Internet Engineering Task Force (IETF)
Internet-Draft
Intended status: Standards Track
Expires: September 15, 2026

M. Welen, Ed.
Atea Sverige AB

March 15, 2026

Universal Authenticated Sender Identity (UASI) draft-uasi-framework-00

Abstract

This document defines the Universal Authenticated Sender Identity (UASI) framework, a protocol-agnostic mechanism for cryptographic sender identity assertion and verification across Internet communication protocols. UASI provides a unified method by which any sending entity -- whether a mail server, an API endpoint, an IoT device, or a messaging service -- can assert its identity in a manner that receivers can independently verify using DNS-anchored trust. The framework is designed to be incrementally deployable, backwards-compatible with existing authentication mechanisms, and simple enough to encourage default-on adoption.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 15, 2026.

Table of Contents

1. Introduction

1.1. Background

The Internet was designed in an era of implicit trust between communicating parties. As the network grew from a small research community to a global infrastructure carrying commerce, governance, and personal communications, the absence of built-in sender authentication became an increasingly severe liability.

The consequences are visible across every layer of the stack. Email suffers from phishing and spoofing despite decades of incremental improvements through SPF [RFC7208], DKIM [RFC6376], and DMARC [RFC7489]. Webhook receivers have no standardized way to verify that an incoming HTTP request genuinely originated from a claimed service. IoT devices routinely accept commands without verifying the identity of the sender. API consumers rely on shared secrets and bespoke token schemes rather than cryptographic identity.

Each protocol community has addressed this gap independently, producing solutions that are:

- Protocol-specific: DKIM works for email but not MQTT. HTTP Message Signatures [RFC9421] works for HTTP but not CoAP.
- Partial: SPF authenticates the sending IP, DKIM authenticates message content, DMARC ties them to the domain -- yet all three are needed for reasonable email authentication.
- Opt-in: Deployment requires active configuration by both senders and receivers, resulting in decades-long adoption curves.
- Inconsistent: The identity model, key distribution mechanism, signature algorithms, and policy semantics differ across every protocol, making cross-protocol security reasoning difficult.

1.2. Goals

The Universal Authenticated Sender Identity (UASI) framework addresses these deficiencies with the following design goals:

(a) Protocol-agnostic: A single identity assertion and verification model that can be bound to any Internet communication protocol through lightweight protocol-specific bindings.

(b) DNS-anchored trust: Public keys and policy records are published in DNS, leveraging the existing global namespace and DNSSEC [RFC4033] for integrity. No new PKI or certificate authority infrastructure is required.

(c) Incrementally deployable: UASI can coexist with and complement existing authentication mechanisms (DKIM, mTLS, HTTP Signatures). It does not require flag-day adoption.

(d) Default-on simplicity: The framework is designed to be simple enough that platform and library implementors can enable it by default, shifting the paradigm from opt-in authentication to opt-out.

(e) Minimal new primitives: UASI reuses existing cryptographic algorithms, DNS record types (where possible), and proven design patterns. Novelty is in composition, not invention.

1.3. Scope

This document defines:

- The UASI identity model and namespace
- DNS record formats for key publication and policy declaration
- The canonical signature format
- Protocol bindings for SMTP, HTTP, MQTT, CoAP, and WebSocket
- The verification procedure
- The policy evaluation framework
- Migration guidance for environments with existing authentication mechanisms

This document does NOT define:

- End-to-end encryption (UASI authenticates the sender; it does not encrypt the payload)
- Authorization or access control (UASI establishes identity; what you do with that identity is out of scope)
- Human identity (UASI authenticates domains and services, not individual humans)

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.

Sender Domain:
: The DNS domain name that a sending entity claims as its identity. This is the root of the UASI trust anchor.

UASI Identity (UID):
: A structured identifier of the form <selector>._uasi.<domain> that uniquely identifies a sending entity or class of sending entities within a domain.

Selector:
: A label that allows a domain to publish multiple UASI keys, enabling key rotation, per-service keys, and delegation to third parties.

Assertion:
: A cryptographic signature over a canonical representation of the message or event, produced by the sender using the private

key corresponding to a published UASI record.

Verifier:
Any entity that receives a UASI-signed message and performs the verification procedure defined in Section 9.

Protocol Binding:
A specification of how UASI assertions are encoded and transported within a specific communication protocol (e.g., as an SMTP header, an HTTP header, or an MQTT user property).

Policy Record:
A DNS TXT record under _uasi-policy.<domain> that declares the sender domain's authentication policy -- analogous to DMARC policy for email, but generalized.

Nonce:
A unique, non-repeating value included in a signature to enable intra-protocol replay detection by the receiver.

3. Problem Statement

3.1. The Fragmentation Problem

The current landscape of sender authentication on the Internet can be characterized as a patchwork of protocol-specific solutions, each developed independently and each with its own identity model, key management approach, and policy semantics.

Consider an organization (example.com) that operates:

- An email system requiring SPF, DKIM, and DMARC
- A REST API sending webhook callbacks
- An IoT fleet publishing MQTT telemetry
- A real-time service using WebSocket connections
- A microservice mesh using gRPC

Today, this organization must configure and maintain five or more completely separate authentication systems. Each has its own:

- Key generation and rotation procedures
- DNS records or certificate infrastructure
- Signature algorithm negotiation
- Verification libraries
- Failure handling and reporting

This fragmentation imposes costs at every level:

- Operational complexity: Security teams must be expert in multiple unrelated authentication systems.

- Inconsistent coverage: Organizations typically secure their most visible protocol (email) and leave others unprotected.

- Reinvention: Every new protocol reinvents sender authentication, usually starting from shared secrets and only later (if ever) migrating to cryptographic identity.

- Audit difficulty: Answering "can we verify the sender of every inbound communication?" requires protocol-by-protocol analysis.

3.2. The Adoption Problem

Even within individual protocols, authentication adoption is

slow. DMARC, specified in 2015, still sees incomplete adoption a decade later. The reasons are structural:

- Complexity: Configuring DKIM alone requires generating keys, publishing DNS records, configuring the MTA, and testing. Adding SPF and DMARC adds further steps.
- Fragility: Misconfigurations (e.g., SPF record exceeding the 10-lookup limit) silently degrade authentication.
- No universal default: Most software ships with authentication disabled; administrators must actively enable it.
- Receiver-side cost: Verifiers must implement protocol-specific verification logic for each mechanism.

UASI addresses the adoption problem by providing a single mechanism that, once implemented in a library or platform, works across all protocol bindings. This dramatically reduces the per-protocol implementation cost and enables platform vendors to ship with UASI enabled by default.

3.3. The IoT and Machine-to-Machine Gap

The fastest-growing category of Internet communication -- machine-to-machine messaging via MQTT, CoAP, AMQP, and proprietary protocols -- has the weakest sender authentication. Many IoT deployments rely on:

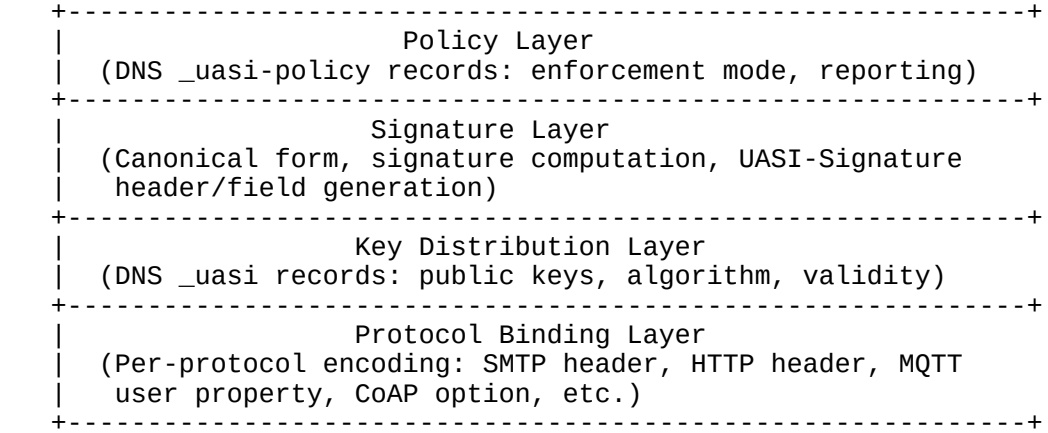
- Pre-shared keys (PSK) with no rotation mechanism
- Username/password over TLS with no domain-level identity
- No authentication at all on constrained networks

The consequences include large-scale IoT botnets, unauthorized command injection, and telemetry spoofing. UASI's lightweight design and DNS-based key distribution make it feasible to deploy on constrained devices, particularly those that already perform DNS resolution.

4. Architecture Overview

4.1. Layered Design

UASI is structured in four layers:



This layering ensures that the core cryptographic operations (key lookup, signature generation, signature verification) are identical regardless of the transport protocol. Only the

outermost layer -- the protocol binding -- differs.

4.2. Data Flow

The end-to-end UASI flow is:

1. Key Publication (one-time setup per selector):
   The sender domain generates a key pair and publishes the public key as a DNS record under <selector>._uasi.<domain>.

