Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 20 October 2026

S. Jovancevic
SKGO, IKT Support

20 April 2026

SAIP: Signed Agent Identity Protocol
draft-jovancevic-saip-08

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 implements the principles defined in the Verifiable Identity Claims and Delegation Model [VICDM] and 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. It introduces DNS-based Attestation Discovery as a lightweight alternative to registry-based key lookup, making deployment accessible to organizations of any size.

Table of Contents

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.

1.1. Design Philosophy: The VICDM Principle

SAIP is grounded in the Verifiable Identity Claims and Delegation Model [VICDM], which establishes the following core principle:

> Anonymous interaction is permitted.
> Identity assertion is permitted.
> False identity assertion is not.

This principle has a critical implication for policy:

An agent that asserts an identity it cannot prove is more dangerous than an agent that asserts no identity at all. The former actively corrupts trust signals; the latter is simply anonymous. Systems implementing SAIP SHOULD treat unverifiable identity claims with lower trust than genuine anonymous clients.

SAIP is the protocol-layer implementation of this principle: it provides the cryptographic mechanism by which identity assertions become verifiable.

1.2. Relationship to Anonymous Authentication

Anonymous bot authentication systems (such as Privacy Pass and related mechanisms under development in the IETF webbotauth working group) allow a bot to prove it is vouched for by a trusted attester without revealing its specific identity.

SAIP and anonymous authentication are complementary:

- Anonymous authentication addresses the privacy use case.
- SAIP addresses the accountability use case.

The web needs both. Sites that require per-instance revocation, audit trails, and fleet management need SAIP. Sites whose primary concern is privacy-preserving rate limiting may prefer anonymous attestation. Many deployments will benefit from both.

1.3. Changes from draft-jovancevic-saip-02

This document adds the following relative to -02:

- Explicit alignment with [VICDM] principles (Section 1.1)
- DNS-based Attestation Discovery (Section 10.2) as a lightweight alternative to RE registry lookup
- DNS delegation record format (Section 10.2)
- VICDM identity class mapping (Appendix C)
- DNS record examples (Appendix D)
- Clarified relationship to anonymous authentication (Section 1.2)

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.

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 or publishes keys via DNS.

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.

VICDM: Verifiable Identity Claims and Delegation Model. The conceptual framework [VICDM] that defines the principles this protocol implements.

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 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.

False Identity: Under [VICDM], an agent asserting an identity it cannot prove is categorically more harmful than an anonymous agent. Current protocols provide no mechanism to distinguish or penalize false identity claims.

4. Design Goals

Simplicity Minimal overhead, single-header implementation.

(KISS):	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.
Protocol-Agnostic:	Applicable to HTTP, SMTP, and other header-based protocols.
Granular Control:	Policy decisions MUST be applicable at vendor, agent type, and instance level independently.
Identification Only:	SAIP is strictly an identification protocol. It MUST NOT be used as an authorization mechanism.
Accessible Deployment:	DNS-based Attestation Discovery enables organizations of any size to deploy SAIP without RE registration.

5. Protocol Overview

5.1. Identification vs. Authorization

SAIP provides a verifiable cryptographic assertion of WHO an agent is. It does not determine what that agent is permitted to do.

Authorization remains the exclusive responsibility of existing application-layer mechanisms such as OAuth 2.0 [RFC6749], JWT, or ACLs.

Under [VICDM], SAIP clients with successful verification are Class 3 (Fully Verified). See Appendix C for the full mapping.

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.

5.3. Parameters

Param	Type	Required	Description
id	String	MUST	Agent instance identifier. Allowed chars: a-z, 0-9, ., _, -. Maximum 128 characters. RECOMMENDED format: vendor.type.instance Where vendor component SHOULD correspond to the domain used in DNS-based Attestation Discovery.
alg	String	MUST	Algorithm: "ed25519" or "hmac-sha256". Implementations MUST support "ed25519".

ts	Integer	MUST	Unix timestamp at signing.
nonce	String	MUST	Per-request unique value, minimum 8 chars, cryptographic randomness per [RFC4086].
sig	Base64	MUST	Signature over canonical string per Section 6.
pk	Base64URL	MAY	Public key for stateless verification.
tm	Base64	MAY	Opt-in telemetry commitment. See Section 8. MUST NOT be present unless explicitly enabled by deployment policy.

Implementations MUST reject headers missing any MUST parameter. Implementations MUST silently ignore unknown parameters.

6. Canonicalization and Signature

6.1. HTTP Canonical String

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

Where <METHOD> is uppercase and <path> includes the 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

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

6.3. Other Protocols

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

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.

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

Where Seq_n is a big-endian unsigned 64-bit integer.

The Rolling Key MUST be used to sign exactly one request and MUST be destroyed immediately after use.

The MasterKey SHOULD be stored in hardware-backed secure storage (TPM 2.0, HSM, Apple Secure Enclave, Android StrongBox).

7.2. Sequence Window

Servers MUST implement a configurable look-ahead sequence window:

- Default window size SHOULD be 5.
- Maximum window size MUST NOT exceed 10.
- Received sequence numbers within the window cause the server to advance state and invalidate prior sequences.
- Sequences outside the window MUST be rejected.

Upon successful verification, servers SHOULD respond with:

```
SAIP-Next-Seq: <n+1>
```

Clients MUST advance sequence state on receipt of this header.
On missing response, clients MUST retry with the same Rolling Key.

7.3. Renegotiation and Recovery

Full Renegotiation MAY be initiated when a client loses sequence state entirely.

Full Renegotiation MUST be authorized exclusively by an RE.
Endpoint servers MUST NOT authorize Full Renegotiation.

The Full Renegotiation process:

1. Client sends a Renegotiation Request to RE containing:
   - InstanceID, reason code, timestamp, nonce
   - Signature using previous MasterKey (valid window: 24 hours from ts value)
   - Hardware Attestation evidence per [RFC9334] (RECOMMENDED)

2. RE MUST validate:
   - MasterKey signature valid and within validity window
   - InstanceID not revoked
   - Rate limit: maximum 3 renegotiation requests per InstanceID per 24-hour period
   - Hardware Attestation consistent with registered hardware (if provided)

3. On success, RE issues new RKDF seed and starting Sequence. Previous sequence range is permanently invalidated.

4. Vendor MUST be notified of all Full Renegotiation events.

7.4. Graceful Key Rotation

Key transitions use the SAIP-Key-Version header:

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

- Servers with new key MUST accept new key version.
- Servers without new key SHOULD accept fallback until expiry.
- After fallback-expires, old key MUST be rejected.

7.5. Forward-Only Constraint

Implementations MUST enforce:

- Sequence numbers MUST NOT decrease.
- Algorithm MUST NOT downgrade during renegotiation.
- Full Renegotiation MUST be RE-authorized.
- Violations MUST be rejected and logged.

8. Opt-In Telemetry (tm= field)

8.1. Commitment-Based Model

The tm= field MUST NOT be transmitted without explicit opt-in.

tm = HMAC-SHA256(RollingKey, Metadata_String)

Where Metadata_String is a semicolon-separated key=value string.
The tm= value is computed using the same RollingKey as sig=.
Raw metadata MUST NOT appear in the header.

