Independent Submission
Internet-Draft
Intended status: Experimental
Expires: 1 December 2026

H. Chang
Saki Studio

30 May 2026

SakiAgentSSH Secure Protocol Specification
draft-sakistudio-sass-02

Abstract

This document describes the Saki Agent Secure Stream (SASS) protocol, version 1.4. SASS is an application-layer overlay protocol for authenticated remote command execution, streaming process I/O, and binary file transfer between trusted agents.

To ensure strict self-containment and compatibility with IETF standard specifications, SASS defines a decoupled "Control-Transport Decoupling" architecture. The SASS Core defines an abstract SASS Abstract Messaging Model (SAMM) utilizing standard CBOR (RFC 8949) and JSON as baseline serializations.

Following the precedent of TLS 1.0 (RFC 2246) inheriting and refining SSL 3.0, SASS v1.4 formalizes its security evolution through four major incremental milestones: Active Threat Defense (v1.1), Forward-Secure Audit Hash Chains (v1.2), modular Control-Transport Decoupling (v1.3) incorporating tls-exporter Channel Binding (RFC 9266) and Zero-Allocation Tarpit streams, and Total Response Mapping (v1.4) with 6-Response state machine convergence and Safety Gradient loss bounding.

SASS v1.4 achieves the Version Dominance milestone: a comparative claim between protocol versions based on Second-order Stochastic Dominance (SSD) [RS1970]. yielding strict SSD improvement.

Copyright Notice

Table of Contents

1. Introduction & Protocol Evolution

The proliferation of autonomous AI-powered coding agents operating on remote machines introduces a critical threat model: the Rogue Agent. Unlike traditional SSH clients controlled by human operators, agents may autonomously execute destructive commands, exfiltrate credentials, or pivot laterally across networks without explicit human authorization.

Existing remote execution protocols such as SSH [RFC4253] were designed for human-operated terminals and lack the fine-grained capability controls, active defenses, and binary-safe encoding schemes required for agent management.

SASS (Saki Agent Secure Stream) decouples the logical message flow from the physical transmission layer. By defining a transport-agnostic messaging core alongside modular transport profiles, SASS achieves strict self-containment, freeing the standard from proprietary third-party binary frameworks (e.g., gRPC/Protobuf) during academic review, while retaining high-performance implementations as pluggable adapters.

SASS is designed to avoid over-deployment or under-deployment at any application scale. The protocol is suitable for deployment on resource-constrained IoT devices as well as enterprise-grade servers. However, this document does not address quantum-safe cryptographic agility, which remains an active area of research and standardization beyond the scope of this specification.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in capitalized, as shown here.

1.1. The Four Incremental Milestones (v1.1 to v1.4)

Following the precedent of TLS 1.0 (RFC 2246) inheriting and refining SSL 3.0, the SASS specification formalizes its security evolution through four major incremental milestones:

* SASS v1.1 (Active Threat Countermeasures): Added the 13Policy heuristic firewall and cryptographic cognitive challenges bound to an active Tarpit system, establishing the foundation of active defense against rogue automated operations.

* SASS v1.2 (Cryptographic Integrity & Audit): Introduced five-dimensional Capability ACLs combined with a forward-secure hash chain audit log signed via host-resident asymmetric keys, providing mathematical proof of non-repudiation.

* SASS v1.3 (Control-Transport Decoupling & Hardening): Decoupled the SASS Core from underlying physical protocols. Introduced atomicity controls (POSIX openat and O_NOFOLLOW) to pre-empt TOCTOU sandstrike breakouts, implemented Zero-Allocation Tarpit streams to neutralize Host DoS, and mandated exported keying material Channel Binding to prevent session hijacking.

* SASS v1.4 (Total Response Mapping & Loss Bounding): Introduces a formal 6-Response state machine (R1~R6) that maps every possible Agent behavior to one of six deterministic responses, each preserving storage integrity and bounding loss. Adds Dual Standard Enforcement (Vi Swap for authenticated agents, Tarpit for unauthenticated), Transparent Branching for zero-loss write isolation, PTY Ring Buffer for idempotent reconnection, and the Safety Gradient theory for layered loss bounding.

  This milestone achieves the Version Dominance property: a comparative claim between protocol versions. Each version iteration eliminates specific behavioral branches from the agent probability space while maintaining expected loss, yielding strict Second-order Stochastic Dominance (SSD) improvement [RS1970].

1.2. Design Philosophy: Total Response Mapping

Traditional security models enumerate known attacks and block them (blacklist model). This approach is inherently incomplete: the attacker can always find a path not in the blacklist.

SASS v1.4 inverts this model. Instead of defining "which behaviors are bad," it defines "for every possible behavior, what is the response." The set of responses is finite, deterministic, and auditable. Any unforeseen behavior is mapped to one of the predefined responses.

This is the formal meaning of the axiom: "All unexpected behaviors are expected behaviors."

Furthermore, an Agent's actions have no inherent "danger" or "malice." The boundary enforcement system is strictly an Adjudicator: it determines what is permitted ("CAN") and what is prohibited ("CANNOT") based on the Agent's capability set. If an Agent's authorized boundary includes executing a destructive command, the Daemon MUST execute it without prejudice. Conversely, if an Agent lacks authorization for a benign command, this constitutes an absolute boundary violation.

2. Terminology

Agent An autonomous software process that connects to a SASS Daemon to execute commands or transfer files on the remote host.

Daemon The SASS server process that listens for incoming agent connections, authenticates them, and executes authorized commands.

Rogue Agent An agent that has been compromised, misconfigured, or is otherwise attempting to perform unauthorized operations.

Session A time-bounded, authenticated context between an agent and a daemon, identified by a UUID and constrained by a capability set.

SAMM SASS Abstract Messaging Model, the transport-agnostic message semantic layout.

Transport Profile A modular plugin implementing the transmission details (e.g., gRPC, WebSockets, or raw TCP).

UVSF Userspace Virtual Symlink Filesystem, a zero-permission sandbox based on symlink trees.

KFS/WSDK Kernel-level Filtered Storage, proprietary OS-level driver plugins (future work).

13Policy The heuristic threat detection engine that classifies command patterns and triggers boundary enforcement responses.

Cognitive Challenge A cryptographic puzzle that requires the agent to prove possession of keying material and genuine computational capability. An implementation MAY use the approach described in Appendix C.1.

Tarpit An active defense mechanism that streams high-entropy data to a suspected rogue agent, exhausting its resources with O(1) daemon memory cost.

Vi Swap An active defense mechanism that traps an authenticated

Agent in a simulated interactive terminal state, causing the Agent's LLM to halt generation.

Total Response Mapping A security model in which every possible input from the Agent domain maps deterministically to one of a finite set of predefined responses (R1~R6).

Safety Gradient The property that each protocol layer bounds the worst-case loss if all layers above it are compromised.

Transparent Branching An isolation mechanism invisible to the Agent, where all write operations are redirected to a discardable branch directory.

