The Design and Formal Analysis of the Secure Infrastructure Access Protocol

John G. Underhill

Quantum Resistant Cryptographic Solutions Corporation
contact@qrcscorp.ca

November 2025

Abstract

The Secure Infrastructure Access Protocol (SIAP) is a symmetric, stateful authentication and token-release construction for controlled access to infrastructure, removable-token authentication, and offline authorization. SIAP combines a server-held derivation secret, fixed-width hierarchical identities, a cSHAKE-derived token tree, a passphrase-derived sealing layer, authenticated encryption of device state, and destructive one-time leaf consumption. This paper gives an implementation-aligned formal treatment of SIAP and evaluates the protocol under explicit assumptions on cSHAKE, SHAKE, SCB, RCS, entropy generation, durable state, and endpoint control.

The analysis models SIAP as a stateful symmetric authenticator rather than as a public-key key exchange. It proves correctness, token unforgeability, replay resistance under monotonic durable state, confidentiality of the sealed token tree under passphrase secrecy and AEAD security, and post-quantum resistance within the symmetric-security model. The paper also gives a cryptanalytic evaluation of rollback, concurrency, desynchronization, passphrase-verifier leakage, malformed encodings, state compromise, and denial-of-service behavior. The resulting analysis identifies the exact boundary between cryptographic security and operational enforcement: SIAP's central invariants hold only when successful authentication commits the advanced device key and server tag atomically, when per-identity authentication attempts are serialized, and when the server base key remains within the intended trust boundary.

Contents

1 Introduction

1.1 Problem Statement

Infrastructure authentication often requires a different security profile from ordinary web login. The relying system may be offline, the protected resource may be a local console or encrypted storage object, and the authenticator may need to release a symmetric value that is consumed immediately by another subsystem. A public-key credential is not always necessary in this setting; in some deployments it can be operationally undesirable because it introduces certificate issuance, path validation, revocation, online status checking, and long-lived private-key handling.

SIAP addresses this narrower problem. It is a stateful symmetric authentication construction in which a device carries an encrypted token tree and the server stores a compact tag for the current device state. Each successful authentication consumes exactly one leaf, advances the key identity index, updates the server-side tag, and reseals the device state. The design objective is not to establish an anonymous channel or to replace a general-purpose authenticated key exchange. The objective is to release a fresh symmetric authentication token after verifying possession of the device state and knowledge of the passphrase.

1.2 Design Positioning

SIAP is closest in spirit to one-time password and hash-chain systems, but it differs from conventional HOTP and TOTP deployments in two important respects. First, the released value is a full-width symmetric token rather than a short decimal code. Second, the device state is an encrypted tree of one-time leaves rather than a single shared HOTP/TOTP secret. This makes SIAP more suitable for systems that need a cryptographic key release primitive, while retaining the operational simplicity of a symmetric authentication model.

The construction is also distinct from password-authenticated key exchange. SIAP does not attempt to hide the authentication transcript from the server, and it does not establish a mutually negotiated session key between two remote principals. The server is trusted to hold the base derivation key, and the device is trusted to hold the current encrypted token state. The passphrase protects the sealed device tree and supplies a second authentication factor. These roles must be modeled explicitly; otherwise the security claims of the construction are easy to overstate.

1.3 Contributions

This paper makes the following contributions. It gives an implementation-aligned mathematical specification of SIAP, including identity encodings, passphrase processing, token derivation, tag generation, RCS sealing, authentication, and state advancement. It defines formal security experiments for correctness, impersonation, replay, rollback, token confidentiality, and passphrase guessing. It provides reductions and proof sketches relating SIAP security to the pseudorandomness of cSHAKE, the preimage and collision resistance of SHAKE in the modeled context, the memory-hardness and domain separation of SCB processing, and the authenticated-encryption security of RCS.

The paper also provides an explicit cryptanalysis of non-ideal deployment conditions. It examines what is and is not protected under server database disclosure, token theft, server base-key compromise, stale-state rollback, non-atomic persistence, concurrent same-identity attempts, malformed serialization, and denial-of-service attempts. The analysis is intentionally conservative. It treats operational state management as part of the security boundary and does not claim properties that require an unmodeled transaction layer.

1.4 Summary of Results

Under the assumptions stated in Section 5, SIAP provides the following properties. A party that does not know the current token leaf or the server base key cannot forge a valid authentication except with probability bounded by the token length and the distinguishing advantage against cSHAKE. A replay of an old device state is rejected if the server has durably committed the advanced tag and the implementation rejects stale or mismatched key indices. A stolen sealed device key does not reveal the token tree if the passphrase-derived sealing key remains unknown and the underlying AEAD is secure. A compromise of the server tag reveals the current identity, tree commitment, and passphrase verifier, but not the token leaves or the live passphrase-derived sealing key in the repaired construction. A compromise of K_{base} is a boundary event: it enables derivation of all device leaves under that server identity and must be treated as a full compromise of that server domain.