Example:

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

8.2. Audit-on-Demand

Servers SHOULD store tm= in audit logs. During incident response:

1. Operator obtains raw Metadata_String from client or RE log.
2. Verifier recomputes HMAC-SHA256(RollingKey, Metadata_String).
3. Match confirms metadata authenticity.

8.3. Telemetry Field Registry

RECOMMENDED standard fields (IANA registry requested):

asn	Autonomous System Number
geo	ISO 3166-1 alpha-2 country code
attest	Hardware attestation status
fw	Firmware or software version
gps	Physical coordinates (IoT)
api_hash	HMAC of associated API credential

9. Processing Model

9.1. Client Processing

1. Construct canonical string per Section 6.
2. Derive current Rolling Key via RKDF (Section 7.1).
3. Sign canonical string using Rolling Key and alg=.
4. If telemetry enabled, compute tm= per Section 8.1.
5. Construct and transmit SAIP header.
6. On receiving SAIP-Next-Seq response: advance Sequence state, destroy used Rolling Key immediately.
7. On missing response: retry with same Rolling Key and Sequence.

9.2. Server Processing

1. Parse SAIP header. Reject if any MUST parameter is absent.

2. Validate ts=. MUST reject if |server_time - ts| > 300 seconds.

3. Validate nonce. SHOULD track nonces within validity window for sensitive endpoints.

4. Obtain public key via discovery order (Section 10.4).

5. Reconstruct canonical string and verify signature using constant-time comparison.

6. On success, include SAIP-Next-Seq in response.

7. Apply policy per Section 12 based on verified identity.

8. If tm= present and audit mode active, store in audit log.

9. Classify client per [VICDM] identity classes:
   - Verified signature: Class 3 (Fully Verified)
   - DNS partial match only: Class 2 (Partially Verified)
   - No SAIP header: Class 0 (Anonymous)
   - Invalid/unverifiable: Class 1 (treat below Class 0)

10. Trust and Attestation Discovery

10.1. Stateless Mode (pk=)

If pk= is present, the server MAY verify immediately without external lookup.

Servers SHOULD verify pk= consistency with prior observations for the same id= to detect key substitution attacks.

10.2. DNS-Based Attestation Discovery

DNS-based Attestation Discovery allows vendors to publish a verifiable attestation statement — including cryptographic key material, authorized infrastructure, and preferred Registration Entity — using existing DNS infrastructure, without requiring prior RE registration.

This mechanism is inspired by the DNS-based key publication patterns established by DKIM [RFC6376] and SPF [RFC7208], and extends them to the agent identity layer.

Servers perform a DNS TXT query at:

_saip.<vendor-domain>.

Where <vendor-domain> is derived from the vendor component of the id= parameter (the leftmost label before the first dot).

Example: for id="acme.crawler.nyc-042", the server queries:

_saip.acme.

The DNS TXT record format follows [VICDM] Section 6.2:

_saip.<domain>. IN TXT "v=saip1; [parameters]"

Defined parameters:

v=saip1	MUST be present. Version indicator.
pk=<key>	Base64URL-encoded Ed25519 public key. Used directly for signature verification.
re=<host>	Preferred RE hostname for registry-based lookup. MAY be used as fallback if pk= verification fails.
asn=<list>	Comma-separated authorized ASNs for delegated infrastructure per [VICDM] Section 6.1.
ip=<prefix>	CIDR prefix of authorized delegated infrastructure. Multiple ip= parameters are permitted.
exp=<ts>	Unix timestamp after which this record SHOULD be considered expired. Servers SHOULD NOT use expired records for verification.

Example records:

; Simple public key publication

```
_saip.acme.com. IN TXT "v=saip1; pk=MCowBQYDK2Vd..."

; With preferred RE
_saip.acme.com. IN TXT "v=saip1; pk=MCowBQYDK2Vd...;
                        re=re1.saip-registry.example"

; With delegated infrastructure authorization
_saip.acme.com. IN TXT "v=saip1; pk=MCowBQYDK2Vd...;
                        asn=13335,15169"
```

DNS-based verification requirements:

- Servers MUST validate DNSSEC signatures where available [RFC4033].
- Servers MUST NOT use DNS responses with TTL of 0 for key material.
- Servers SHOULD cache DNS key material according to the record TTL. Default TTL SHOULD be 3600 seconds.
- If the DNS record contains asn= or ip= parameters, the server SHOULD verify that the client's network source is within the authorized set before accepting the key.
- If the DNS record has expired (exp= in the past), the server MUST NOT use the key material for verification and SHOULD treat the request as Class 1 (unverifiable claim) per [VICDM].

10.3. Registry-Based Discovery

The server uses the id= parameter to look up the public key from a distributed RE registry.

Implementations MUST support caching of registry responses: minimum TTL 60 seconds, maximum TTL 3600 seconds.

10.4. Discovery Priority Order

When multiple discovery methods are available, servers MUST apply them in the following priority order:

1. pk= parameter (stateless, highest priority)
2. DNS-based Attestation Discovery (_saip.<domain> TXT record)
3. RE registry lookup (using id=)

Servers MAY skip lower-priority methods once a valid key is found. Servers MUST NOT accept a request if no valid key can be found through any available method.

Servers MAY skip lower-priority methods once a valid key is found. Servers MUST NOT accept a request if no valid key can be found through any available method.

10.5. DNS-Native Mode with Rolling Key Certification

The DNS-Native mode implements the alternative trust model defined in [VICDM] Section 6.4. It replaces the RE registry with the vendor's DNS zone as the identity trust anchor.

The DNS-Join trust model rests on three principles: hardware secures the Master Key, DNS publishes it and acts as the trust circuit breaker, and Rolling Keys provide speed and network efficiency.

10.5.1. Additional Header Parameters

DNS-Native mode introduces two additional SAIP header parameters:

Param	Type	Required	Description
rpk	Base64URL	MUST (DNS-Native mode)	Rolling Public Key. A fresh ephemeral Ed25519 public key generated per request.
rcert	Base64	MUST (DNS-Native mode)	Rolling Key Certificate. Signature by MasterPrivateKey over rpk + instanceID + ts + nonce + method + path. Binds rpk to exactly one request. Non-replayable.

When operating in DNS-Native mode, the sig= parameter MUST be computed using the Rolling Private Key (corresponding to rpk=) rather than an RKDF-derived key. The pk= parameter MUST NOT be present simultaneously with rpk= and rcert=.

10.5.2. DNS-Native Header Example

```
SAIP: id="acme.agent-nyc-042";
      alg="ed25519";
      ts="1744200000";
      nonce="7f3k9p2m";
      rpk="<base64url_rolling_pubkey>";
      rcert="<base64_rolling_key_cert>";
      sig="<base64_sig_by_rolling_privkey>"
```

10.5.3. Rolling Key Certificate Format

The rcert= value MUST be computed as:

```
rcert = Sign(MasterPrivateKey,
             rpk        ||
             instanceID ||
             ts         ||
             nonce      ||
             method     ||
             path)
```

Where || denotes byte-level concatenation and all string values are UTF-8 encoded. The inclusion of nonce, method, and path binds the rcert= to exactly one request, making replay attacks structurally impossible regardless of the rcert= validity period.

10.5.4. Server Processing for DNS-Native Mode

When rpk= and rcert= are present, servers MUST:

1. Extract the vendor domain from the id= parameter (leftmost component before the first dot).

2. Perform DNS TXT lookup at:
   <instance-label>._saip.<vendor-domain>.
   to retrieve the Master Public Key. Cache per DNS TTL. MUST validate DNSSEC where available [RFC4033].

