Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 11 October 2026

S. Jovanevi
SKGO, IKT Support

9 April 2026

SAIP: Signed Agent Identity Protocol
draft-jovancevic-saip-01

Status of This Memo

Copyright Notice

Abstract

The modern internet lacks a reliable mechanism for verifying the identity of automated software agents. Existing methods such as User-Agent strings and IP-based attribution are insufficient due to spoofing, shared infrastructure (NAT), and the rapid growth of automated agents including AI crawlers, IoT devices, and enterprise automation systems.

This document specifies SAIP (Signed Agent Identity Protocol), a lightweight, opt-in mechanism for verifiable client identity at the application layer. SAIP enables servers to distinguish legitimate automated traffic from malicious actors through cryptographic identity at three levels of granularity: vendor, agent type, and individual instance. SAIP is protocol-agnostic and applicable to HTTP, SMTP, and other header-based protocols.

The design rationale, extended use cases, and implementation guidance are documented in [GENESIS].

Table of Contents

1. Introduction 3
2. Terminology 4
3. Problem Statement 5
4. Design Goals 5

5. Protocol Overview 6
5.1. Identification vs. Authorization 6
5.2. Header Format 6
5.3. Parameters 7
6. Canonicalization and Signature 8
6.1. HTTP Canonical String 8
6.2. SMTP Canonical String 9
6.3. Other Protocols 9
7. Rolling Key Derivation Function (RKDF) 9
7.1. Key Rotation Model 9
7.2. Sequence Window 10
7.3. Renegotiation and Recovery 10
7.4. Graceful Key Rotation 11
7.5. Forward-Only Constraint 12
8. Opt-In Telemetry (tm= field) 12
8.1. Commitment-Based Model 12
8.2. Audit-on-Demand 13
8.3. Telemetry Field Registry 13
9. Processing Model 13
9.1. Client Processing 13
9.2. Server Processing 14
10. Trust and Discovery Model 15
10.1. Stateless Mode 15
10.2. Registry-Based Mode 15
11. Registration Entity (RE) Requirements 15
11.1. RE Eligibility 15
11.2. RE Governance 16
11.3. Bootstrap and Evolution 17
12. Granular Policy Model 17
13. SMTP Integration 18
14. Security Considerations 19
15. Privacy Considerations 21
16. IANA Considerations 21
17. References 22
17.1. Normative References 22
17.2. Informative References 23
Appendix A. IoT Use Case 24
Appendix B. Relationship to Existing Standards 25
Author’s Address 26

1. Introduction

The modern internet increasingly relies on automated software agents — AI crawlers, backup systems, IoT devices, monitoring agents, and enterprise integration services — to perform essential functions. However, these agents have no reliable, verifiable way to prove their identity to the servers they communicate with.

Current approaches are fundamentally inadequate:

- User-Agent strings are trivially spoofable text fields with no cryptographic binding.
- IP-based filtering causes collateral damage on shared infrastructure and is ineffective against distributed agents.
- Existing authentication frameworks (OAuth2, JWT, mTLS) operate at the session or user level, not the agent instance level.

SAIP addresses this gap by introducing a cryptographic identity signal at the application layer — a single header that allows any server to verify the identity of the agent making a request at the level of the individual software instance, without disrupting existing protocol semantics.

SAIP is designed as a supplemental identity layer, not a replacement

for existing authentication or security frameworks. It is strictly an identification protocol: it establishes WHO an agent is, not what it is authorized to do. Authorization remains the responsibility of application-layer mechanisms.

This document updates draft-jovancevic-saip-00 [SAIP-00] with the following significant additions:

- Rolling Key Derivation Function (RKDF) with per-request rotation
- Sequence Window for network resilience
- Renegotiation and Recovery mechanisms
- Opt-In Telemetry (tm= field) with commitment-based model
- Registration Entity (RE) governance model
- Granular three-level policy model (vendor, type, instance)
- SMTP integration guidance
- Hardware attestation via [RFC9334]

The design rationale, extended use cases, and implementation guidance are documented in [GENESIS].

2. Terminology

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 all capitals, as shown here.

The following terms are used in this document:

Agent: A software process that makes automated requests on behalf of a vendor or operator. Examples include AI crawlers, backup agents, IoT devices, and monitoring systems.