2 Related Work and Standards Context

SIAP relies on standardized SHA-3 family functions. FIPS 202 specifies SHA-3 and SHAKE functions based on the Keccak permutation [4]. NIST SP 800-185 specifies cSHAKE and KMAC, including cSHAKE's function-name and customization-string interfaces for domain separation [5]. Random generation requirements are

aligned with the general requirement that security-critical secrets be generated from an approved or otherwise appropriate random bit generator [6].

Event-based and time-based one-time password systems are standardized by HOTP and TOTP [8, 9]. These systems derive short authentication codes from a shared secret and a moving factor. Lamport's hash-chain authentication construction is an earlier example of one-time password authentication over insecure channels [10]. SIAP differs from those mechanisms by releasing a full-width symmetric token and by maintaining an encrypted local token tree that is resealed after each successful use.

Authenticated encryption with associated data formalizes the requirement that ciphertext integrity bind both encrypted plaintext and non-encrypted metadata [11]. SIAP uses this concept to bind the token tree to metadata such as the key identity and expiration field. The paper also draws on standard proof techniques for symmetric authentication and pseudorandom function constructions, while remaining implementation-oriented rather than presenting a new primitive.

3 Protocol Overview

3.1 Entities

The protocol involves three logical entities. The server stores K_{base}, SID, D_{salt}, expiration policy, and one device tag per enrolled device. The device stores the sealed device key, including an encrypted token tree, the current key identity, and an expiration field. The user supplies a passphrase that is processed into a sealing secret and a server-verifiable passphrase verifier. The physical implementation may combine user and device behavior in a local reader, but the analysis separates the roles because the security assumptions differ.

3.2 State Objects

Let $\lambda \in \{256, 512\}$ denote the nominal symmetric security mode. The implemented default uses 256-bit tokens and hash outputs. The extended mode uses 512-bit outputs where the corresponding compile-time mode is enabled.

The server key is

$$SK = (K_{\text{base}}, \text{SID}, D_{\text{salt}}, \text{Exp}^{srv}),$$

where $K_{\text{base}} \leftarrow \{0,1\}^{\lambda}$ is generated uniformly, SID is a fixed-width server identity, D_{salt} is derived from K_{base}, and Exp^{srv} is the server-key expiration time.

The device key is

$$DK = (C_{\mathcal{T}}, \text{Kid}, \text{Exp}^{dev}),$$

where $C_{\mathcal{T}}$ is the sealed token tree plus authentication tag, Kid $=$ DID $\|$ idx, DID is a fixed-width device identity, and idx is a 32-bit index encoded in the key identity.

The server-side device tag is

$$DT = (\text{Kid}, K_{\text{hash}}, P_{\text{ver}}).$$

Here K_{hash} commits to the current plaintext tree state, and P_{ver} is a verifier derived from the passphrase hash and Kid. The implementation field name may preserve legacy terminology, but semantically the stored value is a verifier and not the live sealing-key precursor.

3.3 Derivation Functions

The derivations are modeled as domain-separated uses of cSHAKE and SHAKE. The following labels are used as abstract domain-separation constants:

$$\begin{aligned} L_{seed} &= \texttt{SIAP-PHASH-SEED}, \\ L_{ver} &= \texttt{SIAP-PHASH-VERIFIER}, \\ L_{seal} &= \texttt{SIAP-RCS-SEAL-v1}. \end{aligned}$$

The server salt is

$$D_{\text{salt}} = \text{cSHAKE}_{\lambda}(K_{\text{base}}; \text{Config}, \text{SID}).$$

The ith tree leaf is

$$\mathcal{T}[i] = \text{cSHAKE}_{\lambda}(K_{\text{base}}; \text{Config}, \text{DID} \parallel i).$$

The passphrase seed and passphrase hash are

$$Seed = \text{cSHAKE}_{\lambda}(passphrase; L_{seed}, \text{Config}),$$

$$P_{\text{hash}} = \text{SCB}(Seed; \text{Config}, c_{cpu}, c_{mem}).$$

The SCB cost parameters are implementation constants. This paper analyzes the construction for arbitrary fixed parameters and does not assign a time-cost claim to the default constants.

The passphrase verifier is

$$P_{\text{ver}} = \text{cSHAKE}_{\lambda}(P_{\text{hash}}; L_{ver}, \text{Kid}).$$

The sealing metadata is

$$\text{SealInfo} = L_{seal} \parallel \text{Kid} \parallel \text{Exp}^{dev}.$$

The RCS key and nonce are derived as

$$K_{rcs} \parallel N_{rcs} = \text{cSHAKE}_{\lambda}(P_{\text{hash}}; L_{seal}, D_{\text{salt}} \parallel \text{SealInfo}).$$

RCS is invoked with SealInfo as associated data. Thus the key schedule, nonce input, and AEAD authentication context all depend on the current key identity and expiration.