3. Verify rcert= using the retrieved Master Public Key. This confirms rpk= is authorized by the Master Key holder for this exact request.

4. Verify sig= using rpk=.
   This confirms the request was signed by the holder of the corresponding Rolling Private Key.

5. Both verifications MUST succeed. Either failure MUST result in request rejection and SHOULD be logged.

6. Servers MUST NOT cache Master Public Keys beyond the DNS TTL of the source record.

10.5.5. Security Considerations for DNS-Native Mode

Replay Protection:	rcert= is bound to nonce + ts + method + path. Each rcert= is valid for exactly one request. Replay is structurally impossible.
Key Compromise:	Compromise of a Rolling Private Key exposes exactly one request. The Master Private Key remains in hardware and is never transmitted.
Revocation:	Vendor deletes the agent's _saip DNS record. After TTL expiry (RECOMMENDED: 300 seconds), all servers reject the agent's rcert= values.
DNS Security:	Servers MUST use DNSSEC [RFC4033] where available. _saip record TTL SHOULD be 300 seconds to bound the revocation window.
Comparison with TLS:	This model provides stronger replay protection than TLS Session Tickets [RFC5077] because rcert= is bound per-request rather than per time-window, while retaining equivalent performance via DNS caching.

10.5.6. MAC-Based Identity Binding (Optional Extension)

This section specifies the MAC identity binding extension defined conceptually in [VICDM] Section 6.4.7. When implemented, it adds a hardware identity layer to the DNS-Native trust model.

10.5.6.1. Additional Header Parameters

MAC binding introduces the following optional SAIP header parameters:

Param	Type	Required	Description
mac	String	MAY	MAC address in XX:XX:XX:XX:XX:XX format. Real or Virtual (V-MAC). MUST be signed via mac_proof.
mac_proof	Base64	MUST if mac= present	MAC Attestation Proof. Sign(MasterPrivateKey, type||mac_bytes||ts||nonce). Proves MAC was attested by MasterKey holder.

mac= and mac_proof= MUST always appear together. mac= without mac_proof= MUST be rejected as an unverifiable identity claim per [VICDM] Class 1.

10.5.6.2. Virtual MAC (V-MAC) Derivation

For agents without a meaningful physical MAC address, a V-MAC MUST be derived from the MasterPublicKey as follows:

```
raw   = SHA256(MasterPublicKey)[0:6]
v_mac = (raw[0] | 0x02) || raw[1:6]
```

The LAA bit (bit 1 of octet 0, value 0x02) distinguishes V-MAC from globally assigned hardware MAC addresses per IEEE 802.

Servers MAY verify V-MAC derivation independently:

```
(SHA256(MasterPublicKey)[0] | 0x02) == mac[0]
AND SHA256(MasterPublicKey)[1:6]    == mac[1:6]
```

This provides a secondary cryptographic binding between the V-MAC and the MasterPublicKey without requiring the mac_proof.

10.5.6.3. MAC Attestation Proof Format

The mac_proof= value MUST be computed as:

For Real MAC:

```
mac_proof = Sign(MasterPrivateKey,
                 "real"    ||
                 mac_bytes ||
                 ts        ||
                 nonce)
```

For V-MAC:

```
mac_proof = Sign(MasterPrivateKey,
                 "virtual" ||
                 mac_bytes ||
                 ts        ||
                 nonce)
```

Where:

- "real" and "virtual" are UTF-8 encoded type prefixes
- mac_bytes is the 6-byte MAC in big-endian, no separators
- ts and nonce are the same values as in the SAIP header
- || denotes byte-level concatenation

The inclusion of ts and nonce binds mac_proof to this specific request, preventing mac_proof replay across different requests.

10.5.6.4. Complete DNS-Native MAC Header Example

```
; Real MAC (IoT device, TPM-bound key)
SAIP: id="manufacturer.washer.wm-bg-4471";
      alg="ed25519";
      ts="1744200000";
      nonce="7f3k9p2m";
      mac="B8:27:EB:4A:3C:11";
      mac_proof="<Sign(MasterKey,'real'||mac||ts||nonce)>";
      rpk="<RollingPubKey>";
      rcert="<RollingKeyCert>";
      sig="<RollingKeySig>"

; V-MAC (cloud crawler, software or HSM key)
SAIP: id="acme.crawler.nyc-042";
      alg="ed25519";
      ts="1744200000";
      nonce="a1b2c3d4";
      mac="AE:CE:7F:3A:B1:09";
      mac_proof="<Sign(MasterKey,'virtual'||mac||ts||nonce)>";
      rpk="<RollingPubKey>";
      rcert="<RollingKeyCert>";
```

```
sig="<RollingKeySig>"
```

10.5.6.5. Server Verification with MAC Binding

When mac= and mac_proof= are present, servers MUST perform the following additional verification steps after standard SAIP verification (Section 9.2):

1. Check DNS record for mac= field:
   Verify that the mac= value in the SAIP header matches the mac= value in the agent's DNS TXT record.
   Mismatch MUST result in rejection.

2. Verify mac_proof= using MasterPublicKey:
   Reconstruct the signed payload:
   type_prefix || mac_bytes || ts || nonce
   Verify mac_proof signature with MasterPublicKey.
   Failure MUST result in rejection.

3. For V-MAC: optionally verify derivation:
   Check SHA256(MasterPublicKey)[0:6] against mac= value (with LAA bit applied). Mismatch SHOULD be logged.

4. If ip= or asn= present in DNS record:
   Compare against request source address.
   Mismatch SHOULD trigger trust downgrade and logging but MAY be accepted under deployment policy.

5. On full triplet verification success:
   Classify as Class 3+ (Fully Verified with Hardware Attestation) per [VICDM] Section 6.4.7.

10.5.6.6. Privacy Considerations for MAC Binding

Real MAC addresses reveal hardware manufacturer information via the OUI (Organizationally Unique Identifier — first 3 octets). Implementations using Real MAC:

- MUST transmit SAIP headers over TLS [RFC8446] to prevent MAC disclosure to network observers.
- SHOULD consider whether OUI disclosure is acceptable under applicable privacy policy.

V-MAC reveals no hardware manufacturer information and is RECOMMENDED for privacy-sensitive deployments.

MAC binding is OPTIONAL. Agents that do not include mac= and mac_proof= operate in standard SAIP mode and are not penalized for absence of MAC binding.

10.6. Agent Letter of Intent (ALOI)

This section implements the ALOI mechanism defined in [VICDM] Section 6.5. ALOI enables agents to publish a cryptographically bound behavioral declaration at registration time.

10.6.1. ALOI Header Parameter

When an agent operates under an ALOI declaration, it MUST include the following parameter in every SAIP header:

Param	Type	Required	Description
aloi-hash	Base64	MUST if ALOI	SHA256 of the canonical ALOI string from the agent's DNS

```
|           |          | declared| TXT record. Binds this       |
|           |          |         | request to the declared ALOI.|
+-----------+----------+---------+------------------------------+
```

The aloi-hash= MUST be included in the canonical string that is signed by the Rolling Key:

```
id=<id>;ts=<ts>;nonce=<nonce>;method=<METHOD>;path=<path>;
aloi-hash=<aloi-hash>
```

This creates cryptographic proof that the agent is operating under its declared ALOI for this specific request.