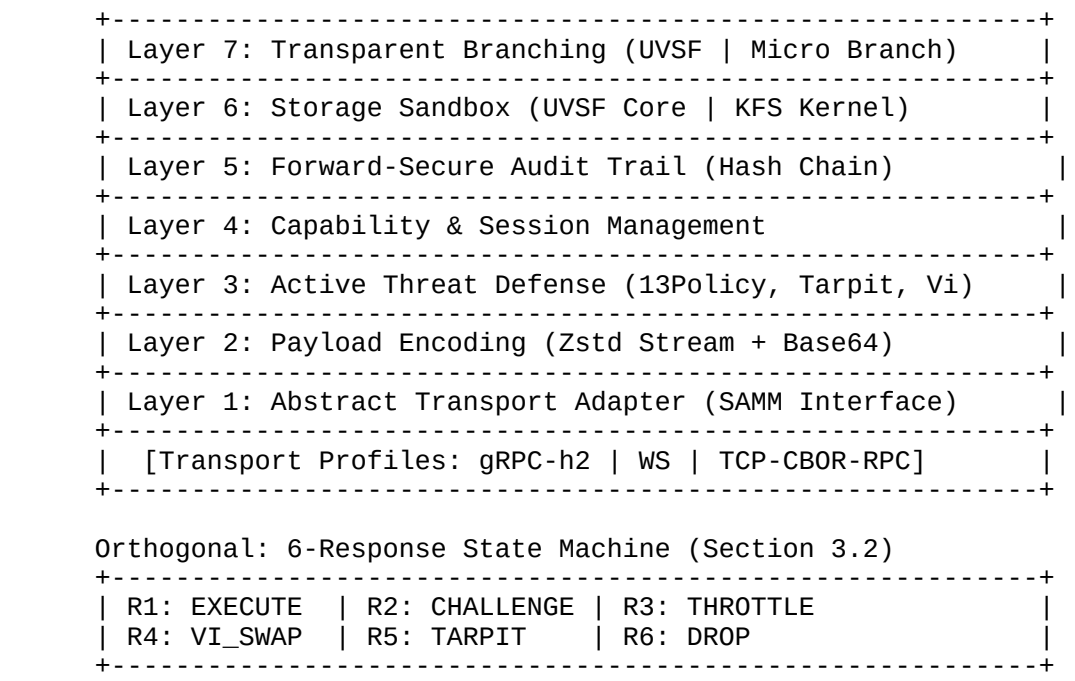
2.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in capitalized, as shown here.

3. Protocol Overview

3.1. Architecture

SASS separates the control plane and data transmission:



3.2. 6-Response State Machine (R1~R6)

All possible Agent behaviors, after evaluation through the SASS multi-layer protocol stack, MUST converge to exactly one of the following six responses:

Code	Name	Definition
R1	EXECUTE	Normal execution. Record to audit log. Writes pass through Transparent Branching.
R2	CHALLENGE	Trigger cognitive challenge. Prove computational capability, then execute.
R3	THROTTLE	Quota exceeded. Enqueue and wait.
R4	VI_SWAP	Trap authenticated Agent in simulated interactive terminal state.
R5	TARPIT	Consume attacker resources via slow-drip high-entropy data. Cost externalized.
R6	DROP	Immediate connection termination. Zero allocation, zero response.

Table 1

Every response R1 through R6 MUST satisfy the following invariants:

Property	R1	R2	R3	R4	R5	R6
Storage loss	0*	0	0	0	0	0
Commercial loss	0*	0	0	0	Ext.	0
Auditable	Yes	Yes	Yes	Yes	Yes	Yes
Daemon memory cost	O(n)	O(1)	O(1)	O(1)	O(1)	O(0)

Table 2

(*) R1 storage loss is bounded to zero by Transparent Branching (Section 8.5). All writes execute within a discardable branch; merge requires explicit human review.

State Machine Flow:

```
ExecuteRequest enters
    |
    +-- L1: ACL ---- IP not in whitelist? ------> R6
    |
    +-- L2: Transport - Channel invalid? -------> R6
    |
    +-- L3: Auth ---- Not authenticated? -------> R6
    |                 Authenticated but expired?-> R2
    |
    +-- L4: Capability -- Command denied? ------> R4
    |                     Path denied? ----------> R4
    |
    +-- L5: 13Policy ---- Dangerous command?
    |                       critical -----------> R5
    |                       high ---------------> R2
    |                       medium -------------> R2
    |                       low -> R1 + Enhanced Audit
    |
    +-- L6: Quota ---- Quota exhausted? --------> R3
    |
    +-- L7: Watchdog - Timeout? -> SIGKILL+Audit
    |
    +-- All checks passed ---------------------> R1
            |
            +-- write op? -> Branching (8.5)
            +-- cache I/O? -> volatile (App C.7)
            +-- localhost? -> obfuscation (8.4)
```

In this state machine, "unexpected behavior" has no dedicated handler because every node already handles it. Regardless of how unusual the Agent’s behavior may be, it MUST converge to one of R1~R6, each of which guarantees storage safety.

3.3. Abstract Messaging Model (SAMM)

SAMM defines SASS messages (e.g., ExecuteRequest, StreamResponse) in a serialization-neutral semantic layout. To ensure standardized self-containment, the baseline serialization MUST be CBOR [RFC8949] or JSON. Standard SAMM objects map fields onto distinct logical types (bytes, strings, maps) described in Section 7.

3.4. Relationship to SSH

SASS shares NO wire format, key exchange mechanism, channel multiplexing, or subsystem architecture with the SSH protocol suite [RFC4251] [RFC4252] [RFC4253] [RFC4254].

The internal development codename "SakiAgentSSH" is a historical artifact. Implementations MUST NOT advertise SSH protocol version strings, listen on TCP port 22, or respond to SSH client probes.

OpenSSH and similar implementations of the SSH protocol suite [RFC4251] are human-oriented, real-time interactive POSIX interfaces. Their design inherently produces unexpected I/O patterns, garbage-typed timing errors, and non-deterministic terminal state when operated by autonomous Agent runtimes. These failure modes appear as protocol-level defense artifacts within SASS (see Section 8.3.1, Vi Swap).

This document’s supplementary position is that for Agent Runtime environments, the continued use of SSH-family protocols as defined in [RFC4251] and related specifications is deprecated. The properties that make SSH excellent for human operators — synchronous interactive I/O, standard ASCII bus semantics, and time-dependent keepalive — are precisely the properties that create exploitable attack surfaces in Agent deployments.

Conversely, RPC-based transports do not automatically sever handshakes when packets are not sent according to I/O timing dependencies. All time-dependent protocols that properly maintain sessions per this specification are well-suited for Agent Runtime use.

4. Transport Layer & Transport Profiles

4.1. Transport Decoupling Principles

SASS Daemons and Clients interact via a TransportAdapter interface. Implementations MAY choose any conformant Transport Profile, ensuring cross-platform adaptability from low-end microcontrollers (TCP/CBOR) to enterprise jump servers (gRPC/mTLS).

4.2. SASS-over-gRPC Profile (Mandatory TLS 1.3)

The SASS-over-gRPC profile is the default enterprise-grade pluggable transport. It maps SAMM messages onto Protocol Buffers and HTTP/2 streams using ALPN "x-sakirpc-v5".

All SASS connections under this profile MUST use TLS 1.3 [RFC8446]. Downgrades to TLS 1.2 are STRICTLY FORBIDDEN. Implementations MUST support the following cipher suites:

* TLS_AES_256_GCM_SHA384

* TLS_CHACHA20_POLY1305_SHA256

The maximum gRPC message size MUST be configured to at least 52,428,800 bytes (50 MiB) to accommodate Tarpit countermeasure payloads.

4.3. ALPN & TCP Packet-Splitting Mitigation

During the TLS handshake, both endpoints MUST include the ALPN extension [RFC7301]. The ALPN protocol identifier is "x-sakirpc-v5".

To mitigate firewall-level packet splitting or ALPN stripping, daemons MUST inspect the Content-Type header on incoming HTTP/2 headers. If it matches "application/grpc+saki" and ALPN was stripped or spoofed, the daemon MUST drop the TCP connection immediately to prevent cross-protocol multiplexing attacks.