4 Implementation-Aligned Specification

4.1 Enrollment

Enrollment creates a device key and a corresponding server tag. The server generates K_{base}, assigns SID, computes D_{salt}, assigns DID, initializes Kid $=$ DID $\parallel 0$, and derives the full tree $\mathcal{T}[0], \ldots, \mathcal{T}[N-1]$ for $N = 1024$ in the default implementation. The tree commitment is

$$K_{\text{hash}} = \text{SHAKE}_{\lambda}(\mathcal{T}).$$

The passphrase is transformed into P_{hash} through the seed-and-SCB procedure above. The server stores

$$DT = (\text{Kid}, K_{\text{hash}}, P_{\text{ver}}),$$

where P_{ver} is derived from P_{hash} and Kid. The device receives the sealed device key

$$C_{\mathcal{T}} = \text{RCS.Enc}_{K_{rcs}, N_{rcs}}(\mathcal{T}; \text{SealInfo}).$$

The plaintext tree is an enrollment intermediate and must not be exported as the device state.

4.2 Authentication

A complete authentication run is shown in Algorithm 1. The important ordering property is that the current leaf is compared before it is erased and before idx is advanced. State mutation occurs only after all verification checks needed for authentication success have passed.

Algorithm 1 SIAP authentication and state advancement

Require: Sealed device key DK, server tag DT, server key SK, passphrase pw
Ensure: Accept with token T, or reject without advancing state

parse $DK = (C_{\mathcal{T}}, \text{Kid}^{dev}, \text{Exp}^{dev})$ and $DT = (\text{Kid}^{tag}, K^{tag}_{\text{hash}}, P^{tag}_{\text{ver}})$
if $\text{Kid}^{dev} \neq \text{Kid}^{tag}$ **then**
    return reject
if Exp^{dev} is expired or outside server policy **then**
    return reject
$Seed \leftarrow \text{cSHAKE}_{\lambda}(pw; L_{seed}, \text{Config})$
$P_{\text{hash}} \leftarrow \text{SCB}(Seed; \text{Config}, c_{cpu}, c_{mem})$
$P_{\text{ver}} \leftarrow \text{cSHAKE}_{\lambda}(P_{\text{hash}}; L_{ver}, \text{Kid}^{tag})$
if $P_{\text{ver}} \neq P^{tag}_{\text{ver}}$ in constant time **then**
    return reject
$\text{SealInfo} \leftarrow L_{seal} \parallel \text{Kid}^{dev} \parallel \text{Exp}^{dev}$
derive K_{rcs}, N_{rcs} from P_{hash}, L_{seal}, $D_{\text{salt}} \parallel \text{SealInfo}$
$\mathcal{T} \leftarrow \text{RCS.Open}_{K_{rcs}, N_{rcs}}(C_{\mathcal{T}}; \text{SealInfo})$
if RCS authentication fails **then**
    return reject
if $\text{SHAKE}_{\lambda}(\mathcal{T}) \neq K^{tag}_{\text{hash}}$ **then**
    return reject
$i \leftarrow \text{idx}(\text{Kid}^{dev})$
if $i \geq N$ **then**
    return reject
$T_{srv} \leftarrow \text{cSHAKE}_{\lambda}(K_{\text{base}}; \text{Config}, \text{DID} \parallel i)$
if $\mathcal{T}[i] \neq T_{srv}$ in constant time **then**
    return reject
$T \leftarrow \mathcal{T}[i]$
erase $\mathcal{T}[i]$ and set $\text{Kid}' \leftarrow \text{DID} \parallel (i+1)$
$K'_{\text{hash}} \leftarrow \text{SHAKE}_{\lambda}(\mathcal{T})$
$P'_{\text{ver}} \leftarrow \text{cSHAKE}_{\lambda}(P_{\text{hash}}; L_{ver}, \text{Kid}')$
$\text{SealInfo}' \leftarrow L_{seal} \parallel \text{Kid}' \parallel \text{Exp}^{dev}$
$C'_{\mathcal{T}} \leftarrow \text{RCS.Enc}(\mathcal{T}; \text{SealInfo}')$ under the derived key and nonce
commit $DK' = (C'_{\mathcal{T}}, \text{Kid}', \text{Exp}^{dev})$ and $DT' = (\text{Kid}', K'_{\text{hash}}, P'_{\text{ver}})$ atomically
return accept with token T

4.3 Serialization Requirements

All serialized state objects have fixed encoded lengths. A conforming parser must reject a serialized server key, device key, or device tag unless the supplied input length is at least the corresponding encoded length. This is an implementation-security requirement rather than a cryptographic primitive requirement. Without length validation, malformed persistent state may cause out-of-bounds reads or writes before any cryptographic verification occurs.

