CRYPTO 2005, vol. 3621, pp. 17–36. Springer, Berlin Heidelberg (2005). https://doi.org/10.1007/11535218_2
14. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Advances in Cryptology – EUROCRYPT 2005, vol. 3494, pp. 19–35. Springer, Berlin Heidelberg (2005). https://doi.org/10.1007/11426639_2
15. National Institute of Standards and Technology (NIST): Hash Functions (2017). <https://csrc.nist.gov/projects/hash-functions>
16. National Institute of Standards and Technology (NIST): NIST Retires SHA-1 Cryptographic Algorithm (2022). <https://www.nist.gov/news-events/news/2022/12/nist-retires-sha-1-cryptographic-algorithm>
17. Provos, N., Mazieres, D.: A future-adaptable password scheme. In: 1999 USENIX Annual Technical Conference (USENIX ATC 99). USENIX Association, Monterey, CA (1999). <https://www.usenix.org/conference/1999-use-nix-annual-technical-conference/future-adaptable-password-scheme>
18. Biryukov, A., Dinu, D., Khovratovich, D.: Argon2: new generation of memory-hard functions for password hashing and other applications. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 292–302 (2016). <https://doi.org/10.1109/EuroSP.2016.31>
19. Biryukov, A., Dinu, D., Khovratovich, D., Josefsson, S.: RFC 9106: Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications (2021). <https://doi.org/10.17487/RFC9106>
20. Bai, W., Blocki, J., Ameri, M.H.: Cost-asymmetric memory hard password hashing. Information and Computation 297, 105134 (2024). <https://doi.org/10.1016/j.ic.2023.105134>
21. Manber, U.: A simple scheme to make passwords based on one-way functions much harder to crack. Computers & Security 15(2), 171–176 (1996). [https://doi.org/10.1016/0167-4048\(96\)00003-X](https://doi.org/10.1016/0167-4048(96)00003-X)
22. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-Hashing for Message Authentication. RFC 2104. Internet Engineering Task Force (1997). <https://doi.org/10.17487/RFC2104>
23. Chen, L.: Recommendation for Key Derivation Using Pseudorandom Functions. NIST Special Publication 800-108 Rev. 1. National Institute of Standards and Technology (2022). <https://doi.org/10.6028/NIST.SP.800-108r1>
24. Najjar, M.: Using improved d-HMAC for password storage. Computer and Information Science 10(3), 1–9 (2017). <https://doi.org/10.5539/cis.v10n3p1>

25. Najjar, M.: d-HMAC – an improved HMAC algorithm. *International Journal of Computer Science and Information Security* 13, 89–96 (2015)
26. Obiri-Tetteh, J.A., Turkson, R.A., Frimpong, E.A., Yussif, A.-S., Asante, D.A., Attipoe, E.: Bcrypt-based optimized password hashing against targeted Internet of Things attacks. *Cureus Journal of Computer Science* (2025). <https://doi.org/10.7759/s44389-025-05167-y>
27. Adjadeh, N.K., Narita, M.: CAYENNE: Spicing Up Password Security with Dynamic Peppers. In: *Advanced Information Networking and Applications*, vol. 297, pp. 387–398. Springer Nature Switzerland (2026). https://doi.org/10.1007/978-3-032-23304-2_31
28. Nielsen, J.: Response Times: The 3 Important Limits (1993). <https://www.nngroup.com/articles/response-times-3-important-limits/>