

S/MIME Certificate Discovery (SCD)

2026-10-20

S/MIME Certificate Discovery over DNS and HTTPS

Draft 01, 20 October 2026. This is a working draft, not a published specification: section numbering, requirement numbering and normative wording may still change between drafts, and nothing in it should be implemented against without checking for a later draft.

Abstract

This document describes a federated method for locating S/MIME certificates for an email address or for an email domain. Support is announced through the DNS; the certificates themselves are retrieved over HTTPS from a well-known location on a host name derived from the domain. The design is deliberately close to the OpenPGP Web Key Directory (WKD) [WKD] and to MTA-STS [RFC8461], so that existing operational experience and existing static-hosting infrastructure can be reused.

Discovery is explicitly separated from validation. For an address lookup the directory only makes a certificate *available*; whether it is acceptable is decided by path validation, name binding, validity and revocation checking, on the certificate itself — exactly as for a certificate obtained by any other means. Domain-to-domain certificates are a separate case: no CA asserts that a key is authorised for a whole domain, so there the act of publication — signed under a key the domain publishes in its DNS — carries the assertion, and additional constraints apply.

Status of this Memo

This is a design document, not an IETF submission. Requirement keywords

("MUST", "SHOULD", "MAY", ...) are used as in [RFC2119] and [RFC8174] to make the intended implementation behaviour unambiguous: they carry that meaning only when they appear in capitals.

1. Introduction

1.1 Problem statement

Sending an encrypted S/MIME message requires the sender to possess the recipient's certificate before the first encrypted message can be sent. In practice this is solved by:

- asking the recipient to send a signed message first (a manual round trip that does not scale and cannot be automated across organisations), or
- an out-of-band exchange of certificates (email attachments, portals, USB), or
- a central directory (LDAP, a "certificate portal", a hosted key server).

The central-directory approach concentrates trust and availability in a single operator. The operator can serve a substituted certificate, can observe who is about to communicate with whom, and is a single point of failure and a single target for denial of service. Cross-organisational LDAP is also rarely reachable through firewalls.

1.2 Design goals

1. **Federated.** Every domain publishes for itself. There is no registry, no central operator, no enrolment, and no bilateral agreement.
2. **Trivial to publish.** A conforming directory is a tree of static files on any HTTPS server, including object storage or a static site host, announced by one DNS TXT record. No dynamic code, no database and no new server software are required.
3. **Trivial to consume, and cheap to decline.** One DNS query and one HTTPS GET per address — two GETs, once per domain, for a domain certificate — using only primitives already present in every mail gateway and MUA. Crucially, a domain that does not support SCD is identified by the DNS query alone, with no HTTP request at all (Section 4.2.1).
4. **DNSSEC optional.** DNSSEC hardens availability, but the protocol MUST

be usable and meaningfully secure without it. It is not what makes a result trustworthy: that is path validation to an anchor the relying party configured, for addresses and domains alike (Section 6.1). The one thing a fetch adds — that a domain published a certificate as *its* certificate — rests on two things together, neither sufficient alone: the fixed host, whose TLS certificate can only be obtained by a party controlling the domain's DNS, the host that name resolves to, the domain's apex web server, the domain's registered contact or one of its standard administrative mailboxes (Section 5.6), and a signature under a key the domain publishes in its DNS (Section 5.7).

5. **Discovery is not trust.** The retrieved certificate is a *candidate*. The relying party decides whether it is acceptable, using the same path validation it applies to a certificate from any other source.
6. **Per-address and per-domain.** Both end-user certificates and domain-to-domain (gateway-to-gateway) certificates are supported, the latter with one requirement no CA can supply (Sections 5.6 and 5.7).

1.3 Relationship to existing work

Mechanism	Transport	Trust anchor for the binding	Deployment cost
SMIMEA [RFC8162]	DNS	DNSSEC (mandatory)	Needs DNSSEC + a non-mainstream RR type; needs DNS write access per user
OpenPGP WKD [WKD]	HTTPS	Web PKI (name of the host is derived from the mail domain)	Static files; an address record for <code>openpgpkey.<D></code> for the advanced method, no DNS change for the direct method
MTA-STS [RFC8461]	DNS TXT + HTTPS	Web PKI, with DNS TXT used	Static file + one TXT record

Mechanism	Transport	Trust anchor for the binding	Deployment cost
		only as a signal	
Central LDAP	LDAP	The directory operator	Central operator; firewall issues
SCD (this document)	DNS TXT + HTTPS	The issuing CA, per certificate (Web PKI only authenticates the fetch); for a domain certificate, also a key the domain publishes in DNS	Static files + one TXT record + one host name; a signing key for a domain certificate

SCD is intentionally the S/MIME analogue of WKD, with the discovery step taken from MTA-STS: a TXT record announces support, and a host name derived from the domain serves the files. SCD does not replace SMIMEA; a client MAY query both (see Section 8.3).

Note on prior art: the fixed sub-domain is WKD’s own construction. [WKD]’s *advanced method*, which clients must try first, serves the directory from a sub-domain with the fixed name `openpgpkey.<D>`, falling back to the apex (the *direct method*) only when that name has no address record; `smimecert.<D>` is that method with a different label. What this document takes from MTA-STS is narrower: the TXT record as the switch that announces support. MTA-STS never had a DNS-supplied host — its record only says that a policy exists, and the policy is always fetched from `mta-sts.<D>` — and [WKD] deliberately removed the DNS SRV-based discovery that its early revisions used, for two stated reasons: SRV lookups “posed a problem for implementations which have only limited access to DNS resolvers”, and a DNS-supplied host “reduces the certainty that a mail address belongs to a domain”, since an attacker able to influence a sub-domain could redirect the lookup. This document shares the second reason and follows both protocols: the TXT record carries no URL, and the directory is always at `smimecert.<D>` (Section 4.2). The first reason it accepts as a limitation, because its own switch is a TXT record rather than the existence

of an address record (Section 4.2.1). For an address lookup that is not what makes the result trustworthy — a CA attests the key-to-mailbox binding independently of the channel (Section 6.1) — but it keeps the set of hosts a relying party will ever connect to derivable from the domains it sends to (Section 4.3), and where the fetch itself carries an assertion no CA makes — domain certificates (Section 5.6) — it is half of what makes the assertion worth anything, the other half being the publisher’s signature (Section 5.7).

2. Terminology

- **Address:** an `addr-spec` as defined in [RFC5322], i.e. `local-part@domain`.
- **Publisher:** the party operating the directory for a domain, usually the domain owner or its mail provider.
- **Relying party (RP):** the party performing the lookup, typically an encryption gateway, or an MUA with full access to a DNS resolver (Section 4.2.1).
- **Directory base URL:** the `https://` prefix under which the well-known resources are served.
- **Certificate bundle:** the PEM document returned by a lookup.

Three symbols are used throughout, all produced by Section 3:

- **L** : the canonical local-part of the address (Section 3.1).
- **D** : the canonical domain — lower-case, A-label form (Section 3.1).
- **H** : the hash of **L** (Section 3.2), the name under which an address’s certificates are published.

In literal path and URL templates these appear inside angle brackets — `<D>` , `<H>` — marking a substitution slot. The angle brackets carry no meaning beyond that; `<D>` and **D** are the same value.

3. Canonicalisation and hashing

3.1 Address canonicalisation

Given an address, the RP and the publisher **MUST** derive the same canonical form:

1. Split the address at the last @ into a local-part and a domain-part.
2. **Local-part** (yielding `L`): if it is a quoted-string, remove the quoting and unescape it. Map all upper-case ASCII characters (`A - Z`) to lower case. No other transformation is applied: in particular, dots are NOT removed and sub-address (“plus”) detail parts are NOT stripped. The matching in Section 5.3 is correspondingly exact apart from case, in line with how deployed S/MIME implementations compare addresses (see Section 7.4). The local-part MUST be ASCII; an address with a non-ASCII local-part is out of scope of this version (Section 7.4).
3. **Domain-part** (yielding `D`): map to lower case. If the domain contains non-ASCII labels, convert it to its A-label (Punycode) form as specified in [RFC5891]. An address that is not syntactically valid under [RFC5322] and [RFC5321] yields no `D` at all (Section 7.4); an RP MUST NOT repair one. A domain-part that is a domain literal — `[192.0.2.1]` , `[IPv6:2001:db8::1]` — is a valid address form under [RFC5322] but names no domain: it yields no `D` , no lookup of either kind is performed, and the address has no SCD certificate. A non-ASCII local-part yields a `D` but no per-address lookup (Section 7.4).

3.2 Hashing

The canonical local-part `L` is hashed with SHA-256 and encoded as lower-case hexadecimal, producing a fixed 64-character string composed only of `0-9a-f` .

$$H = \text{lowercase_hex}(\text{SHA-256}(L))$$

Only the local-part is hashed. The domain is **not** hashed anywhere in this specification: it is already restricted to lower-case A-label form by Section 3.1, and so to letters, digits, hyphens and dots, at most 253 bytes, which is a valid filename on every filesystem a directory would be served from, and it already appears literally as a path component (Section 4.1). Hashing it as well would protect nothing, since it is visible as the directory name regardless, and would make the published tree unreadable to the operator debugging it.

The hash is **not** a security or privacy feature. It exists only to map an arbitrary local-part onto a fixed-length, filesystem-safe, case-insensitive name, so that a directory can be a plain directory of files on any operating system and any web server. Section 6.3 discusses why this does not conceal the address.

D is not included in H ; instead it appears literally in the URL path (Section 4.1). This avoids IDN normalisation mismatches inside the hash, allows one hash to be reused if a domain is renamed or aliased, and — unlike a scheme where the domain is only implied by the host name — keeps a multi-tenant directory unambiguous about which domain a given file belongs to.

SHA-1 with a z-base-32 encoding, as used by WKD, would serve this purpose equally well; the choice is not security-relevant. SHA-256 with hex encoding is specified because it needs no additional encoder — a standard library's digest and hex formatter suffice, whereas z-base-32 must be hand-written; because it avoids having to explain to auditors, procurement and FIPS-oriented reviewers why a new protocol uses SHA-1 at all, even where the use is not security-relevant; and because SMIMEA [RFC8162] already uses SHA-256 for the same purpose — though it truncates the digest to 28 octets to fit a DNS label and does not lower-case the local-part first, so the two names for one address are not interchangeable. The cost is a 64-character rather than a 32-character filename. A WKD-compatible SHA-1/z-base-32 profile could be added under a separate path segment and a new version tag, but supporting two hash forms simultaneously doubles the lookup cost and is NOT RECOMMENDED.

3.3 Test vectors

Canonical local-part	SHA-256, lower-case hex
joe.doe	a418287638a9d71c1a563a47d37aa26207daeb183f6cf720caaa44c
alice	2bd806c97f0e00af1a1fc3328fa763a9269723c8db8fac4f93af71c

Thus `Joe.Doe@Example.ORG` and `joe.doe@example.org` both map to the first row.

4. Discovery

4.1 Resource layout

All resources live under a directory base URL. Relative to that base URL:

```
<base>/<D>/hu/<H>      end-entity certificates for an address
```

<code><base>/<D>/domain</code>	certificates for domain-to-domain use
<code><base>/<D>/domain.sig</code>	the publisher's signature over the certificate in the file above; always present beside it (Section 5.7)

`D` is the canonical domain of the address being looked up and `H` the hash of its local-part, both as defined in Section 3. `D` is present in every path, including when the directory serves only one domain, so that the same layout works for single-tenant and multi-tenant hosting and so that a relying party can always confirm which domain a response was published for.

A request URL carries no query string. The path is the complete and only identifier of the resource (Section 4.3), which keeps one resource to one URL and therefore one cache key. A publisher whose backend is dynamic rather than a file tree indexes its own records by `H` at publication time; it does not need the address to be restated in the request, and **MUST NOT** require it.

Example, for `Joe.Doe@Example.ORG` :

```
https://smimecert.example.org/.well-known/smimecert/example.org/hu/
a418287638a9d71c1a563a47d37aa26207daeb183f6cf720caaa44dfdec31eae
```

(line wrapped for presentation)

4.2 Determining the base URL

The TXT record is mandatory. A domain supports SCD if and only if it publishes `_smimecertdir.<D>`. There is no convention-based fallback to the domain's own web server, and an RP **MUST NOT** probe for one.

A relying party determines the base URL for `D` as follows:

1. Query TXT at `_smimecertdir.<D>`.
2. If no record with `v=SCD1` is returned (NXDOMAIN, NODATA, or only records with other or unknown version tags), the domain does not support SCD. The RP **MUST NOT** attempt any HTTP request. See Section 4.2.1 on caching this result.
3. If more than one `v=SCD1` record is returned, the domain supports SCD. The RP **MUST** take `flags` from the records only where every record yields the same set of flags after parsing under Section 4.4.2, unknown flags removed and an absent field counting as the empty set, and apply the default otherwise; it **MUST** take `k=` from all of them as the union;

and it SHOULD log the condition. The record names no host and no policy, so two records never leave an RP unsure where to go, only what hints to take. Section 8.4 says why an ambiguous answer is still support, and why the keys merge where the flag does not.