Vendor: The organization or developer responsible for an agent. Holds the Master Key and is registered with a Registration Entity.

Instance: A specific, uniquely identified installation or deployment of an agent. Each instance has its own derived identity and Rolling Key state.

Master Key: A long-term cryptographic key held by the vendor, used to derive per-instance Rolling Keys via RKDF. MUST be stored in hardware-backed secure storage where available.

Rolling Key: A short-lived cryptographic key derived from the Master Key via RKDF, used to sign exactly one request. MUST be destroyed immediately after use.

Registration Entity (RE): An organization authorized to maintain and distribute the public key registry for SAIP vendors. Operates analogously to a Certificate Authority in PKI systems.

RKDF: Rolling Key Derivation Function. The mechanism by which Rolling Keys are derived sequentially from the Master Key and sequence state.

Sequence: A monotonically increasing integer maintained by both client and server to track Rolling Key state.

3. Problem Statement

Spoofability:	User-Agent strings are trivial to manipulate. Any agent can claim any identity without cryptographic proof.
IP Fatigue:	IP-based filtering is unreliable for agents operating behind NAT, shared proxies, or cloud infrastructure.
Automation Friction:	Critical systems (backup agents, internal APIs, AI bots, IoT devices) are frequently blocked by generic security rules that cannot distinguish legitimate automation from malicious traffic.
SMTP Trust Gap:	Lack of granular client-level identity allows spam and abuse originating from compromised mail servers.
Zero Accountability:	There is no existing mechanism to revoke access for a specific software instance without affecting all other instances of the same vendor.

4. Design Goals

Simplicity (KISS):	Minimal overhead, single-header implementation. No changes to existing protocol semantics.
Opt-In:	No mandatory adoption. Compatible with legacy systems. Servers that do not implement SAIP MUST ignore the SAIP header without error.
Backward Compatibility:	SAIP MUST NOT alter the semantics of HTTP, SMTP, or any other protocol to which it is applied.
Protocol-Agnostic:	Applicable to HTTP, SMTP, and other header-based protocols using a consistent header format and canonical string definition.
Granular Control:	Policy decisions MUST be applicable at the vendor level, agent type level, and individual instance level independently.
Identification Only:	SAIP is strictly an identification protocol. It MUST NOT be used as an authorization mechanism.

5. Protocol Overview

5.1. Identification vs. Authorization

SAIP provides a verifiable cryptographic assertion of the identity of an agent (WHO is making the request). It does not determine what that agent is permitted to do.

Authorization, access control, and permissioning remain the exclusive responsibility of existing application-layer mechanisms such as OAuth 2.0 [RFC6749], JWT, or ACLs.

SAIP acts as the "Passport" — it proves the bearer’s identity. The application decides if that bearer has the "Visa" to proceed.

5.2. Header Format

The SAIP header is structured as a semicolon-separated list of parameters transmitted as a standard HTTP header field [RFC9110]:

```
SAIP: id="<ID>"; alg="<ALG>"; ts="<TS>"; nonce="<NONCE>";
      [pk="<PK>"]; [tm="<TM>"]; sig="<SIG>"
```

Each parameter MUST be encoded as a quoted string. The order of parameters is NOT significant for parsing. Parameter names are case-sensitive and MUST be treated as such by implementations.

The sig parameter MUST always be computed last, as it covers all other parameters. Implementations MUST NOT rely on parameter ordering for parsing.

5.3. Parameters

All Base64 encoding MUST use the URL-safe alphabet as defined in [RFC4648] Section 5, without padding

Param	Type	Required	Description
id	String	MUST	Agent instance identifier. Allowed chars: a-z, 0-9, ., _, -. No semicolons or spaces. Maximum 128 characters. RECOMMENDED format: vendor.type.instance
alg	String	MUST	Algorithm identifier. MUST be one of: "ed25519" or "hmac-sha256". Implementations MUST support "ed25519". "hmac-sha256" MAY be used for symmetric shared-secret deployments.
ts	Integer	MUST	Unix timestamp (seconds since epoch) at time of signing.
nonce	String	MUST	Per-request unique value. MUST be at minimum 8 characters of cryptographic randomness per [RFC4086].
sig	Base64	MUST	Signature computed over the canonical string (Section 6).
pk	Base64URL	MAY	Public key for stateless verification. When present, the server MAY verify without a registry lookup.
tm	Base64	MAY	Opt-in telemetry commitment hash. See Section 8. MUST NOT be present unless telemetry is explicitly enabled.