2. Policy Publication (one-time setup per domain):
   The sender domain publishes a policy record under _uasi-policy.<domain> declaring its authentication posture.

3. Signing (per message/event):
   a. The sending application constructs the message payload per the transport protocol.
   b. The UASI library extracts the signable content according to the protocol binding rules (Section 8).
   c. The library computes a canonical hash of the signable content.
   d. The library signs the hash with the sender's private key.
   e. The library constructs the UASI-Signature field and injects it into the message via the protocol binding.

4. Verification (per message/event):
   a. The receiving application extracts the UASI-Signature field via the protocol binding.
   b. The UASI library parses the signature field to obtain the sender domain and selector.
   c. The library queries DNS for the public key at <selector>._uasi.<domain>, applying caching per Section 9.5.
   d. The library extracts the signable content using the same protocol binding rules.
   e. The library verifies the signature against the canonical hash.
   f. If a nonce is present (Section 7.5), the library checks for replay.
   g. The library queries the policy record at _uasi-policy.<domain> and evaluates the result.
   h. The verification result (pass/fail/none) is returned to the application.

4.3. Trust Model

UASI's trust model is rooted in DNS:

- A domain's UASI public keys are authoritative because they are published under the domain's own DNS zone.

- Integrity of key material depends on DNSSEC. Domains SHOULD sign their zones with DNSSEC. Verifiers SHOULD perform DNSSEC validation where available.

- In the absence of DNSSEC, UASI provides authentication under a Trust-On-First-Use (TOFU) model, similar to how DKIM operates today. This is explicitly acknowledged as a weaker security property but is strictly better than no authentication, which is the status quo for most non-email protocols.

- UASI does NOT require a certificate authority. The DNS zone administrator is the root of trust for all identities within that zone.

This model inherits both the strengths of DNS (global reachability, hierarchical delegation, caching) and its weaknesses (dependence on registrar security, DNSSEC deployment gaps). Section 12 discusses these trade-offs in detail.

Note: The TOFU model, while imperfect, has a proven track record in DKIM, where it has been operationally useful for over a decade without universal DNSSEC. The incremental security benefit of cryptographic domain authentication without DNSSEC still raises the bar significantly above the unauthenticated baseline.

5. Identity Model

5.1. UASI Identity (UID) Format

A UASI Identity is a DNS name of the following form:

```
<selector>._uasi.<domain>
```

Where:

- <domain> is a fully qualified domain name (FQDN) under the control of the sending entity.
- <selector> is a DNS label chosen by the domain administrator. Selectors allow a domain to publish multiple keys for different purposes (e.g., per-service, per-environment, key rotation).
- "_uasi" is a fixed underscore-prefixed label that prevents collision with existing DNS names, per [RFC8552].

Examples:

```
mail._uasi.example.com       (email signing key)
webhooks._uasi.example.com   (webhook signing key)
iot-fleet1._uasi.example.com (IoT device fleet key)
api-v2._uasi.example.com     (API signing key)
```

5.2. Selector Semantics

Selectors are opaque labels from the verifier’s perspective. Their internal structure and naming convention are a matter of local policy for the sender domain. However, the following conventions are RECOMMENDED:

- Use descriptive names that aid operational debugging (e.g., "mail-2026q1" rather than "k1").
- Include a rotation indicator (e.g., date or sequence number) to facilitate key rollover.
- Use separate selectors for separate services to enable independent key management and least-privilege signing.

5.3. Delegation

A domain MAY delegate UASI signing authority to a third party by publishing a CNAME record from the selector to a name controlled by the delegate:

```
webhooks._uasi.example.com. CNAME webhooks._uasi.provider.net.
```

This allows SaaS providers, CDNs, and managed service operators

to sign on behalf of their customers without requiring access to the customer's DNS zone beyond the initial CNAME setup -- a pattern proven effective by DKIM delegation.

5.4. Wildcard Identities

Wildcard selectors (*._uasi.<domain>) are NOT RECOMMENDED due to the difficulty of revoking individual keys and the security implications of a single key compromise affecting all services.

5.5. Selector Management at Scale

Organizations operating large numbers of services (microservice architectures, IoT fleets, multi-tenant SaaS platforms) may accumulate hundreds or thousands of selectors. This section provides operational guidance for managing selectors at scale.

5.5.1. Selector Lifecycle

Each selector SHOULD follow a defined lifecycle:

Active:	Key is published, signing system uses this selector.
Draining:	New key published under new selector; old selector still resolves for in-flight message verification. Duration: at minimum, the old record's TTL plus the maximum expected message delivery delay.
Retired:	DNS record removed. The selector label MAY be reused after a cooling-off period (RECOMMENDED: 30 days minimum) to avoid confusion in log analysis.

5.5.2. Automation Patterns

Manual selector management does not scale beyond a handful of services. Implementations SHOULD support automated lifecycle management:

- Key generation and DNS publication via DNS API (e.g., RFC 2136 dynamic updates, or provider-specific APIs).
- Selector naming conventions that encode creation date and service identity (e.g., "webhooks-20260315-001") to enable automated garbage collection of retired records.
- Monitoring that alerts on selectors approaching key expiration (the "x=" tag in the key record) or exceeding a maximum age threshold.

Appendix C provides reference patterns for common deployment architectures.

5.5.3. Shared Selectors

Where individual per-device selectors are infeasible (e.g., a fleet of 10,000 IoT sensors), a shared fleet selector is acceptable. The trade-off is that key compromise affects the entire fleet rather than a single device. Organizations SHOULD segment fleets into groups of manageable size (RECOMMENDED: no more than 1,000 devices per selector) and use fleet-level selectors (e.g., "fleet-a", "fleet-b") to limit blast radius.

6. DNS Resource Records

6.1. Key Record

UASI public keys are published as DNS TXT records under the

UASI Identity name. The record value is a semicolon-separated list of tag=value pairs.

Required tags:

```
v=UASI1     ; Version (MUST be "UASI1" for this specification)
k=<alg>     ; Key algorithm (see Section 6.3)
p=<key>     ; Base64-encoded public key material
```

Optional tags:

```
t=<flags>   ; Flags (colon-separated):
            ;   "s" = this key is for signing only (not
            ;         encryption; default assumed)
            ;   "y" = testing mode (verifiers SHOULD NOT
            ;         reject based on this key)
h=<hash>    ; Acceptable hash algorithms (colon-separated)
            ;   Default: "sha256"
x=<expiry>  ; Key expiration as Unix timestamp. Verifiers
            ;   MUST treat expired keys as invalid.
n=<notes>   ; Human-readable notes (no semantic meaning)
```

Example:

```
mail._uasi.example.com. 3600 IN TXT (
   "v=UASI1; k=ed25519; "
   "p=MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE...; "
   "x=1735689600; "
   "n=Email signing key 2026-Q1" )
```

6.1.1. TTL Recommendations

The TTL of UASI key records directly affects both the DNS query load generated by verifiers and the speed of key rotation.

The following TTL values are RECOMMENDED:

General use:	3600 seconds (1 hour)
High-frequency APIs:	86400 seconds (24 hours)
Key rotation period:	300 seconds (5 minutes), temporarily, during the Draining phase of key rotation (Section 5.5.1)

Verifiers MUST respect the TTL of cached key records and MUST NOT query DNS more frequently than the TTL permits. See Section 9.5 for verifier-side caching guidance.

6.2. Policy Record

Domain-level UASI policy is published as a DNS TXT record under _uasi-policy.<domain>. The record value uses the same tag=value syntax.

Required tags:

```
v=UASI1     ; Version
p=<policy>  ; Policy mode:
            ;   "none"    = no assertion about UASI usage;
            ;               informational only
            ;   "report"  = UASI is deployed; receivers
            ;               SHOULD verify and report but
            ;               SHOULD NOT reject
            ;   "enforce" = UASI is deployed; receivers
            ;               SHOULD reject unsigned or
            ;               failing messages
```

Optional tags:

```
rua=<uri>   ; Aggregate reporting URI
            ;   (mailto: or https: endpoint)
ruf=<uri>   ; Forensic/failure reporting URI
pct=<int>   ; Percentage of messages to which policy applies
            ;   (0-100; default 100). Enables gradual rollout.
sp=<policy> ; Subdomain policy (overrides p= for subdomains)
b=<list>    ; Protocol bindings to which this policy applies
            ;   (colon-separated; e.g., "smtp:http:mqtt")
            ;   Default: all bindings
rl=<level>  ; Report granularity level (Section 10.2.1):
            ;   "d" = domain-only (most private)
            ;   "s" = selector-level (default)
            ;   "f" = full forensic detail
```

Example:

```
_uasi-policy.example.com. 3600 IN TXT (
   "v=UASI1; p=enforce; pct=50; "
   "rua=mailto:uasi-reports@example.com; "
   "b=smtp:http" )
```

This policy declares that example.com enforces UASI on 50% of its SMTP and HTTP traffic, with aggregate reports sent to the specified mailbox.