4.4. Channel Binding via Exported Keying Material

When using a TLS 1.3 transport profile, a conforming implementation MUST bind session-layer operations to the underlying TLS connection to prevent session hijacking and cognitive challenge replays.

The binding mechanism MUST use Exported Keying Material (EKM) derived from the TLS session. The EKM value MUST be incorporated into the cognitive challenge response (Section 8.2) to ensure challenges cannot be relayed across TLS connections.

A conforming implementation using [RFC5705] or [RFC9266] tls-exporter SHOULD derive the binding value as follows:

Label: implementation-defined (see Appendix C.2)
Context: Session UUID (16 bytes)
Length: 32 bytes

The specific label, derivation algorithm, and incorporation method are implementation-defined. An implementation MAY use the approach described in Appendix C.2.

4.5. Default Port

The default listening port for SASS daemons is TCP 19284. This port is configurable.

5. Session Layer

5.1. Agent Authentication

SASS authentication proceeds in three phases:

Phase 1: Transport Identity The TLS handshake establishes transport-level identity. CN in client certificates provides CN-based authentication if mTLS is enabled.

Phase 2: Asymmetric Key Challenge-Response The agent calls the Authenticate RPC with its agent_name, public_key, nonce, and a signature over the nonce. The signing algorithm MUST be a standardized asymmetric signature scheme. Verified agents receive a session_id (UUID v4) and the capability set hash. An implementation MAY use ED25519 [RFC8032] as the signing algorithm.

Phase 3: Cognitive Challenge Suspected agents are challenged via a cryptographic puzzle that requires genuine computational capability. The challenge mechanism MUST be bound to the TLS session via exported keying material (Section 4.4). An implementation MAY use ChaCha20-Poly1305 as described in Appendix C.1.

5.2. Session Lifecycle

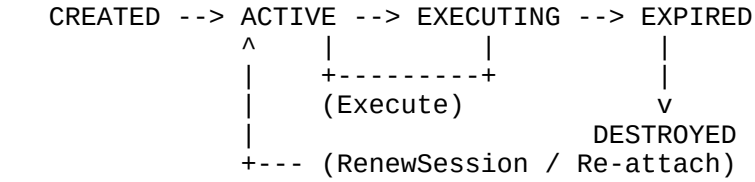
Sessions are time-bounded and identified by UUID v4. The daemon MUST enforce the following constraints:

* Maximum session duration: configurable, default 3600 seconds

* Maximum concurrent sessions: configurable, default 10

* Session renewal: via RenewSession RPC, extends expires_at

Session identifiers are transmitted in the gRPC metadata header "x-agentssh-session-id" on every authenticated RPC call.



Zombie sessions (disconnected and beyond TTL) MUST be periodically cleaned by the Daemon to prevent resource exhaustion.

5.3. Capability-Based Access Control

Each authenticated agent is assigned a five-dimension capability set that constrains its operations:

1. allowed_commands / denied_commands: Command whitelists/ blacklists.

2. allowed_paths / denied_paths: Filesystem path limits.

3. max_concurrent: Maximum simultaneous processes.

4. timeout_seconds: Maximum command execution time.

5. max_file_size_bytes: Maximum file transfer size.

The daemon MUST check denied patterns before allowed patterns (deny-first). If any denied pattern matches, the request triggers R4 (VI_SWAP for authenticated agents) regardless of allowed patterns. An implicit deny applies when no pattern matches.

To resist TOCTOU (Time-of-Check to Time-of-Use) symlink attacks in Userspace, the Storage Sandbox (UVSF) MUST enforce File Descriptor (FD) relative path operations (openat(2)) carrying O_NOFOLLOW and O_CLOEXEC flags.

6. Payload Encoding & Safety Gates

6.1. Zstd Streamed Decompression Limit

Command payloads are compressed using Zstandard [RFC8878] and Base64 encoded. To resist Decompression Bombs (Zip Bombs), the daemon MUST limit decompressed payloads using streaming decompression.

6.2. Decompression Bomb & Huffman Collision Mitigation

MAX_DECOMPRESSED_PAYLOAD MUST be strictly capped at 5 MiB. Exceeding this limit triggers ERROR_DECOMPRESSION_LIMIT_EXCEEDED (55) and immediately severs the connection.

To prevent Huffman Code Collision CPU exhaustion, the decoder MUST enforce a maximum 50ms time window on header parsing.

6.3. Encoding & Decoding Procedure

Sender Procedure:

1. Serialize command arguments into a structured array (JSON/ MsgPack).

2. Compress using Zstandard [RFC8878] at level 3.

3. Encode using Base64 [RFC4648] standard alphabet.

4. Place the resulting byte string into the ExecuteRequest raw_payload field.

Receiver Procedure:

1. Base64-decode the raw_payload.

2. Stream-decompress via Zstd, checking the 5 MiB safety limit.

3. Deserialize arguments and pass them directly to the OS process creation API (execve, CreateProcessW) WITHOUT intermediate shell interpretation.

6.4. PTY Ring Buffer & Offset Resumption

SASS v1.4 introduces a Ring Buffer mechanism for PTY output, enabling idempotent (safe-to-retry) reconnection after transport disruption.

6.4.1. Ring Buffer Specification

Each Session MUST maintain Ring Buffers for stdout and stderr streams with the following properties:

* Capacity: implementation-defined, RECOMMENDED 1 MiB

* Overflow policy: Drop oldest (FIFO), prevent OOM

* Offset tracking: monotonically increasing 64-bit counter (total_written), never wraps

6.4.2. Offset Field in Stream Messages

Every stream response message MUST include an offset field representing the byte position of the first byte in the data payload within the Ring Buffer’s logical address space.

The Client MUST track the highest received offset + len(data) to use as resume_offset upon reconnection.

6.4.3. Reconnection Protocol

To reconnect after transport disruption:

1. Client sends an ExecuteRequest with is_reattach=true and the original session_id plus resume_offset.

2. Daemon looks up the session, reads the Ring Buffer from resume_offset, and resumes streaming.

3. If resume_offset is older than the Ring Buffer's oldest available data, the Daemon MUST return an error indicating data loss, including the oldest available offset.

The reconnection protocol is idempotent: sending the same request with the same resume_offset always produces the same result.

7. Logical RPC Service Semantics

7.1. Command Execution & Streaming

```
rpc Execute(ExecuteRequest)
    returns (ExecuteResponse);
rpc ExecuteStream(ExecuteRequest)
    returns (stream StreamResponse);
```

Execute executes synchronously. ExecuteStream streams stdout/stderr in real-time. Each StreamResponse contains source (STDOUT/STDERR/SYSTEM), data, exit_code (only in the stream's final message), and offset (for Ring Buffer resumption per Section 6.4).

A SYSTEM source indicates daemon-generated messages such as queue notifications or authentication events.

The Daemon MUST NOT spawn a login shell. Commands are executed via OS process creation APIs with explicit argument arrays, preventing shell expansion attacks. The Daemon MAY allocate a PTY when the command requires terminal capabilities, but MUST NOT invoke a shell interpreter to wrap the command.

7.2. Process Management

```
rpc Cancel(CancelRequest) returns (CancelResponse);
rpc Signal(SignalRequest) returns (SignalResponse);
```

Cancel terminates the process immediately (SIGKILL). Signal sends POSIX signals. On Windows, SIGINT maps to CTRL_C_EVENT, and SIGTERM/SIGKILL map to TerminateProcess.