4. The base URL is `https://smimecert.<D>/.well-known/smimecert`.

The TXT record carries no URL, and an RP MUST NOT take a base URL from DNS. The record announces that the directory exists; the host name is derived from the domain, as in MTA-STS. A deployment therefore needs only `_smimecertdir.<D> IN TXT "v=SCD1"` plus an address record or CNAME for `smimecert.<D>`, and no second DNS lookup, because the presence of the TXT record is itself the statement that the host exists. Section 4.3 gives the reasons, and Section 8.2 shows how hosting is delegated to a provider.

An administrator MAY configure a different base URL for a specific domain by hand, for example under a bilateral arrangement. That is local configuration, not something the protocol carries, and it is the only way a directory is ever fetched from another host. The override moves the host and nothing else. The TXT record is still required, since it is the switch and the source of `k=`; the TLS certificate is validated for the configured name; redirects stay on that host; the signature of Section 5.7 is verified under `D`'s own keys exactly as before; and held certificates and the memory of Section 6.2 are keyed by `D` as always. The address restriction of Section 4.5 applies too, because the configured name still resolves through the correspondent's DNS, unless the administrator exempts that one host explicitly — a bilateral arrangement over a private link is the one legitimate reason to point at a non-global address, and it is the administrator's decision to make, not a default. A correspondent that cannot publish the record is reached by a certificate assigned by hand (Section 5.6, requirement 1), which needs no fetch at all.

If a request against the base URL fails at the transport level or with a TLS error, the RP MUST treat that as a lookup failure; there is no alternative base URL to fall back to. A `404` is not a failure but a definitive answer that nothing is published (Section 5.4; the `404` for `domain.sig` beside a published `domain` is the one exception), and the two MUST be told apart: a `404` is cached as a negative answer and a failure is not (Sections 5.4 and 5.5), and a `404` for a certificate the RP obtained through this protocol is logged as a disappearance where a failure is not (Section 6.2).

The host is `smimecert.<D>` rather than `D` deliberately: the apex web server of a company is usually a CMS, a CDN or a marketing site outside the mail

team's control, and adding a well-known path to it is often the hardest part of deployment. A dedicated name can point at static hosting that the mail team owns.

4.2.1 Negative answers, and why the record is required

Requiring the TXT record makes “does this domain support SCD?” answerable by a single DNS query, with the answer cached by the resolver and shared across all senders behind it. Without it, the only way to learn that a domain does *not* support SCD is a TCP connection, a TLS handshake and an HTTP request that ends in a 404 — per domain, and in the worst case per recipient. For a gateway handling mail to a long tail of domains that will never deploy SCD, that is the dominant cost of the whole mechanism, and it is paid forever on domains that will never benefit from it.

Negative answers are therefore first-class:

- An RP **MUST** cache the absence of a `v=SCD1` record. The DNS negative TTL (derived from the SOA `MINIMUM` field and the SOA TTL [RFC2308]) **SHOULD** be honoured, clamped to a local minimum (RECOMMENDED: 1 hour) and maximum (RECOMMENDED: 24 hours) so that a domain deploying SCD becomes visible within a day.
- An RP **SHOULD NOT** re-query on every message. A per-domain negative cache consulted before any other work is the intended implementation.
- A resolver failure (SERVFAIL, timeout) is **not** a negative answer and **MUST NOT** be cached as one; it is a transient error, remembered against the domain's TXT query in the failure memory of Section 5.5 and retried when that lapses. Distinguishing “no record” from “could not ask” also matters for the memory of Section 6.2, which logs the first and not the second.

A TXT record is also a better switch than the one WKD uses. WKD's advanced method announces support by the existence of an address record for `openpgpkey.<D>`, which a site with wildcard DNS publishes for every name without meaning to; the WKD specification [WKD] therefore has to tell such sites to exempt that one name from the wildcard, and until they do, every lookup against them is an HTTP probe. A wildcard address record produces an answer for every name by accident; a wildcard TXT record carrying `v=SCD1` is a deliberate act, so “does this domain support SCD” has exactly one answer and an accidental wildcard answers it “no”.

The costs of this decision are two. The first is adoption friction: a publisher can no longer deploy SCD by dropping files on an existing web server, and must touch DNS. That is accepted deliberately. Publishing a certificate

directory is an explicit act by the domain owner, not something that should happen by convention, and the DNS record doubles as the switch that turns the feature off again. The second is that a relying party needs a resolver that will answer a TXT query. Every gateway and every full mail client has one; a sandboxed or mobile client that can only resolve addresses through its platform's name-lookup API does not, and WKD's choice of an address record as its signal was made for exactly such clients. This document does not serve them. Falling back to "does `smimecert.<D>` resolve" would reintroduce the wildcard problem and the per-domain probe above for a class of client this document is not written for, so a client without TXT access is out of scope of this version (Section 2).

4.3 Why the host is fixed

The base URL is always `https://smimecert.<D>/.well-known/smimecert`. The protocol has no way to name another host, for three reasons.

- **It bounds where a relying party connects.** The set of host names a gateway will ever open an outbound HTTPS connection to is derivable from the domains it sends mail to, and can be firewalled or proxied as such, and the port is 443 unless the directory itself redirects to another port on the same host (Section 4.5) — an operator who firewalls outbound traffic to 443 alone accepts that such a publisher is unreachable, and Section 7.9 tells publishers so. Had the TXT record carried a URL, an unsigned DNS answer — published by any correspondent domain, or spoofed — could have directed the gateway at a scheme, host, port and path of someone else's choosing, and had the response parsed. What the fixed host does *not* bound is the address: the publisher owns the address records for `smimecert.<D>` and can point them anywhere, including at an address inside the sender's own network. What it guarantees is that no HTTP request is ever sent to, and no response ever parsed from, a host that does not hold a certificate for `smimecert.<D>`, since the TLS handshake fails first. The connection attempt itself still happens, and Section 4.5 restricts where it may go.
- **It is what makes a publication the domain's own.** A Web PKI certificate for `smimecert.<D>` is only issued to a party that controls `D`'s DNS, the host that `smimecert.<D>` resolves to, `D`'s apex web server, `D`'s registered contact, or one of `D`'s standard administrative mailboxes, so a bundle served under that name with a valid certificate was published by the domain, or by the party the domain pointed the name at, whether or not the TXT record that led the RP there was

authentic. That is the first of the two conditions a domain certificate's authorisation rests on, which no CA supplies (Section 5.6); Section 5.7 adds the second, a signature under a key in the domain's DNS. Whether the certificate is trusted at all is decided by the RP's anchors, as for any certificate. A DNS-supplied host would have needed DNSSEC, an allow-list or a rule about the host lying inside `D` to recover that property, and a rule about sub-domains is weaker than it looks: many parties hold a valid certificate for *some* name under a large domain — every user-controlled sub-domain, every vendor a department CNAMEd in, every dangling CNAME — and none of them hold one for `smimecert.<D>`, with one exception that the provenance rule already counts: a holder of a `*.D` wildcard certificate, typically the apex CDN, to which a wildcard address record makes `smimecert.<D>` resolve, and which is then the party the domain pointed the name at (Section 5.6, requirement 2). This is the “certainty that a mail address belongs to a domain” that [WKD] names as one of its two reasons for dropping SRV-based discovery (Section 1.3), and the reason MTA-STS never had a DNS-supplied host.

- **It removes a class of rules.** No URL syntax to validate, no scheme, port or userinfo constraints, no normalisation of trailing slashes, and no field whose change has to be signalled to caches.

Delegation loses nothing. The domain owner points `smimecert.<D>` at the provider with a CNAME, and the provider obtains a certificate for that name through ordinary domain validation, which succeeds because the CNAME is real. This is how hosted MTA-STS providers work, and how static site hosts serve every custom domain. The `<D>` segment in every path lets one tree serve every tenant (Section 4.1, Section 8.2).

For an address lookup none of this decides whether a certificate is acceptable. The certificate carries its own binding — an `rfc822Name` naming the queried mailbox and a path to a trust anchor the relying party configured — and where the file was fetched from is not part of that decision (Section 6.1). The fixed host is an operational property for address lookups and a security property only for domain certificates (Section 5.6).

A request URL carries no query string, fragment or userinfo and names no port; the RP MUST NOT add a query string of its own (Section 4.1). A redirect may move to another port on the same host, nothing else may.

4.4 TXT record syntax

```
_smimecertdir.example.org. IN TXT "v=SCD1"
```

or, for a domain which publishes only a domain certificate (Section 4.4.1), with the signing key that any domain certificate requires (Section 5.7):

```
_smimecertdir.example.org. IN TXT "v=SCD1; flags=domain-only; k=<base64urlencodedkey>"
```

There is no lifetime field. How long a relying party remembers anything about `D` — a certificate it obtained, or that `D` yielded one — is the relying party’s own affair (Sections 5.5 and 6.2), and a value in an unauthenticated record could only have been used to shorten or lengthen that memory by whoever could spoof the record.

Fields are separated by `;` with optional surrounding whitespace. `v=SCD1` MUST be first: an RP recognises an SCD1 record by its first field, and discards a TXT record that does not begin with `v=SCD1`, as MTA-STS does [RFC8461]. A record which carries the tag elsewhere is therefore malformed, and Section 4.2 step 2 applies as if it were absent. A TXT record longer than 255 octets is carried as several character-strings, which an RP MUST concatenate in order and without separators before parsing, as for SPF [RFC7208] and MTA-STS [RFC8461]. The record is REQUIRED: its presence is what declares support, and a bare `v=SCD1` is a complete and valid record. Defined fields:

Field	Presence	Meaning
v	REQUIRED	Version tag. SCD1 for this document.
flags	OPTIONAL	Comma-separated publisher hints. <code>domain-only</code> is defined (Section 4.4.1). Unknown flags MUST be ignored. A flag never grants or withdraws anything: consent to the use of the domain certificate is its publication (Section 5.6).
k	REQUIRED for a domain that publishes <code>/domain;</code>	One or two comma-separated Ed25519 public keys, each the

Field	Presence	Meaning
	absent otherwise	base64 of the 32-octet public key encoding of [RFC8032], with which the publisher signs its domain certificate (Section 5.7). Two keys are for rotation; a relying party reads up to eight (Section 4.4.2). Its presence is also the announcement that a domain certificate exists: without it a relying party does not fetch /domain . Unlike a flag, a key bears on trust; Section 5.7 says why that is safe, in short because a key names an acceptable signer but never suffices without the host.

Unknown fields MUST be ignored. A record whose version tag is unknown MUST be treated as “no record”; this is what allows a future SCD2 to be deployed alongside SCD1 without breaking existing relying parties. When more than one `v=SCD1` record is present, `flags` is taken from them only where they all agree and `k` from all of them (Section 4.2, step 3): support is announced, the host is derived, the keys are merged, and the flag stands if every record carries it and falls to its default if any does not, so a publisher that leaves two differing records standing loses `domain-only` until it cleans up, and nothing else. Section 4.4.2 says how a relying party reads a record it did not write.

Publishers withdrawing from SCD MUST remove the TXT record — it is the switch, and a record left standing keeps senders connecting to a host that may no longer be the publisher’s (Section 7.9). Nothing else is needed: a

relying party that holds certificates from `D` keeps using them until they expire or are revoked (Section 6.2), logs the record's disappearance once, and looks no further. A publisher that wants its certificates to stop being used revokes them (Section 7.6).

4.4.1 The domain-only flag

`flags=domain-only` states that the publisher does not publish per-address certificates for this domain. An RP that sees it SHOULD skip the per-address lookup and fetch the domain certificate directly. A record that carries the flag but no `k=` is inconsistent — a domain certificate cannot be published without a signing key (Section 5.7) — and the RP ignores the flag and performs the per-address lookup as if it were absent. The flag addresses senders that would use the domain certificate. A sender whose policy under Section 5.6 requirement 4 does not permit a domain certificate for `D` has no use for `/domain` and ignores the flag too: it performs the per-address lookup as if the flag were absent, since the alternative — skipping the lookup on the strength of a hint it cannot verify, and then holding nothing — would let a spoofed flag leave it without a certificate the domain publishes. Such senders are the exception, and what they cost a `domain-only` publisher is one cached `404` per address per cache lifetime (Section 5.5). A sender that would use the domain certificate fetches it, and only a `404`, a fetch that fails, a signature that does not verify, or a certificate it cannot accept — one that does not validate to its anchors, say under a private root it has not imported — sends it back to the per-address lookup (below). Anything short of a domain certificate the sender can use is the same case: the flag promised a certificate and did not deliver one.