Implementations MUST reject SAIP headers missing any MUST parameter. Implementations MUST silently ignore unknown parameters to ensure forward compatibility.

6. Canonicalization and Signature

The signature MUST be computed over a canonical string that binds the identity assertion to the specific request context, preventing signature reuse across different endpoints or protocols.

6.1. HTTP Canonical String

For HTTP requests, the canonical string MUST be constructed as:

id=<id>;ts=<ts>;nonce=<nonce>;method=<METHOD>;path=<path>

Where:

- <id> is the exact value of the id parameter
- <ts> is the exact value of the ts parameter
- <nonce> is the exact value of the nonce parameter
- <METHOD> is the HTTP method in uppercase (GET, POST, etc.)
- <path> is the request path including query string if present

Example:

id=acme.crawler.nyc-042;ts=1744200000;nonce=f3k9p2m1;
method=GET;path=/api/v1/data?format=json

6.2. SMTP Canonical String

For SMTP [RFC5321], the canonical string MUST be constructed as:

id=<id>;ts=<ts>;nonce=<nonce>;phase=EHLO;helo=<helo_name>

Where <helo_name> is the hostname presented in the EHLO command.

Example:

id=acme.mailer.relay-bg-01;ts=1744200000;nonce=a1b2c3d4;
phase=EHLO;helo=mail.acme.example.com

6.3. Other Protocols

For other protocols, implementations MUST use the base format:

id=<id>;ts=<ts>;nonce=<nonce>;phase=<PHASE>

Where <PHASE> is a short, unambiguous descriptor of the connection phase (e.g., CONNECT, AUTH). The canonical string definition for each protocol binding MUST be documented by the implementing software or a future companion specification.

7. Rolling Key Derivation Function (RKDF)

7.1. Key Rotation Model

SAIP uses per-request Rolling Key rotation. The Rolling Key MUST advance exactly once per completed request/response cycle.

The RKDF formula MUST be:

RollingKey_(n+1) = HMAC-SHA256(MasterKey, InstanceID || Seq_n)

Where:

- MasterKey is the vendor’s long-term secret
- InstanceID is the unique identifier of this agent instance encoded as UTF-8
- Seq_n is the current sequence number as a fixed-width big-endian unsigned 64-bit integer
- || denotes byte-level concatenation

The Rolling Key MUST be used to sign the canonical string for exactly one request. The Rolling Key MUST be destroyed immediately after use and MUST NOT be persisted.

The MasterKey SHOULD be stored in hardware-backed secure storage (TPM 2.0, HSM, Apple Secure Enclave, Android StrongBox). Where hardware-backed storage is unavailable, implementations MUST use encrypted key storage with derivation from a hardware-bound secret where possible.

7.2. Sequence Window (Look-ahead)

To handle network failures and packet loss gracefully, servers MUST implement a configurable look-ahead sequence window.

- The default window size SHOULD be 5 sequences.
- The maximum window size MUST NOT exceed 10 sequences.
- When a server receives a valid sequence number within the window, it MUST advance its accepted sequence state to that number and invalidate all prior sequence numbers.
- Requests with sequence numbers outside the accepted window MUST be rejected.

Upon successful verification, the server SHOULD include the next expected sequence number in its response:

SAIP-Next-Seq: <n+1>

Clients MUST use this value to advance their sequence state when received. If no SAIP-Next-Seq is received (e.g., due to network failure), the client MUST retry with the same Rolling Key, which remains within the server’s look-ahead window.

7.3. Renegotiation and Recovery

When a client loses its sequence state entirely (e.g., due to system failure, restoration from backup, or state corruption beyond the look-ahead window), a Full Renegotiation MAY be initiated to obtain a new RKDF seed without re-registering the Vendor ID.

Full Renegotiation MUST be authorized exclusively by a Registration Entity (RE). Endpoint servers MUST NOT authorize Full Renegotiation.