6.3. Supported Algorithms

UASI implementations MUST support the following algorithms:

```
Ed25519  [RFC8032]  -- REQUIRED (mandatory to implement)
ECDSA P-256 (ES256) -- RECOMMENDED
RSA-2048 (RS256)    -- OPTIONAL (for legacy compatibility)
```

Ed25519 is designated as the mandatory-to-implement algorithm due to its small key size (32 bytes public key, 64 bytes signature -- both suitable for DNS TXT records and constrained protocol fields), fast verification, and resistance to implementation pitfalls.

Implementations MUST NOT support key sizes below 2048 bits for RSA or curves below 256 bits for ECDSA.

Note: Symmetric HMAC (hmac-sha256) is intentionally excluded from the core algorithm set. Publishing symmetric keys in DNS would be architecturally unsound. For constrained devices that cannot perform public key cryptography, the KDF-hint model defined in Section 12.4 provides HMAC-based authentication without exposing the shared secret.

6.4. Key Rotation

Key rotation is accomplished by:

1. Publishing a new key under a new selector.
2. Configuring the signing system to use the new selector.
3. Allowing sufficient time for DNS caches to expire the old key record (at minimum, the old record’s TTL plus the maximum expected message delivery delay for the protocol).
4. Removing the old key record.

During the transition period, both keys are valid, and verifiers will accept signatures from either. The selector mechanism ensures that each signature unambiguously identifies which key to use, avoiding the ambiguity problems that arise with key

rollover in other systems.

For automated rotation guidance, see Section 5.5.

7. Signature Format

7.1. UASI-Signature Field

The UASI-Signature field is the carrier for UASI assertions. Its syntax is a semicolon-separated list of tag=value pairs, conceptually similar to a DKIM-Signature header but generalized for any protocol.

Required tags:

```
v=1          ; Signature version
a=<alg>      ; Signing algorithm (e.g., "ed25519-sha256")
d=<domain>   ; Sender domain (the "d=" domain)
s=<selector> ; Selector
t=<ts>       ; Signature timestamp (Unix seconds)
bh=<hash>    ; Base64-encoded hash of the canonicalized
             ; message body/payload
b=<sig>      ; Base64-encoded signature value
c=<canon>    ; Canonicalization method (Section 7.2)
z=<context>  ; Protocol context tag (e.g., "smtp", "http",
             ;   "mqtt"). REQUIRED. Prevents cross-protocol
             ;   replay. (Section 7.4)
```

Optional tags:

```
x=<expiry>   ; Signature expiration (Unix seconds)
h=<fields>   ; Colon-separated list of signed header/metadata
             ; field names (protocol-binding specific)
q=<method>   ; Query method for key retrieval
             ;   "dns/txt" (default)
l=<length>   ; Body length limit (number of octets signed)
n=<nonce>    ; Unique nonce for intra-protocol replay
             ;   detection (Section 7.5). RECOMMENDED for
             ;   HTTP and WebSocket bindings.
```

Example (line breaks for display; actual value is continuous):

```
v=1; a=ed25519-sha256; d=example.com; s=webhooks;
t=1710500000; x=1710503600; z=http;
n=a1b2c3d4e5f6;
h=content-type:x-request-id;
bh=base64encodedHash==;
b=base64encodedSignature==;
c=strict
```

7.2. Canonicalization

To ensure that signatures survive transport-layer transformations (e.g., header reordering, whitespace normalization), UASI defines three canonicalization modes:

Simple:
The signable content is used as-is, with no normalization. This mode is suitable for binary protocols (MQTT, CoAP) where message content is unlikely to be transformed in transit.

Relaxed:
- Header/metadata field names are lowercased.
- Header/metadata field values have leading and trailing

whitespace removed, and internal sequences of whitespace collapsed to a single space.
- Empty lines at the end of the body are removed.
- Line endings are normalized to CRLF.

This mode is suitable for text-based protocols (SMTP, HTTP) where intermediaries may modify whitespace.

Strict:
- All rules of "relaxed" mode apply.
- Additionally, ONLY fields listed in "h=" are considered part of the signing input. If a field listed in "h=" is absent from the message, it is included as an empty value (field_name ":" SP "" CRLF). This prevents an attacker from adding a signed field that wasn't present at signing time.
- The body hash covers the EXACT byte sequence of the body after transfer-encoding removal, with no further normalization.

This mode is RECOMMENDED for the HTTP binding (Section 8.2) because it limits the signing input to fields that the sender controls, making the signature resilient to header additions or modifications by intermediaries (proxies, CDNs, WAFs).

The choice of canonicalization mode is declared in the "c=" tag of the UASI-Signature field.

7.2.1. Canonicalization Selection Guidance

The following table summarizes the RECOMMENDED canonicalization mode for each protocol binding:

Protocol	RECOMMENDED	Rationale
SMTP	relaxed	MTAs modify whitespace
HTTP	strict	CDNs/proxies add/modify headers
MQTT v5	simple	Brokers preserve payload
CoAP	simple	No intermediary modification
WebSocket	strict	Proxy traversal possible

Senders MAY use a stricter mode than recommended (e.g., "simple" for MQTT is acceptable even though "strict" would also work). Senders SHOULD NOT use a less strict mode than recommended without understanding the signature breakage risks.

7.3. Canonical Signing Input

The signing input is constructed as follows:

1. For each field name listed in "h=", in the order listed:
   a. Apply canonicalization to the field name and value.
   b. If the field is absent from the message and c=strict, use an empty value.
   c. Concatenate: canonicalized_name ":" SP canonicalized_value CRLF

2. Concatenate the protocol context: "z:" SP context_value CRLF

3. If a nonce is present: "n:" SP nonce_value CRLF

4. Concatenate the body hash: "bh:" SP body_hash_value CRLF

5. Concatenate the UASI-Signature field value itself, with the

"b=" tag value empty (i.e., "b=;"), canonicalized.

6. The signature is computed over the concatenation from steps 1-5.

This construction ensures that both selected metadata fields and the message body are covered, and that the signature is self-referential (signing its own parameters prevents parameter substitution attacks).

7.4. Preventing Cross-Protocol Replay

The "z=" (protocol context) tag is REQUIRED and is always included in the signing input. This ensures that a valid UASI signature for an HTTP webhook cannot be replayed as an MQTT message, even if the payload is identical. Verifiers MUST reject signatures where the "z=" value does not match the protocol over which the message was received.

Note: The "z=" tag prevents cross-protocol replay only. For intra-protocol replay protection (e.g., re-sending the same HTTP request to the same endpoint), see Section 7.5.

7.5. Nonce-Based Intra-Protocol Replay Detection

The "z=" context tag prevents an attacker from replaying a signed message across different protocols (e.g., an HTTP signature used in MQTT). However, it does not prevent intra-protocol replay -- re-sending an identical, validly signed HTTP request to the same receiver.

Replay protection is fundamentally a stateful problem: the receiver must remember what it has already seen. UASI provides an optional "n=" (nonce) tag to support this use case without mandating statefulness for all deployments.

7.5.1. Nonce Generation

When included, the "n=" value MUST be unique per signature. Acceptable generation strategies include:

- UUIDv4 (RECOMMENDED for general use)
- UUIDv7 or ULID (RECOMMENDED when time-ordering aids debugging)
- Cryptographically random hex string (minimum 128 bits)
- Monotonically increasing counter per selector (acceptable for single-signer deployments)

The nonce value MUST NOT exceed 128 characters and MUST consist only of alphanumeric characters and hyphens.

7.5.2. Nonce Verification

When a UASI-Signature contains an "n=" tag, verifiers SHOULD maintain a nonce cache indexed by (d=, s=, n=) tuples:

1. After successful signature verification, check whether the (d=, s=, n=) tuple exists in the cache.
2. If it exists, return result = "fail" (replay detected).
3. If it does not exist, add the tuple to the cache with a TTL equal to the signature’s validity window (x= minus t=, or a configured maximum if x= is absent).

The nonce cache MAY be bounded by size. When the cache is full, the oldest entries SHOULD be evicted. Eviction of unexpired entries weakens replay protection; operators should size the

cache according to their traffic volume and risk tolerance.

7.5.2.1. Cache Saturation and Forced Eviction

An attacker who can generate valid signed messages (e.g., a compromised sender, or a legitimate sender whose messages are observed and collected) can flood a verifier with messages carrying unique nonces, forcing the nonce cache to evict legitimate entries. Once a legitimate nonce is evicted, the attacker can replay the corresponding message.

This is a fundamental trade-off: unbounded caches are vulnerable to memory exhaustion; bounded caches are vulnerable to forced eviction. Verifiers MUST choose a strategy appropriate to their threat model:

Fail-Closed (High Security):

When the nonce cache reaches its configured maximum size, the verifier rejects all new messages carrying nonces (result = "temperror") until existing entries expire naturally. This prevents forced-eviction replays but introduces a denial-of-service risk: an attacker can cause legitimate traffic to be rejected by filling the cache.