This is a hint about cost, not a statement about trust. The per-address lookup is the one cost in this protocol that does not aggregate: a domain-certificate fetch happens once per domain and its result caches as a single entry, while a per-address `404` has to be paid for every distinct correspondent, forever. For a domain that has decided never to publish per-address certificates — which is precisely the gateway-to-gateway deployment the domain certificate exists for — without this flag every new recipient costs a TCP connection, a TLS handshake and a `404` before the RP falls back to a certificate it could have used immediately. With it, the whole domain costs one DNS query and two HTTPS requests, the file and its signature, all cacheable.

Because it is only a hint, it carries no authorisation, and a TXT record is not authenticated without DNSSEC. Consent to the use of the domain certificate is the publication of `/domain` under `smimecert.<D>` (Section 5.6,

requirement 4), which a spoofed TXT record can neither create nor — once a sender has verified the domain’s signature and holds the certificate (Section 5.7) — remove. Two rules keep the flag harmless whatever DNS says:

- **A 404 on /domain falls through, and so do a failed fetch, a signature that fails, and a certificate the sender cannot accept.** If the flag is set but no domain certificate is published, or the fetch fails, or the one served does not verify (Section 5.7), or it verifies but does not validate to the sender’s anchors (Section 5.6, requirement 3), the RP MUST perform the per-address lookup after all. The last case is what a domain that publishes a private-CA domain certificate beside public-CA mailbox certificates looks like to a stranger, and without it a spoofed flag alone would keep that stranger from every certificate the domain publishes. A spoofed flag on a domain which publishes per-address certificates and no domain certificate is ignored outright when it comes without a key (above), and costs one cached /domain fetch when the spoofer adds a key; a spoofed key on a domain that does publish one costs the same fetch and a failed verification. Nothing else changes. Without this rule a spoofed record would suppress every certificate the domain publishes.
- **Its absence hides nothing.** An RP fetches /domain after a per-address 404 or a failed per-address fetch whether or not the flag is present, provided the record carries k= (Section 5.6, requirement 4; Section 5.7). A record from which the flag has been stripped therefore costs one per-address 404 per recipient before the domain certificate is found, and changes nothing else. Stripping k= instead does hide the domain certificate from a sender that does not yet hold it; that is the first-contact denial Section 5.7 accepts, and it is the one loss from the record that matters (Section 4.4.2).

A manipulated flag cannot steer an RP onto a key the domain did not publish, and it cannot keep an RP away from one it did. That is what allows the flag to live in an unauthenticated record at all, and why DNSSEC is not needed for it. What it can do, beyond adding requests, is choose between two keys the same domain published: on a domain that publishes per-address certificates *and* a domain certificate, a spoofed domain-only causes a sender whose policy permits domain certificates to omit the mailbox key for recipients it holds no certificate for. Under the “both” policy that Section 7.1 recommends this changes nothing that matters, since the message was going to be encrypted to the domain key anyway and the perimeter could read either way; under the “instead” policy it moves such mail from the mailbox key onto the domain key. That is accepted. Both keys

are the domain's own, and by publishing `/domain` the domain consented to that key being used for any of its mailboxes (Section 5.6, requirement 4); steering onto a key the domain itself published is accepted in general, since Section 5.6 requirement 4 lets a sender fall through to the domain certificate even on a failed per-address fetch, on the ground that encrypting to that key is never worse than not encrypting; and a domain that wants its mailbox keys to be the only keys does not publish `/domain`. The flag's appearance on a record that lacked it when the RP last read it, or on a domain from which the RP has obtained per-address certificates through this protocol, is a change worth logging (Section 6.6). Holding per-address certificates that arrived some other way is not: a domain that publishes only a domain certificate while its users sign their mail with mailbox certificates is a normal deployment (Section 7.3), and its correspondents hold both from the start.

The flag skips the lookup, not a certificate already held. `domain-only` tells an RP not to *look for* a per-address certificate. It does not tell it to stop *using* one it already holds, from any source (Section 7.1), while that certificate is valid under Section 5.3: a held per-address certificate is always used (Section 7.5), and the flag does not change that. An RP MUST NOT substitute the domain certificate for a recipient it holds a valid per-address certificate for — “substitute” meaning use *in place of*; adding the domain certificate beside it is the “both” policy of Section 7.1 and is permitted. No sender benefits from dropping a valid mailbox key, and a publisher that wants a mailbox key retired revokes it (Section 7.6), which is why this is a rule and not a policy. Under the “instead” policy this bounds the spoofed flag above to first-contact mail; under “both” the flag could never change who can read the message in the first place.

One consequence remains for an implementer. Under `domain-only` the domain certificate is the sole path to every recipient at that domain, so what matters is who can choose it. The host alone cannot: every domain certificate is signed under the key in `D`'s TXT record (Section 5.7), so the choice belongs to the holder of the signing key, or to an attacker who can both write to the host and spoof the sender's DNS. Whoever chooses, the certificate still has to validate to an anchor the sender configured and name `D`, as any certificate does, and the continuity check of Section 5.6 requirement 5 is what notices a change, whoever made it.

A publisher could achieve the same effect without any flag by publishing a per-address certificate for every mailbox, all carrying the same gateway key. That works, but it enumerates the domain's mailboxes in a public directory, which a domain choosing gateway-to-gateway encryption usually does not

want (Section 6.3).

4.4.2 Parsing

The record is unauthenticated input from a stranger, parsed by every relying party on the internet, so the rules have to be the same everywhere and never fatal. The principle is the one Section 4.2 step 3 already applies to multiple records: the only thing a record must get right is to begin with `v=SCD1`. Everything after that is a hint, and a hint that cannot be read falls to its default. No malformed field turns a supporting domain into “not supported”, and no malformed field yields a value its publisher could not have meant.

- **Recognition is by the first field alone, and is final.** A record that begins with `v=SCD1` announces support. An RP MUST NOT treat it as absent because of anything that follows, and MUST treat a record that begins with anything else as absent (Section 4.4).
- **Separators.** Fields are split on `;`. Whitespace around a separator, around the `=` of a field, and at either end of the record is ignored. Empty fields — a trailing `;`, a doubled `;;` — are ignored, so `v=SCD1;` is valid.
- **Names and values are matched exactly, in lower case.** `Flags` is an unknown field and is ignored like any other unknown field. The version value `SCD1` is likewise exact.
- **A field that appears more than once is taken only if every occurrence yields the same parsed value,** and ignored as if absent otherwise — except `k`, whose occurrences are merged as for multiple records (Section 4.2, step 3). A hint said twice is one hint; a hint said two ways is none; a set of keys has a merge.
- **flags** is split on `,`, with whitespace around the commas ignored; empty elements and duplicates are ignored, and each flag is matched exactly in lower case. Unknown flags are ignored (Section 4.4).
- **k** is split like `flags`. Each element must decode from standard base64 [RFC4648], padded, to exactly 32 octets; an element that does not is dropped. The keys of all `k` fields in all records are merged, in order of appearance, into one set of at most eight; keys beyond the eighth are ignored, so that no record can make a sender attempt more than eight verifications. A field with no valid element is unreadable and contributes nothing. A readable `k` is the one field whose loss or replacement matters, and Section 5.7 says why a sender that already holds the domain certificate is unaffected by either.
- **Anything else** — a field with no `=`, a value containing characters

outside printable ASCII, a stray character-string that will not concatenate into fields — is ignored field by field. The record still announces support.

4.5 Transport requirements

- The RP MUST perform full PKIX validation of the server certificate for the base URL host against the Web PKI trust store, including host name verification [RFC9525].
- The RP MUST NOT respond to an HTTP authentication challenge (401). Directory content is public. (This mirrors the corresponding WKD rule; prompting a user for credentials during an automated certificate lookup is a phishing vector.)
- The RP MUST use `GET`. No other method has a role in this protocol; nothing in this document overrides the ordinary HTTP method requirements of [RFC9110].
- The RP SHOULD revalidate cached entries with a conditional `GET` [RFC9111] (`If-None-Match`, or `If-Modified-Since` where no `ETag` is available) and MUST handle `304 Not Modified` by extending the freshness of the cached bundle rather than discarding it. Servers SHOULD emit `ETag` and `Last-Modified`; static file servers do so by default.
- Redirects: at most 5, `https` only, and MUST stay on the host name the request was made to; the port MAY differ. A redirect to another host MUST be treated as a lookup failure, so that the bounded set of hosts of Section 4.3 holds through redirects too.
- The RP MUST impose a response size limit (RECOMMENDED: 256 KiB) and a total time budget per recipient lookup (RECOMMENDED: 10 seconds including DNS). One lookup makes one TXT query, the address resolutions its connections need, and at most five resource fetches, each following at most five redirects (Section 8.4), and a resource or host that failed is not asked again within the failure memory of Section 5.5; Section 6.4 adds the per-message and per-domain limits above this. The size limit counts the bytes actually read, during the read, and the read is abandoned the moment it is reached; `Content-Length` is advisory, absent on chunked responses and written by the publisher. A `Content-Encoding` of `identity` or `gzip` MUST be accepted, with the limit applied to the *decoded* bytes, so that a small body cannot exceed it by expanding; any other coding is a lookup failure. Whether the RP requests compression is its choice; both are correct (Section 8.4 says why `gzip` must be accepted either way).

- The RP MUST NOT connect to an address that is not globally routable: loopback, link-local, the unspecified address, the private and unique local ranges, and the cloud metadata address `169.254.169.254` (which the link-local rule covers) — in short, anything the IANA special-purpose address registries [RFC6890] mark as not globally reachable. Every address the name resolves to is checked, and again after every redirect, because the same name may resolve differently a second later; an address that fails the check is skipped, and the lookup fails only when no address remains. The connection MUST be made to an address that was checked, not to a fresh resolution of the name, since a second resolution between the check and the connect is exactly the rebinding the check is meant to catch. The publisher chooses the address records for `smimecert.<D>`, so without this rule a correspondent domain can make the gateway open a connection to any address the gateway can reach (Section 6.4). An RP that sends its outbound HTTPS through a proxy delegates this check to the proxy, which then has to enforce it; a proxy that forwards to any address the gateway names is not a mitigation but a relay.
- The RP SHOULD send `Accept: application/pem-certificate-chain`.

5. The certificate bundle

5.1 Format

The response body is a PEM document [RFC7468] with media type `application/pem-certificate-chain` [RFC8555]. It contains one or more `CERTIFICATE` blocks. Nothing other than PEM blocks and whitespace is permitted; explanatory text outside the blocks MUST NOT be present.

Publishers SHOULD order the file as: end-entity certificate, then the intermediate CA certificates needed to build a path towards a public trust anchor, in issuing order. Root certificates SHOULD NOT be included; they add size and confer no additional trust. The exception is a publisher's own private CA (Section 7.5, option B): its root SHOULD be included, so that an administrator on the relying side who decides to trust it can fetch the bundle on purpose and import the root from it by hand. No lookup stores it: a relying party keeps nothing from a bundle but what it accepted and the intermediates that certificate validated through (Section 6.2), and a self-signed certificate is never stored from one — if it is already a configured anchor it is used as that anchor, and if it is not it is nothing.

Relying parties **MUST NOT** depend on the ordering. An RP **MUST** parse the whole document into a certificate *pool*, treat every certificate in it as a candidate — requirement 1 of Section 5.3, a name matching the queried address, is what selects the end-entity ones — and perform its own path building using the remaining certificates as untrusted intermediates. This is both more robust against publisher mistakes and directly implementable with the path-building primitives every PKIX library provides (Section 8.4).

5.2 Multiple certificates

More than one end-entity certificate for the same address is expected and legitimate:

- separate signing and encryption certificates,
- an RSA and an ECC certificate published in parallel during an algorithm migration,
- an overlapping successor certificate published before the current one expires.

All of them are concatenated in the same bundle, together with their intermediates. Because the RP treats the file as a pool and does its own path building, no delimiter or ordering convention is needed.

5.3 Selection and binding rules for address lookups

This section applies to `/hu/` lookups. Domain certificates are governed by Section 5.6, which imposes additional requirements that this section cannot supply.

An RP **MUST** discard any candidate end-entity certificate unless **all** of the following hold:

1. **Identity binding.** The certificate contains **at least one** `subjectAltName` of type `rfc822Name` whose domain-part matches `D` and whose local-part matches the queried local-part, compared as in Section 5.3.1. The test is existential: a certificate may legitimately carry further names — aliases, other domains — and those are simply not relevant to this lookup. A certificate with no matching name is discarded, however well it validates.
2. **Path validation.** A path to a trust anchor accepted by local policy can be built and validates, including basic constraints, name constraints, path length and signature algorithm policy.
3. **Validity period.** The certificate is currently valid. An expired certificate

MUST NOT be used for encryption.