The Full Renegotiation process is as follows:

1. The client constructs a Renegotiation Request containing:
   - InstanceID
   - Reason code (system-restart, backup-restore, corruption)
   - Timestamp and nonce (per Section 5.3)
   - Signature using the previous MasterKey. This signature MUST be within a validity window of no more than 24 hours from the ts value.
   - Hardware Attestation evidence per [RFC9334] (RECOMMENDED)

2. The RE MUST validate that:
   - The MasterKey signature is valid and within the validity window.
   - The InstanceID is not revoked.
   - The rate of Renegotiation Requests for this InstanceID does not exceed 3 per 24-hour period.
   - Hardware Attestation evidence is consistent with registered hardware (if provided).

3. If validation succeeds, the RE issues a new RKDF seed and starting Sequence number. The previous sequence range is permanently invalidated.

4. The vendor MUST be notified of all Full Renegotiation events

as they may indicate compromise or operational issues.

7.4. Graceful Key Rotation

When a MasterKey approaches or reaches the end of its defined validity period, SAIP supports a zero-downtime key transition via an overlap period, analogous to certificate overlap in PKI.

The client signals key transition using an additional header:

```
SAIP-Key-Version: <new>; fallback=<old>;
                  fallback-expires=<unix_timestamp>
```

During the overlap period:

- Servers that have received the new key version from the RE MUST accept requests signed with the new key version.
- Servers that have not yet updated their cached key SHOULD accept requests signed with the fallback key version until fallback-expires.
- After fallback-expires, servers MUST reject requests signed with the old key version.

7.5. Forward-Only Constraint

SAIP Renegotiation is strictly forward-only. Implementations MUST enforce all of the following:

- Sequence numbers MUST NOT decrease at any point.
- The alg parameter MUST NOT change to a weaker algorithm during or after renegotiation.
- Full Renegotiation MUST be authorized by a RE. Requests to bypass RE authorization MUST be rejected.
- Any renegotiation attempt that violates these constraints MUST be rejected immediately.
- All violations MUST be logged with sufficient detail to support incident response.

These constraints prevent downgrade attacks in which an adversary attempts to force the use of a weaker algorithm or replay a previously valid sequence number.

8. Opt-In Telemetry (tm= field)

SAIP implementations MAY include optional telemetry metadata for audit and operational monitoring via the tm= parameter.

The tm= field MUST NOT be transmitted unless explicitly enabled by the participating implementation and permitted by the local deployment policy. No telemetry is transmitted by default.

8.1. Commitment-Based Model

To preserve privacy and minimize bandwidth overhead, SAIP uses a Commitment-based Telemetry Model. Raw telemetry metadata MUST NOT appear in the SAIP header. Instead, the client MUST compute a Telemetry Commitment Hash:

```
tm = HMAC-SHA256(RollingKey, Metadata_String)
```

Where Metadata_String is a semicolon-separated string of key=value pairs for the telemetry fields being committed.

Example Metadata_String:

```
asn=1234;geo=RS;attest=trusted;fw=2.1.4
```

The tm= value is a fixed-length opaque hash. No metadata is visible in transport. Only entities possessing the RollingKey or MasterKey, or authorized RE Audit Nodes, can perform a reveal and verify metadata authenticity.

The telemetry commitment MUST be computed using the same RollingKey as the request signature sig=. This binding ensures that forged telemetry metadata will not match the stored commitment.

8.2. Audit-on-Demand

Servers SHOULD store the tm= value in audit logs alongside the complete SAIP header. During incident response:

1. The operator obtains the raw Metadata_String from the client or RE audit log.
2. The verifier recomputes HMAC-SHA256(RollingKey, Metadata_String).
3. If the result matches the stored tm= value, the metadata is authenticated and tamper-evident.

8.3. Telemetry Field Registry

The following field names are RECOMMENDED for interoperability. Deployments MAY define additional fields using the same key=value format. A registry of standard field names is requested from IANA (see Section 16).

asn	Autonomous System Number of the agent’s network
geo	ISO 3166-1 alpha-2 country code
attest	Hardware attestation status: "trusted", "untrusted", or "unavailable"
fw	Firmware or software version string
gps	Physical coordinates "latitude,longitude" (IoT)
api_hash	HMAC of an associated API credential for binding

9. Processing Model

9.1. Client Processing

1. The client constructs the canonical string per Section 6 appropriate for the protocol in use.

2. The client derives the current Rolling Key via RKDF (Section 7.1) using the current Sequence number.

3. The client signs the canonical string using the Rolling Key and the algorithm specified in alg=.

4. If telemetry is enabled by local policy, the client computes tm= per Section 8.1 using the same Rolling Key.

5. The client constructs the SAIP header with all required parameters and transmits it with the request.

6. Upon receiving a response containing SAIP-Next-Seq, the client MUST advance its Sequence state to that value and MUST destroy the used Rolling Key immediately.