This strategy is RECOMMENDED for:
- Financial transaction webhooks
- Any endpoint where replay has irreversible consequences

Fail-Open (High Availability):

When the nonce cache reaches its configured maximum size, the verifier evicts the oldest unexpired entries to make room. The verifier SHOULD emit a security alert (log entry, monitoring event) indicating that the replay protection window has been compromised by volume.

This strategy is RECOMMENDED for:
- General-purpose webhook receivers
- Endpoints where occasional replay is tolerable and availability is prioritized

Implementation guidance:

- Verifiers SHOULD use a TTL-aware cache where entries expire automatically based on the signature validity window (x= minus t=). This minimizes the need for forced eviction under normal traffic patterns.

- For high-volume endpoints, a distributed, TTL-aware cache shared across multiple verifier instances reduces per-instance memory pressure and provides consistent nonce checking across a load-balanced deployment.

- Verifiers SHOULD monitor cache utilization and alert when utilization exceeds 80% of the configured maximum, as this may indicate either a traffic spike or an eviction attack.

- The configured maximum cache size SHOULD be at least 10x the expected peak message rate multiplied by the maximum signature validity window. For example, an endpoint receiving 1,000 messages/second with a 300-second validity window should configure a cache of at least 3,000,000 entries.

7.5.3. When to Use Nonces

The "n=" tag is RECOMMENDED for:

- HTTP webhook delivery (where replay can trigger duplicate side effects)
- WebSocket per-message signing
- Any binding where the receiver is stateful and replay has business-logic consequences

The "n=" tag is NOT RECOMMENDED for:

- SMTP (email has its own Message-ID-based deduplication)
- IoT telemetry (stateless, idempotent sensor readings)
- High-frequency, low-latency streams where nonce cache overhead is prohibitive

Signers MUST NOT assume that receivers enforce nonce uniqueness. The "n=" tag is a cooperative mechanism: it enables replay detection for receivers that implement it, but does not guarantee it.

8. Protocol Bindings

This section defines how UASI-Signature fields are encoded and transported within specific protocols. Each binding specifies:

- Where the UASI-Signature field is placed
- Which protocol-specific fields are signable (eligible for inclusion in "h=")
- Any protocol-specific canonicalization rules
- The value of the "z=" context tag

8.1. SMTP Binding

Context tag: z=smtp

Placement:
The UASI-Signature field is placed as an RFC 5322 message header field named "UASI-Signature". It is added by the originating MTA or MSA, similar to DKIM-Signature.

Signable fields:
Any RFC 5322 header field MAY be included in "h=". The following fields are RECOMMENDED:

> from, to, subject, date, message-id, mime-version, content-type

Canonicalization:
SMTP bindings SHOULD use "relaxed" canonicalization due to the well-known tendency of MTAs to modify whitespace in headers.

Body:
The body hash (bh=) covers the MIME body of the message after transfer encoding (i.e., the decoded body content).

Nonce:
The "n=" tag is NOT RECOMMENDED for SMTP. Email already has Message-ID-based deduplication, and the store-and-forward nature of email makes nonce caching impractical across the delivery chain.

Coexistence with DKIM:
UASI-Signature and DKIM-Signature are independent header fields. An MTA MAY generate both. A verifier MAY verify both. UASI does not replace DKIM; it provides a superset

that happens to work the same way for SMTP and can also
work for other protocols.

Example SMTP headers:

```
UASI-Signature: v=1; a=ed25519-sha256; d=example.com;
   s=mail-2026q1; t=1710500000; z=smtp; c=relaxed;
   h=from:to:subject:date:message-id;
   bh=abcdef1234567890==;
   b=SIGVALUEHERE==
DKIM-Signature: [existing DKIM signature, if any]
From: sender@example.com
To: recipient@example.org
Subject: Q1 Report
Date: Fri, 15 Mar 2026 10:00:00 +0000
Message-ID: <msg-123@example.com>
```

8.2. HTTP Binding

Context tag: z=http

Placement:
The UASI-Signature field is placed as an HTTP header field named "UASI-Signature". For HTTP/1.1 and HTTP/2, this is a standard header. For HTTP/3, it is a header in the HEADERS frame.

Signable fields:
Any HTTP header field MAY be included in "h=". The following fields are RECOMMENDED for webhook/callback use cases:

```
x-request-id, x-webhook-event
```

The content-type header MAY be included in "h=" but is subject to intermediary modification (see Section 8.2.2). Senders who include it MUST pre-normalize the value.

Additionally, the following pseudo-fields are defined for the HTTP binding:

```
@method      -- the HTTP method (GET, POST, etc.)
@target-uri  -- the full request-target URI
@authority   -- the Host or :authority value
```

These pseudo-fields follow the convention established by HTTP Message Signatures [RFC9421].

Signable field guidance:
Senders SHOULD sign only headers that they control and that are semantically meaningful to the receiver. Senders SHOULD NOT sign headers that intermediaries are known to modify (e.g., content-length, via, x-forwarded-for, x-real-ip).

The following headers MUST NOT be included in "h=" because they are routinely modified by intermediaries:

```
content-length, transfer-encoding, via, x-forwarded-for,
x-forwarded-proto, x-real-ip, connection, keep-alive,
proxy-authorization, te, trailer
```

Canonicalization:
"strict" is RECOMMENDED for the HTTP binding. This ensures that the signature covers only sender-controlled fields and survives intermediary modifications to other headers.

Body:

The body hash covers the HTTP message body (the entity body after any transfer-encoding is decoded).

Nonce:
The "n=" tag is RECOMMENDED for the HTTP binding, as webhook delivery is particularly susceptible to replay attacks that trigger duplicate side effects.

Relationship to HTTP Message Signatures:
UASI's HTTP binding and RFC 9421 (HTTP Message Signatures) serve overlapping but distinct purposes. RFC 9421 is a general-purpose HTTP signing mechanism. UASI's HTTP binding is a profile that additionally specifies DNS-based key distribution, domain-level policy, and cross-protocol compatibility. Implementations MAY support both. A future document may define a mapping between the two.

Example HTTP request:

```
POST /webhook/orders HTTP/1.1
Host: receiver.example.org
Content-Type: application/json
X-Request-Id: req-456
UASI-Signature: v=1; a=ed25519-sha256; d=sender.example.com;
   s=webhooks; t=1710500000; z=http; c=strict;
   n=550e8400-e29b-41d4-a716-446655440000;
   h=@method:@target-uri:content-type:x-request-id;
   bh=bodyHashHere==;
   b=SIGVALUEHERE==

{"order_id": "12345", "status": "shipped"}
```

8.2.1. Intermediary Guidance for the HTTP Binding

HTTP intermediaries (reverse proxies, CDNs, WAFs, API gateways, load balancers) SHOULD preserve the UASI-Signature header without modification.

Intermediaries MUST NOT:

- Remove or modify the UASI-Signature header.
- Modify the body content without being aware that doing so will invalidate the body hash and therefore the signature.

Intermediaries MAY:

- Add new headers (these will not be in the "h=" list and therefore will not affect signature verification).
- Modify headers that are NOT listed in the "h=" set of the UASI-Signature.
- Re-sign the message with their own UASI-Signature if they are an authorized intermediary (adding a second signature; see Section 9.4 on multiple signatures).

When deploying UASI in environments with intermediaries, senders SHOULD test signature survival through their actual delivery path during the "monitor" phase (Section 11.2) before progressing to enforcement.

If an intermediary must modify the body (e.g., content transformation, compression), it SHOULD either:

(a) Strip the original UASI-Signature and re-sign with its own identity, or
(b) Verify the original signature before modification and record the verification result in a trace header

(e.g., "UASI-Authentication-Results").

8.2.2. Volatile Header Parameters

Certain HTTP headers carry parameters that intermediaries frequently add, remove, or normalize. The most common case is Content-Type, where a sender transmits:

```
Content-Type: application/json
```

and an intermediary rewrites it to:

```
Content-Type: application/json; charset=utf-8
```

This modification changes the byte representation of the header value and will cause a signature failure if Content-Type is included in the "h=" list, even under "strict" canonicalization.

This is not a canonicalization deficiency -- it is a semantic modification of the header value by the intermediary. UASI cannot define normalization rules for all possible header parameter variations without becoming a parser for every media type registry.

Signer guidance:

Senders who include Content-Type (or any parameterized header) in the "h=" list MUST pre-normalize the value before signing. Specifically:

- Include all parameters that the sender intends to be present in the signed value. For example, if the sender sends JSON, sign "application/json; charset=utf-8" rather than "application/json", so that an intermediary adding the charset parameter does not change the signed value.
- Lowercase the media type and all parameter names (per RFC 2045, media types are case-insensitive).
- Order parameters consistently (RECOMMENDED: alphabetical by parameter name).

Alternative approach -- omit parameterized headers:

Senders who cannot control intermediary behavior MAY omit Content-Type from the "h=" list entirely. The body hash (bh=) already provides integrity protection for the message payload. The Content-Type header indicates how to *interpret* the body but does not affect the body’s byte content; a receiver can verify that the body is intact via bh= even if Content-Type has been modified.