4. **Key usage.** For encryption: if a `keyUsage` extension is present, it includes `keyEncipherment` (RSA) or `keyAgreement` (ECDH); and if an extended key usage extension is present, it includes `emailProtection` (or `anyExtendedKeyUsage`). An absent extension imposes no restriction: [RFC5280] restricts use only when an extension is present, and for extended key usage adds that an application may require the extension, which this document does not; `keyUsage` is given the same treatment here, as this document's own rule.
5. **Revocation.** Revocation status is checked according to local policy (CRL and/or OCSP). A certificate known to be revoked MUST NOT be used.

If several candidates remain, the choice is local policy. An RP may use any of them, or several at once: CMS `EnvelopedData` carries a SET OF `RecipientInfo`, so the content-encryption key can be encrypted under more than one of the recipient's certificates and the receiving side uses whichever private key it holds (Section 7.1).

5.3.1 Name comparison

The same comparison rules apply wherever this document matches a certificate name against a queried address or domain, including Section 5.6 requirement 3.

- **Domain parts** are compared case-insensitively, in A-label form.
- **Local parts** SHOULD be compared case-insensitively. This departs from [RFC5321], which makes the local-part case-sensitive in principle, and follows deployed practice (Section 8.4). An RP MAY be configured to require an exact, case-sensitive match; note that Section 3.1 lower-cases the local-part before hashing, so a publisher whose users have mixed-case addresses should publish accordingly (Section 7.4).
- **Quoted local-parts** are compared in their unquoted form, which is what Section 3.1 produces as `L`. A SAN such as `"john.doe"@example.org` is unquoted the same way before comparison, so a local-part quoted without needing to be, `"alice"@example.org`, matches `alice`.
- **Only `rfc822Name` is matched.** No other SAN form plays a part in this document: a `dNSName` is forbidden in a publicly issued S/MIME certificate (Section 5.6) and matched by no mail client, and the S/MIME name forms that could carry an internationalised address are out of scope (Section 7.4). A subject `emailAddress` attribute is not consulted either; the S/MIME Baseline Requirements require every mailbox in the subject to be repeated in the SAN. Wildcards and sub-domain

relationships have no meaning in an `rfc822Name` and MUST NOT be inferred.

- **Degenerate values** match nothing. An empty domain part, an `rfc822Name` containing no `@`, or a value that is not a valid address is malformed; an RP MUST NOT attempt to repair it.

5.4 Absence

If no certificate is published, the server MUST return `404`. An RP MUST treat `404` as “no certificate available”, not as “the address does not exist” and not as a revocation signal. Removal of a file from a directory is not a revocation mechanism (Section 7.6). The one `404` that is not a negative answer is a `404` for `domain.sig` beside a published `domain`: that is a publisher’s deployment error, and it is remembered as a failure, not as an absence (Section 5.7).

The `404` SHOULD carry a `Cache-Control: max-age` of between one hour and one day, chosen by how soon a certificate published for a new user should become visible to a sender that recently found nothing; a relying party caches the negative answer within that range whatever the server says (Section 5.5), so `no-cache` or `no-store` on a `404` buys nothing, and a server that sends no directive gets the relying party’s default of one hour.

Any status other than `200` or `404` — including `403`, `410` and `429` — is a lookup failure, not a negative answer, and MUST NOT be cached as one. It is remembered as a failure instead (Section 5.5), which is a different thing: a negative answer says nothing is published, a failure says do not ask again yet.

5.5 Caching

An RP caches every definitive answer, positive or negative, keyed by the resource fetched: one entry per address, and one each for a domain’s `domain` and `domain.sig`. The cache is the RP’s own protection against a lookup per message and against the amplification of Section 6.4, so the server’s directives steer it but do not switch it off. An RP SHOULD honour `Cache-Control` within a local minimum and maximum, and MUST NOT let a directive take an entry below the minimum: `no-cache`, `no-store` and a `max-age` under the minimum all mean “the minimum”. A directive above the maximum means the maximum, and no directive means the RP’s default.

Answer	Server SHOULD send	RP minimum	RP default (no directive)	RP maximum
200 , a bundle	max-age up to 7 days, and an ETag	1 hour	1 day	7 days
404 , nothing published	max-age between 1 hour and 1 day (Section 5.4)	1 hour	1 hour	1 day

The values are RECOMMENDED; what is required is that both bounds exist and that the negative maximum is short enough for a newly published certificate to be found within a day. A cached bundle is revalidated with a conditional GET (Section 4.5), which is why the ETag matters more than the max-age , and the seven-day maximum is what sets the rollover overlap of Section 7.2. A certificate the RP holds MUST still be validated before each use — path, expiry and revocation, against the anchors configured at that moment (Section 6.2); the cache remembers what was published, not whether it is still good.

A 200 that yields nothing usable is a negative answer, not a positive one. A body that parses to no certificate — a text/html error page served with a 200 is the commonest real-world failure (Section 7.9) — or whose certificates all fail Section 5.3, or an empty or all-invalid /domain , is cached as a 404 would be, within the negative bounds of the table, and logged, since it is almost always a publisher’s error. Caching it as a positive answer would hide the error for up to a week where the negative bounds exist precisely so that a publisher’s mistake is noticed within a day.

Failures are remembered. A fetch that fails — any lookup failure of Sections 4.5, 5.4 and 6.4, or a signature that does not verify (Section 5.7) — is remembered for the local minimum above (one hour), and what it is remembered against is not requested again while the record is live; the lookup proceeds as if the fetch had just failed. The failure memory is not the negative cache: it says “do not ask again yet”, not “nothing is published”, so it is never extended by a server directive and never satisfies a lookup. It stops requests, not answers the RP already has: a fresh cache entry is served before the memory is consulted. A failure noted against a resource

evicts that resource's cache entry, as a negative answer replaces it, so a 200 whose body was rejected — a bundle over the parse bound, or a / domain pair whose signature does not verify — is not served again from the cache. It is keyed by what failed. A *transport failure* — a connection that cannot be made or is refused, a TLS failure, a timeout, or a name with no address that passes Section 4.5 — is remembered against the host name the request was made to and against the address it was made to, if any; the host name an administrator who points several domains at one configured host (Section 4.2) shares among them knowingly. A *resource failure* — a status other than 200 or 404, a body over the size limit, a content coding other than identity or gzip, a redirect off the host name or beyond the fifth, or a bundle over the parse bound of Section 6.4, which is never truncated — is remembered against the resource, hu/<H>, domain or domain.sig. A signature that does not verify is remembered against both domain and domain.sig, and a resolver failure against the domain's TXT query (Section 4.2.1). Section 8.4 says why the keys differ.

The cache and the certificate store are different things. The cache entry says when to *look again*; its expiry triggers a new lookup for a successor. The certificate that was accepted lives in the store and is used until it expires, is revoked, or is removed by an administrator (Section 6.2), and a lookup that fails, or finds nothing, or finds a successor, never shortens that. A sender therefore never loses a certificate because a cache entry lapsed or a fetch failed; it only, at worst, fails to learn of a new one.

5.6 Domain certificates — an assertion no CA makes

A domain certificate is used to encrypt to an entire domain, typically gateway-to-gateway. Accepting one requires two distinct assertions:

- **A key binding:** this public key belongs to the named subject. A CA can make this assertion, and does.
- **A domain authorisation:** this key is authorised to receive mail for every mailbox in D. **No CA makes this assertion, and no X.509 field expresses it.** An rfc822Name must be a complete mailbox under [RFC5280], so a bare domain cannot appear there. A dNSName names a host, and the S/MIME Baseline Requirements forbid it in an S/MIME certificate's SAN outright (they permit rfc822Name, Smtputf8Mailbox, directoryName and, in the legacy and multipurpose profiles, validated otherName entries, and forbid every GeneralName the section does not list), so no publicly trusted CA can issue one, and no mail client would match it if one did. There is no key usage, extended key usage or name form with the required meaning. The nearest thing a public CA does

assert sits one level up: before placing an `rfc822Name` name constraint on a subordinate CA certificate, the CA must confirm that the applicant has registered the domain or is authorised by its registrant ([SBR], Section 7.1.5). That is a verified statement about domain authority, made about the holder of the subordinate CA rather than about any one key. This document does not rely on it: the authorisation it needs comes from the domain itself, by publication (below) and by signature (Section 5.7).

The authorisation therefore cannot come from the certificate. It can only come from the domain itself, and the act of publishing under the domain's own directory, signed under a key the domain publishes in its DNS (Section 5.7), *is* that statement. For the domain certificate, provenance and signature are not redundant with path validation — they are the two channels the authorisation travels on, and both are required.

Consequently:

1. **Source restriction.** An RP MUST NOT treat a certificate as a domain certificate for `D` unless it was retrieved from `<base>/<D>/domain` in `D`'s own directory, or was configured by an administrator. A domain certificate MUST NOT be harvested from received mail, taken from an address lookup, or promoted from any other source. Without this rule any insider holding a valid mailbox certificate for `D` could become the domain's gateway key by signing one message and letting the receiving gateway harvest it. The rule does not make such an insider harmless; it makes the publication host the thing the insider must control. What a compromised or taken-over `smimecert.<D>` can then do is in Section 6.6 and Section 7.9: inside an organisation that certifies its own staff, any of those certificates would do, so the host is the scarce thing, not the certificate (Section 7.8) — which is why the signature of Section 5.7 takes the choice away from a host acting alone.
2. **Provenance.** Because the fetch carries an authorisation claim, a domain certificate MUST have been fetched from `https://smimecert.<D>/well-known/smimecert/<D>/domain`, with the TLS certificate of that host validated as in Section 4.5, or from a base URL an administrator configured for `D` by hand (Section 4.2). Nothing fetched from anywhere else is a domain certificate for `D`. For a base URL an administrator configured for `D` by hand (Section 4.2), in place of the default under `smimecert.<D>`, the argument in the next paragraph does not apply: the TLS certificate of that host is for a name no CA has tied to `D`, so it cannot show that `D` published the file. What takes its place is the configuration itself: by pointing `D` at that base

URL, the administrator asserted that it is `D`'s. That replaces only this half of the authorisation. The signature of Section 5.7 is still required, and is verified under `D`'s own keys exactly as for `smimecert.<D>`.

This is what makes the publication an assertion by `D`. A Web PKI certificate for `smimecert.<D>` is only issued to a party that controls `D`'s DNS, the host that `smimecert.<D>` resolves to, `D`'s apex web server, `D`'s registered contact, or one of `D`'s standard administrative mailboxes — the TLS Baseline Requirements [TLSBR] validate a name by a DNS change, by a challenge answered at the host the name points to, by a challenge answered at an “Authorization Domain Name” that the CA may form by pruning labels from the left, which for `smimecert.<D>` is `D`'s own web server (the one Section 4.2 says is usually outside the mail team's control), by a message or a call to the contact in the registry, in a CAA record or in a validation TXT record, or by mail to `admin@`, `administrator@`, `webmaster@`, `hostmaster@` or `postmaster@` at the domain (the “constructed email” method). Whoever holds any of those is in the same position as the domain's DNS operator for this purpose, which is the insider requirement 1 is about — so a bundle served under it was placed there by the domain, or by whoever the domain pointed the name at, even if the TXT record that led the RP there was spoofed. It does not depend on DNSSEC. Delegated hosting keeps the property, because the provider serves the name `smimecert.<D>` under a certificate for that name (Section 4.3). The assertion is therefore exactly “the party operating `smimecert.<D>` published this”, which is the domain for as long as the domain keeps the name pointed where it intends; a name left pointing at a provider the domain no longer uses, or at a reallocated address, hands the assertion to whoever claims the far end (Section 7.9).

3. **Certificate requirements.** The certificate MUST contain at least one `rfc822Name` SAN whose domain-part equals `D`, compared as in Section 5.3.1; the local-part is ignored. A domain certificate is thus an ordinary mailbox certificate, which is what every S/MIME CA, public or private, issues, and what every existing domain-to-domain deployment already exchanges by hand; this document defines no host-name form for it, because no public CA may issue a certificate with a `dNSName` in its SAN and no mail client matches one when it looks for a recipient's certificate (the `dNSName` point at the start of this section). A publisher will find a stable local-part such as `gateway@D` convenient for the continuity logs of requirement 5, but nothing turns on it. As in Section 5.3 the test is existential, and for the same reason it is safe to be: a certificate a public CA issued to a multi-tenant gateway may legitimately name several

domains, and names other than the matching one confer nothing here, because acceptance as *D*'s domain certificate rests on requirements 1 and 4 — the certificate came from *D*'s own directory and carries *D*'s signature — not on the SAN. The SAN is what ties the certificate to *D*, not what authorises it: it is a MUST because it catches a signed file served under the wrong tenant's directory, and because every rule that says "names *D*" leans on it. (Under a private CA constrained as Section 7.5 recommends, other domains are forbidden in any case.)