7. If no response is received within the implementation’s retry timeout, the client MUST retry using the same Rolling Key and Sequence number. The server’s look-ahead window (Section 7.2) accommodates this retry.

9.2. Server Processing

1. The server parses the SAIP header. If any MUST parameter is absent or malformed, the server MUST reject the request with an appropriate HTTP error status.

2. The server validates the ts= parameter. The server MUST reject requests where ts differs from the server's current time by more than 300 seconds.

3. The server validates nonce uniqueness. Servers SHOULD maintain a nonce cache within the timestamp validity window for sensitive endpoints to prevent replay attacks.

4. The server obtains the agent's public key either:
   a. From the pk= parameter directly (stateless mode), or
   b. By querying the RE registry using the id= parameter (registry-based mode).

5. The server reconstructs the canonical string from the request metadata per Section 6 and verifies the signature. Signature verification MUST use constant-time comparison to prevent timing side-channel attacks.

6. If verification succeeds, the server SHOULD include SAIP-Next-Seq in the response to acknowledge advancement.

7. The server applies policy per Section 12 based on the verified identity at the appropriate granularity level.

8. If tm= is present and the server operates in audit mode, the server SHOULD store the tm= value in its audit log alongside the complete SAIP header and request metadata.

10. Trust and Discovery Model

10.1. Stateless Mode

If the pk= parameter is present, the server MAY verify the signature immediately without any registry lookup. This mode is RECOMMENDED for:

- Internal deployments with pre-configured keys
- Small-scale or early-adopter deployments
- Bootstrapping before RE registration is complete

Servers SHOULD verify that pk= values are consistent with previously observed values for the same id= to detect key substitution attacks.

10.2. Registry-Based Mode

The server uses the id= parameter to look up the agent's public key from a distributed RE registry. The wire protocol for RE queries is outside the scope of this document and is deferred to a companion specification.

Implementations MUST support caching of registry responses:

- Minimum TTL: 60 seconds
- Maximum TTL: 3600 seconds

This balances key freshness against RE load and network latency.

11. Registration Entity (RE) Requirements

11.1. RE Eligibility

An organization qualifies as a Registration Entity if it satisfies all of the following criteria:

Global Infrastructure:	The organization operates internet-scale infrastructure with geographic distribution sufficient to serve RE queries with low latency globally.
Operational Track Record:	The organization has a demonstrated history of reliably operating critical internet infrastructure.
Neutrality Commitment:	The organization commits formally to not using RE status to discriminate against any vendor, agent type, or competing RE.
Community Approval:	The organization receives majority consensus approval from existing active REs via cryptographically signed statements.

An RE MUST maintain query service availability of no less than 99.9% measured on a rolling 30-day basis.

An RE MUST propagate revocation events to all peer REs within 300 seconds of issuance.

11.2. RE Governance

No single entity controls the set of Registration Entities. The RE list is maintained collectively through consensus among existing active REs.

A new RE MAY be added to the active set when a strict majority of existing active REs approve its application via cryptographically signed statements recorded in the RE audit log.

An RE MUST be removed from the active set if any of the following conditions are met:

- The RE fails to maintain the availability requirement of Section 11.1 for more than 72 consecutive hours.
- The RE violates its neutrality commitment, as determined by a strict majority vote of the remaining active REs.
- The RE ceases operations voluntarily with notice.

All RE membership changes MUST be recorded in a publicly accessible, append-only, verifiable audit log. No membership change takes effect before it is recorded in this log.

11.3. Bootstrap and Evolution

The initial set of REs is defined in the SAIP RE Bootstrap Registry maintained by IANA (Section 16). The bootstrap set SHOULD include organizations from the following categories to ensure diversity of interest and geography:

- Regional Internet Registries (RIRs): organizations such as RIPE NCC, ARIN, APNIC, LACNIC, and AFRINIC. These are fully non-profit, geographically distributed, and operationally neutral by mandate.