This approach is RECOMMENDED for senders whose delivery path traverses multiple intermediaries (e.g., CDN -> load balancer -> reverse proxy -> application), where header parameter normalization is unpredictable.

Headers with known volatility:

In addition to Content-Type, the following headers are known to be subject to parameter modification by intermediaries and SHOULD be treated with the same caution:

```
Accept, Accept-Encoding, Accept-Language, Cache-Control
```

Senders SHOULD prefer signing application-specific headers

(e.g., X-Webhook-Event, X-Request-Id) that intermediaries have no reason to modify.

8.3. MQTT Binding (v5)

Context tag: z=mqtt5

Placement:
The UASI-Signature is placed as an MQTT v5 User Property with the name "UASI-Signature" on the PUBLISH packet.

Signable fields:
The following MQTT properties are signable using symbolic names:

```
@topic          -- the PUBLISH topic name
@qos            -- the QoS level
@retain         -- the retain flag (0 or 1)
@content-type   -- the Content Type property (if present)
@response-topic -- the Response Topic property (if present)
@correlation-data -- the Correlation Data property
                     (base64-encoded, if present)
```

Additional User Properties MAY be included by name.

Canonicalization:
"simple" is RECOMMENDED. MQTT brokers generally do not modify packet contents.

Body:
The body hash covers the Application Message (payload) of the PUBLISH packet.

Constrained devices:
For devices unable to perform public key cryptography, UASI defines a KDF-hint constrained device model (see Section 12.4) where the verifier and sender share a secret established out-of-band, with DNS providing only a public salt for key derivation.

Example MQTT v5 PUBLISH User Properties:

```
UASI-Signature: v=1; a=ed25519-sha256;
   d=sensor-fleet.example.com; s=iot-fleet1;
   t=1710500000; z=mqtt5; c=simple;
   h=@topic:@qos:@content-type;
   bh=payloadHash==;
   b=SIGVALUEHERE==
```

8.4. CoAP Binding

Context tag: z=coap

Placement:
The UASI-Signature is placed as a CoAP Option. A new option number is requested from IANA (see Section 13).

Signable fields:
The following CoAP fields are signable:

```
@uri-path     -- the Uri-Path option value(s)
@uri-host     -- the Uri-Host option value
@content-format -- the Content-Format option value
```

Canonicalization:
"simple" is RECOMMENDED.

Body:
The body hash covers the CoAP payload.

Note: CoAP's constrained environment makes UASI's compact Ed25519 signatures (64 bytes) particularly attractive compared to RSA alternatives.

8.5. WebSocket Binding

Context tag: z=websocket

Placement:
UASI authentication for WebSocket is performed during the opening handshake as an HTTP header on the Upgrade request (following the HTTP binding). For per-message authentication after the handshake, the UASI-Signature is placed in a JSON wrapper or as a protocol-defined extension frame.

Per-message signing (application-layer):
For WebSocket messages, the RECOMMENDED approach is:

```
{
   "uasi_signature": "<UASI-Signature value>",
   "payload": { ... }
}
```

Where the body hash covers the JSON-serialized "payload" value.

Signable fields:
@origin -- the WebSocket origin
@protocol -- the negotiated subprotocol

Nonce:
The "n=" tag is RECOMMENDED for WebSocket per-message signing.

8.6. Defining New Bindings

New protocol bindings can be defined in separate documents by specifying:

1. A unique context tag value (registered with IANA)
2. The placement of the UASI-Signature field
3. The set of signable protocol-specific fields
4. Any protocol-specific canonicalization rules
5. How the body/payload hash input is determined
6. Whether the "n=" nonce tag is recommended for the binding
7. Intermediary behavior requirements (if applicable)

9. Verification Procedure

9.1. Overview

Verification is performed by the receiving entity or an intermediary acting on its behalf. The procedure is the same regardless of protocol, operating on the abstract UASI-Signature field and protocol binding.

9.2. Step-by-Step Procedure

1. EXTRACT the UASI-Signature field from the received message using the appropriate protocol binding rules.

2. PARSE the tag=value pairs. Verify that all required tags (v, a, d, s, t, bh, b, c, z) are present and syntactically valid. If not, return result = "permerror".

3. CHECK the protocol context: if the "z=" value does not match the protocol over which the message was received, return result = "fail" (cross-protocol replay detected).

4. CHECK the timestamp: if "x=" is present and the current time exceeds the expiration, return result = "fail" (expired signature).

5. QUERY DNS for the key record at <s>._uasi.<d>, respecting cache entries per Section 9.5. Apply DNSSEC validation if the verifier supports it; distinguish between insecure and bogus responses per Section 12.1.1.
   a. If no record is found, return result = "none" (no key published).
   b. If the DNSSEC response is bogus, return result = "temperror" (see Section 12.1.1).
   c. If the record's "x=" tag indicates key expiration and the key has expired, return result = "fail".
   d. Parse the key record and extract the public key and algorithm.

6. CHECK algorithm compatibility: the algorithm in the key record ("k=") must be compatible with the algorithm in the signature ("a="). If not, return result = "permerror".

7. CANONICALIZE the signed fields and body according to the "c=" method and protocol binding rules.

8. COMPUTE the body hash and compare to "bh=". If they do not match, return result = "fail" (body modified).

9. RECONSTRUCT the signing input per Section 7.3.

10. VERIFY the signature "b=" against the signing input using the public key from step 5. If verification fails, return result = "fail".

11. If "n=" is present and the verifier implements nonce checking (Section 7.5.2), verify nonce uniqueness. If the nonce has been seen before within the validity window, return result = "fail" (replay detected).

12. Return result = "pass".

9.3. Result Codes

The verification procedure produces one of the following results:

```
pass      -- Signature verified successfully.
fail      -- Signature verification failed (invalid signature,
             expired, cross-protocol mismatch, body hash
             mismatch, or replay detected).
none      -- No UASI-Signature was present, or no key record
             was found. The message is unsigned.
permerror -- A permanent error occurred (e.g., malformed
             signature, DNS error, unsupported algorithm).
temperror -- A temporary error occurred (e.g., DNS timeout).
             The verifier SHOULD retry.
```

9.4. Multiple Signatures

A message MAY carry multiple UASI-Signature fields (e.g., one

from the originator and one from an intermediary). Verifiers SHOULD evaluate all signatures and make policy decisions based on the aggregate result.

A common pattern is:

- The originating sender signs the message (first signature).
- An authorized intermediary (e.g., an API gateway) verifies the original signature, then adds its own signature attesting to the verified provenance.

Verifiers receiving multiply-signed messages can choose to trust the intermediary's signature (if they have a relationship with the intermediary) or verify the original sender's signature directly (if they have DNS access to the sender's key).

9.5. Verifier Caching

UASI verification requires DNS queries to retrieve key records and policy records. In high-frequency scenarios (e.g., API webhooks at thousands of requests per second), naive per-message DNS resolution would impose unacceptable latency and load.

9.5.1. Key Record Caching

Verifiers MUST cache UASI key records according to the DNS TTL of the record. A verifier that receives multiple messages from the same (d=, s=) pair within a TTL window MUST reuse the cached key without issuing additional DNS queries.

Implementation note: In practice, a webhook receiver that processes traffic from saas.example.com with selector "webhooks" and a TTL of 3600 seconds will issue one DNS query per hour for that sender, regardless of message volume.

9.5.2. Policy Record Caching

Verifiers SHOULD cache policy records independently of key records. Policy records typically change less frequently than key records and may use longer TTLs.

9.5.3. Negative Caching

If a DNS query for a key record returns NXDOMAIN or NODATA, the verifier SHOULD cache the negative result for the lesser of:

- The SOA minimum TTL from the authoritative response, or
- 300 seconds (5 minutes)

This prevents repeated DNS queries for non-existent selectors, which could be used as a denial-of-service vector against the verifier's DNS resolver.

9.5.4. Cache Warming

For verifiers with a known set of sender domains (e.g., a webhook receiver that has registered with specific SaaS providers), pre-fetching key records at startup and refreshing them periodically (at TTL intervals) is RECOMMENDED. This eliminates cold-start latency on the first message from each sender.

10. Policy Layer

10.1. Policy Evaluation

After verification, the verifier queries the sender domain's policy record at _uasi-policy.<d> and evaluates the result against the policy.

Policy evaluation follows this logic:

Verification Result	p=none	p=report	p=enforce
pass	accept	accept	accept
fail	accept	accept*	reject*
none (unsigned)	accept	accept*	reject*
permerror	accept	accept*	reject*
temperror	accept	accept	defer

* Subject to the pct= percentage. If a random value (0-99) is >= pct, treat as if policy were "none" for this message.

"accept" = deliver/process the message normally
"reject" = refuse the message (protocol-appropriate error)
"defer" = temporarily refuse; request retry

10.2. Reporting

If rua= or ruf= is specified in the policy record, verifiers SHOULD send reports:

Aggregate reports (rua=):
Periodic reports (RECOMMENDED: daily) summarizing UASI verification results, analogous to DMARC aggregate reports. Format: JSON, schema defined in a companion document.