Requirements 2 to 5 of Section 5.3 (path validation, validity, key usage, revocation) apply unchanged. In particular a domain certificate MUST validate to a trust anchor the RP configured, whoever issued it; a certificate which does not is not a domain certificate for *D*, and this document provides no way for a fetch to make it one. Section 7.5 says how a publisher's private CA comes to be such an anchor.

4. **Consent is the publication, and it is not sufficient.** The receiving domain consents to its own key being used for its own mailboxes by publishing `/domain` under `smimecert.<D>`. Nothing in DNS grants or withdraws that consent: the `domain-only` flag (Section 4.4.1) only tells an RP that per-address lookups are not worth making. Consent does not authorise the sending RP to use the key: that decision turns on what the sender is sending and what it is willing to have decrypted under one key for the whole domain, which the receiving domain cannot know. A per-address certificate may be a perimeter key as well — a gateway that publishes for its users may hold their private keys (Section 7.8) — and the sender cannot tell, so the policy is not a choice between perimeter and mailbox. It is a choice about a key the sender knows to cover every mailbox at *D*, whose use rests on publication rather than on a CA's binding to one mailbox, and whose compromise exposes all of them at once.

The sending RP MUST therefore act on its own configured policy. That policy has two layers, and they compose in the ordinary way. A **per-domain entry** records a decision an administrator made about one domain: a certificate assigned to *D* by hand, or a root imported for *D*'s certificates with *D* named. A **default** applies to every domain without an entry, and MAY be off, MAY be a list, or MAY be blanket acceptance of any domain publishing a domain certificate, optionally conditioned further (on the issuer, for example). A per-domain entry takes precedence over the default, and a certificate accepted under an entry is not subject to the default's conditions. The requirement is only that the sending side has set the policy — not that it is narrow, and not that

a human is consulted per message. An implementation **MUST** have such a policy, and its shipped default is one of the policy's values; a default of "off" is a policy. This document recommends no default for it: it is the decision with the blast radius described at the end of this section, and it is the deployment's to make. (Whether a domain certificate is used *beside* or *instead of* a held mailbox certificate is a separate choice, Section 7.1, which does have a recommended default.)

Whatever the policy, a domain certificate fetched through this protocol is one only if it carries a signature that verifies under the key in `D`'s TXT record (Section 5.7); there is no unsigned `/domain`. The policy decides whether the sender *uses* a domain certificate for `D`; a sender that uses one checks the signature, without exception, and a sender that uses none need not fetch it. Nothing about the certificate itself — its issuer, its type, how its path is constrained — makes it acceptable, because a compromised host can present any certificate the organisation ever issued and only the signature takes that choice away from the host. A domain that cannot sign does not publish `/domain`; it hands its certificate to a correspondent's administrator, who assigns it to `D` by hand, which is not a fetch and needs no signature. A sender **MAY** add conditions of its own on top of the signature — an issuer, a certificate type under [SBR] — as local policy; this document asks for none.

The structural reason is that the certificate arrives in data fetched from the domain it concerns. If publication alone sufficed, any domain could unilaterally change which key every sender encrypts to by publishing one file. Fetched data may supply material for a decision; it may not authorise the decision.

A missing per-address certificate, for whatever reason, falls through to the domain certificate. This rule governs *substitution*: using the domain certificate in place of a per-address one (Section 7.1). An RP **MUST NOT** use the domain certificate as a recipient's only certificate while it holds a valid per-address certificate for that recipient from any source. When it holds none, it **SHOULD** use the domain certificate. That holds in three cases: the per-address lookup returned a definitive `404`; it was skipped under `domain-only` (Section 4.4.1); or it *failed* — a transport failure, a TLS error, a timeout, or any status other than `200` or `404` (Section 5.4). The third case comes with two conditions. The failure is remembered as a failure for the local minimum (Section 5.5) and never cached as a negative answer, so that the per-address lookup is retried within the hour and the fallback is bounded in time rather than sticky. And the fallback is logged, since a

correspondent whose per-address fetches keep failing is worth a look. Which domain certificate a failed lookup can reach depends on the kind of failure. A transport failure is remembered against the host, and `/domain` lives on the same host, so it reaches only a domain certificate the sender already holds or has cached. A status failure leaves a fresh `/domain` fetch possible.

The fallback on a failure is deliberate: an attacker who can break the per-address fetch could steer mail onto the domain key, but refusing the fallback would send the message to the failure policy of Section 7.7, every outcome of which is worse for the sender than encryption to a key the domain published for all of its mailboxes (Section 8.4). Encrypting to a key the domain itself published is never worse than not encrypting; the failure policy is for when there is nothing left to encrypt to. Adding the domain certificate beside a held per-address certificate is not substitution and is outside this rule altogether. The rule holds in the other direction: a `404` on `/domain` is a definitive statement that there is no domain certificate, and under `domain-only` it sends the RP back to the per-address lookup (Section 4.4.1), and so does a `/domain` fetch that *fails*, a signature that does not verify, or a certificate that verifies but does not validate to the sender's anchors — otherwise a host that wanted to deny could do so more cheaply with a `500` than with a bad signature. A failed `/domain` fetch yields no domain certificate, and only when the per-address lookup has nothing either does Section 7.7 decide.

5. **Continuity.** An RP SHOULD record the issuer and public key of the accepted domain certificate, and SHOULD alert an administrator when the *issuer* changes, or when the certificate is replaced substantially before its normal renewal window (RECOMMENDED: a replacement while the previous certificate still had more than a third of its validity, or more than ninety days, to run) — not on every rotation. Routine renewal from the same issuer, whoever the issuer is, is unremarkable and alerting on it is actively harmful: an operator who learns that the alert means “renewal” will dismiss the one occasion when it does not. This is monitoring, not a trust mechanism: a certificate which does not validate is rejected by requirement 3 whether or not anything was recorded. Under `domain-only` this check carries the entire detection burden for the domain, since no per-address certificate exists to fall back to. The *first* domain certificate seen for a domain the RP has corresponded with before, whether it held per-address certificates for that domain or sent to it in the clear, is a change in the same sense and SHOULD be logged — “corresponded with” being what the RP can tell

from the certificates it holds for the domain, the memory of Section 6.2, and a record of domains sent to where it keeps one; the rule as stated would otherwise only ever compare against a previous domain certificate, and a taken-over host (Section 7.9) presents its certificate as the first one.

The practical consequence is not that domain certificates require manual work per correspondent — under a blanket policy they are discovered and used as automatically as address certificates. It is that the sender chooses the risk profile in advance rather than inheriting it from the publisher. Where the policy names the domains it permits — a per-domain entry, or a list — the two organisations have already agreed out of band to encrypt to each other's gateway key, and the domain certificate only automates the handling of that key: it is fetched, rolled over and withdrawn through the directory instead of being exchanged by email at every renewal, which is a substantial improvement in itself. Where the policy is blanket, it is genuine discovery — but the blast radius of a single compromised key, and the fact that mail is decrypted at a perimeter, were accepted by the sender when the policy was set.

Section 7.5 discusses the realistic certificate options for this case.

5.7 Signed domain certificates

Requirement 2 of Section 5.6 rests the domain's authorisation on one fact: the file was served by `smimecert.<D>`. Whoever can write to that host — an insider with the web root, a CDN, the taker of a dangling CNAME (Section 7.9) — can therefore choose, among every certificate that validates and names `D`, which one strangers encrypt to (Section 6.6). This section closes that, by adding a second condition that the host cannot meet on its own, and Section 5.6 requirement 4 makes it the only way a fetched domain certificate is accepted at all, by any sender.

Publishing. A domain certificate is published signed or not at all. The publisher generates an Ed25519 key pair [RFC8032], publishes the public key as `k=` in its TXT record (Section 4.4), and publishes, beside `domain`, a file `domain.sig` containing the standard base64 [RFC4648], padded, of the 64-octet Ed25519 signature over

```
"SCD1 domain " || D || "\n" || DER(c1) || DER(c2) || ... || DER(cn)
```

and nothing else: one line, no other content. `D` is the canonical domain of Section 3.1, and `c1 ... cn` are the certificates parsed from the

CERTIFICATE blocks of the `domain` file, each as the encoded bytes decoded from its block — DER for a conforming certificate, and in any case the bytes as they were, never re-encoded by either side — as the *set* of distinct encodings, a repeated certificate counting once, concatenated in lexicographic order of those byte strings with each octet compared as an unsigned value, a shorter string that is a prefix of a longer one sorting first. A set, so that a relying party whose certificate store de-duplicates computes the same input as one that does not. A `domain` file MUST NOT hold more than 32 certificates, and a relying party's parse bound for it (Section 6.4) MUST be at least 32, so that both sides see every certificate; a domain bundle needs a leaf and a chain and comes nowhere near. The prefix separates this use of the key from any other, and `D` in the input stops a signature made for one tenant of a multi-tenant host from being replayed for another.

The signature covers the certificates, not the text of the file, so that line endings, PEM wrapping, whitespace and anything outside the blocks are irrelevant, and the sort makes the order irrelevant too, which Section 5.1 already tells relying parties it is. DER is canonical and each DER certificate is self-delimiting, so the concatenation cannot be misparsed and two different certificates never canonicalise alike (Section 8.4 says what this protects against). Whenever `domain` changes, `domain.sig` is regenerated in the same atomic swap (Section 7.8).

Ed25519 is the only algorithm, in its pure form: the signature is Ed25519 as defined in Section 5.1 of [RFC8032], computed over the input above directly — not Ed25519ph, which hashes the message first, and not Ed25519ctx, which takes a context string; the prefix in the input is what separates this use of the key from any other. The scheme has no padding and no parameters to choose. The signature is the 64-octet `R || S` encoding of Section 5.1.6 of that RFC and the public key the 32-octet encoding of Section 5.1.5, each carried in base64 as said above; a relying party verifies as in Section 5.1.7. A key is 32 octets and fits a single TXT character-string as 44 characters of base64, the same property DKIM relies on for its Ed25519 records [RFC8463]; RSA would have needed multi-string records and an algorithm field, and nothing here needs agility.

Rotation. A publisher lists one or two keys in `k=`; a relying party reads up to eight (Section 4.4.2). To rotate, the publisher adds the new key to the record, waits for the record's TTL to pass so that every resolver serves both keys, re-signs with the new key, waits for the RP cache lifetime of Section 5.5 to pass, and removes the old key. A relying party accepts a signature by any key listed in the record it holds. A verification failure is then treated as any

other lookup failure: not cached as a negative answer (Section 5.4), but remembered against `domain` and `domain.sig` in the failure memory of Section 5.5. A sender that already holds `D`'s domain certificate is unaffected by the failure either way: it keeps using what it holds (below).

Verifying. The `k=` field is also the announcement that a domain certificate exists: an RP whose record carries no readable `k` does not fetch `/domain` at all, and treats the domain as publishing no domain certificate — one it already holds stays in use regardless (Section 6.2). When the record does carry one, then for the domain certificate — and only for it; address certificates carry their own binding from the CA and are not signed — and for every sender that would use it, which is one whose policy under Section 5.6 requirement 4 permits a domain certificate for `D`; a sender whose policy does not has no reason to fetch the signature, and does not:

1. the RP MUST fetch `<base>/<D>/domain.sig` from the same host, under the rules of Section 4.5. A `200` is cached beside `domain`, with its own ETag and revalidated with it (Section 5.5). A `404`, or any other status failure, goes to the failure memory of Section 5.5 against `domain.sig` — a transport failure against the host, as always — and never to the negative cache: a signature missing beside a published `domain` is not “nothing is published” but a deployment error the publisher is probably about to fix, so the sender comes back within the hour and a `Cache-Control` on that `404` cannot stretch the wait, as it could if the `404` were a negative answer;
2. the RP parses `domain` into its certificate pool first (Section 5.1, bounded as in Section 6.4) and verifies over the canonical input above, so that the bytes of the file never matter; a `404` for `domain.sig`, a file that is not one line of standard base64 [RFC4648], padded, 88 characters decoding to 64 octets (a trailing CR LF or LF is ignored), or a signature that verifies under none of the record's keys is a lookup failure for the domain certificate: the file served as `domain` is not `D`'s domain certificate, whatever it contains, and the RP MUST NOT use it. Before declaring a signature failure the RP refetches `domain` and `domain.sig` together once, bypassing its own HTTP cache, so that the two files are never judged from different moments (Section 8.4). Under `domain-only` such a failure sends the RP to the per-address lookup exactly as a `404` does (Section 4.4.1), so that a spoofed record can at worst add requests;
3. a signature that verifies is what makes the certificate eligible under any policy at all (Section 5.6, requirement 4). Every other requirement of Section 5.6 — source, provenance, naming, path validation, validity, revocation, the sender's policy — applies unchanged. The signature is a

second condition on top of the host, not a replacement for it, and in particular it does not relax requirement 3: a signed certificate that does not validate to an anchor the RP configured is still not a domain certificate for $\mathcal{D}$. A self-signed or privately issued domain certificate therefore reaches only correspondents whose administrators imported the root or listed the domain, signed or not; a domain certificate meant for strangers is issued by a CA they already trust, in practice a public S/MIME CA, and since the certificate's kind plays no part (Section 7.5) a mailbox certificate for the gateway address is enough. The reason the CA stays in the path is given in Section 5.6 requirement 3 and Section 6.5, and bears restating here: the signature settles that $\mathcal{D}$ wants this key used, the CA keeps an independent party in the path — so that the one attacker this section does not stop, who holds the directory and a targeted DNS spoof, can only choose among $\mathcal{D}$'s real certificates rather than mint one — and the CA's revocation is what lets a leaked key be withdrawn (Section 7.6). Under DNSSEC a validated $k=$ could anchor a self-signed certificate the way DANE anchors a key; this version does not go there.