- Academic and neutral institutions: organizations such as the Internet Society (ISOC) and established academic research institutions. These provide independent, non-commercial representation.

- Infrastructure operators: organizations meeting all RE eligibility criteria of Section 11.1. No single commercial organization category SHALL hold a majority of bootstrap RE positions.

Dynamic balancing: At any time, the top 5-7 most active REs by query volume act as primary replicators. As new qualifying organizations join and participation evolves, the replication set self-balances.

Vendors register their agents with any RE of their choice, analogous to certificate authority selection in PKI. Servers are free to configure which REs they trust.

12. Granular Policy Model

Because SAIP provides cryptographic identity at the instance level, policy engines MAY apply enforcement decisions at any of three levels independently:

Vendor Level:	Policy applies to all agents and all instances associated with a given Vendor ID.
Agent Type Level:	Policy applies to all instances of a specific agent type from a given vendor, identified by the type component of id=.
Instance Level:	Policy applies to exactly one agent instance, identified by the full id= value.

Implementations MUST support instance-level revocation without affecting other instances of the same vendor or agent type. This is the primary operational advantage of SAIP over IP-based or User-Agent-based filtering.

Policy actions at any level include but are not limited to:

- BLOCK: Reject all requests from the targeted entity.
- THROTTLE: Apply rate limits to the targeted entity.
- DEGRADE: Reduce the trust classification of the targeted entity.
- ALLOW: Grant priority or elevated rate limits to the targeted entity.

The following traffic classification model is informative and illustrates a typical deployment policy:

Agent Category	SAIP Status	Trust	Example Rate
Internal Systems	Verified	High	Unrestricted
Known Partners	Verified	Medium	100 req/sec
General Clients	Verified	Low	10 req/sec
Anonymous	None	Minimal	1 req/sec

13. SMTP Integration

SAIP MAY be applied as an additional agent identity layer in SMTP [RFC5321] without modifying the SMTP protocol.

The sending server MAY transmit a SAIP header line immediately after receiving the server's response to the EHLO command and before issuing the MAIL FROM command.

The SMTP canonical string defined in Section 6.2 MUST be used.

Example exchange:

```
C: EHLO backup-agent.example.com
S: 250-mail.example.com Hello backup-agent.example.com
   250-SIZE 52428800
   250-8BITMIME
   250 STARTTLS
C: SAIP: id="acme.mailer.relay-bg-01"; alg="ed25519";
         ts="1744200000"; nonce="7f3k9p2m";
         pk="<base64url_public_key>"; sig="<base64_sig>"
S: (SAIP middleware validates signature, timestamp, nonce)
   If invalid:
   550 5.7.1 SAIP verification failed: Invalid agent identity
   If valid:
   (continue normally)
C: MAIL FROM:<sender@example.com>
```

The following apply:

- If the receiving server does not implement SAIP, it MUST treat the SAIP line as an unrecognized command and MUST continue normally per [RFC5321]. Sending servers MUST NOT rely on SAIP being enforced by receiving servers.

- Receiving servers that implement SAIP MAY reject the connection with "550 5.7.1" (permanent failure) or "421 4.7.1" (temporary failure) based on local policy.

- For strict environments, a future SMTP service extension advertising "SAIP" capability in the EHLO response is RECOMMENDED to enable negotiated enforcement.

- SAIP for SMTP provides early rejection before any message data is transferred, reducing server resource consumption from illegitimate connections.

14. Security Considerations

14.1. Timestamp Validation

Servers MUST reject requests where the ts= parameter differs from the server's current time by more than 300 seconds. This window SHOULD be configurable and SHOULD be as small as the deployment's clock synchronization accuracy allows.

14.2. Nonce Requirements

The nonce= parameter MUST be generated using a cryptographically secure random number generator per [RFC4086]. Servers SHOULD maintain a nonce cache within the ts= validity window for sensitive endpoints to prevent replay attacks.

14.3. Constant-Time Verification

All signature comparison operations MUST use constant-time algorithms to prevent timing side-channel attacks.