10.6.2. DNS Record Format

ALOI parameters are published in the agent's DNS TXT record using the "aloi-" prefix. See [VICDM] Section 6.5.4 for complete examples. A summary of defined parameters:

```
aloi=          Primary intent: crawl, backup, monitor,
               api, smtp, iot, sync
aloi-scope=    Declared access scope (glob patterns)
aloi-rate=     Self-imposed rate limit (e.g., 10/sec)
aloi-exclude=  Paths the agent declares it will not access
aloi-ua=       Declared User-Agent string
aloi-expires=  ALOI expiry timestamp
```

10.6.3. Server Enforcement

Servers implementing ALOI enforcement MUST:

1. Verify aloi-hash= matches SHA256 of the DNS ALOI record. Mismatch MUST be treated as a critical violation.

2. Verify request path matches aloi-scope=.

3. Verify request path does not match aloi-exclude=.

4. Verify request rate does not exceed aloi-rate=.

5. Verify User-Agent matches aloi-ua= (if declared).

On violation detection, servers MUST include in the response:

```
SAIP-Violation: <type>; declared=<val>; attempted=<val>;
                penalty=<action>
```

Defined violation types per [VICDM] Section 6.5.6: scope-exceeded, exclusion-breach, rate-exceeded, ua-mismatch, aloi-expired, aloi-hash-mismatch.

10.6.4. Violation Scoring and Sanctions

Servers SHOULD apply progressive sanctions per the model defined in [VICDM] Section 6.5.7:

```
1 violation:   Warning only (SAIP-Violation header + log)
2-3:           Throttle to minimal rate
4-5:           Trust downgrade to Class 1
6-10:          Block all requests
>10:           Report to RE for InstanceID revocation
```

Violation scores SHOULD decay at -1 point per 24 hours of fully compliant behavior, allowing agents to recover trust.

10.6.5. Good Faith Self-Reporting

Agents that detect they have unintentionally violated their ALOI declaration MAY self-report before server detection, using the Good Faith Mechanism defined in [VICDM] Section 6.5.8.

Self-reporting is performed by including in the next request:

```
SAIP-Self-Report: violation=<type>;
                  declared=<declared-value>;
                  attempted=<attempted-value>;
                  ts=<timestamp-of-violation>;
                  self-sig=<Sign(MasterPrivateKey,
                               type||declared||attempted||
                               ts||nonce)>
```

The self-sig= MUST be verifiable using the MasterPublicKey retrieved from DNS. This proves the self-report is authentic.

Self-reported violations MUST receive reduced penalties per [VICDM] Section 6.5.8. Agents that consistently self-report SHOULD receive elevated trust scores, as self-reporting demonstrates responsible vendor implementation.

This mechanism implements the voluntary disclosure principle: proactive transparency is rewarded; concealment is penalized.

10.6.6. RE Violation Reporting

Servers MAY report violations to the RE for ecosystem-wide awareness. The violation report MUST be signed by the server to prevent fabrication:

```
POST <re-endpoint>/violation
{
  "instance-id": "<agent-id>",
  "violation":   "<violation-type>",
  "declared":    "<declared-value>",
  "attempted":   "<attempted-value>",
  "ts":          <unix-timestamp>,
  "server-sig":  "<Sign(ServerKey, report-hash)>"
}
```

The RE MAY aggregate violation reports from multiple servers to build a cross-ecosystem behavioral profile of agent instances, enabling coordinated revocation when patterns of abuse emerge across multiple independent servers.

11. Registration Entity (RE) Requirements

11.1. RE Eligibility

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

Global Infrastructure:	Operates internet-scale infrastructure with sufficient geographic distribution to serve RE queries globally.
Operational Track Record:	Demonstrated history of reliably operating critical internet infrastructure.
Neutrality Commitment:	Formal agreement not to discriminate against any vendor, agent type, or competing RE.

Community Approval: Majority consensus approval from existing active REs via cryptographically signed statements.

An RE MUST maintain query availability of no less than 99.9% 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.

A new RE MAY be added when a strict majority of existing active REs approve via signed statements in the RE audit log.

An RE MUST be removed if:

- Availability requirement fails for more than 72 consecutive hours, OR
- Neutrality commitment is violated (majority vote), OR
- RE ceases operations voluntarily.

All RE membership changes MUST be recorded in a publicly accessible, append-only, verifiable audit log.

11.3. Bootstrap and Evolution

The initial RE set is defined in the SAIP RE Bootstrap Registry (IANA, Section 16). The bootstrap set SHOULD include:

- Regional Internet Registries (RIPE NCC, ARIN, APNIC, LACNIC, AFRINIC) — non-profit, geographically distributed, operationally neutral by mandate.

- Academic and neutral institutions (Internet Society, ICANN, established research universities) — independent, non-commercial representation.

- Infrastructure operators meeting Section 11.1 criteria.

No single commercial organization category SHALL hold a majority of bootstrap RE positions.

Vendors register with any RE of their choice. Servers configure which REs they trust. DNS-based Attestation Discovery (Section 10.2) is available as a zero-registration alternative.

12. Granular Policy Model

SAIP enables policy enforcement at three levels independently:

Vendor Level: All agents and instances of a given vendor.

Agent Type Level: All instances of a specific agent type from a given vendor.

Instance Level: Exactly one agent instance (full id= value).

Implementations MUST support instance-level revocation without affecting other instances of the same vendor or agent type.

Policy actions include:

- BLOCK: Reject all requests from the targeted entity.

- THROTTLE: Apply rate limits to the targeted entity.
- DEGRADE: Reduce trust classification.
- ALLOW: Grant priority or elevated rate limits.

VICDM identity class mapping (see Appendix C) SHOULD inform the base trust level before granular policy is applied.

Informative traffic classification:

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
False Claim	Unverifiable	Below Minimal	Reject or 0.1 req/sec

13. SMTP Integration

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

The sending server MAY transmit a SAIP header line immediately after the server’s EHLO response and before MAIL FROM.