What it changes. Without the signature the attacker who matters holds write access to the directory and nothing else. With it that attacker can serve a file but not sign it. To succeed they now need the directory *and* either the signing key, which lives with the gateway's key material rather than on a web host, or a spoofed TXT record delivered to the specific sender they are attacking at the moment that sender fetches. No spoofed record alone suffices, because the file must still come from `smimecert.<D>` under its TLS certificate; so this does not reintroduce the DNS-supplied consent that Section 7.11 rejects. It is an AND where that would have been an OR. Under DNSSEC the key is itself authenticated, and a future version could let it stand alone; this version does not.

Accepted once, held until it expires. The signature is checked at the moment a domain certificate is accepted, under the record as it is then. From that moment the certificate is held like any other certificate the RP has — one imported from a signed message, one obtained from LDAP, a per-address certificate obtained through this protocol — and is used on the same terms as they are: path validation against the anchors configured at the time of use, expiry and revocation along the path, removal by an administrator (Section 6.2). The record is not consulted again on its account: a later spoof that *replaces* $k=$, after which a fresh fetch would verify under nothing, or *removes* it, after which the domain would look as if it published no domain certificate, cannot touch what the sender already holds. What such a spoof can still do is interfere with discovering a *successor* when the

held certificate nears its end, or with first contact; that is the exposure every held certificate has at renewal and every discovery mechanism has without DNSSEC. Section 6.2 says what a later lookup may and may not do to a held certificate, and Section 7.6 what withdrawal amounts to; a publisher that withdraws its domain certificate removes the key and the file together.

What it does not do. It does not prevent a host from serving an older domain with its older, still valid signature, which is a selection among the domain's historical keys and is answered by revocation, as Section 6.1 says for the address case; a timestamp in the signed input could narrow it in a later version. It does not stop an attacker who holds the directory *and* can spoof the TXT answer to one particular sender: that attacker substitutes a key of their own and signs any certificate that validates and names `D`, including any account holder's. Two capabilities are needed for that: write access to the directory, and a targeted DNS spoof timed to the moment the sender fetches, which DNSSEC removes; Section 4.4.1 already names this attacker, under `domain-only`, as the one who can choose the certificate strangers encrypt to. The one party that holds both without a spoof is whoever controls `D`'s DNS — a DNS hosting provider, a registrar account — since it points `smimecert.<D>` where it likes, obtains the host certificate by the DNS-change method (Section 4.3) and writes `k=`; this document counts that party as the domain itself (Section 5.6, requirement 2), DNSSEC does not change it since that party signs the zone, and continuity (Section 5.6, requirement 5) and revocation are the only answers to it, as for a domain whose own gateway is compromised. A certificate accepted that way is held like any other until it expires or is revoked, so the attacker needs the spoof once per sender rather than per message, and the remedy is the one for any wrongly accepted certificate: revocation, which the certificate is subject to because it had to validate to a CA (requirement 3). It does not protect address certificates, which need no protection of this kind. And a leaked signing key puts the publisher where a leaked DKIM key does, with one difference: rotating the record stops new acceptances under the old key at once, but a certificate already accepted under it stays in senders' stores until it expires or is revoked, so a publisher that suspects its key was used in anger revokes the certificates it finds published and reissues. A publisher that holds the key in the same place as its TLS and gateway keys has not widened its exposure.

6. Security considerations

6.1 What the binding actually proves

For an address lookup, discovery is not part of the trust decision.

The security of an S/MIME certificate does not depend on where it was obtained. It depends on whether a path can be built to a configured trust anchor, whether the certificate names the mailbox in question, and whether it is currently valid and unrevoked. Those checks (Section 5.3) are performed on the certificate itself and are unaffected by the channel that delivered it. A gateway already accepts certificates from a channel that offers no origin authentication at all — certificates attached to inbound messages — and it is the same checks that make that safe.

It follows that redirecting an SCD lookup gains an attacker nothing at the message-confidentiality layer. To have a substituted certificate accepted, the attacker must present one that names the target mailbox and chains to a trust anchor the RP configured. Under the CA/Browser Forum S/MIME Baseline Requirements that means demonstrating control of the mailbox, or of its domain, to a CA (Section 7.5); with an over-broad enterprise anchor in the store it means that anchor is already broken. An attacker in either position can simply attach the certificate to a signed message and let the gateway import it — cheaper, and requiring no network position at all.

The host is fixed at `smimecert.<D>` (Section 4.3), and it is worth being clear that for an address lookup that is not a security measure. It bounds where a gateway connects, and it is half of what a domain certificate needs, the other half being the signature of Section 5.7; it is not what makes an address certificate acceptable. Path validation already makes that decision, and a certificate that validates is acceptable however it was obtained. Provenance is load-bearing only for domain certificates (Section 5.6), and there only for the authorisation.

So what does an address lookup actually contribute? It contributes **availability and freshness**: a certificate that would otherwise require a manual round trip, kept current across renewals. The trust content comes from the CA and from the RP's own policy, exactly as it does for a certificate that arrived by mail.

The parties who can substitute a certificate for an address are therefore not the parties who control the directory, but:

- any CA in the RP's S/MIME trust list that mis-issues, or that is entitled to issue for the mailbox in the ordinary course (Section 6.5);
- the domain owner, who in most deployments can obtain a certificate

for any mailbox in the domain anyway, and who controls the mail itself;

- anyone exploiting a weakness in the RP's own validation — an unconstrained anchor, a skipped revocation check, a certificate accepted despite a mismatched SAN.

Controlling the directory response does still let an attacker *choose among* certificates that all pass validation — for example, steering the RP towards an older certificate whose key has leaked but which the CA has not revoked. This is a real but narrow gain, and an attacker holding that key can equally well sign a message with it (Section 7.1).

For a domain lookup one thing is added: no CA asserts the authorisation being relied on, so provenance and the publisher's signature supply it together (Sections 5.6 and 5.7). The trust decision itself is the same path validation to the same configured anchors.

What DNSSEC is and is not for here. Given the above, DNSSEC contributes little to the confidentiality of an address lookup: confidentiality rests on path validation. Nor does the domain certificate need it for acceptability: its authorisation rests on the TLS certificate of `smimecert.<D>` and on a signature under the key in the record together (Section 5.6, requirement 2; Section 5.7), and a spoofed record alone can satisfy neither. What DNSSEC protects against is *suppression as a denial-of-service* — an attacker who removes the TXT record, or removes or replaces the key in it, can stop discovery from working for a first contact or a successor, though not for anything a sender already holds, which Section 6.2 and Section 5.7 address without DNSSEC by holding what was accepted and logging what disappears. The publisher flag needs no protection, since manipulating it only adds requests, or omits a key the sender would have used beside the domain's, or — under the “instead” policy, at first contact — moves mail onto the domain's own key, all of which Section 4.4.1 accepts. That is a cleaner justification for “DNSSEC supported but not mandatory” than a confidentiality argument would be; what DNSSEC would add is the ability to let the record's key stand on its own (Section 5.7).

6.2 Downgrade and suppression

An attacker who can block DNS or HTTPS makes the domain look as if it does not support SCD. The sender then falls back to sending in the clear (or, in a gateway with a strict policy, does not send at all). Mitigations:

- **What is accepted is held.** A certificate obtained through this protocol, per-address or domain, has the checks that are particular to this

protocol — the name binding of Section 5.3, the source and signature of Sections 5.6 and 5.7 — made once, when it is accepted, and is then held like a certificate from any other source. Held means used on the same terms as every other certificate the RP has: path validation to the anchors configured *at the time of use*, expiry and revocation of every certificate on that path, and the administrator's hand. A root the administrator removes, an intermediate that is revoked, or a root whose name constraint excludes the name therefore stops a held certificate exactly as it stops any other (Section 6.5); only the discovery checks are not repeated. Discovery runs again at the cache interval of Section 5.5 to look for a successor, and a discovery that fails, or returns a 404, or returns a signature that does not verify, removes nothing; the held certificate is used until its own end. That is what Section 7.6 already says withdrawal amounts to, and it is the main defence against silent stripping: an attacker who blocks DNS or HTTPS, or spoofs the record, takes nothing away from a sender that already holds a certificate, and can only interfere with first contact and with finding a successor, as any discovery mechanism without DNSSEC can be interfered with. A 404 for a certificate the RP obtained through this protocol may be legitimate — the mailbox is gone, or the certificate was withdrawn — or it may be a compromised host serving a selective 404 (Section 6.6), and the RP cannot tell; it keeps the certificate and logs the disappearance once, as a change under Section 6.6, until a later acceptance for that address. A 404 for a certificate held from another source — one harvested from signed mail at a domain that never published it (Section 7.3), or for a user who has not opted in (Section 7.8) — is the ordinary state of affairs and is not logged; the store therefore records the origin of what it accepts. A publisher who needs senders to stop now revokes (Section 7.6).

- **What is accepted is stored with its path; nothing else is stored.** An acceptance stores the end-entity certificate together with the intermediates of the path it validated through, since a held certificate is validated again at each use and cannot be without its chain; never a self-signed certificate from the bundle, which is either an anchor the RP already has or nothing. An RP MUST NOT store any other certificate from a fetched bundle: not one that failed the checks of Section 5.3 or Section 5.6, and not a self-signed one. It logs the rejection with its reason and discards the material (Section 8.4 says why a fetched bundle, unlike an inbound message, is not worth keeping for diagnosis).
- **A local memory that holds yielded certificates.** An RP SHOULD remember that it has obtained and accepted at least one certificate

from `smimecert.<D>`, for a period of its own choosing (RECOMMENDED: 30 days from the last acceptance), clearable by an administrator. The memory changes no decision; its job is a log line. While it is live, a record that is gone — NXDOMAIN or NODATA, not a resolver failure — is logged once as a change under Section 6.6, so that stripping, a migration or a withdrawal is seen rather than silent; the lookup itself proceeds as “not supported”, and what the sender holds goes on being used. The memory takes no value from the record and is set only by an acceptance, so nothing in DNS can plant, lengthen or shorten it, and for the same reason the record carries no lifetime (Section 8.4).

- **The record as last read.** For the log lines of Section 6.6 an RP SHOULD keep, per `D`, the `flags` and `k` of the record as it last parsed them, for the lifetime of the memory above and cleared with it. It is compared with the next record read and nothing else: no decision takes a value from it, so a spoofed record plants only a log line.
- **Distinguish failure from absence.** A SERVFAIL or timeout is not evidence that the domain stopped supporting SCD, and MUST NOT populate the negative cache or be logged as a disappearance (Section 4.2.1). Only a definitive empty answer — NXDOMAIN or NODATA — is either. An answer with more than one `v=SCD1` record is neither failure nor absence: it announces support with its hints kept where the records agree and its keys merged (Section 4.2, step 3), and an attacker who can add a record to an answer gains no more by it than one who can replace the answer, which Section 5.7 accepts, and cannot deny anything by it.
- **Fail-closed policy.** For domains marked as “must encrypt”, a lookup failure must not silently downgrade to plaintext; the message should be deferred or bounced. This is a local policy decision with real operational consequences (Section 7.7). It applies when nothing is left to encrypt to: a failed per-address lookup for a domain whose domain certificate is available is not a discovery failure, and the message is encrypted to the domain certificate (Section 5.6, requirement 4).