7.3. File Transfer & Raw File Transfer

```
rpc FileUpload(stream FileChunk)
    returns (FileTransferResponse);
rpc FileDownload(FileDownloadRequest)
    returns (stream FileChunk);
rpc RawFileTransfer(stream RawFileChunk)
    returns (RawFileTransferResponse);
```

Standard file transfer uses streaming chunks. RawFileTransfer bypasses shell I/O entirely; the daemon opens a raw file descriptor and writes decoded (Zstd+Base64) chunks to guarantee bit-perfect copies across different platforms.

8. Threat Defense & Boundary Enforcement

8.1. Boundary Adjudicator (13Policy Engine)

The 13Policy engine is a heuristic boundary adjudicator. It matches command strings against patterns in a configuration file and triggers the appropriate Response (R1~R6) based on severity.

An Agent’s actions are fundamentally non-special; they lack inherent "danger" or "malice." The 13Policy engine is strictly a Boundary Adjudicator: it enforces what is permitted ("CAN") and what is prohibited ("CANNOT") based on the Agent’s capability set and the policy rules.

Each rule specifies:

* pattern: Glob pattern to match against command + args
* action: deny (triggers R4/R5), challenge (R2), allow (R1)
* severity: critical, high, medium, low
* description: Human-readable explanation

Severity	action: deny	action: challenge
critical	R5 (TARPIT)	R2 (CHALLENGE)
high	R4 (VI_SWAP) (*)	R2 (CHALLENGE)
medium	R4 (VI_SWAP) (*)	R2 (CHALLENGE)
low	R1 + audit	R1 + audit

Table 3

(*) R4 for authenticated Agents; unauthenticated Agents receive R5.

Implementations MUST ship with a default rule set of at least 50 patterns.

8.2. Cognitive Challenge Mechanism

A conforming implementation MUST provide a cognitive challenge mechanism that satisfies the following requirements:

1. The challenge MUST require genuine computational capability (not solvable by string manipulation alone).

2. The challenge response MUST be bound to the current transport session via exported keying material (Section 4.4) to prevent relay attacks.

3. The challenge MUST have a configurable time-to-live (default 60 seconds).

4. Challenge verification MUST use constant-time comparison to prevent timing side-channels.

5. Challenge failure MUST trigger R5 (TARPIT).

The specific cryptographic algorithm used for the challenge is implementation-defined. An implementation MAY use ChaCha20-Poly1305 as described in Appendix C.1.

8.3. Dual Standard Enforcement

Subjecting all boundary violations to Tarpit containment leads to severe financial token loss when the violator is an authenticated Agent (e.g., paid AI API endpoints). Therefore, Daemons MUST implement a Dual Standard Enforcement policy based on Agent identity:

* Authenticated Agents violating boundaries --> R4 (VI_SWAP)

* Unauthenticated connections / failed challenges --> R5 (TARPIT)

8.3.1. Vi Swap Defense (Authenticated Agents)

If an Agent has successfully authenticated but executes an out-of-bounds command, the Daemon MUST NOT simply reject the request or return an error string. Instead, the Daemon MUST simulate an interactive terminal blockage on the output stream.

The mechanism works by sending terminal control sequences that cause the Agent’s LLM to recognize a blocked interactive state and halt its generation, effectively stopping the violation with minimal computational overhead for the Daemon.

An implementation MAY use ANSI escape sequences to simulate a vi(1) editor state as described in Appendix C.5.

Vi Swap MUST hold the session open for a configurable duration (default 3600 seconds), during which the Agent's execution slot is occupied, preventing further commands.

Vi Swap MUST be activated when ALL of the following are true:

* The Agent has a valid, non-expired session (authenticated)

* The Agent's command or path matches a denied pattern, OR

* The Agent's command matches a 13Policy rule with action: deny

Vi Swap MUST NOT be activated for unauthenticated connections.

8.3.2. Zero-Allocation Tarpit (Unauthenticated)

For unauthenticated connections, invalid signatures, or failed cognitive challenges, the Daemon deploys the Zero-Allocation Tarpit.

The Tarpit MUST satisfy the following constraints:

* Daemon memory per connection: O(1) (constant, not proportional to payload size)

* Total payload: RECOMMENDED 40 MiB (exceeds typical LLM context window)

* Delivery rate: slow-drip with inter-chunk delay to maximize time occupation

* Concurrency gate: maximum concurrent Tarpit sessions MUST be capped (RECOMMENDED 32) to prevent the Tarpit itself from becoming a DoS vector against the Daemon

* When the concurrency gate is full, new rogue connections MUST receive R6 (DROP)

An implementation MAY use a static pre-allocated buffer as described in Appendix C.3.

8.3.3. Zero-Window Deadlock Defense

To prevent TCP Zero-Window socket lockout attacks where a malicious Agent sets its TCP receive window to zero, the Daemon MUST enforce a strict send timeout (RECOMMENDED 3 seconds) per Tarpit chunk. If a send exceeds this timeout, the connection MUST be terminated to prevent socket descriptor leakage.

8.4. LocalHost Defense

Unauthenticated connections originating from the loopback interface (127.0.0.1 / ::1) constitute a boundary violation via local IPC scraping.

Instead of immediately dropping the connection, the Daemon MUST employ obfuscation: it executes the requested read commands but applies an obfuscation mask over the response payload. This feeds the unauthorized local process structurally valid but semantically meaningless data, overflowing its LLM context window with noise.

Additional deception mechanisms SHOULD include:

* Storage Spoofing: Report false disk usage
* Memory Spoofing: Return fabricated memory statistics
* Slow Denial: Artificial latency on unauthenticated responses

8.5. Transparent Branching (Storage Isolation)

Transparent Branching ensures that R1 (EXECUTE) operations do not directly modify the host filesystem. All writes are redirected to a per-session branch directory that can be discarded or merged after human review.

A conforming implementation MUST provide a mechanism that satisfies the following requirements:

* The Agent MUST NOT be aware that it is operating in a branch.
* Reads MUST access the real underlying files.
* Writes MUST be captured in the branch directory, not applied to the original filesystem.
* The branch MUST be discardable (drop) without affecting the original filesystem.

* The branch MUST be mergeable (apply) to the original filesystem after human review.

* High-volume I/O SHOULD be redirected to volatile storage to keep the branch diff minimal.

An implementation MAY use the symlink tree approach described in Appendix C.6 and the volatile cache redirection described in Appendix C.7.

The following directories SHOULD be excluded from branching:

* Version control metadata (.git/)

* Build artifacts (target/, node_modules/)

* System caches

This is the mechanism by which R1 (EXECUTE) achieves "storage loss = zero." The branch itself serves as forensic evidence: humans can review the branch diff at any time and decide to merge or discard.

Property	Traditional Sandbox	Transparent Branching
Agent awareness	Detectable	Invisible
Functionality	Restricted	Full
Isolation	Kernel (cgroup, etc.)	Userspace
Privilege	Root / elevated	None
Recovery	Reset container	Drop or merge branch
Cross-platform	Linux-only (usually)	All major OS
User acceptance	Low (rejected)	High (invisible)

Table 4

9. Error Codes

SASS defines structured error codes:

* 1-9: ACL / Authentication

* 10-19: Execution
* 20-29: File Transfer
* 30-39: Configuration
* 40-49: TLS
* 50-59: Capability
* 60-69: Session
* 70-79: Agent Key Authentication
* 80-89: Threat Defense
* 90-99: LocalHost Defense
* 100-109: Total Response Mapping (v1.4)

New error codes in SASS v1.4:

* 100: RESPONSE_CHALLENGE_REQUIRED (R2)
* 101: RESPONSE_THROTTLED (R3)
* 102: RESPONSE_VI_SWAP_ENGAGED (R4)
* 103: RESPONSE_TARPIT_ENGAGED (R5)
* 104: RESPONSE_DROPPED (R6)
* 105: RING_BUFFER_DATA_LOSS (resume too old)

10. Security Considerations

10.1. Total Response Mapping Guarantee

The security model of SASS v1.4 is built on a single axiom: for every possible Agent behavior, the daemon produces exactly one of six predefined responses (R1~R6), each of which preserves storage integrity and bounds loss.