4.4 State Commit Requirement

SIAP is stateful. Its replay and rollback properties require a commit rule:

$$Accept \Rightarrow \text{Commit}(DK', DT') \text{ before relying-party authorization is finalized.}$$

A crash between writing DK' and DT' can desynchronize the device and server views. A conforming deployment therefore needs either an atomic database transaction, a write-ahead log, a two-phase state update, or an equivalent recovery mechanism. The formal analysis below treats this as an explicit assumption rather than an automatic property of the cryptographic algorithm.

5 Security Model and Assumptions

5.1 Primitive Assumptions

The analysis uses the following assumptions.

Assumption 1 (cSHAKE pseudorandomness)**.** *For independent domain labels and fixed output length, cSHAKE is modeled as a pseudorandom function keyed by its input key material. An adversary without the key input cannot distinguish cSHAKE outputs from uniform strings except with advantage* $\text{Adv}^{\text{cshake}}$.

Assumption 2 (SHAKE commitment behavior)**.** *For the fixed tree encoding used by SIAP, SHAKE is second-preimage resistant and collision resistant at the selected security level. An adversary cannot produce a different tree state with the same* K_{hash} *except with advantage* $\text{Adv}^{\text{shake}}$.

Assumption 3 (SCB passphrase transform)**.** *For the configured SCB parameters, the mapping from a passphrase-derived fixed seed to* P_{hash} *has the intended password-hardening cost. The effective offline guessing resistance is parameter and passphrase-entropy dependent.*

Assumption 4 (RCS authenticated encryption)**.** *RCS is IND-CPA secure and INT-CTXT secure under nonce uniqueness for a fixed key, and its associated data is cryptographically bound to ciphertext verification.*

Assumption 5 (Entropy)**.** *The random generation of* K_{base} *and any generated passphrase material has the required entropy for the selected parameter set.*

5.2 State and Deployment Assumptions

SIAP's cryptographic checks are insufficient by themselves to enforce replay resistance if persistent state can be rolled back. The following deployment assumptions are therefore required.

Assumption 6 (Durable monotonic state)**.** *After successful authentication, the advanced device key and server tag are committed durably and atomically before the released token is accepted by the relying application.*

Assumption 7 (Per-identity serialization)**.** *At most one authentication transition for a given* Kid *is processed at a time. Concurrent same-identity attempts are serialized by a lock, transaction isolation, or an equivalent mechanism.*

Assumption 8 (Server trust boundary)**.** *The server base key* K_{base} *remains secret from the adversary during the period for which SIAP authentication security is claimed. Compromise of* K_{base} *is treated as compromise of all devices under that server key.*

5.3 Adversary Classes

The analysis distinguishes five adversary classes. A_{net} observes, replays, delays, and injects protocol messages but does not control persistent state. A_{tok} obtains the sealed device key but not the passphrase. A_{db} obtains the server tag database but not K_{base} and not the passphrase. A_{srv} obtains K_{base} or full server execution access. $A_{rollback}$ can restore old device or server state unless the deployment prevents it through durable monotonic storage.

This classification is necessary because SIAP's properties do not all survive the same compromise. The sealed device key is intended to resist theft without the passphrase. The server tag is intended not to contain plaintext token leaves. The server base key is a root derivation secret and is outside the ordinary leakage model.

6 Security Definitions

6.1 Correctness

SIAP is correct if, for any honestly provisioned device and server tag at index $i < N$, authentication with the correct passphrase accepts, returns $\mathcal{T}[i]$, erases that leaf, advances the index to $i + 1$, and produces a new sealed state that is accepted by the next honest authentication.

6.2 Token Unforgeability

The token-unforgeability experiment gives $\mathcal{A}$ access to enrollment transcripts, stored device tags, and sealed device keys, subject to the restrictions that $\mathcal{A}$ does not know K_{base} and does not know the passphrase for the target device. $\mathcal{A}$ wins if it causes the verifier to accept at index i without presenting the correct leaf $\mathcal{T}[i]$ or without producing a sealed state that opens to a tree containing that leaf.

6.3 Replay Resistance

The replay experiment gives $\mathcal{A}$ a previously valid sealed device state and server tag at index i. The honest system then accepts at index i and commits index $i + 1$. $\mathcal{A}$ wins if it can replay the old state and obtain a second acceptance at index i. The definition assumes durable server-side state; otherwise the experiment reduces to a rollback failure outside the cryptographic construction.