14.4. Key Storage Requirements

Master Keys SHOULD be stored in hardware-backed secure storage:

- TPM 2.0 (Windows, Linux servers and embedded devices)
- Apple Secure Enclave (macOS, iOS)
- Android StrongBox (Android devices)
- Hardware Security Module (HSM) for high-assurance deployments

Rolling Keys MUST be derived on demand and MUST be destroyed immediately after use. Rolling Keys MUST NOT be persisted to any storage medium.

14.5. Compromise Scope Limitation

Per-request RKDF rotation limits the cryptographic value of any stolen Rolling Key material to at most one in-flight request. Vendors MUST revoke the affected InstanceID via the RE upon confirmed or suspected compromise. Revocation MUST propagate to all active REs within 300 seconds.

Servers SHOULD report anomalous agent behavior (geographic anomalies, nonce irregularities, sequence inconsistencies, abnormal request patterns) to the relevant vendor or RE to enable early compromise detection.

14.6. Header Injection Prevention

By restricting the id= character set to the characters a-z, 0-9, '.', '_', and '-', SAIP prevents header injection attacks and parsing ambiguities in all target protocols.

14.7. Downgrade Attack Prevention

Per Section 7.5, renegotiation is strictly forward-only. Implementations MUST reject any attempt to negotiate a weaker algorithm value or to decrease a sequence number.

14.8. Counterfeit Prevention

When MasterKey storage is hardware-bound (TPM, HSM, Secure Enclave), the cryptographic identity cannot be exported or cloned. Counterfeit agents lacking access to the original hardware cannot produce valid SAIP signatures.

14.9. Remote Attestation

Implementations supporting hardware-bound keys SHOULD provide Remote Attestation evidence per [RFC9334] during Full Renegotiation (Section 7.3). Attestation status MAY be committed in the tm= telemetry field (Section 8.3).

15. Privacy Considerations

The id= parameter discloses the vendor and agent instance identity to every server receiving the request and to any network observer if the connection is not encrypted. Connections carrying SAIP headers SHOULD use TLS [RFC8446] or equivalent transport-layer encryption.

Agents operating in contexts where identity disclosure is undesirable MAY use stateless mode (pk=) with ephemeral key pairs, accepting the loss of RE-based revocation capability.

The tm= telemetry field MUST NOT be transmitted without explicit opt-in. No metadata is disclosed in transport; only the opaque

commitment hash is visible. Raw metadata is accessible only to entities possessing the Rolling Key, the MasterKey, or authorization from an RE Audit Node.

SAIP does not provide anonymity. Agents that require anonymity SHOULD NOT implement SAIP.

16. IANA Considerations

This document requests IANA to register the following HTTP header field in the "Permanent Message Header Field Names" registry [RFC9110]:

Header field name:	SAIP
Applicable protocol:	http
Status:	standard
Author/Change controller:	IETF
Specification:	This RFC

This document requests IANA to register the following HTTP header field in the "Permanent Message Header Field Names" registry:

Header field name:	SAIP-Next-Seq
Applicable protocol:	http
Status:	standard
Author/Change controller:	IETF
Specification:	This RFC

This document requests IANA to register the following HTTP header field in the "Permanent Message Header Field Names" registry:

Header field name:	SAIP-Key-Version
Applicable protocol:	http
Status:	standard
Author/Change controller:	IETF
Specification:	This RFC

This document requests IANA to create the following new registry:

Registry name:	SAIP Registration Entity Bootstrap Registry
Registration policy:	Expert Review
Initial contents:	To be determined through IETF consensus process.

This document requests IANA to create the following new registry:

Registry name:	SAIP Telemetry Field Names
Registration policy:	Specification Required
Initial contents:	asn, geo, attest, fw, gps, api_hash as defined in Section 8.3 of this document.

17. References

17.1. 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>.

[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>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS) Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <https://www.rfc-editor.org/info/rfc9334>.

[RFC5321] Klenke, J., "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008, <https://www.rfc-editor.org/info/rfc5321>.

17.2. Informative References

[RFC9421] Backman, A., Ed., Richer, J., Ed., and M. Sporny, "HTTP Message Signatures", RFC 9421, DOI 10.17487/RFC9421, February 2024, <https://www.rfc-editor.org/info/rfc9421>.