Forensic reports (ruf=):
Per-failure reports containing details of individual verification failures. Due to privacy implications, forensic reports are OPTIONAL and SHOULD be rate-limited.

10.2.1. Report Granularity (rl= Tag)

The UASI policy record MAY include an "rl=" (report level) tag that controls the granularity of information included in aggregate reports. This enables senders to balance operational visibility with privacy protection.

Values:

rl=d (Domain):
Aggregate reports summarize results at the organizational domain level only. Individual selectors, message timestamps, and source IP addresses are NOT included. This is the most privacy-preserving option.

RECOMMENDED for: Large IoT fleet operators, organizations subject to strict data minimization requirements, senders who want to confirm UASI is working without exposing communication patterns.

rl=s (Selector):
Aggregate reports include results broken down by selector. This enables senders to identify which service or key is experiencing failures. Source IP addresses are included in aggregated form (IP ranges, not individual IPs). This is the DEFAULT if rl= is not specified.

RECOMMENDED for: Most deployments. Provides sufficient granularity for operational debugging without exposing individual message metadata.

rl=f (Full):
Forensic-level detail is permitted in aggregate reports, including individual message identifiers, timestamps, and source IPs. This is equivalent to enabling forensic reporting (ruf=) within the aggregate report stream.

RECOMMENDED for: Initial deployment and debugging only. Senders SHOULD NOT leave rl=f enabled in production unless they have a specific operational need for per-message visibility.

Interaction with ruf= (forensic reports):

The rl= tag controls aggregate report granularity. It does not affect forensic reports, which are independently controlled by the presence or absence of the ruf= tag. However, senders who set rl=d (domain-level aggregation) and also specify ruf= should be aware that the forensic reports will contain more detail than the aggregate reports, potentially undermining the privacy intent of rl=d.

Example:

```
_uasi-policy.example.com. TXT (
   "v=UASI1; p=enforce; rl=d; "
   "rua=https://uasi-reports.example.com/aggregate" )
```

This policy enforces UASI and receives aggregate reports with domain-level granularity only -- no selector-level or per-message detail is included.

10.3. Protocol-Specific Policy

The "b=" tag in the policy record allows domains to apply different policies to different protocols. For example:

```
_uasi-policy.example.com. TXT (
   "v=UASI1; p=enforce; b=smtp:http; "
   "sp=report; "
   "rua=https://uasi-reports.example.com/aggregate" )
```

This enforces UASI on SMTP and HTTP traffic but applies only reporting mode to subdomains.

11. Migration and Coexistence

11.1. Relationship to Existing Mechanisms

UASI is designed to coexist with, not replace, existing protocol-specific authentication mechanisms:

DKIM:
UASI’s SMTP binding is functionally similar to DKIM. During migration, domains can publish both DKIM and UASI signatures. Receivers verify both independently. Over time, if UASI achieves sufficient adoption, domains may choose to deprecate DKIM, but this is not required.

SPF:
SPF authenticates the sending IP address, which UASI does

not. SPF and UASI are complementary. Domains SHOULD continue to publish SPF records alongside UASI.

DMARC:
: DMARC aligns SPF and DKIM with the From: domain. A future specification may define how DMARC evaluation can incorporate UASI results alongside DKIM.

HTTP Message Signatures (RFC 9421):
: UASI's HTTP binding can be seen as a profile of HTTP Message Signatures with standardized DNS-based key distribution. A mapping document is anticipated.

mTLS:
: mTLS authenticates the transport endpoint. UASI authenticates the application-layer sender. The two are complementary: mTLS ensures you're talking to the right server; UASI ensures the message was sent by the claimed domain.

11.2. Incremental Deployment Path

The recommended deployment sequence is:

Phase 1: Monitor
: Publish a UASI key record and policy record with p=none. Begin signing outbound messages. Monitor via aggregate reports. No receiver-side impact.

  During this phase, senders SHOULD verify that signatures survive their actual delivery path, including any intermediaries (CDNs, proxies, load balancers). If signatures are broken by intermediaries, adjust the "h=" field list to exclude modified headers before proceeding.

Phase 2: Report
: Change policy to p=report. Cooperating receivers begin verifying and sending reports. Sender analyzes reports to identify unsigned or failing traffic (e.g., third-party services sending on behalf of the domain).

Phase 3: Gradual Enforcement
: Change policy to p=enforce with pct=10, then 25, 50, and finally 100 as report data confirms readiness.

Phase 4: Full Enforcement
: Policy at p=enforce; pct=100 across all relevant protocol bindings.

This mirrors the proven DMARC deployment model but applies it uniformly across all protocols.

11.2.1. Recommended Protocol Ordering

Organizations deploying UASI across multiple protocol bindings SHOULD NOT attempt to enable all bindings simultaneously. Instead, the following ordering is RECOMMENDED:

1. HTTP (webhooks/APIs):
   Start here. HTTP traffic is the easiest to test (request/response is synchronous), the delivery path is typically shortest (fewer intermediaries than email), and webhook receivers are the most likely early adopters. The "strict" canonicalization mode and nonce support make HTTP the most robust binding.

2. SMTP (email):

Add second. Email authentication has the most mature ecosystem (SPF, DKIM, DMARC already exist), so UASI deployment can leverage existing operational familiarity. However, email's store-and-forward nature and MTA diversity make canonicalization issues more likely; allow extra time in the Monitor and Report phases.

3. MQTT/CoAP (IoT):
   Add last. IoT deployments typically involve fleet-wide changes that are harder to roll back. The "simple" canonicalization mode reduces breakage risk, but the KDF-hint model (if used) requires out-of-band secret provisioning that adds deployment complexity. Ensure HTTP and SMTP bindings are stable before adding IoT.

Each binding progresses through the four phases independently. An organization may be at Phase 4 (Full Enforcement) for HTTP while still at Phase 2 (Report) for SMTP. The "b=" tag in the policy record (Section 6.2) enables per-protocol policy declaration to support this.

12. Security Considerations

12.1. DNS Dependency

UASI's security is fundamentally dependent on the integrity of DNS responses. Without DNSSEC, an attacker who can forge DNS responses (e.g., via cache poisoning or on-path attacks) can substitute their own public key and forge valid UASI signatures.

Mitigation:
- Domains SHOULD deploy DNSSEC.
- Verifiers SHOULD perform DNSSEC validation.
- In the absence of DNSSEC, UASI provides TOFU-level security: the first observed key for a selector is cached and subsequent changes trigger warnings. This is imperfect but strictly better than no authentication. DKIM has operated usefully under this model for over a decade.
- Verifiers SHOULD use encrypted DNS transports (DNS-over-TLS [RFC7858] or DNS-over-HTTPS [RFC8484]) to protect key material in transit from passive observation and on-path modification. See Section 12.6.

12.1.1. DNSSEC Validation States

When a verifier performs DNSSEC validation on a UASI key record query, the response falls into one of three states, as defined in [RFC4033]:

Secure:
   The response has a valid DNSSEC signature chain to a trust anchor. The verifier has strong assurance that the key record is authentic.

   Action: Proceed with verification normally.

Insecure:
   The domain does not deploy DNSSEC, or there is a provable break in the chain of trust (e.g., an unsigned delegation). The response is not signed but there is no evidence of tampering.

   Action: The verifier MAY proceed with verification using TOFU semantics. This is the expected state for the majority of domains today and provides the same security level as

DKIM without DNSSEC.

Bogus:
The DNSSEC signature is present but fails validation. This indicates either a misconfigured zone or an active attack (e.g., DNS cache poisoning, BGP hijacking, on-path key substitution).

Action: The verifier MUST NOT fall back to TOFU. A bogus response means the verifier cannot trust the key material in the response. The verification result MUST be set to "temperror" (if the verifier believes the condition may be transient, e.g., a zone re-signing event) or "permerror" (if the condition persists across multiple queries over a period of at least 1 hour).

Verifiers SHOULD log bogus DNSSEC responses as security events. Persistent bogus responses for a previously-secure domain are a strong indicator of an active attack and SHOULD trigger operator alerts.

The distinction between "insecure" and "bogus" is critical. Treating both as equivalent (i.e., falling back to TOFU in both cases) would allow an attacker to downgrade a DNSSEC-signed domain to TOFU by injecting bogus responses, defeating the purpose of DNSSEC deployment.

12.2. Key Compromise

If a UASI private key is compromised, the attacker can forge signatures for the affected selector until the key is rotated.

Mitigation:
- Use per-service selectors to limit blast radius.
- Set short key expiration times (x= tag).
- Monitor aggregate reports for unexpected signing activity.
- Rotate keys immediately upon suspected compromise.
- Use automated key rotation (Section 5.5) to reduce the window of exposure.

12.3. Replay Attacks

An attacker who observes a validly signed message can replay it.

Mitigation:
- The "z=" context tag prevents cross-protocol replay.
- The "t=" and "x=" tags limit the temporal validity window.
- The "n=" nonce tag (Section 7.5) enables intra-protocol replay detection for receivers that maintain a nonce cache.
- Senders SHOULD use short signature validity windows (x= minus t= no longer than necessary for the use case; RECOMMENDED: 5 minutes for webhooks, 1 hour for email, 5 minutes for IoT commands).