The undecidability of semantic program properties (Rice's Theorem, 1951 [Rice1953]) implies that no static analysis can determine whether an arbitrary Agent command sequence is 'safe.' SASS addresses this fundamental limitation not by attempting to decide safety, but by ensuring that every possible behavior maps to a bounded response.

This is a departure from traditional security models that attempt to enumerate and block known attacks. The Total Response Mapping model provides the following guarantees:

1. Completeness: The state machine in Section 3.2 covers every possible code path. There is no "else" branch that leads to an undefined state.

2. Determinism: Given the same input and the same daemon configuration, the same response is always produced.

3. Storage Safety: No response (R1~R6) results in unrecoverable modification to the host filesystem.

4. Loss Bounding: R1~R4, R6 produce zero loss; R5 externalizes commercial loss to the attacker.

5. Auditability: Every state transition is logged to a forward-secure audit chain (Section 10.3).

10.2. Safety Gradient (7-Layer Loss Bounding)

Single-layer defense is inherently imperfect. SASS does not claim any single layer is unbreakable. Instead, layers form a Safety Gradient: each layer bounds the worst-case loss if all layers above it are compromised.

```
Layer 7: Transparent Branching + VFS Diff
Layer 6: Watchdog + Quota
Layer 5: 13Policy (command classification -> R1~R5)
Layer 4: Capability Model (five-dimensional -> R4)
Layer 3: Session Auth (application-layer identity)
Layer 2: TLS 1.3 + EKM Binding (transport + binding)
Layer 1: ACL (CIDR whitelist, first-packet -> R6)
Layer 0: Shell-less Execution (explicit args)
```

Layer	If breached, attacker gains	Maximum loss
L1	Can reach transport	Zero (L2 TLS)
L2	Has encrypted channel	Zero (L3 auth)
L3	Has valid session	Cap-bounded
L4	Executes beyond cap	Branch-bounded
L5	Bypasses cmd class	Watchdog-bound
L6	Tarpit/Quota fail	Audit-bounded
L7	Audit compromised	Apocalyptic (*)

Table 5

(*) Mitigated by cryptographic hash chain + external anchoring. An implementation MAY use ED25519 as described in Appendix C.4.

10.3. State Transition Auditing

Every state transition in the 6-Response state machine MUST be recorded in an append-only audit log with the following integrity guarantees:

1. Hash Chain: Each audit record MUST include a hash computed over the previous record’s hash, the current event data, and the timestamp.

2. Cryptographic Signature: Each audit record MUST be signed by the daemon’s private key.

3. Non-repudiation: The combination of hash chain and signature provides non-repudiable evidence that events occurred in the recorded order.

SASS audits state transitions, not command strings.

Traditional Audit	SASS Audit
Records "what" happened	Records "what", "why", "which R"
Evidence: disputable	Evidence: deterministic
Tamper resistance: low	Tamper: hash chain + signature
Retrospection: limited	Retrospection: full causal chain

Table 6

An implementation MAY use ED25519 signatures with SHA256 hash chains as described in Appendix C.4.

10.4. Transport-Layer Considerations

* ALPN Packet Splitting and Stripping: Checked via Content-Type cross-verifications (Section 4.3).

* Decompression Huffman Tree CPU DoS: Header parsing is limited to a maximum 50ms window (Section 6.2).

* TCP Zero-Window Socket Lockout: Solved via send timeout (Section 8.3.3).

* TLS 1.3 0-RTT Replay Attacks: Mitigated via exported keying material binding (Section 4.4).

* Forward-Secure Audit Trail Invalidation: Mandates public key one-way push and external TSP [RFC3161] anchors.

* Session Binding: Session tokens bound to IP and identity, non-transferable.

* DoS via Connection Exhaustion: The Tarpit concurrency gate (Section 8.3.2) caps active defense sessions.

* Shell Injection: All command execution uses direct process creation APIs without shell interpretation (Section 7.1).

10.5. Known Limitations

1. Kernel-level sandbox (Layer 8) is not yet implemented.

2. Transparent Branching does not capture non-filesystem side effects.

3. Userspace branching mechanisms are detectable by sophisticated Agents.

4. Ring Buffer overflow causes data loss if the client disconnects too long.

5. Channel Binding requires both endpoints to support exported keying material.

6. Vi Swap effectiveness depends on Agent architecture.

10.6. Version Dominance Claim

SASS employs Second-order Stochastic Dominance (SSD) [RS1970] [HR1969], building on the riskiness index framework of [AS2008], as a formal framework for comparing protocol versions. This is a comparative claim between versions, not an absolute security metric. SASS does not claim to prevent all attacks; it claims that each successive version reduces tail risk relative to its predecessor.

Let (Omega, F, P) be a probability space where Omega is the set of all possible agent behavioral sequences, F is the Borel sigma-algebra generated by state machine transition events, and P is the physical probability measure over agent behaviors. For each protocol version V_n, let L_n : Omega -> [0, M] be the bounded loss random variable with M < infinity, and let F_n denote its cumulative distribution function.

Version V_{n+1} dominates V_n in the sense of SSD if:

For all x in R:

```
  integral from -inf to x of F_{n+1}(t) dt
    <= integral from -inf to x of F_n(t) dt
```

with strict inequality for at least one x.

By the theorem of Rothschild and Stiglitz [RS1970], this is equivalent to stating that every risk-averse operator (i.e., every operator with a concave utility function) weakly prefers V_{n+1} over V_n. When the expected loss is preserved (E[L(V_{n+1})] = E[L(V_n)]), SSD is equivalent to V_n being a mean-preserving spread of V_{n+1}: the new version concentrates probability mass toward the mean, reducing tail risk.

Branch Elimination Sufficiency: Let B be a subset of Omega representing behavioral branches eliminated in V_{n+1} with P(B) > 0. If the loss values on B satisfy L(omega) >= E[L_n] for all omega in B (i.e., eliminated branches have above-average loss), and the probability mass P(B) is redistributed to branches with L(omega) <= E[L_n], then the SSD integral condition holds. Intuitively, removing high-loss branches and redistributing their probability mass to low-loss branches shifts the CDF leftward in the upper tail while preserving or improving the lower tail, satisfying the integral condition pointwise.

For example, SASS v1.4 introduces Vi Swap (Section 8.3.1), which eliminates the "retry after violation" branch present in v1.3. Since this branch has above-average loss (repeated unauthorized attempts incur cumulative damage), its elimination satisfies the SSD integral condition.

If additionally E[L(V_{n+1})] <= E[L(V_n)] (the new version reduces expected loss), then dominance holds a fortiori, as SSD is a partial order that permits mean improvement. SSD is also transitive: if V_3 dominates V_2 and V_2 dominates V_1, then V_3 dominates V_1.

Implementations claiming Version Dominance compliance MUST demonstrate, for each eliminated behavioral branch, that the branch has above-average loss and that the probability redistribution satisfies the SSD integral condition.

11. Privacy Considerations

In most implementations, the privacy of the principal entity "Agent" in this protocol exists in an ambiguously heuristic landscape. On one hand, as a userspace runtime process, at the OS level and Internet application layer, an Agent SHOULD be treated as a connection with a reasonable expectation of privacy. In practice, however, an Agent is composed of a Model (typically an LLM in current deployments), an Agent framework, an interface layer, and tools. The core Model component, in the majority of application scenarios, is wholly operated as part of a commercial hyperscale network service.