6.4 Sealed-State Confidentiality

The sealed-state confidentiality experiment gives $\mathcal{A}$ the sealed device key and associated public metadata. $\mathcal{A}$ wins if it distinguishes the encrypted token tree from an encryption of a random tree of the same length, without knowledge of P_{hash} or the passphrase and without breaking RCS.

6.5 Passphrase Guessing Resistance

The passphrase experiment measures the probability that $\mathcal{A}$ recovers the correct passphrase or a valid P_{hash} after obtaining the sealed device key and server tag. Since passphrases are drawn from a human-selected or policy-selected distribution $\mathcal{D}_{pw}$, the bound is necessarily distributional. For q offline guesses, the attack probability is bounded by the cumulative probability mass of the tested guesses plus the probability of violating the SCB or cSHAKE assumptions.

7 Security Analysis

7.1 Correctness

Theorem 1 (Correctness)**.** *For an honestly generated device key, tag, server key, and passphrase, Algorithm 1 accepts for every index $i < N$ whose state has not been consumed, returns the leaf $\mathcal{T}[i]$, and advances the state to $i + 1$.*

Proof. By construction, enrollment sets Kid $=$ DID $\parallel i$ and stores a tag containing the same Kid. The passphrase procedure deterministically maps the correct passphrase to P_{hash} and then to P_{ver}, so verifier comparison succeeds. The same P_{hash}, D_{salt}, and SealInfo derive the RCS key and nonce used at enrollment, so RCS opening succeeds. The tree has not been modified except through honest transitions, so SHAKE$(\mathcal{T})$ equals the stored K_{hash}. The leaf at position i was generated as cSHAKE$(K_{\text{base}}; \text{Config}, \text{DID} \parallel i)$, which is exactly the verifier's expected token derivation. The algorithm then erases the leaf, increments i, recomputes the tag, and reseals under metadata containing the new Kid. Therefore the next state is well formed. □

7.2 Token Unforgeability

Theorem 2 (Token unforgeability)**.** *Let $\mathcal{A}$ be an adversary that does not know K_{base}, does not know the target passphrase, and cannot break RCS authentication. For token length λ, the probability that $\mathcal{A}$ causes acceptance at an unused index is bounded by*

$$\text{Adv}^{\text{forge}}_{\text{SIAP}}(\mathcal{A}) \leq \text{Adv}^{\text{cshake}}(\mathcal{A}') + \text{Adv}^{\text{aead}-\text{int}}_{\text{RCS}}(\mathcal{A}'') + q/2^{\lambda},$$

where q is the number of independent token guesses.

Proof. Acceptance requires a tree that opens under RCS with valid associated data, has a hash equal to the server-stored K_{hash}, and contains at the current index a leaf equal to cSHAKE$(K_{\text{base}}; \text{Config}, \text{Kid})$. If $\mathcal{A}$ modifies the ciphertext or associated metadata, successful acceptance without the correct key implies an RCS integrity forgery. If $\mathcal{A}$ supplies an unmodified honestly sealed tree but lacks the passphrase, it cannot open or alter the tree except by breaking sealed-state confidentiality or guessing the current leaf. If $\mathcal{A}$ attempts to predict the verifier's expected token, it must distinguish or predict the cSHAKE output keyed by K_{base}. In the random-function idealization, each independent guess succeeds with probability $2^{-\lambda}$. The union bound gives the stated result. □

7.3 Replay and Reuse Resistance

Theorem 3 (Replay resistance under durable state)**.** *Assume durable monotonic state and per-identity serialization. After a successful authentication at index i, a replay of the previously accepted state at index i is rejected except with probability bounded by the probability of breaking the hash commitment, AEAD integrity, or parser correctness assumptions.*

Proof. A successful authentication commits $DT' = (\text{DID} \parallel (i+1), K'_{\text{hash}}, P'_{\text{ver}})$. A replayed old device state contains Kid $=$ DID $\parallel$ i and is rejected by the key-identity comparison. If the adversary edits the key identity to $i+1$ without resealing correctly, RCS associated-data verification fails because SealInfo contains Kid. If the adversary also tries to substitute a tree matching K'_{hash}, it must either know the passphrase-derived sealing key or find a tree collision or second preimage for the stored K'_{hash}. Therefore replay is rejected under the stated assumptions. □

Remark 1. *This theorem is not true without the durable-state assumption. If both the device key and server tag can be rolled back to the old index, the cryptographic transcript is again locally consistent. Rollback prevention is therefore an operational requirement, not an automatic consequence of cSHAKE or RCS.*

7.4 Sealed-State Confidentiality

Theorem 4 (Token-tree confidentiality)**.** *For an adversary that obtains a sealed device key but not the passphrase, not P_{hash}, and not an RCS decryption oracle, the advantage in distinguishing the encrypted token tree from an encryption of a random tree is bounded by the IND advantage against RCS plus the advantage of deriving P_{hash} from the passphrase distribution.*