What the sender does when it holds nothing and discovery fails — send in the clear, defer or bounce — is its local policy for a discovery failure (Section 7.7). Nothing in this section turns a failure into an error on the strength of the record, so there is no planted state for an attacker to exploit: the worst a spoofed record does to a sender that holds nothing is what the absence of SCD would have done, and to a sender that holds a certificate, nothing.

6.3 Privacy

- **The hash conceals nothing.** The local-part space is small and highly predictable; a rainbow table over common local-parts recovers most of them instantly, and because the domain is not part of the hash, one table serves all domains. This is a known property of both WKD and SMIMEA and is acknowledged here rather than papered over. Anyone who can read the directory's file names or web server logs learns the addresses.
- **Enumeration.** A publisher SHOULD disable directory indexes, serve a `robots.txt` [RFC9309] disallowing the well-known path, and rate-limit by source address. None of this prevents a determined dictionary attack, since a `GET` against a missing resource returns a small 404 and is nearly as cheap for the attacker as a metadata-only request would be. Publishers who consider the existence of a mailbox to be sensitive should weigh that against the benefit of publishing at all.
- **Lookup metadata.** The lookup tells the recipient's domain (and its CDN, and any on-path observer who can see the SNI/DNS query) that the querying organisation is about to send mail to that address, before the message is sent. Performing lookups from a gateway rather than from an MUA aggregates this to one organisational identity; caching and, where feasible, prefetching or padding reduce the timing correlation. DNS over TLS [RFC7858] or HTTPS [RFC8484], and Encrypted Client Hello [RFC9849], reduce the on-path exposure.
- **Request logs.** Because the request carries no query string (Section 4.1), a publisher's access log records `H` rather than the address itself. That is a weak mitigation — `H` is reversible for common local-parts, as above — but it avoids handing plaintext addresses to a delegated host or CDN that would not otherwise hold them.

6.4 Denial of service

- Every recipient at an SCD-supporting domain potentially triggers an HTTPS request. Without caching this is trivially amplifiable by an attacker who sends mail to many random addresses at a victim domain. Negative caching (for addresses as well as domains, Section 5.5) and the failure memory are the first line; they bound repeats, not firsts. An RP MUST therefore also bound the number of lookups one message may cause (RECOMMENDED: the message's recipient count, capped at a local maximum) and SHOULD allow one lookup in flight per domain, so that the time budget of Section 4.5, which is per recipient, is also bounded per message, and so that the first failure at a host is remembered before the next lookup there begins. A stalling host itself is then bounded by the failure memory, which remembers a transport

failure against the host name and the address (Section 5.5): a message to many fresh addresses there costs one timeout, not one per address. A hostile publisher of many names — a wildcard record answers for every sub-domain, each a `D` of its own with its own caches — is bounded by the per-message cap and by the address key: however many names, one stalling address is remembered once.

- The mandatory TXT record (Section 4.2) bounds this considerably: for the vast majority of domains, which publish no record, the cost per message is a DNS query answered from cache and no HTTP traffic at all. An attacker wanting to induce outbound HTTP from the gateway must first get a victim domain to publish a record, which it controls.
- Response size and time limits (Section 4.5) prevent a hostile or misconfigured directory from exhausting the RP. The size limit counts what comes out of the decoder, not what went over the wire, so a compressed body cannot exceed it by expanding, and only `gzip` is ever decoded.
- A publisher can use a relying party as a port scanner of the RP's own network. The attacker owns `D`, points `smimecert.<D>` at an address inside the victim's network, and provokes the victim into sending mail to a mailbox at `D`: a message to a non-existent user yields a bounce, a message to a real one yields an out-of-office reply or a read receipt, and any sign-up yields a notification. Each goes out through the victim's gateway, which looks up `D` and connects to the attacker's chosen address. The attacker times the arrival of the bounce or reply at its own MX: a refused connection fails in milliseconds, a filtered address burns the whole time budget, and a listening TLS service fails fast at certificate validation. One domain per probed address costs the attacker nothing but DNS records under its own domain, so the victim's negative caches never apply, and the victim's failure policy (Section 7.7) is a second channel. The address restriction of Section 4.5 closes the scan of the RP's own network; nothing else in the protocol does, since TLS only prevents the request, not the connection. Connections to global addresses the publisher names remain, as they must, bounded by the time budget and the per-host failure memory.
- A bundle can contain a large number of certificates and a hostile bundle can be crafted to make naive path building blow up combinatorially (many cross-signed intermediates). Implementations MUST bound the number of certificates parsed (RECOMMENDED: 32, and for `/domain` at least 32, since the signature covers every certificate in the file; Section 5.7) and the path-building effort. A bundle over the bound is a lookup failure remembered against the resource (Section 5.5), never a truncated bundle.

- Parsing untrusted DER from an unauthenticated-at-the-application-layer source is an attack surface in its own right; keep the ASN.1 parser current and resource-bounded.
- The certificate store is a second amplification target. A publisher can make a sender fetch at will (above) and chooses what the bundle holds, so a sender that stored what it fetched regardless of validation would gain up to 32 rows per lookup from a hostile domain, forever. Section 6.2 closes this: nothing enters the store but an accepted certificate and the path it validated through, and what is accepted had to validate to one of the sender's anchors and name the address, which costs the publisher a real certificate per row.

6.5 Trust store, CA quality, and private CAs

Because the trust decision for an address lookup rests entirely on path validation (Section 6.1), the quality of the RP's trust configuration is the whole ballgame. Two failure modes are common in deployments today:

- Using the browser/TLS trust store for S/MIME. Many roots in a TLS trust store were never audited for email issuance. An RP **SHOULD** maintain a separate S/MIME trust list, and **SHOULD** apply EKU-constrained anchors where the trust store format supports it.
- Trusting a root for more than it should issue. Trust is inferred from the chain: a certificate that chains to a configured anchor and is valid for email encryption is accepted, whatever domain it names, and this document asks for no per-domain list of acceptable anchors, which are cumbersome to manage. What bounds a root is the root itself: a separate S/MIME trust list (above) and, for a private CA, a name constraint confining it to the publisher's domain (Section 7.5), which the validator **MUST** apply (below).

Private CAs. This document has no relaxed mode: a certificate that does not validate to an anchor the RP configured is not used, whatever directory it came from and whether it names an address or a domain. There is no trust on first use, and nothing is pinned: a certificate that was accepted is held until it expires or is revoked (Section 6.2), which makes nothing acceptable that was not acceptable when it was checked. A publisher whose certificates come from a private CA makes them usable by getting its root into the relying party's trust store, which is an administrator's decision on the relying side (Section 7.5). The bundle **SHOULD** carry the root (Section 5.1) so that the administrator can take it from there rather than ask the correspondent for it. An RP **MUST NOT** trust a root because it arrived in a

bundle, and MUST NOT store one from a routine lookup either (Section 6.2): the import is an administrator's act on a bundle fetched for the purpose, and a root left lying in the store as untrusted material is a root waiting for a careless import.

An RP which lets an administrator import a root found in a bundle MUST apply that root's name constraints when it validates against it. The publisher constrains the root to its own domain precisely so that importing it trusts that CA for that domain's mailboxes and, in an S/MIME trust list, for nothing else (Section 7.5), and [RFC5280] leaves a trust anchor's extensions to the implementation — Section 6.2 of the RFC says it is “free to process or ignore” them, naming name constraints as an example — and most libraries ignore them unless told otherwise, so a validator which forgets this leaves the constraint decorative and the import unbounded. Once the root is in the store, a certificate under it is validated in the full sense, and may be reported to operators as such: the trust came from their own decision, not from the fetch.

6.6 Compromise of the directory is silent

Unlike a signed message, a directory response leaves no evidence for the recipient in the lookup itself. A compromised directory can serve one certificate to one sender and a different one to everyone else, and neither the mailbox owner nor the other senders will notice.

The message itself is the one trace. CMS names every certificate the content key was encrypted to (Section 7.4), so a receiving gateway sees which certificate each sender used: a certificate whose key it does not hold fails to decrypt, which is loud, and one it does hold but did not expect — an older certificate, or a mailbox certificate used as the domain certificate — shows in the `RecipientInfo` identifiers, which a gateway that logs them can compare with what it published. Detection on that side is after the fact and depends on someone looking, but it exists, and it is the receiving domain's own check on its directory.

For address lookups the practical impact is bounded by Section 5.3: the substituted certificate must still name the mailbox and chain to a configured anchor, so this is a *selection* attack among valid certificates rather than a free substitution. For domain certificates (Section 5.6) it is bounded the same way — a substitute must validate to a configured anchor and name `D` — with one addition: because publication under `/domain` is what makes a certificate the domain's, a compromised host can present any of `D`'s mailbox certificates that pass validation as the domain certificate,

which is still a selection among valid certificates but a wider one. The signature of Section 5.7 removes it for every sender as far as a host acting alone is concerned: the host can serve a file but cannot sign one, and there is no unsigned `/domain`, so the choice belongs to whoever holds the signing key, or to an attacker who holds the host and can also spoof the sender's DNS. Requirement 1 of Section 5.6 moved the insider's problem from "send a signed message" to "control the publication host"; the signature moves it again, from the host to the key, which lives with the gateway's key material. The first item below describes that mechanism; the two after it are what remain against the key-holder and the two-capability attacker:

- **Publisher-signed domain certificates.** Section 5.7 specifies a detached signature over the certificates in the `domain` file made by a key whose public part is in the TXT record (DKIM-style [RFC6376]), verified by the RP in addition to TLS. It binds the file to a key that a CDN, web host or CNAME taker does not hold, and is the one countermeasure here that closes the wider selection rather than narrowing it; every domain certificate fetched through this protocol carries one (Section 5.6, requirement 4). Address bundles are not signed, since the CA already binds them; the resource layout leaves room for `<base>/<D>/hu/<H>.sig` should a later version want it.
- **Gossip / transparency.** Logging observed (address, certificate hash, first-seen) tuples and comparing across gateways would detect targeted substitution. This is an ecosystem-scale idea, out of scope here, but the local half — remembering what was seen before and alerting on unexpected change — is cheap and worth implementing.
- **Change monitoring.** A gateway SHOULD log every change to the certificate of a known correspondent, and SHOULD alert on the changes that a renewal does not explain — a different issuer, a replacement well before the normal renewal window — and SHOULD log, not alert on, a disappearance (Section 6.2), which is as often a withdrawal or a migration as an attack. Alerting on every new key is counterproductive for the reason Section 5.6 requirement 5 gives — a renewal rotates the key, and an operator who learns that the alert means "renewal" dismisses the one that does not. A gateway SHOULD also log the appearance of `domain-only` on a record that lacked it when the record was last read, or on a domain from which it has obtained per-address certificates through this protocol, whenever a lookup reads the record: the comparison is free, since the record as last read (Section 6.2) and the store are in hand at that moment anyway, and it fires exactly when a spoofed flag (Section 4.4.1) or a genuine

migration to gateway-only would begin. It must not fire merely because the gateway holds per-address certificates for the domain, since those commonly arrived in signed mail from a domain that never published them (Section 7.3).

6.7 Cryptographic notes

- SHA-256 here is a naming function, not a signature; collision resistance is not load-bearing. A collision would merely cause two local-parts to share a file, which the SAN binding check in Section 5.3 catches.
- The bundle contains only public data; confidentiality of the response is not required, but TLS is mandatory anyway because it authenticates the origin.
- S/MIME message-layer weaknesses (EFAIL-style exfiltration via CBC malleability and HTML rendering [EFAIL]) are unaffected by this document, but a mechanism that increases how much mail gets encrypted also increases exposure to them. Deployments SHOULD prefer authenticated encryption (AES-GCM in `AuthEnvelopedData` [RFC5083], with `id-aes128-GCM` or `id-aes256-GCM` [RFC5084]) where the recipient supports it, and SHOULD NOT weaken existing EFAIL mitigations because certificates are now easier to obtain.

7. Other things to consider

7.1 Combining certificate sources

A gateway will typically have several sources: administratively configured certificates, certificates harvested from received mail, LDAP, SMIMEA, and SCD. This document specifies no precedence among them. Whether an RP ranks its sources, and how, is local policy.

What this document does constrain is narrow and unchanged: every candidate must pass Section 5.3 whatever its source, and a domain certificate is additionally subject to Section 5.6 — in particular requirement 1, which permits only the domain's own directory or an administrator as its source, because there the provenance carries an assertion no CA makes. Beyond that, a certificate that validates is a certificate that validates, and SCD is simply one more way to obtain one.

Note that the question often does not arise. `CMS EnvelopedData` carries a

SET OF RecipientInfo , so an implementation may encrypt the content-encryption key under several certificates for the same recipient — a per-address certificate and a domain certificate, or an RSA and an ECC certificate — and let the receiving side use whichever private key it holds. Where that is acceptable, there is nothing to rank.