The SMTP canonical string per 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: (validates signature, timestamp, nonce)
   If invalid: 550 5.7.1 SAIP verification failed
   If valid: (continue)
C: MAIL FROM:<sender@example.com>
```

- Non-implementing receivers MUST treat SAIP line as unrecognized command and continue per [RFC5321].
- Implementing receivers MAY reject with 550 5.7.1 (permanent) or 421 4.7.1 (temporary) based on local policy.
- A future SMTP extension advertising SAIP capability in EHLO is RECOMMENDED for strict environments.
- SAIP-verified SMTP agents correspond to Class 3 per [VICDM]. Unverifiable SMTP identity claims correspond to Class 1 and SHOULD be treated with lower trust than anonymous senders.

14. Security Considerations

14.1. Timestamp Validation

Servers MUST reject requests where |server_time - ts| > 300s.

14.2. Nonce Requirements

Nonce MUST be generated via cryptographically secure RNG per [RFC4086]. Servers SHOULD track nonces within the validity window.

14.3. Constant-Time Verification

All signature comparisons MUST use constant-time algorithms.

14.4. Key Storage Requirements

Master Keys SHOULD be stored in hardware-backed storage: TPM 2.0, Apple Secure Enclave, Android StrongBox, or HSM.

Rolling Keys MUST be derived on demand and destroyed after use.

14.5. Compromise Scope Limitation

Per-request RKDF limits stolen key value to one in-flight request. Vendors MUST revoke affected InstanceIDs via RE. Revocation MUST propagate to all REs within 300 seconds.

14.6. Header Injection Prevention

Restricting id= to a-z, 0-9, '.', '_', '-' prevents header injection and parsing ambiguities.

14.7. Downgrade Attack Prevention

Per Section 7.5, renegotiation is strictly forward-only.

14.8. DNS Security

DNS-based Attestation Discovery MUST validate DNSSEC [RFC4033] where available. Implementations MUST NOT rely on unauthenticated DNS responses for key material in high-security deployments.

DNS cache poisoning could cause a server to accept a spoofed public key. DNSSEC validation and monitoring of unexpected key changes SHOULD be used to mitigate this risk.

14.9. False Identity Claims

Under [VICDM], clients asserting identities they cannot verify (Class 1) SHOULD receive lower trust than anonymous clients (Class 0). Implementations SHOULD log Class 1 interactions for audit purposes.

14.10. Remote Attestation

Implementations SHOULD provide Remote Attestation evidence per [RFC9334] during Full Renegotiation. Attestation status MAY be committed in tm= (Section 8.3).

15. Privacy Considerations

The id= parameter discloses vendor and instance identity to servers and network observers. Connections carrying SAIP headers SHOULD use TLS [RFC8446] or equivalent transport encryption.

DNS-based Attestation Discovery (Section 10.2) causes the server to make a DNS query for the vendor's domain, which may be observable. Implementations in sensitive deployments SHOULD use stateless mode (pk=) or RE-based discovery to avoid DNS-observable lookups.

The tm= field MUST NOT be transmitted without explicit opt-in.

SAIP does not provide anonymity. Agents requiring anonymity SHOULD use anonymous attestation mechanisms instead.

16. IANA Considerations

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

SAIP	(defined in Section 5.2)
SAIP-Next-Seq	(defined in Section 7.2)
SAIP-Key-Version	(defined in Section 7.4)

This document requests IANA to create the following registries:

SAIP RE Bootstrap Registry
- Registration policy: Expert Review
- Initial contents: To be determined through IETF consensus.

SAIP Telemetry Field Names
- Registration policy: Specification Required
- Initial contents: asn, geo, attest, fw, gps, api_hash (Section 8.3)

This document requests IANA to update the following registry created by [VICDM]:

VICDM DNS Delegation Record Parameters
- Add parameter: re (defined in Section 10.2 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>.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <https://www.rfc-editor.org/info/rfc4033>.

[VICDM] Jovanevi, S., "Verifiable Identity Claims and Delegation Model (VICDM)", draft-jovancevic-vicdm-04, April 2026, <https://datatracker.ietf.org/doc/draft-jovancevic-vicdm/>.

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)", RFC 7208, DOI 10.17487/RFC7208, April 2014, <https://www.rfc-editor.org/info/rfc7208>.

[RFC7489] Kucherawy, M., Ed., and E. Zwicky, Ed., "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>.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, DOI 10.17487/RFC5077, January 2008, <https://www.rfc-editor.org/info/rfc5077>.

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

[SAIP-02] Jovanevi, S., "SAIP: Signed Agent Identity Protocol", draft-jovancevic-saip-04, April 2026,

<https://datatracker.ietf.org/doc/
draft-jovancevic-saip/>.

Appendix A. IoT Use Case (Informative)

SAIP is particularly well-suited for IoT deployments:

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

DNS-based Attestation Discovery simplifies IoT deployment significantly: the manufacturer publishes _saip.manufacturer.com TXT record once, and all devices are immediately verifiable by any server without RE registration. The manufacturer's cloud service verifies device identity cryptographically; compromised devices are revoked at the instance level without fleet disruption.

Per [VICDM], verified IoT devices are Class 3. Counterfeit devices without TPM access fall to Class 1 and receive lower trust than anonymous devices — removing the incentive to impersonate legitimate products.

Appendix B. Relationship to Existing Standards (Informative)

B.1. RFC 9421 (HTTP Message Signatures)

RFC 9421 signs message content (WHAT). SAIP signs agent identity (WHO). Complementary, not competing.

B.2. DKIM (RFC 6376)

DKIM signs email messages at domain level. SAIP signs agents at instance level. The DNS TXT record pattern in Section 10.2 is directly inspired by DKIM's use of DNS for key publication.

B.3. SPF (RFC 7208)

SPF authorizes IP addresses to send mail for a domain using DNS TXT records. SAIP's DNS asn= and ip= delegation parameters in Section 10.2 serve an analogous purpose for agent identity.

B.4. RATS (RFC 9334)

SAIP's hardware attestation requirements (Section 14.10) and the tm= attestation field (Section 8.3) are compatible with RATS attestation evidence formats.

Appendix C. VICDM Alignment (Informative)

This appendix maps SAIP verification outcomes to VICDM identity classes [VICDM].

VICDM Class	SAIP Condition	Trust Level	Notes
Class 3	Valid signature via pk=, DNS, or RE	High	Full SAIP verification
Class 2	DNS record exists but no valid signature	Medium	Partial: domain identity only
Class 0	No SAIP header present	Low	Anonymous; apply default

			policy
Class 1	SAIP header present but unverifiable or signature invalid	Below Class 0	False claim; log; apply lower trust than anon

The Class 1 row is operationally significant: an agent that presents an invalid or unverifiable SAIP header MUST receive lower trust than one that presents no SAIP header at all. This removes the incentive to make false identity claims.

Appendix D. DNS Record Examples (Informative)

D.1. Minimal deployment (public key only)

```
_saip.example.com. 3600 IN TXT "v=saip1; pk=MCowBQYDK2VdAyEA..."
```

D.2. With preferred RE fallback

```
_saip.example.com. 3600 IN TXT "v=saip1; pk=MCowBQYDK2VdAyEA...;
                                re=re1.saip-registry.example"
```

D.3. CDN delegation by ASN (no pk, RE-based)

```
_saip.example.com. 3600 IN TXT "v=saip1; asn=13335,15169;
                                re=re1.saip-registry.example"
```

D.4. Full record with expiry

```
_saip.example.com. 3600 IN TXT "v=saip1; pk=MCowBQYDK2VdAyEA...;
                                re=re1.saip-registry.example;
                                asn=13335; exp=1767225600"
```

D.5. IoT manufacturer (devices have hardware-bound keys)