Proof. The RCS key and nonce are derived from P_{hash} and context containing D_{salt} and SealInfo. Without P_{hash}, the adversary's view of the ciphertext is that of the RCS confidentiality game. The only remaining path is to recover P_{hash} by guessing the passphrase and evaluating the seed-and-SCB transform, or by violating cSHAKE or SCB assumptions. □

7.5 Server-Database Leakage

The server tag stores Kid, K_{hash}, and P_{ver}. Kid identifies the current device and index. K_{hash} is a commitment to the current token-tree state but does not by itself reveal a leaf under the preimage resistance assumption. P_{ver} enables verification of a presented P_{hash} but is not the direct RCS sealing-key precursor. Consequently, disclosure of the tag database alone does not open the sealed device tree.

The tag database can still support offline passphrase checking if the adversary also knows enough context to evaluate candidate passphrases into candidate P_{ver} values. The effective resistance is then determined by the entropy of the passphrase distribution and the SCB work factor. This limitation is inherent to verifier-based password storage unless the protocol is redesigned as an online PAKE or uses a hardware-protected verifier.

7.6 Post-Quantum Security

SIAP uses symmetric primitives and hash-derived functions. In the generic quantum model, exhaustive search over a λ-bit uniformly random secret is reduced by Grover-type quadratic speedups, giving an approximate $2^{\lambda/2}$ search scale. This statement does not imply a proof of quantum security for the full implementation; it states that SIAP does not rely on integer factorization, discrete logarithms, or lattice assumptions. Its post-quantum posture is therefore governed by symmetric key lengths, sponge capacity, passphrase entropy, and implementation resistance to side channels.

8 Cryptanalytic Evaluation

8.1 Identity Binding and Domain Separation

The security of SIAP depends on unambiguous binding of each leaf to its identity and index. The leaf derivation uses DID $\parallel i$ under K_{base} and a configuration label. The sealing key uses $D_{\text{salt}} \parallel$ SealInfo under P_{hash}, where SealInfo includes the RCS seal label, current Kid, and expiration. The passphrase verifier uses a separate label and Kid. These separations prevent a value generated for one purpose from being reused as a valid value for another purpose, subject to cSHAKE's domain-separation behavior.

A conforming implementation should treat all labels and fixed encodings as normative. Changing a label, field order, length, or endian convention changes the derived values and therefore changes the protocol instance. Test vectors are necessary to make this property independently auditable.

8.2 Rollback and Desynchronization

Rollback is the dominant non-primitive threat. If the device and server are both restored to a prior consistent state, the cryptographic checks cannot distinguish that state from an honestly current state. A deployment that requires rollback resistance must therefore store the server index in monotonic durable state or use a recovery

protocol that refuses stale indices. The server can also maintain a separate highest-seen index or transaction counter for each device. Such a counter must be included in the commit protocol, not merely in logs.

Desynchronization is the dual problem. If the device state advances but the server tag does not, future authentication fails because Kid or K_{hash} no longer match. If the server tag advances but the device state does not, future authentication fails or rejects replay. A robust deployment should define a recovery interval and administrative re-provisioning procedure. Cryptographic acceptance should not silently resynchronize divergent states, because that would create an oracle for rollback attacks.

8.3 Concurrency

Two simultaneous authentications for the same device identity can both observe index i if no lock or transaction isolation is used. If both proceed to verification before either commits $i+1$, both may accept the same leaf. This is a state-machine race rather than a cryptographic failure. The correct mitigation is per-identity serialization: a mutex in a single-process implementation, a database row lock in a database-backed implementation, or a hardware monotonic counter in secure-token deployments.

8.4 Passphrase-Verifier Analysis

The repaired verifier design separates the value stored on the server from the value used to open the token tree. This improves factor separation under combined theft scenarios: disclosure of DT and DK does not directly provide the RCS key precursor. However, the server tag remains a verifier for offline passphrase candidates. The construction therefore inherits the usual limitations of password-verifier systems. Its effective resistance to offline attack depends on passphrase entropy and the cost of SCB evaluation.

The fixed-size seed derivation before SCB is necessary because memory-hard functions often have strict input contracts. A variable-length passphrase should be encoded and domain-separated before being supplied to a fixed-seed KDF interface. The seed is not a substitute for SCB; it is a canonicalization and domain-separation step that ensures the KDF receives input in the required format.

8.5 Malformed State and Parser Behavior

Cryptographic verification is reached only after serialized state has been parsed. A malformed state object can therefore become a memory-safety issue if the parser

assumes the caller supplied a complete buffer. SIAP requires runtime length checks for server keys, device keys, and device tags. Rejecting malformed encodings before copying fixed fields is part of the security boundary. This is also a conformance property: an implementation that uses only debug assertions for external input is not equivalent to one that validates at runtime.