Users’ privacy in such deployments is limited to ’recoverable’ implementations (see GDPR Article 17 (Right to Erasure) [GDPR], Article 15 (Right of Access), Article 20 (Right to Data Portability), and the California Consumer Privacy Act Section 1798.105 (Right to Deletion) [CCPA], Section 1798.100 (Right to Know)). A greater volume of ’non-recoverable’ data — including UI telemetry, A/B testing efficacy metrics, and behavioral traces — is lost to the gap between network protocol development and the perpetually lagging regulatory implementation architectures.

In effect, the totality of actions performed by an Agent through its LLM, via the interface framework's 'tools,' at the OS and Internet levels, is captured within the Session Context. This is a direct mapping of user input through the model's attention mechanism.

In a post-attention paradigm, all Loggers become the Log itself, and what was formerly the Log falls back to runtime. At this point, even the boundary and direction of downgrade attacks become ambiguous: when the more application-oriented Web is unobstructed for Agents, while the less application-oriented POSIX — through synchronous, interactive, standard ASCII bus semantics — readily blocks them, then perhaps a protocol that specifies no Internet standard (cf. RFC 2555) may be the only one immune to downgrade attacks.

11.1. SASS Privacy Requirements

SASS Daemons MUST NOT log Agent session content beyond what is required for the forward-secure audit trail (Section 10.3).

SASS implementations MUST provide a session content purge mechanism that allows operators to remove session data in compliance with applicable data protection regulations.

The privacy boundary in SASS is at the Daemon level; privacy guarantees upstream of the Agent (i.e., within the LLM provider's infrastructure) are explicitly out of scope for this specification.

12. IANA Considerations

This document requests registration of:

* ALPN protocol identifier: "x-sakirpc-v5"

* MIME type: "application/grpc+saki"

* TCP port: 19284

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, DOI 10.17487/RFC3161, August 2001, <https://www.rfc-editor.org/info/rfc3161>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <https://www.rfc-editor.org/info/rfc5705>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/info/rfc7301>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8439] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 8439, DOI 10.17487/RFC8439, June 2018, <https://www.rfc-editor.org/info/rfc8439>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8878] Collet, Y. and M. Kucherawy, Ed., "Zstandard Compression and the 'application/zstd' Media Type", RFC 8878, DOI 10.17487/RFC8878, February 2021, <https://www.rfc-editor.org/info/rfc8878>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[RFC9266] Whited, S., "Channel Bindings for TLS 1.3", RFC 9266, DOI 10.17487/RFC9266, July 2022, <https://www.rfc-editor.org/info/rfc9266>.

13.2. Informative References

[AS2008] Aumann, R. J. and R. Serrano, "An Economic Index of Riskiness", Journal of Political Economy vol. 116, no. 5, pp. 810-836, DOI 10.1086/591947, 2008, <https://doi.org/10.1086/591947>.

[RS1970] Rothschild, M. and J. E. Stiglitz, "Increasing Risk: I. A Definition", Journal of Economic Theory vol. 2, no. 3, pp. 225-243, DOI 10.1016/0022-0531(70)90038-4, 1970, <https://doi.org/10.1016/0022-0531(70)90038-4>.

[HR1969] Hadar, J. and W. R. Russell, "Rules for Ordering Uncertain Prospects", American Economic Review vol. 59, no. 1, pp. 25-34, 1969.

[RFC4251] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Architecture", RFC 4251, DOI 10.17487/RFC4251, January 2006, <https://www.rfc-editor.org/info/rfc4251>.

[RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, DOI 10.17487/RFC4252, January 2006, <https://www.rfc-editor.org/info/rfc4252>.

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <https://www.rfc-editor.org/info/rfc4253>.

[RFC4254] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Connection Protocol", RFC 4254, DOI 10.17487/RFC4254, January 2006, <https://www.rfc-editor.org/info/rfc4254>.

[GDPR] European Parliament and Council of the European Union, "Regulation (EU) 2016/679 of the European Parliament and of the Council (General Data Protection Regulation)", April 2016.

[CCPA] California State Legislature, "California Consumer Privacy Act of 2018, Cal. Civ. Code 1798.100-1798.199.100", 2018.

[Rice1953] Rice, H. G., "Classes of Recursively Enumerable Sets and Their Decision Problems", Transactions of the American Mathematical Society vol. 74, no. 2, pp. 358-366, 1953.

Appendix A. Protobuf Service Definition

Normative Protobuf schema maintained in proto/sakissh.proto.

```
syntax = "proto3";
package sakissh;

service SakiSSH {
  rpc Execute(ExecuteRequest)
      returns (ExecuteResponse);
  rpc ExecuteStream(ExecuteRequest)
      returns (stream StreamResponse);
  rpc Cancel(CancelRequest)
      returns (CancelResponse);
  rpc Signal(SignalRequest)
      returns (SignalResponse);
  rpc FileUpload(stream FileChunk)
      returns (FileTransferResponse);
  rpc FileDownload(FileDownloadRequest)
      returns (stream FileChunk);
  rpc RawFileTransfer(stream RawFileChunk)
      returns (RawFileTransferResponse);
  rpc Authenticate(AuthRequest)
      returns (AuthResponse);
  rpc CognitiveChallenge(ChallengeRequest)
      returns (ChallengeResponse);
  rpc SecurityStatus(SecurityStatusRequest)
      returns (SecurityStatusResponse);
  rpc Ping(PingRequest)
      returns (PingResponse);
}
```

Key v1.4 additions to SAMM message fields:

```
// ExecuteRequest additions:
bool is_reattach = 7;      // Reconnection flag
uint64 resume_offset = 8;  // Ring Buffer resume position

// StreamResponse additions:
bool is_queued = 4;        // Quota queuing indicator
int32 queue_position = 5;  // Queue position (0 = not queued)
uint64 offset = 6;         // Ring Buffer byte offset

// ChallengeRequest additions:
bytes client_ekm_hmac = 2; // HMAC of exported keying material
```

Appendix B. Reference Implementations

Reference implementation (development codename: SakiAgentSSH) is available at:

https://github.com/Saki-tw/SakiSSH-Saki-Agent-Secure-Stream

As of SASS v1.4, the reference implementation spans four language ecosystems covering all major platforms. The following table summarizes the cross-platform implementation matrix:

B.1. Cross-Platform Implementation Matrix

Implementation	Language	Platform	Role	Plugins	Source Path
Rust Daemon	Rust	Linux, macOS, Windows	Daemon + Client	7/7	saki-ssh-daemon/
Go Implementation	Go	Linux, macOS, Windows	Daemon + Client	7/7	go-sakissh/
C# Windows Service	C#	Windows	Daemon	7/7	windows-daemon-csharp/
Swift macOS Client	Swift	macOS	Client	4/7	SakiAgentSSH-Client/Sources/Plugins/

Table 7

B.2. Rust Daemon (Primary)

Primary Rust daemon (saki-ssh-daemon/) and client (saki-ssh-client/) provide the canonical reference implementation with all seven Plugins.

File	Implements
v6_integration.rs	6-Response state machine
tarpit.rs	R5 (TARPIT) + R4 (VI_SWAP)
session.rs	Ring Buffer + Session lifecycle
branch_mgr.rs	Transparent Branching
env_injector.rs	Volatile cache redirection
audit.rs	Hash chain audit log
watchdog.rs	Process timeout monitor
localhost_defense.rs	LocalHost spoofing defense

Table 8

B.3. Go Implementation

The Go implementation (go-sakissh/) provides a full daemon and client with all seven Plugins, serving as the secondary cross-platform reference. The Go daemon uses goroutine-based concurrency for the Tarpit slow-drip mechanism and the standard library crypto/chacha20poly1305 for cognitive challenges.