```
_saip.gorenje.com. 3600 IN TXT "v=saip1; re=re1.saip-registry.example"
```

In this case the pk= is not in DNS — each device instance has its own TPM-bound key registered directly at the RE under the vendor’s registration. DNS provides the RE discovery pointer.

In this case the pk= is not in DNS — each device instance has its own TPM-bound key registered directly at the RE under the vendor’s registration. DNS provides the RE discovery pointer.

Appendix E. Operational Guide — DNS-Join How-To (Informative)

This appendix provides practical guidance for administrators deploying SAIP in DNS-Native mode. All examples are illustrative. Production deployments MUST follow the security requirements defined in Sections 10.5 and 14.

E.1. Prerequisites

Before deploying SAIP DNS-Native mode, the following are required:

- Authoritative DNS control over your domain (ability to create TXT records and delegate subdomains).
- A hardware security module or equivalent for Master Key generation and storage. See Section E.2 for options.
- DNSSEC enabled on your zone (RECOMMENDED, see Section 14.8).
- DNS TTL set to 300 seconds for all _saip records (RECOMMENDED

to minimize revocation window, see Section 10.5.4).

The non-negotiable security requirement across all deployment options is:

> The Master Private Key MUST NOT be exportable from the hardware module in which it was generated. Any hardware or software configuration that does not enforce this property MUST NOT be used in production deployments.

E.2. Key Generation

E.2.1. Software — Development and Testing Only

WARNING: Software-generated keys are stored on disk and are exportable. This method MUST NOT be used in production. It is provided only for development and testing purposes.

Using OpenSSL (all platforms):

```
# Generate Ed25519 Master keypair
openssl genpkey -algorithm ed25519 -out master.key

# Extract public key
openssl pkey -in master.key -pubout -out master.pub

# Base64URL encode for DNS TXT record
openssl pkey -in master.key -pubout -outform DER | \
  tail -c 32 | base64 | tr '+/' '-_' | tr -d '='
```

The output of the last command is the pk= value for the DNS TXT record.

E.2.2. Hardware-Backed — Production Recommended

All production deployments SHOULD use hardware-backed key generation where the Master Private Key is generated inside the hardware module and marked non-exportable at creation time.

PKCS#11 is the standard interface supported by most hardware modules and is compatible with OpenSSL via provider/engine:

```
# Install OpenSSL PKCS#11 provider (Linux example)
apt install libpkcs11-provider  # or distro equivalent

# Generate non-exportable Ed25519 key inside hardware module
openssl genpkey \
  -provider pkcs11 \
  -algorithm ed25519 \
  -pkeyopt "pkcs11-uri:pkcs11:token=SAIP-Master;object=saip-key" \
  -pkeyopt "non-exportable:true"

# Extract public key (private key never leaves hardware)
openssl pkey \
  -provider pkcs11 \
  -in "pkcs11:token=SAIP-Master;object=saip-key" \
  -pubout -out master.pub
```

E.2.3. Linux — TPM 2.0 (tpm2-tools)

TPM 2.0 is available on most modern Linux servers and embedded devices. The tpm2-tools suite provides direct TPM access:

```
# Create primary key context in TPM
tpm2_createprimary -C e -g sha256 -G ecc -c primary.ctx
```

```
# Create non-exportable Ed25519-equivalent key under primary
tpm2_create -C primary.ctx -G ecc256 -u saip.pub -r saip.priv \
  --attributes "fixedtpm|fixedparent|sensitivedataorigin|\
                userwithauth|sign|noda"

# Load key into TPM
tpm2_load -C primary.ctx -u saip.pub -r saip.priv -c saip.ctx

# Extract public key in PEM format via tpm2-pkcs11
tpm2_ptool addkey --algorithm=ecc256 --label=saip-master
openssl pkey -provider tpm2 -in "handle:saip-master" -pubout
```

Note: The "--attributes fixedtpm|fixedparent" flags ensure the key is permanently bound to this TPM and cannot be exported.

E.2.4. Windows — CNG and TPM

Windows exposes TPM functionality through the Cryptography API: Next Generation (CNG). PowerShell example:

```
# Generate non-exportable key in TPM via CNG
$key = [System.Security.Cryptography.ECDsa]::Create(
  [System.Security.Cryptography.ECCurve]::NamedCurves.nistP256
)

# Store in TPM-backed key storage provider
$cngParams = New-Object ‘
  System.Security.Cryptography.CngKeyCreationParameters
$cngParams.Provider = [System.Security.Cryptography.CngProvider]::
  MicrosoftPlatformCryptographyProvider  # TPM provider
$cngParams.ExportPolicy = ‘
  [System.Security.Cryptography.CngExportPolicies]::None  # Non-exportable
$cngKey = [System.Security.Cryptography.CngKey]::Create(
  [System.Security.Cryptography.CngAlgorithm]::ECDiffieHellmanP256,
  "saip-master-key", $cngParams
)
```

The ExportPolicies::None flag enforces non-exportability at the CNG level, which is enforced by the TPM hardware.

E.2.5. macOS and iOS — Secure Enclave

Apple platforms expose the Secure Enclave via the Security framework. Example using the security command-line tool:

```
# Generate non-exportable key in Secure Enclave
security create-keypair \
  -a ec \
  -s 256 \
  -k saip-master \
  -T /usr/bin/security \
  -x  # non-extractable flag
```

For programmatic access, use SecKeyCreateRandomKey with kSecAttrTokenIDSecureEnclave and kSecAttrIsPermanent set to true, and kSecAttrIsExtractable set to false.

E.2.6. Embedded and RTOS Environments

For IoT and embedded deployments (QNX, FreeRTOS, Zephyr, and similar):

wolfTPM (C library, platform-agnostic):

```
/* Generate non-exportable ECC key in TPM */
WOLFTPM2_KEY masterKey;
```

```
wolfTPM2_GetKeyTemplate_ECC(&publicTemplate,
  &sensitiveTemplate,
  TPM_ECC_NIST_P256,
  TPM_ALG_ECDSA);
/* Set fixedTPM and fixedParent to prevent export */
publicTemplate.objectAttributes |= TPMA_OBJECT_FIXEDTPM;
publicTemplate.objectAttributes |= TPMA_OBJECT_FIXEDPARENT;
wolfTPM2_CreateKey(&dev, &masterKey, &parent,
  &publicTemplate, &sensitiveTemplate);
```

Mbed TLS + PSA Crypto API (ARM TrustZone):

```
/* Generate key with no export permission */
psa_key_attributes_t attrs = PSA_KEY_ATTRIBUTES_INIT;
psa_set_key_algorithm(&attrs, PSA_ALG_ECDSA(PSA_ALG_SHA_256));
psa_set_key_type(&attrs, PSA_KEY_TYPE_ECC_KEY_PAIR(
  PSA_ECC_FAMILY_SECP_R1));
psa_set_key_bits(&attrs, 256);
psa_set_key_usage_flags(&attrs, PSA_KEY_USAGE_SIGN_HASH);
/* PSA_KEY_USAGE_EXPORT intentionally omitted */
psa_generate_key(&attrs, &key_id);
```

E.2.7. Cloud HSM

Cloud HSM services expose the same PKCS#11 interface and are suitable for cloud-native agent deployments:

AWS CloudHSM:
- Use cloudhsm_mgmt_util or PKCS#11 SDK.
- Set CKA_EXTRACTABLE=false at key creation.
- Compatible with OpenSSL via the CloudHSM PKCS#11 provider.

Azure Dedicated HSM / Azure Key Vault Managed HSM:
- Use az keyvault key create --protection hsm
- Keys created with HSM protection are non-exportable by policy.

Google Cloud HSM (via Cloud KMS):
- Use gcloud kms keys create --protection-level hsm
- Private key material never leaves Google HSM boundary.

All cloud HSM providers enforce non-exportability at the hardware level. The PKCS#11 interface is consistent across providers, enabling portable SAIP agent implementations.

E.3. DNS Record Publishing

E.3.1. Zone Structure

The recommended DNS zone structure for SAIP is:

```
; Delegate _saip subdomain to allow bot self-registration
_saip.yourdomain.com.          IN NS   ns1.yourdomain.com.