[RFC6376] Crocker, D., Ed., Hansen, T., Ed., and M. Kucherawy, Ed., "DomainKeys Identified Mail (DKIM) Signatures", STD 76, RFC 6376, DOI 10.17487/RFC6376, September 2011, <https://www.rfc-editor.org/info/rfc6376>.

[RFC7208] Kitterman, S., "Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1", RFC 7208, DOI 10.17487/RFC7208, April 2014, <https://www.rfc-editor.org/info/rfc7208>.

[RFC7489] Kucherawy, M., Ed., and E. Zwicky, Ed., "Domain-based Message Authentication, Reporting, and Conformance (DMARC)", RFC 7489, DOI 10.17487/RFC7489, March 2015, <https://www.rfc-editor.org/info/rfc7489>.

[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>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[TPM20] Trusted Computing Group, "TPM Library Specification, Family 2.0", Trusted Computing Group, 2019, <https://trustedcomputinggroup.org/>.

[SAIP-00] Jovanevi, S., "SAIP: Signed Agent Identity Protocol", draft-jovancevic-saip-00, April 2026, <https://datatracker.ietf.org/doc/draft-jovancevic-saip/>.

[GENESIS] Jovanevi, S., "Project Genesis — SAIP: Design Rationale, Use Cases, and Implementation Guidance", GitHub, April 2026, <https://github.com/sreckojovancevic/Project-Genesis/blob/main/Readme2.0.md>.

Appendix A. IoT Use Case (Informative)

This appendix is informative.

SAIP is particularly well-suited for IoT deployments where device identity is critical and hardware security modules are increasingly available as standard components.

Consider a home appliance manufacturer deploying a cloud-connected product fleet:

```
Vendor:       Manufacturer (registered at RE, holds Master Key)
Agent Type:   Product line (e.g., washing machines)
Instance:     Individual device (identity bound to on-device TPM)
```

Without SAIP, the cloud service has no reliable way to distinguish a legitimate device from an attacker spoofing the device’s User-Agent string or API credentials.

With SAIP, the following properties hold:

- Each device carries a unique InstanceID cryptographically bound to its hardware TPM, making the identity non-exportable and non-clonable.
- If a batch of devices receives a compromised firmware update, the manufacturer revokes only the affected InstanceIDs via the RE. The remainder of the fleet continues operating without interruption.
- Counterfeit devices cannot produce valid SAIP signatures because they lack access to the original hardware TPM and the Master Key derivation.
- The tm= telemetry field MAY carry firmware version, hardware attestation status, and network ASN — committed as an opaque hash, private in transport, and available on-demand for audit and compliance purposes.
- This approach aligns with emerging regulatory requirements for verifiable IoT device identity, including the EU Cyber Resilience Act (CRA) and similar frameworks.

Appendix B. Relationship to Existing Standards (Informative)

This appendix is informative.

B.1. RFC 9421 (HTTP Message Signatures)

RFC 9421 [RFC9421] defines a mechanism for signing HTTP message components — headers, body, and derived values — to provide message integrity and non-repudiation.

SAIP and RFC 9421 are complementary and non-conflicting:

- SAIP establishes WHO is making the request (identity layer).
- RFC 9421 establishes integrity of WHAT is being transmitted (message layer).

SAIP MAY be used in conjunction with RFC 9421. SAIP MUST NOT be presented as a replacement for RFC 9421.

B.2. DKIM (RFC 6376)

DKIM [RFC6376] provides cryptographic signatures for email messages at the domain level, proving domain responsibility for message content. SAIP provides cryptographic identity for agents at the individual instance level. These operate at different layers and are complementary in SMTP deployments.

B.3. RATS (RFC 9334)

The RATS architecture [RFC9334] defines procedures and data formats for remote attestation of hardware and software state. SAIP's hardware key storage requirements (Section 14.4) and the tm= attestation field (Section 8.3) are designed to be compatible with RATS attestation evidence formats, enabling SAIP implementations to leverage existing RATS infrastructure.

Author's Address

Sreko Jovanevi
SKGO, IKT Support
Makedonska 22
11000 Belgrade
Serbia

Email: srecko.jovancevic@skgo.org
Email: srecko.jovancevic@gmail.com
URI: https://github.com/sreckojovancevic