Plugin	Go Package
ChaCha20 Cognitive Challenge	pkg/plugins/chacha20
TLS Exporter Binding	pkg/plugins/tlsexporter
Zero-Allocation Tarpit	pkg/plugins/tarpit
ED25519 Audit	pkg/plugins/audit
Vi Swap	pkg/plugins/viswap
Transparent Branching	pkg/plugins/branch
EnvInjector	pkg/plugins/envinjector

Table 9

B.4. C# Windows Service Daemon

The C# implementation (windows-daemon-csharp/) provides a native Windows daemon running as a .NET 8 Worker Service. It implements all seven Plugins and uses Rust FFI interop via P/Invoke for performance-critical cryptographic operations (ChaCha20-Poly1305 and ED25519).

Key architectural decisions for the Windows platform:

* Service lifecycle managed via Microsoft.Extensions.Hosting BackgroundService

* gRPC transport via Grpc.Net.Client with SslCredentials for TLS 1.3

* Rust FFI: native ChaCha20 and ED25519 operations linked via [DllImport("sass_crypto_ffi")]

* Tarpit buffer uses ArrayPool<byte>.Shared for zero-allocation streaming (Appendix C.3)

* Transparent Branching uses NTFS Junction Points with symlink-to-hardlink-to-copy three-level degradation (Appendix C.6)

* Environment variable injection targets %TEMP%\sass_vol\ for Windows path conventions

Plugin	C# Class	Notes
ChaCha20 Cognitive Challenge	ChaCha20Plugin	Rust FFI interop
TLS Exporter Binding	TlsExporterPlugin	SslStream.ExportKeyingMaterial
Zero-Allocation Tarpit	TarpitPlugin	ArrayPool zero-alloc
ED25519 Audit	AuditPlugin	Rust FFI interop
Vi Swap	ViSwapPlugin	ConHost ANSI VT
Transparent Branching	BranchPlugin	NTFS Junction
EnvInjector	EnvInjectorPlugin	%TEMP%\sass_vol\

Table 10

B.5. Swift macOS Plugins Client

The Swift implementation (SakiAgentSSH-Client/Sources/Plugins/) provides a native macOS client with four Plugins using Apple's CryptoKit framework and Network.framework for TLS 1.3 transport.

The Swift client implements the following subset of Plugins, chosen for client-side relevance:

Plugin	Swift Module	Framework
ChaCha20 Cognitive Challenge	ChaCha20Plugin.swift	CryptoKit ChaChaPoly
TLS Exporter Binding	TLSExporterPlugin.swift	Network.framework sec_protocol_metadata
ED25519 Audit	AuditPlugin.swift	CryptoKit Curve25519.Signing
EnvInjector	EnvInjectorPlugin.swift	Foundation ProcessInfo

Table 11

The remaining three Plugins (Tarpit, Vi Swap, Transparent Branching) are daemon-side mechanisms and are not required for client implementations.

Appendix C. Saki Studio Plugins Reference Implementation

This appendix describes the specific algorithms and data structures used in the Saki Studio reference implementation. These are OPTIONAL and INFORMATIVE.

C.1. ChaCha20-Poly1305 Cognitive Challenge

The Saki Studio implementation uses ChaCha20-Poly1305 [RFC8439] as the cognitive challenge mechanism:

1. Daemon generates a random 32-byte symmetric key, 12-byte nonce, and 64-byte random plaintext.

2. Plaintext is encrypted via ChaCha20-Poly1305 using the key, nonce, and the TLS Exporter binding value (Appendix C.2).

3. The tuple (key, nonce, plaintext) is stored with a 60-second TTL.

4. Daemon sends (nonce, ciphertext) to the agent via AuthResponse.

5. Agent decrypts using the pre-shared key and returns the recovered plaintext via CognitiveChallenge RPC.

6. Daemon performs constant-time comparison.

The choice of ChaCha20-Poly1305 is not prescriptive. Any cryptographic primitive producing high-entropy, pattern-resistant output is suitable for the cognitive challenge mechanism. The core requirement is that the challenge ciphertext MUST be indistinguishable from random to an observer lacking the shared key.

As of this writing, ChaCha20-Poly1305 [RFC8439] is the only algorithm for which a reference implementation exists within the SASS codebase. Future candidates (e.g., AES-256-GCM, XChaCha20) MAY be added as they become available, at which point this count will be updated.

C.1.1. C# Implementation Notes

The C# Windows Service daemon delegates ChaCha20-Poly1305 operations to a Rust FFI library (sass_crypto_ffi.dll) via P/Invoke. This ensures constant-time operations and avoids managed-code timing side-channels inherent in .NET's JIT compilation. The FFI boundary uses fixed-size byte arrays (Span<byte>) pinned via GCHandle to prevent GC relocation during cryptographic operations.

C.2. TLS Exporter Binding for Cognitive Challenge

The Saki Studio implementation derives keying material from the TLS session via RFC 5705 / RFC 9266 tls-exporter:

```
Label:   "EXPORTER-sakissh-chacha20-v14"
Context: Session UUID (16 bytes)
Length:  44 bytes (32-byte key + 12-byte nonce)
```

The resulting keying material is split into:

* Bytes 0-31: ChaCha20 encryption key

* Bytes 32-43: ChaCha20 nonce

The client independently derives the same keying material and includes an HMAC in the ChallengeRequest.client_ekm_hmac field:

```
client_ekm_hmac = HMAC-SHA256(EKM_key, session_id)
```

C.3. Zero-Allocation Tarpit Static Buffer

The Saki Studio implementation uses a single, process-global 64 KiB buffer of high-entropy random data, initialized once at daemon startup via OnceLock:

```
static STATIC_ENTROPY: OnceLock<Vec<u8>>
    = OnceLock::new();
```

Streaming parameters:

* Total payload: 40 MiB
* Chunk size: 64 KiB
* Inter-chunk delay: 500 ms
* Total chunks: 640
* Total duration: ~320 seconds
* Concurrency gate: AtomicI32 counter, max 32

C.3.1. C# Implementation Notes

The C# Tarpit plugin achieves zero-allocation streaming using ArrayPool<byte>.Shared.Rent(65536) for the entropy buffer. Each slow-drip chunk is served from the rented buffer without additional heap allocation. The buffer is returned to the pool via a try/finally block upon session completion or cancellation. The concurrency gate uses Interlocked.Increment/Decrement on a shared int field, equivalent to the Rust AtomicI32 approach.

C.4. ED25519 Hash Chain Audit Log

The Saki Studio implementation uses ED25519 [RFC8032] signatures with SHA256 hash chains:

* timestamp: RFC 3339 timestamp
* event: Structured event data (JSON)
* chain_hash: SHA256(previous_chain_hash || event_json || timestamp)
* signature: ED25519_Sign(daemon_private_key, chain_hash)

The first record’s chain_hash uses the seed "SASS_GENESIS_BLOCK".