Intra-protocol replay protection is intentionally OPTIONAL because it requires receiver-side state, which imposes costs that are not justified for all use cases (e.g., idempotent sensor telemetry). The "n=" tag provides the mechanism; receivers decide whether to use it based on their risk profile.

12.4. Constrained Devices and the KDF-Hint Model

Some IoT devices lack the computational resources for public key cryptography. A naive approach would be to publish symmetric (HMAC) keys directly in DNS TXT records, but this is architecturally unsound: a "secret" key published in a globally

queryable database is not secret, rendering the authentication effectively useless against any attacker who can perform a DNS lookup.

UASI addresses this with a KDF-hint model that separates the public parameter (a salt, published in DNS) from the secret parameter (a shared key established out-of-band).

12.4.1. KDF-Hint Key Record

For constrained device deployments, the UASI key record uses a new algorithm identifier:

```
v=UASI1; k=kdf-hmac-sha256; p=<base64-encoded-salt>;
kdf=hkdf-sha256; n=<notes>
```

Where:

k=kdf-hmac-sha256 -- indicates this selector uses the KDF-hint model. The signature is an HMAC-SHA256 tag, not a digital signature.

p=<salt> -- a base64-encoded public salt value (RECOMMENDED: 128 bits minimum). This is NOT the shared secret.

kdf=hkdf-sha256 -- the key derivation function [RFC5869] used to derive the HMAC key from the shared secret and salt.

The actual shared secret is provisioned out-of-band between the sender (IoT device fleet) and the verifier (data ingestion platform). This may be done via:

- Device provisioning during manufacturing
- A secure enrollment protocol (e.g., EST [RFC7030])
- Manual configuration in a fleet management system

12.4.2. Key Derivation

The HMAC key is derived as follows:

```
HMAC_key = HKDF-Expand(
   HKDF-Extract(salt=<p value from DNS>, IKM=<shared_secret>),
   info="UASI-KDF-HMAC" || d= || s=,
   L=32
)
```

Where:

- <shared_secret> is the out-of-band provisioned secret.
- The "info" parameter includes the domain and selector to ensure that the derived key is unique per selector, even if the same shared secret is used across multiple selectors.
- L=32 produces a 256-bit HMAC key.

The HMAC is then computed over the canonical signing input (Section 7.3) using HMAC-SHA256 with the derived key.

12.4.3. Security Properties

The KDF-hint model provides the following properties:

- The shared secret is NEVER published in DNS. An attacker who queries DNS learns only the salt, which is useless without the shared secret.

- The salt in DNS enables key derivation without transmitting

the derived key, reducing the risk of key exposure.

- Salt rotation (by publishing a new salt in DNS) forces re-derivation of the HMAC key, providing a lightweight rotation mechanism without re-provisioning the shared secret.

- Third-party verifiability is NOT provided. Only entities that possess the shared secret can verify signatures. This is an inherent limitation of symmetric authentication.

Limitations:

- The KDF-hint model requires an out-of-band channel for shared secret provisioning. This adds deployment complexity compared to the pure public-key model.

- Compromise of the shared secret compromises all devices that use it. Organizations SHOULD use per-fleet or per-device-group secrets, not a single global secret.

- The KDF-hint model MUST NOT be used for high-sensitivity applications where third-party verifiability is required.

12.5. Canonicalization Brittleness

Canonicalization -- the process of normalizing message content before signing -- is the most common cause of spurious signature failures in deployed authentication systems. DKIM has suffered from this problem for over a decade, and UASI inherits the risk for text-based protocols.

The "strict" canonicalization mode (Section 7.2) is designed to mitigate this risk by signing only sender-controlled fields. However, even with "strict" mode, the following scenarios can cause legitimate signature failures:

- An intermediary rewrites the message body (e.g., content transformation, virus scanning that modifies attachments).

- An intermediary removes the UASI-Signature header entirely.

- Character encoding normalization (e.g., UTF-8 NFC vs NFD) changes the byte representation of the body.

- An intermediary adds or modifies header parameters (e.g., appending "charset=utf-8" to Content-Type). See Section 8.2.2 for specific guidance on volatile headers.

Mitigation:

- Senders SHOULD test signature survival through their actual delivery path before enabling enforcement.

- Senders SHOULD use the "strict" mode for HTTP and WebSocket bindings, and "simple" mode for binary protocols.

- The Phase 1 (Monitor) and Phase 2 (Report) stages of the deployment path (Section 11.2) exist specifically to identify and resolve canonicalization issues before enforcement.

- Verifiers operating in "enforce" mode SHOULD implement a configurable "fail-open" option for the initial deployment period, where verification failures generate alerts but do not reject messages. This is distinct from "report" mode in that the verifier’s local policy treats failures as actionable alerts while the domain’s published policy is

already at "enforce".

12.6. DNS Query Privacy

UASI verification requires DNS queries of the form <selector>._uasi.<domain> for every distinct sender. These queries reveal to DNS resolvers (and any entity on the network path to the resolver) which domains are communicating with the verifier.

While sender identity is generally visible in transport-layer headers (and therefore not a new exposure), UASI formalizes this into a structured, easily observable DNS query pattern that could be used for traffic analysis.

Mitigation:

- Verifiers SHOULD use encrypted DNS transports:
  - DNS-over-TLS (DoT) [RFC7858] encrypts the query between the verifier and its resolver.
  - DNS-over-HTTPS (DoH) [RFC8484] provides similar protection and may traverse network middleboxes more reliably.
- Verifiers SHOULD use a trusted recursive resolver (e.g., operated by the verifier’s own organization) to limit exposure of query patterns to third parties.
- Aggressive caching (Section 9.5) reduces the frequency of queries, limiting the metadata window. A key record with a TTL of 86400 seconds generates at most one query per day per sender-selector pair.
- QNAME minimization [RFC9156] reduces the information exposed to authoritative servers along the resolution chain.

Senders who are concerned about receiver-side query privacy (e.g., in adversarial environments) should be aware that UASI verification will generate DNS queries that correlate with message receipt. This is an inherent trade-off of DNS-anchored trust.

12.7. Privacy Considerations

UASI signatures reveal the sender domain and selector to any on-path observer. This is generally not a new privacy exposure (the sender is typically visible in transport headers), but it does formalize the association.

Aggregate and forensic reports may contain metadata about message flows. Operators SHOULD handle reports as sensitive data and apply appropriate access controls.

13. IANA Considerations

13.1. DNS Underscore Label Registration

This document registers the following underscore label in the "Underscored and Globally Scoped DNS Node Names" registry:

```
_uasi         -- UASI key records
_uasi-policy  -- UASI policy records
```

13.2. HTTP Header Field Registration

This document registers the following HTTP header field:

Field name: UASI-Signature
Status: permanent
Specification: This document, Section 8.2

This document also registers:

Field name: UASI-Authentication-Results
Status: permanent
Specification: This document, Section 8.2.1

13.3. CoAP Option Number Registration

This document requests assignment of a CoAP Option Number for the UASI-Signature option.

Option Name: UASI-Signature
Option Number: TBD
Reference: This document, Section 8.4

13.4. UASI Algorithm Registry

IANA is requested to create a "UASI Signing Algorithms" registry with the following initial entries:

Algorithm	Status	Reference
ed25519	REQUIRED	RFC 8032
es256	RECOMMENDED	(*)
rs256	OPTIONAL	-
kdf-hmac-sha256	OPTIONAL	Sec 12.4

(*) ECDSA P-256 as defined in FIPS 186-4.

Note: The kdf-hmac-sha256 algorithm uses the KDF-hint model (Section 12.4) rather than publishing raw symmetric keys in DNS.

13.5. UASI KDF Algorithm Registry

IANA is requested to create a "UASI Key Derivation Functions" registry with the following initial entry:

KDF	Status	Reference
hkdf-sha256	REQUIRED	RFC 5869

13.6. UASI Protocol Context Registry

IANA is requested to create a "UASI Protocol Context Tags" registry with the following initial entries:

Context	Reference
smtp	Sec 8.1
http	Sec 8.2
mqtt5	Sec 8.3
coap	Sec 8.4
websocket	Sec 8.5

14. References

14.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4033] Arends, R., et al., "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, May 2010.

[RFC6376] Crocker, D., Hansen, T., and M. Kucherawy, "DomainKeys Identified Mail (DKIM) Signatures", STD 76, RFC 6376, September 2011.

[RFC7030] Pritikin, M., et al., "Enrollment over Secure Transport", RFC 7030, October 2013.

[RFC7208] Kitterman, S., "Sender Policy Framework (SPF)", RFC 7208, April 2014.

[RFC7858] Hu, Z., et al., "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, May 2016.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, January 2017.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, May 2017.

[RFC8484] Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", RFC 8484, October 2018.

[RFC8552] Crocker, D., "Scoped Interpretation of DNS Resource Records through ’Underscored’ Naming of Attribute Leaves", BCP 222, RFC 8552, March 2019.