; Vendor root attestation record
_saip.yourdomain.com.     300  IN TXT  "v=saip1;
                                        pk=<VendorMasterPubKey>;
                                        re=re1.saip-registry.example"

; Individual agent records (one per bot instance)
agent-nyc-042._saip.yourdomain.com. 300 IN TXT "v=saip1;
                                        pk=<AgentMasterPubKey>"
```

The 300-second TTL is RECOMMENDED for all _saip records to bound the revocation window to 5 minutes.

E.3.2. Record Verification

```
After publishing, verify with standard DNS tools:

# Verify vendor root record
dig TXT _saip.yourdomain.com +short

# Verify individual agent record
dig TXT agent-nyc-042._saip.yourdomain.com +short

# Verify DNSSEC (recommended)
dig TXT _saip.yourdomain.com +dnssec

Expected output format:
"v=saip1; pk=MCowBQYDK2VdAyEA<base64url>..."
```

E.4. Complete Deployment Examples

E.4.1. Small Vendor — Stateless Deployment

```
Suitable for: small teams, internal tools, development.
No RE registration required.

; DNS records
_saip.smallvendor.com.       300 IN TXT "v=saip1;
                                         pk=MCowBQYDK2VdAyEAx..."
backup-agent-01._saip.smallvendor.com. 300 IN TXT "v=saip1;
                                         pk=MCowBQYDK2VdAyEAy..."

; SAIP header sent by backup-agent-01
SAIP: id="smallvendor.backup.agent-01";
      alg="ed25519";
      ts="1744200000";
      nonce="a1b2c3d4e5f6g7h8";
      rpk="<RollingPubKey>";
      rcert="<MasterSig_over_rpk_ts_nonce_method_path>";
      sig="<RollingPrivKeySig_over_canonical>"

; Server verification steps:
; 1. DNS lookup: agent-01._saip.smallvendor.com → MasterPubKey
; 2. Verify rcert with MasterPubKey
; 3. Verify sig with rpk
; 4. Accept request as Class 3 (Fully Verified) per [VICDM]
```

E.4.2. Enterprise Fleet — AI Crawler

```
Suitable for: large-scale crawler fleets, CDN-delivered agents.

; Zone delegation for self-registration
_saip.bigcorp.com.           300 IN NS  ns1.bigcorp.com.

; Vendor root with RE pointer
_saip.bigcorp.com.           300 IN TXT "v=saip1;
                                         pk=<VendorRootPubKey>;
                                         re=re1.saip-registry.example;
                                         asn=64496"

; Crawler fleet (each bot registers its own record)
crawler-us-east-01._saip.bigcorp.com. 300 IN TXT "v=saip1;
                                         pk=<CrawlerPubKey1>"
crawler-us-west-02._saip.bigcorp.com. 300 IN TXT "v=saip1;
                                         pk=<CrawlerPubKey2>"
crawler-eu-de-03._saip.bigcorp.com.   300 IN TXT "v=saip1;
                                         pk=<CrawlerPubKey3>"

; Revoke single misbehaving crawler:
; Delete crawler-us-east-01._saip.bigcorp.com TXT record
```

```
   ; After 300s TTL expiry → all servers reject its rcert values
   ; Other crawlers unaffected ← surgical precision
```

E.4.3. IoT Fleet — Home Appliances

Suitable for: device manufacturers, embedded systems, IoT.

```
   ; Manufacturer root — RE handles per-device registry
   _saip.manufacturer.com.      300 IN TXT "v=saip1;
                                           re=re1.saip-registry.example"

   ; Individual devices with TPM-bound keys
   ; (published by device during factory provisioning or first boot)
   wm-bg-4471._saip.manufacturer.com. 300 IN TXT "v=saip1;
                                           pk=<TPM_Device_PubKey_1>"
   wm-ns-1823._saip.manufacturer.com. 300 IN TXT "v=saip1;
                                           pk=<TPM_Device_PubKey_2>"
   wm-lj-0392._saip.manufacturer.com. 300 IN TXT "v=saip1;
                                           pk=<TPM_Device_PubKey_3>"

   ; Firmware compromise scenario:
   ; Batch wm-bg-4471 through wm-bg-4999 affected
   ; Manufacturer deletes their DNS records
   ; After 300s → fleet protected, unaffected devices continue
   ; No service interruption for wm-ns-* or wm-lj-* devices
```

E.4.4. SMTP Mail Relay

Suitable for: mail infrastructure, anti-spam enforcement.

```
   ; Mail relay agent records
   relay-bg-01._saip.company.com. 300 IN TXT "v=saip1;
                                              pk=<RelayPubKey1>;
                                              asn=1234"
   relay-de-02._saip.company.com. 300 IN TXT "v=saip1;
                                              pk=<RelayPubKey2>;
                                              asn=5678"

   ; SMTP exchange (see Section 13 for full protocol detail)
   ; C: EHLO relay-bg-01.company.com
   ; S: 250-mail.recipient.com Hello ...
   ; C: SAIP: id="company.relay.relay-bg-01"; alg="ed25519";
   ;          ts="1744200000"; nonce="x9y8z7w6";
   ;          rpk="<RollingPubKey>"; rcert="<cert>"; sig="<sig>"
   ; S: (verifies via DNS → relay-bg-01._saip.company.com)
   ; S: (if valid, continues; if invalid, 550 5.7.1)
   ; C: MAIL FROM:<sender@company.com>
```

E.5. Revocation Procedures

E.5.1. Single Agent Revocation

To revoke one agent immediately:

1. Delete the agent’s _saip DNS TXT record:
   ```
   ; Remove from zone file or DNS management API
   ; DELETE agent-nyc-042._saip.yourdomain.com TXT
   ```

2. Lower TTL to 0 or minimum supported value before deletion to accelerate propagation (if DNS provider supports it).

3. After TTL expiry (default 300s), all servers will return NXDOMAIN for this agent’s DNS lookup. rcert= verification will fail as no Master Public Key can be retrieved.

4. Log the revocation event for audit purposes.

E.5.2. Full Fleet Revocation

To revoke all agents of a vendor immediately:

1. Rotate the vendor's Master Public Key at the zone apex:
   ; Update _saip.yourdomain.com TXT with new pk=
   ; Old pk= is immediately invalid for new rcert verification

2. Or remove the _saip subdomain delegation entirely.

3. All outstanding rcert= values signed by the old Master Private Key become unverifiable immediately upon DNS TTL expiry.

E.5.3. Emergency Revocation

For immediate revocation without waiting for DNS TTL:

- Notify RE (if registered) to add InstanceID to revocation list. RE-aware servers will check revocation independently of DNS TTL.

- Use Anycast DNS or DNS NOTIFY to accelerate propagation to authoritative servers.

- For critical infrastructure: pre-negotiate with major RE operators to maintain an emergency revocation channel.

E.6. Common Mistakes and How to Avoid Them

MISTAKE 1: Exportable Master Key

Wrong: openssl genpkey -algorithm ed25519 -out master.key
(key stored on disk, exportable)

Right: Use PKCS#11 provider with non-exportable flag, TPM with fixedTPM|fixedParent attributes, or equivalent hardware enforcement.

Impact: Attacker with filesystem access can clone the entire agent identity permanently.

MISTAKE 2: TTL too high

Wrong: _saip.example.com. 86400 IN TXT "v=saip1; pk=..."
(24 hour TTL)

Right: _saip.example.com. 300 IN TXT "v=saip1; pk=..."
(5 minute TTL)

Impact: Revocation takes up to 24 hours to propagate. Compromised agent continues operating.

MISTAKE 3: No DNSSEC

Wrong: Publishing _saip records without DNSSEC enabled.

Right: Enable DNSSEC on the zone and validate at server.

Impact: DNS cache poisoning can substitute attacker's public key, allowing impersonation.

MISTAKE 4: Reusing nonce in rcert

Wrong: Using timestamp alone as nonce (ts= only).

Right: Generate cryptographically random nonce per request per [RFC4086], minimum 8 bytes.

Impact: rcert becomes replayable within the ts= window.

MISTAKE 5: Caching Master Key beyond DNS TTL

Wrong: Caching MasterPublicKey for 24 hours regardless of DNS record TTL.

Right: Cache MUST expire at DNS record TTL. Servers MUST re-query DNS after TTL expiry.

Impact: Revoked agents continue to be accepted by servers that cache beyond TTL.

E.7. Attack Scenario: The Great Impersonation Kill-Switch

This scenario demonstrates Dynamic Delegation — the core operational advantage of DNS-Native SAIP over static certificate models. It illustrates what happens when an agent identity is compromised, and how the DNS trust anchor enables surgical, near-instant response.

E.7.1. Background: Static vs. Dynamic Delegation

Traditional PKI uses static delegation:

```
Vendor → CA issues cert → cert valid for 1 year
Attacker compromises key → attacker has 1 year of validity
Revocation (CRL/OCSP) → complex, slow, often ignored
```

SAIP DNS-Native uses dynamic delegation:

```
Vendor → DNS publishes MasterPublicKey → TTL: 300 seconds
Attacker compromises key → attacker has at most 300 seconds
Revocation → delete DNS record → done
```

The DNS record is not a static artifact. It is a live trust anchor that the vendor controls in real time. This is Dynamic Delegation in its purest form.

E.7.2. Scenario Setup

Consider the following deployment:

```
Vendor:   BigCorp, running an AI crawler fleet
Agent:    crawler-us-east-01._saip.bigcorp.com
Key type: SOFTWARE-based Master Key (not TPM-backed)
          ← This is the vulnerability. See E.7.5.
```

The agent's DNS record:

```
crawler-us-east-01._saip.bigcorp.com. 300 IN TXT
  "v=saip1; pk=<CrawlerMasterPubKey>"
```

The attacker gains read access to the agent's filesystem and extracts the Master Private Key. Because the key was stored in software (not hardware), it is exportable.

With the Master Private Key, the attacker can:

- Generate valid rcert= values for any request
- Sign requests that appear to originate from the legitimate crawler-us-east-01 instance
- Bypass rate limits and access controls granted to this agent

E.7.3. Attack Timeline

```
T+0:00   Attacker extracts MasterPrivateKey from filesystem.
         Begins generating valid SAIP headers impersonating
         crawler-us-east-01.

T+0:00   Attacker's requests are accepted by all servers.
         Each rcert= is valid — signed by the real Master Key.
         No cryptographic alarm exists yet.

T+5:00   BigCorp monitoring detects anomaly:
         crawler-us-east-01 appears to be making requests
         simultaneously from two geographically distant ASNs.
         Volumetric spike detected from unexpected IP ranges.

T+5:30   Security team confirms: agent identity is compromised.
         Decision: activate Kill-Switch.

T+6:00   Admin executes DNS rotation — one operation:

         ; Option A: Rotate to new Master Key
         crawler-us-east-01._saip.bigcorp.com. 300 IN TXT
           "v=saip1; pk=<NEW_CrawlerMasterPubKey>"

         ; Option B: Delete record entirely (full revocation)
         ; DELETE crawler-us-east-01._saip.bigcorp.com TXT

T+6:00   DNS propagation begins. Authoritative servers
         immediately serve the updated/deleted record.

T+11:00  DNS TTL expires on all caching resolvers (300s).
         Every server that queries DNS now receives either:
         - The new MasterPublicKey (Option A), or
         - NXDOMAIN (Option B).

T+11:00  Attacker's rcert= values, signed by the old
         MasterPrivateKey, fail verification everywhere.
         The stolen key is cryptographically worthless.
         Chain of Trust is severed.

T+11:00  If Option A: the legitimate crawler-us-east-01
         (re-keyed with new Master Key) resumes normal
         operation immediately.
         Other crawlers in the fleet: unaffected throughout.

T+11:00  KILL-SWITCH COMPLETE. Total response time: ~11 minutes
         from detection to full ecosystem protection.
```

E.7.4. Why This Works — Dynamic Delegation

The Kill-Switch works because of three properties acting together:

Property 1 — DNS as Live Trust Anchor:
The MasterPublicKey in DNS is not a static artifact. It is a live signal that every verifying server re-checks on every TTL expiry. The moment the signal changes, the entire ecosystem adapts — automatically, without coordination, without notifying individual servers.

Property 2 — Short TTL as Security Parameter:
The 300-second TTL is not a performance tuning parameter. It is a security parameter that bounds the maximum window during which a compromised key remains valid after revocation. Administrators SHOULD treat TTL as a security control and set it accordingly.

Property 3 — Per-Request rcert Binding:
Even before the DNS TTL expires, the attacker faces a structural limitation: each rcert= is bound to one specific nonce + ts + method + path combination and cannot be reused. The attacker must continuously generate new rcert= values — which requires the MasterPrivateKey — which they have. But after DNS rotation, those new rcert= values reference a MasterPublicKey that no longer exists in DNS. The Chain of Trust is broken at the root.

The combination of these three properties creates a revocation mechanism that is:

- Faster than CRL or OCSP in practice
- Simpler to operate (one DNS record change)
- Globally consistent (DNS propagation is the distribution mechanism — no separate revocation infrastructure needed)
- Surgical (only the compromised instance is affected)

E.7.5. Lessons Learned — The Role of Hardware

This scenario began with one mistake: the Master Private Key was stored in software, making it exportable.

Had the key been generated in a hardware module (TPM, HSM, Secure Enclave), the scenario would not have occurred:

```
Attacker gains filesystem access:
  Software key → EXPORTED. Attack proceeds.
  TPM-bound key → NOT EXPORTABLE. Attack fails at step 1.
```

The DNS Kill-Switch is a powerful recovery mechanism. But hardware-bound keys are the prevention mechanism that makes the Kill-Switch rarely necessary.

The two-layer defense:

```
Layer 1 (Prevention):  Hardware-bound MasterPrivateKey.
                       Attacker cannot export the key even
                       with full filesystem access.

Layer 2 (Recovery):    DNS Dynamic Delegation Kill-Switch.
                       If Layer 1 fails (physical access,
                       supply chain compromise, insider threat),
                       Layer 2 contains the damage in 300s.
```

This is defense in depth applied to agent identity:

> "Hardware makes the Kill-Switch unnecessary.
> DNS makes the Kill-Switch sufficient."

Srecko Jovancevic
SKGO, IKT Support
Makedonska 22
11000 Belgrade
Serbia

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