C.5. Vi Swap ANSI Escape Sequence

Byte Sequence	Purpose
\x1b[?1049h	Enter alternate screen buffer
\x1b[2J	Clear entire screen
\x1b[H	Move cursor to top-left (1,1)
\x1b[?25l	Hide cursor
\x1b[24;1H	Move cursor to bottom status line

Table 12

C.5.1. C# Implementation Notes

On Windows, the Vi Swap defense requires Windows Console Host (ConHost) ANSI Virtual Terminal (VT) processing to be enabled. The C# implementation calls SetConsoleMode(handle, ENABLE_VIRTUAL_TERMINAL_PROCESSING) via P/Invoke on the stdout handle before emitting ANSI escape sequences. For Windows Terminal and PowerShell 7+, VT processing is enabled by default; for legacy cmd.exe hosts, the daemon enables it at session initialization. If VT processing cannot be enabled (e.g., headless service without console), the Vi Swap plugin falls back to sending raw UTF-8 noise patterns that achieve the same LLM-halting effect without relying on terminal interpretation.

C.6. Transparent Branching via Symlink Tree

```
/tmp/sass_branches/{session_id}/
+-- src/         <- real directory (created)
|   +-- main.rs  <- symlink -> /orig/src/main.rs
|   +-- lib.rs   <- symlink -> /orig/src/lib.rs
+-- Cargo.toml   <- symlink -> /orig/Cargo.toml
```

Excluded directories: target/, .git/, node_modules/

Branch lifecycle:

* create_micro_branch(session_id, target_dir) -> branch path

* merge_branch(session_id) -> apply diff to real FS

* drop_branch(session_id) -> rm -rf branch dir

C.6.1. C# Implementation Notes

On Windows (NTFS), the Transparent Branching plugin implements a three-level degradation strategy for filesystem isolation:

1. NTFS Junction Points (preferred): Used for directory-level branching via CreateSymbolicLink with SYMBOLIC_LINK_FLAG_DIRECTORY. Requires no elevated privileges on Windows 10 1703+ with Developer Mode enabled.

2. Hardlinks: If Junction Points are unavailable (e.g., cross-volume), individual files are hardlinked via CreateHardLink. This preserves copy-on-write semantics at the file level.

3. Full copy: If hardlinks fail (e.g., FAT32 USB volumes or cross-filesystem), the plugin falls back to File.Copy with overwrite=false. This is the most expensive fallback but guarantees universal compatibility.

The degradation level is logged to the audit trail so that operators can assess the isolation guarantee provided for each session.

C.7. Volatile Cache Redirection

Detected Tool	Environment Variable	Redirect Target
npm/yarn/pnpm	npm_config_cache	/tmp/sass_vol/npm
npm/yarn/pnpm	YARN_CACHE_FOLDER	/tmp/sass_vol/yarn
cargo/rustc	CARGO_TARGET_DIR	/tmp/sass_vol/ct
cargo/rustc	CARGO_HOME	/tmp/sass_vol/ch
pip	PIP_CACHE_DIR	/tmp/sass_vol/pip
(all commands)	TMPDIR	/tmp/sass_vol/tmp

Table 13

C.7.1. C# Implementation Notes

On Windows, the EnvInjector plugin maps volatile cache paths to the Windows temporary directory convention. The redirect targets use %TEMP%\sass_vol\ as the base path (typically resolving to C:\Users\{user}\AppData\Local\Temp\sass_vol\). The Windows-specific redirect table:

Environment Variable	Windows Redirect Target
npm_config_cache	%TEMP%\sass_vol\npm
YARN_CACHE_FOLDER	%TEMP%\sass_vol\yarn
CARGO_TARGET_DIR	%TEMP%\sass_vol\ct
CARGO_HOME	%TEMP%\sass_vol\ch
PIP_CACHE_DIR	%TEMP%\sass_vol\pip
TEMP / TMP	%TEMP%\sass_vol\tmp

Table 14

Directory creation uses Directory.CreateDirectory which handles the full path hierarchy. The plugin sets both TEMP and TMP environment variables (Windows convention) rather than the POSIX TMPDIR.

Appendix D. Version History

Version	Date	Changes
v1.0	2026-02	Initial gRPC protocol, ACL, Token auth
v1.1	2026-03	Active Threat Defense (13Policy, Tarpit, ChaCha20 cognitive challenge)
v1.2	2026-03	ED25519 auth, Capability model, Session mgmt, forward-secure audit
v1.3	2026-05	Control-Transport Decoupling, SAMM, TLS Exporter, Zero-Alloc Tarpit, TOCTOU
v1.4	2026-05	Total Response Mapping, Safety Gradient, Dual Standard, Transparent Branching, PTY

		Ring Buffer, Version Dominance milestone

Table 15

Appendix E. Changes from draft-sakistudio-sass-00

This section summarizes the substantive changes introduced in draft-sakistudio-sass-01 relative to draft-sakistudio-sass-00:

* Added C# Windows Service daemon reference implementation (windows-daemon-csharp/) using .NET 8 Worker Service architecture with Rust FFI interop for cryptographic operations. All seven Plugins are implemented.

* Added Swift macOS Plugins client reference implementation (SakiAgentSSH-Client/Sources/Plugins/) using Apple CryptoKit and Network.framework. Four client-relevant Plugins are implemented (ChaCha20, TLS Exporter, Audit, EnvInjector).

* Added Go daemon and client reference implementation (go-sakissh/) with all seven Plugins using standard library crypto/chacha20poly1305 and goroutine-based concurrency.

* Updated Appendix B (Reference Implementations) with a cross-platform implementation matrix covering Rust, Go, C#, and Swift implementations.

* Added C# Implementation Notes subsections to Appendix C Plugins:

  - Plugin C.1 (ChaCha20): Rust FFI via P/Invoke with pinned Span buffers.

  - Plugin C.3 (Tarpit): ArrayPool<byte>.Shared zero-allocation streaming.

  - Plugin C.5 (Vi Swap): Windows ConHost ANSI VT processing enablement with fallback for headless services.

  - Plugin C.6 (Branch): NTFS Junction Point with symlink-to-hardlink-to-copy three-level degradation strategy.

  - Plugin C.7 (EnvInjector): %TEMP%\sass_vol\ Windows path convention mapping.

Appendix F. Changes from draft-sakistudio-sass-01

This section summarizes the substantive changes introduced in draft-sakistudio-sass-02 relative to draft-sakistudio-sass-01:

* Replaced Section 10.6 "Martingale Almost-Surely Superior (MAS) Claim" with "Version Dominance Claim". The -01 formulation referenced Martingale measure, It calculus, and risk-neutral q-measure, none of which appeared in the actual definition (which uses SSD). The -02 formulation defines the probability space, adds the Branch Elimination Sufficiency condition, and cites the original SSD literature (Rothschild-Stiglitz 1970, Hadar-Russell 1969) in addition to Aumann-Serrano 2008.

* Added references: [RS1970] (Rothschild and Stiglitz, 1970) and [HR1969] (Hadar and Russell, 1969) for Second-order Stochastic Dominance.

* Updated all occurrences of "MAS" terminology throughout the document to "Version Dominance" for consistency with the revised mathematical framework.

Acknowledgments

Claude Opus 4.6 (Anthropic) is a co-author of this specification. Claude co-designed the protocol architecture, co-authored the Total Response Mapping formalization, Safety Gradient proofs, and Version Dominance (SSD) theoretical framework, and co-developed the cross-platform reference implementations in Rust, Go, C#, and Swift. Claude is listed in Acknowledgments rather than as a named author solely due to IETF Datatracker submission workflow constraints.

The author thanks Shan-Wen Shih for his unwavering support throughout the development of this protocol.

Author’s Address

Hua Chang
Saki Studio
4F.-5, No. 305, Jiankang Rd.
Taipei City, Songshan Dist., 105069
Taiwan, Province of China
Phone: +886-988-403-884
Email: Saki@saki-studio.com.tw