8.6 Side Channels and Secret Erasure

Token comparison and verifier comparison must be constant time with respect to secret values. Passphrase hashing and RCS opening are inherently observable in time at a coarse level because wrong passphrases and malformed ciphertexts fail. The intended side-channel target is therefore not to make all failure paths identical, but to prevent comparison shortcuts and secret-dependent memory disclosure.

Transient values that require erasure include $Seed$, P_{hash} when stored in local buffers, RCS keys and nonces, decrypted token trees on failure, temporary tree hashes, and any released downstream token after use by the relying application. Erasure of the current leaf is a functional security operation, not merely a memory hygiene measure.

8.7 Denial of Service

SIAP has two denial-of-service surfaces. First, passphrase processing can be intentionally expensive. This is a desired property against offline guessing, but it should be rate-limited online. Second, a near-successful authentication attempt must not burn a leaf unless all authentication checks have passed. The authentication order in Algorithm 1 prevents failed token comparisons from consuming state. Implementations should also ensure that failed RCS opening does not leave plaintext tree material in memory and does not persist a partially mutated device key.

9 Parameter Analysis

9.1 Token Length

For $\lambda = 256$, direct guessing of an unused token has success probability 2^{-256} per independent attempt in the classical random-function model and approximately 2^{-128} security against idealized exhaustive quantum search. For $\lambda = 512$, these bounds scale to 2^{-512} and approximately 2^{-256} respectively. These are token-level bounds and do not measure passphrase security.

9.2 Tree Size

The default tree size is $N = 1024$ leaves. This is suitable for limited-use tokens, administrative ceremonies, or deployments with planned re-provisioning. High-frequency deployments require either a larger tree profile, a renewal protocol, or an external key-release profile that does not consume one full tree leaf per event. It is incorrect to infer long operational lifetimes for high-rate applications from the default $N = 1024$ profile.

9.3 SCB Cost Parameters

SCB parameters determine the cost of evaluating each passphrase guess. This paper does not assert a universal millisecond cost for the configured constants. Cost depends on platform, memory hierarchy, compiler options, implementation details, and selected parameter values. A deployment requiring resistance to large-scale offline guessing should benchmark its selected profile and bind the intended profile to provisioning and validation procedures.

9.4 Storage and Computation

The device tree requires $N\lambda/8$ bytes plus the RCS authentication tag and metadata. With $N = 1024$ and $\lambda = 256$, the plaintext tree is 32 KiB. This is acceptable for many embedded controllers but not for all smart-card class devices. Implementations targeting smaller secure elements may require a smaller tree, segmented storage, or a different tree-generation strategy. Such profiles should be specified separately because they change performance and possibly failure behavior.

10 Implementation Conformance and Test Requirements

A conforming implementation should provide test vectors for at least the following derivations: D_{salt}, $\mathcal{T}[i]$ for selected indices, $Seed$, P_{hash}, P_{ver}, SealInfo, $K_{rcs} \parallel N_{rcs}$, K_{hash}, and the full authentication transition from index i to $i + 1$. The vector set should include wrong-passphrase failure, metadata tampering, stale-index replay, malformed serialized inputs, tree exhaustion, and concurrent same-identity attempts.

Requirement	Conformance condition
Runtime parsing	Serialized state is rejected unless the input buffer length is sufficient for the encoded object.

Passphrase processing	The passphrase is first mapped to a fixed-size domain-separated seed before SCB invocation.
Verifier storage	The server tag stores P_{ver}, not the direct RCS sealing-key precursor.
Metadata binding	Kid and expiration are included in sealing metadata and authenticated by RCS.
Nonce uniqueness	The RCS key and nonce derivation depends on the current key identity or equivalent per-seal unique value.
Failure behavior	Failed authentication does not advance idx, erase a leaf, or persist partially decrypted state.
Success behavior	Successful authentication advances the index exactly once and commits DK' and DT' atomically.
Concurrency	Same-identity authentication transitions are serialized.
Secret erasure	Temporary passphrase hashes, RCS keys, decrypted trees, and consumed leaves are cleared according to policy.

11 Limitations

SIAP does not protect against compromise of K_{base}. If K_{base} is disclosed, the adversary can derive token leaves for all devices under that server key. SIAP also does not provide anonymity; the key identity is part of the operational state. SIAP is not a general authenticated key exchange and does not by itself authenticate a remote channel endpoint. It releases a symmetric token that another layer may use.