[RFC9156] Bortzmeyer, S., et al., "DNS Query Name Minimisation to Improve Privacy", RFC 9156, November 2021.

[RFC9421] Backman, A., et al., "HTTP Message Signatures", RFC 9421, February 2024.

14.2. Informative References

[RFC7489] Kucherawy, M. and E. Zwicky, "Domain-based Message Authentication, Reporting, and Conformance (DMARC)", RFC 7489, March 2015.

Note on normative references to Informational RFCs: This document makes normative references to RFC 5869 (HKDF) and RFC 8032 (EdDSA). RFC 5869 defines the key derivation function required by the UASI KDF Algorithm Registry (Section 13.5). RFC 8032 defines the mandatory-to-implement signing algorithm. Both are widely referenced normatively by Standards Track documents and are expected to receive IESG approval as downrefs during Last Call.

Appendix A. Example Flows

A.1. Webhook Verification with Nonce

This example shows a SaaS provider (saas.example.com) sending a webhook to a customer (customer.example.org), using the "strict" canonicalization mode and nonce-based replay detection.

Setup (one-time):

1. saas.example.com generates an Ed25519 key pair.
2. saas.example.com publishes:

```
webhooks._uasi.saas.example.com. 86400 IN TXT (
   "v=UASI1; k=ed25519; p=BASE64PUBKEY==;" )

_uasi-policy.saas.example.com. 86400 IN TXT (
   "v=UASI1; p=enforce; b=http" )
```

Note: TTL of 86400 (24 hours) per Section 6.1.1 recommendation for high-frequency APIs.

Sending:

3. saas.example.com sends a webhook:

```
POST /webhooks/orders HTTP/1.1
Host: customer.example.org
Content-Type: application/json
X-Webhook-Event: order.completed
X-Request-Id: req-789
UASI-Signature: v=1; a=ed25519-sha256;
   d=saas.example.com; s=webhooks; t=1710500000;
   x=1710500300; z=http; c=strict;
   n=550e8400-e29b-41d4-a716-446655440000;
   h=@method:@target-uri:content-type:x-webhook-event:
     x-request-id;
   bh=Abc123BodyHash==;
   b=Ed25519SignatureValue==

{"order_id":"789","total":99.50}
```

Note: Signature expires in 300 seconds (5 minutes) per Section 12.3 recommendation. Nonce is a UUIDv4.

Verification:

4. customer.example.org extracts UASI-Signature from the HTTP header.

5. Checks cache for webhooks._uasi.saas.example.com. Cache hit (TTL not expired from previous webhook) -- no DNS query needed.

6. Verifies the signature using cached key. Result: pass.

7. Checks nonce cache for (saas.example.com, webhooks, 550e8400-...). Not found -- nonce is fresh.

8. Adds nonce to cache with TTL = 300 seconds (matching signature validity window).

9. Queries DNS for _uasi-policy.saas.example.com (cached). Policy: enforce. Proceeds to process the webhook.

If an attacker replays this exact request within 5 minutes: Step 7 finds the nonce in cache. Result: fail (replay).

A.2. IoT Telemetry with KDF-Hint Model

A temperature sensor (part of fleet managed by iot.example.com) publishes readings via MQTT using the KDF-hint constrained device model.

Setup:

1. iot.example.com provisions a shared secret to the fleet-a device group during manufacturing:
   shared_secret = <32 random bytes>

2. iot.example.com publishes:

```
fleet-a._uasi.iot.example.com. 3600 IN TXT (
   "v=UASI1; k=kdf-hmac-sha256; "
   "p=BASE64SALT==; kdf=hkdf-sha256; "
   "n=Fleet A temperature sensors" )

_uasi-policy.iot.example.com. 3600 IN TXT (
   "v=UASI1; p=enforce; b=mqtt5" )
```

3. Both the device and the data ingestion platform derive the HMAC key:

```
HMAC_key = HKDF-Expand(
   HKDF-Extract(salt=<p value>, IKM=shared_secret),
   info="UASI-KDF-HMAC" || "iot.example.com" || "fleet-a",
   L=32
)
```

Sending (MQTT v5 PUBLISH):

4. Topic: sensors/building-7/temp
   Payload: {"temp_c": 22.5, "ts": 1710500000}
   User Property:

```
UASI-Signature: v=1; a=kdf-hmac-sha256;
   d=iot.example.com; s=fleet-a; t=1710500000;
   z=mqtt5; c=simple;
   h=@topic:@content-type;
   bh=PayloadHash==;
   b=HMACValue==
```

Verification:

5. The data ingestion platform extracts the UASI-Signature.
6. Queries DNS for fleet-a._uasi.iot.example.com (cached). Sees k=kdf-hmac-sha256 -- this is a KDF-hint record.
7. Derives the HMAC key using the cached salt and the locally stored shared secret.
8. Verifies the HMAC. Result: pass.
9. No nonce checking (telemetry is idempotent; replay is low-risk).

Note: An attacker who queries DNS learns only the salt, not the shared secret. Without the shared secret, they cannot derive the HMAC key and cannot forge signatures.

A.3. Email with UASI and DKIM Coexistence

A domain migrating from DKIM-only to UASI+DKIM:

```
UASI-Signature: v=1; a=ed25519-sha256; d=example.com;
   s=mail-2026q1; t=1710500000; z=smtp; c=relaxed;
   h=from:to:subject:date:message-id;
```

```
      bh=BodyHash==;
      b=UASISignature==
   DKIM-Signature: v=1; a=rsa-sha256; d=example.com;
      s=dkim2024; ... [standard DKIM signature]
   From: ceo@example.com
   To: board@example.org
   Subject: Q1 Results
```

The receiving MTA verifies both signatures independently. DKIM result feeds into DMARC evaluation. UASI result feeds into UASI policy evaluation. Both can pass, providing defense in depth.

Appendix B. Comparison with Existing Mechanisms

Feature	DKIM	SPF	RFC9421	mTLS	UASI
Protocol scope	Email	Email	HTTP	Any/TLS	Any
Authenticates	Domain	IP	Signer	Cert	Domain
Key distribution	DNS	DNS	App	PKI/CA	DNS
Policy framework	(DMARC)	(DMARC)	No	No	Yes
Body signing	Yes	No	Yes	No(1)	Yes
Cross-protocol	No	No	No	N/A	Yes
Replay protect	Partial	N/A	Partial	Yes	Yes(2)
IoT suitable	No	No	No	Heavy	Yes(3)
Delegation	CNAME	incl	Manual	SubCA	CNAME
DNS privacy	N/A	N/A	N/A	N/A	(4)

(1) mTLS secures the channel, not individual messages.
(2) Cross-protocol via z= tag. Intra-protocol via optional n= nonce tag (Section 7.5).
(3) Public key for capable devices; KDF-hint model (Sec 12.4) for constrained devices.
(4) See Section 12.6 for DNS query privacy considerations and DoT/DoH mitigation.

Appendix C. Selector Management Patterns

C.1. Microservice Architecture

A microservice deployment with 50 services, each with its own selector:

Naming convention:
```
<service-name>-<YYYYMMDD>._uasi.<domain>
```

Example:
```
order-svc-20260315._uasi.api.example.com
payment-svc-20260315._uasi.api.example.com
notification-svc-20260315._uasi.api.example.com
```

Automation:
The CI/CD pipeline generates a new key pair on each quarterly rotation, publishes the new selector via DNS API, updates the service's signing configuration, and schedules the old selector for deletion after 48 hours (old TTL + delivery buffer).

Monitoring:
A cron job queries all *._uasi.api.example.com selectors and alerts if any key's "x=" expiry is within 7 days.

C.2. IoT Fleet

A fleet of 5,000 sensors divided into 5 groups of 1,000:

Naming convention:
fleet-<group>-<YYYYMM>._uasi.<domain>

Example:
fleet-a-202603._uasi.iot.example.com
fleet-b-202603._uasi.iot.example.com
...
fleet-e-202603._uasi.iot.example.com

Key material:
Each group uses the KDF-hint model (Section 12.4) with a
per-group shared secret provisioned during device enrollment.

Salt rotation:
Monthly salt rotation via DNS update. Devices derive the new
HMAC key on next boot or configuration refresh. No secret
re-provisioning needed.

C.3. SaaS Provider with Customer Delegation

A SaaS provider (saas.example.com) sending webhooks on behalf
of 1,000 customers:

Customer DNS setup (one CNAME per customer):
webhooks._uasi.customer1.com. CNAME wh._uasi.saas.example.com.
webhooks._uasi.customer2.com. CNAME wh._uasi.saas.example.com.

SaaS provider manages:
A single key pair under wh._uasi.saas.example.com.
Signatures use d=customer1.com (the customer's domain).
Verifiers resolve via CNAME to the SaaS provider's key.

This scales linearly: adding a new customer requires one CNAME
record, not a new key pair.

Authors' Addresses

Martin Welen (editor)
Atea Sverige AB
Sweden
Email: martin@welen.com