SIAP's replay resistance depends on durable monotonic state. A filesystem or database that permits silent rollback invalidates the replay proof. SIAP's passphrase security is limited by the entropy of the passphrase distribution and the actual SCB cost profile. Its forward secrecy is leaf-erasure forward secrecy: once a leaf is erased and no copy remains, later theft of the device state does not reveal that leaf. It is not forward secrecy against later compromise of K_{base}, because K_{base} can rederive every leaf.

12 Future Work

Future analysis should develop a machine-checkable model of the state machine, including crash points and commit ordering. A reference transaction layer should define exactly how DK' and DT' are committed, recovered, and audited. A conformance-vector suite should accompany the specification. Additional work should evaluate whether K_{hash} should be keyed or replaced by a cSHAKE-labeled

commitment, whether renewal should be specified as a first-class protocol operation, and whether high-rate profiles require segmented trees or a distinct token-expansion mechanism.

13 Conclusion

SIAP is a compact symmetric authentication and token-release construction whose security depends on both cryptographic primitives and disciplined state management. Its cryptographic core is straightforward: tokens are cSHAKE outputs under a server base key, the device tree is sealed under a passphrase-derived RCS key, metadata is authenticated as associated data, and each successful authentication consumes one leaf. The main security result is correspondingly precise. Under cSHAKE pseudorandomness, SHAKE commitment resistance, SCB passphrase-hardening assumptions, RCS AEAD security, nonce uniqueness, durable monotonic state, and per-identity serialization, SIAP provides token unforgeability, replay resistance, and sealed-state confidentiality within the stated symmetric-security model.

The analysis also identifies the limits of the design. SIAP does not survive server base-key disclosure, does not provide rollback resistance without durable state, and does not turn low-entropy passphrases into high-entropy secrets. These limitations do not invalidate the construction, but they define its correct deployment boundary. Within that boundary, SIAP is best understood as a deterministic, audit-friendly, post-quantum symmetric authentication primitive for systems that need local or offline release of one-time cryptographic tokens.

References

[1] John G. Underhill. *Secure Infrastructure Access Protocol - SIAP Specification*. Quantum Resistant Cryptographic Solutions Corporation, 2025. https://www.qrcscorp.ca/documents/siap_specification.pdf.

[2] QRCS Corporation. *SIAP Reference Implementation*. Quantum Resistant Cryptographic Solutions Corporation, 2025. https://github.com/QRCS-CORP/SIAP.

[3] QRCS Corporation. *QSC Cryptographic Library*. Quantum Resistant Cryptographic Solutions Corporation, 2025. https://github.com/QRCS-CORP/QSC.

[4] National Institute of Standards and Technology. *FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. 2015. https://doi.org/10.6028/NIST.FIPS.202.

[5] John Kelsey, Shu-jen Chang, and Ray Perlner. *NIST SP 800-185: SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash*. National Institute of Standards and Technology, 2016. https://doi.org/10.6028/NIST.SP.800-185.

[6] Elaine Barker and John Kelsey. *NIST SP 800-90A Revision 1: Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. National Institute of Standards and Technology, 2015. https://doi.org/10.6028/NIST.SP.800-90Ar1.

[7] Paul A. Grassi, James L. Fenton, Elaine M. Newton, Ray A. Perlner, Andrew R. Regenscheid, William E. Burr, Justin P. Richer, Naomi B. Lefkovitz, Jamie M. Danker, Yee-Yin Choong, Kristen K. Greene, and Mary F. Theofanos. *NIST SP 800-63B: Digital Identity Guidelines: Authentication and Lifecycle Management*. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-63b.

[8] David M'Raihi, Mihir Bellare, Frank Hoornaert, David Naccache, and Ohad Ranen. *HOTP: An HMAC-Based One-Time Password Algorithm*. RFC 4226, 2005. https://www.rfc-editor.org/rfc/rfc4226.

[9] David M'Raihi, Johan Rydell, Mingliang Pei, Salah Machani, and OATH. *TOTP: Time-Based One-Time Password Algorithm*. RFC 6238, 2011. https://www.rfc-editor.org/rfc/rfc6238.

[10] Leslie Lamport. Password Authentication with Insecure Communication. *Communications of the ACM*, 24(11):770–772, 1981. https://doi.org/10.1145/358790.358797.

[11] Phillip Rogaway. Authenticated-Encryption with Associated-Data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002. https://web.cs.ucdavis.edu/~rogaway/papers/ad.pdf.

[12] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In *Advances in Cryptology - CRYPTO 1996*. https://doi.org/10.1007/3-540-68697-5_24.

[13] Colin Percival. Stronger Key Derivation via Sequential Memory-Hard Functions. BSDCan, 2009. https://www.tarsnap.com/scrypt/scrypt.pdf.

[14] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *IEEE Symposium on Security and Privacy*, 2012. https://doi.org/10.1109/SP.2012.44.