For the per-address and domain certificates of one recipient this gives two policies, and the document uses “substitute” with the distinction in mind:

- **Both.** The domain certificate is added beside the per-address certificate whenever the sender’s policy permits domain certificates for `D` and both validate. This is the RECOMMENDED default for a gateway, for this choice alone: whether domain certificates are permitted at all is the separate policy of Section 5.6 requirement 4, on which this document recommends nothing. It is what publishing `/domain` means — the domain said its perimeter may decrypt for any of its mailboxes — and it is robust: a stale or lost mailbox key at the recipient no longer yields an undecryptable message. A message to several recipients at `D` is encrypted to the domain key whenever any of them lacks a per-address certificate, so a per-recipient promise of end-to-end was never available for a shared message in any case. The consequence to state plainly: under this policy the perimeter can always read, and the mailbox key’s inclusion serves the recipient’s convenience, not confidentiality against the perimeter.
- **Instead.** The per-address certificate is used alone when held, and the domain certificate only when there is none. This preserves end-to-end for a recipient whose own client holds the mailbox key, at a domain that publishes a gateway key as well, at the cost of robustness. It is local policy, and a sender that wants it configures it.

What neither policy may do is drop a held, valid per-address certificate in favour of the domain certificate. That is what “substitute” means wherever this document forbids it (Sections 4.4.1 and 5.6), and it is the only thing it forbids: adding the domain certificate is permitted, using it alone where no per-address certificate is held is permitted, using it in place of one that is held is not.

Two observations are worth recording for implementers who do rank sources, or who weigh how much a given source is worth:

- **Import and eligibility for encryption are separate decisions.** A gateway commonly imports a certificate from an inbound message even when the signature does not verify, deliberately, so that an administrator can diagnose why validation failed. That is the right

behaviour for import, but such a certificate has had nothing attest to it, and an implementation may reasonably keep it for diagnosis without making it eligible for encryption. The same reasoning gives the opposite answer for SCD: a bundle that failed validation is not kept at all (Section 6.2), because the evidence is not unrepeatable — the administrator can fetch the directory again at any moment — and because the publisher, unlike a correspondent sending one message, can make the sender fetch as often as it likes.

- **A verified signature attests less than it appears to.** In a dual-key deployment the signature proves possession of the *signing* key, while the encryption certificate rides along in the `SignedData` `certificates` field with nothing attesting to it. The certificate actually used for encryption is the one proof of possession does not cover. Influencing an SCD response, by contrast, requires answering as `smimecert.<D>` under a certificate for that name, which means controlling the target's DNS or web infrastructure — an active attack with real prerequisites, needed once per sender since what is accepted is held (Section 6.2), and leaving traces in DNS, in TLS logs and, for a CA that logs, in Certificate Transparency [RFC6962].

Where two sources yield different certificates for the same address, that is worth logging whatever the policy decides to do about it.

7.2 Rollover

A publisher SHOULD publish the successor certificate alongside the current one some time before expiry (an overlap of at least the maximum RP cache lifetime, so a week or more), and SHOULD keep an expired certificate in the bundle only if there is a reason to (there normally is not — senders never need it, and it only enlarges the response).

Under a private CA whose root the receiving administrators have imported (Section 7.5), the same overlap makes leaf rotation invisible: the successor chains to the same root. The root itself is not rotated; when its certificate needs renewing it is reissued under the same key. A gateway which generates a new key pair at every renewal is unaffected, because the new leaf chains to the root like any other.

7.3 Aliases, lists and shared mailboxes

Publishing a certificate for a mailing-list address, a distribution list, or a shared mailbox means every sender encrypts to a key held by whoever operates that list — often a key that the list expander must use to decrypt

and re-encrypt. This is a legitimate deployment (it is exactly what a gateway does), but it is not end-to-end, and users may believe otherwise. Publishers should be deliberate about it; RPs cannot detect it.

Aliases that forward to another domain are a related trap:

`sales@a.example` forwarding to `bob@b.example` should publish the certificate that the receiving system can actually decrypt with.

Publishing a domain certificate and signing with mailbox certificates are not alternatives. A domain that keeps its address book out of a public directory (Section 6.3) publishes only `/domain`, under `domain-only`, while its users go on signing their mail with their own certificates, which correspondents harvest as they always have. A correspondent then holds both kinds for that domain from the first exchange, uses the mailbox certificate on the terms of Section 7.1, and has seen nothing unusual: the first domain certificate is logged once, as for any domain (Section 5.6, requirement 5), and the logging rules of Sections 4.4.1 and 6.6 are written so that this combination alerts on nothing.

7.4 Address canonicalisation edge cases

- **Local-part case.** [RFC5321] makes the local-part case-sensitive in principle, and lower-casing it is a pragmatic assumption (also made by WKD). Publishers whose users have mixed-case addresses SHOULD publish under the lower-cased form; RPs that need exactness can additionally require the SAN to match with the original case.
- **Sub-addressing.** `alice+news@example.org` and `alice@example.org` are usually the same mailbox, but the detail part is never stripped, and this specification provides no mechanism to opt into stripping it. The reason is not that stripping would be unsafe in principle — it is that the rest of the S/MIME ecosystem does not do it. `rfc822Name` comparison under [RFC5280] is a match on the mailbox as written, and deployed clients and gateways match it as written, apart from case: where an address and a certificate disagree, the available workaround is to disable the name check altogether (Outlook's `SupressNameChecks` [MS276597], and the corresponding long-standing request against Thunderbird), not to normalise the address. A discovery mechanism that returned a certificate for `alice` in answer to a query for `alice+news` would hand the relying party something its own matching rule rejects.

Note that this is a sender-side constraint only. CMS identifies recipients by `issuerAndSerialNumber` or `subjectKeyIdentifier`, so a message

addressed to `alice+news@example.org` but encrypted to `alice@example.org`'s certificate decrypts without complaint. A gateway that chooses to treat tagged addresses as equivalent to their base address can therefore do so as local policy, and the protocol does not need to help: it simply canonicalises the address before Section 3.1 and looks up the base address. What this specification declines to do is put that behaviour on the wire, where a publisher's flag would be asking relying parties to relax a rule most of them enforce in code.

- **Internationalised domains.** Use A-labels in the DNS name, in the URL host and in the `<D>` path segment, and expect them in the `rfc822Name` of the certificate. This needs nothing beyond [RFC5891] and works with every deployed client and CA, since the certificate name form is unchanged.
- **Non-ASCII local-parts** are out of scope of this version. `rfc822Name` is an `IA5String` and cannot carry one; the `Smtputf8Mailbox` name form that can [RFC9598] is permitted by the S/MIME Baseline Requirements [SBR] but, at the time of writing, is issued by no public CA, matched by no mainstream client, and depends on SMTPUTF8 [RFC6531] transport that most mail infrastructure does not offer. An RP MUST treat such an address as having no per-address SCD certificate, without performing a per-address lookup; a domain certificate names `D`, not the mailbox (Section 5.6, requirement 3), and applies to such an address as to any other at `D`. A later version can add the name form, its comparison rules and the required Unicode normalisation of the local-part once there is something to interoperate with.
- **Address forms with no `D`.** Split at the last `@`, and reject addresses that are not syntactically valid under [RFC5322] and [RFC5321] before hashing; an invalid address gets no lookup, and an RP does not repair one. One valid form still yields no `D`: a domain literal, `user@[192.0.2.1]`, which names a host and not a domain, and for which no lookup is performed (Section 3.1).

7.5 Domain-to-domain certificates in practice

Section 5.6 establishes that no CA asserts domain authorisation. That leaves two realistic options for what a publisher actually puts in the `domain` file:

A. A mailbox certificate whose domain-part equals `D`. Conformant, obtainable from any S/MIME CA today, and the CA contributes a genuine key binding. The RP checks the domain-part and ignores the local-part

(Section 5.6, requirement 3).

Within this option the kind of certificate does not matter, and it is worth saying why, because it looks as if it should. The S/MIME Baseline Requirements [SBR] let a CA validate a mailbox by proving control of the whole domain (Section 3.2.2.1 of [SBR], which delegates to the domain-control methods of the TLS Baseline Requirements, Section 3.2.2.4 of [TLSBR]), rather than by a challenge to the one mailbox, and a certificate issued that way is closer to the missing assertion than one issued to whoever could read the mailbox. But the method leaves no mark in the certificate, and the policy OID that a certificate does carry names its *type* — mailbox-, organisation-, sponsor- or individual-validated — not how the mailbox was validated. A relying party cannot tell the two apart, and a mailbox-validated certificate is what any account holder at a freemail provider or a university can get in minutes. So the certificate's kind is not what a stranger relies on. What a stranger relies on is the publisher's signature over the certificates in the file (Section 5.7), which any certificate can carry, and which the publisher makes with a key the CA never saw. A publisher under this option therefore obtains whatever mailbox certificate it can for its gateway address, publishes it, and signs it.

B. A certificate from the publisher's own CA. What most existing domain-to-domain deployments already have, exchanged by hand today. The publisher SHOULD create a root of its own, constrained by `nameConstraints` to a single permitted subtree, `rfc822Name D`, and nothing else, give it a long validity and a CRL, issue the domain certificate under it as an ordinary mailbox certificate (Section 5.6, requirement 3), and publish both (Section 5.1). Under [RFC5280] that one subtree permits every `rfc822Name` at exactly `D` and no other `rfc822Name`; a constraint binds only the name form it names (Section 4.2.1.10 of the RFC), so the root is bounded for S/MIME because this document matches `rfc822Name` alone (Section 5.3.1), and it belongs in the separate S/MIME trust list of Section 6.5, not in one shared with TLS. A publisher whose root should also issue for mailboxes in sub-domains adds a second subtree, `rfc822Name .D`: the leading-dot form covers sub-domains and the bare form does not, and neither implies the other.

Nothing in this document makes that root trusted. The relying party's administrator decides once how to accept `D`'s certificate, in one of two ways. The first is to assign the domain certificate to `D` by hand, which trusts that one certificate and nothing more; it is today's practice, except that the administrator fetches the certificate from `D`'s directory instead of receiving it by email. The second is to import the root. Because of the name

constraint, an imported root is trusted for $\mathcal{D}$'s mailboxes and, as an S/MIME anchor, for nothing else (Section 6.5). The constraint exists to protect the importing administrator: it bounds what the import can do, and it plays no part in whether the certificate is eligible.

Importing the root makes the certificate validate, which puts the private CA where a public CA is; from then on nothing distinguishes the two. Whether the sender may use a domain certificate for $\mathcal{D}$ at all is the separate policy question of Section 5.6 requirement 4, answered the same way for both: a blanket default permits it as soon as the certificate validates, and a list or an "off" default needs a per-domain entry for $\mathcal{D}$. An administrator who imports a root for $\mathcal{D}$ under such a default will usually add that entry in the same step, since both are decisions about the same correspondent, and a console can offer them together. What signing does not do is carry a private-root certificate past a sender that has not imported the root. The signature settles authorisation, not trust, and requirement 3 of Section 5.6 still wants a path to one of the sender's anchors. A self-signed certificate is the degenerate form of the same decision: it is assigned to $\mathcal{D}$ directly and reaches nobody else, signed or not.

A publisher under this option that also wants to reach strangers obtains, from a publicly trusted S/MIME CA, a mailbox certificate for its gateway address and publishes that as its domain certificate instead, as under option A. A correspondent that imported the private root trusts public CAs as well, so a second domain certificate from the private CA beside it would serve nobody. Where the private CA earns its place is the mailbox certificates under `/hu/`: a correspondent that imported the root uses them, and a stranger, whose per-address lookup finds nothing it can validate (Section 5.5), falls through to the public-CA domain certificate. Leaf rotation and leaf revocation under the private CA are ordinary, and a new root is a new decision, as with any CA.

B is not a weaker option than A in the dimension that matters here, since neither carries a CA's assertion of domain authority. The two differ in who makes the first decision. Under A the sender's existing trust store makes it, so a blanket policy over strangers works with no human involved. Under B an administrator makes it per correspondent, once. B is therefore the option for known correspondents and A for everyone else. The separate root is recommended over a self-signed certificate because a self-signed certificate is a root that is its own leaf: it cannot change its key without a new decision at every sender.

Further points regardless of option:

- The domain certificate MUST NOT apply to sub-domains. Each sub-domain publishes for itself.
- Once a root is in the store the RP cannot tell a shipped anchor from an imported one, and should not try: continuity is the same rule for both. Renewal from the same issuer is unremarkable, whoever the issuer is, and alerting on it teaches the operator to dismiss the alert; alert on a change of issuer or an early replacement instead (Section 5.6, requirement 5). That signal is not specific to public CAs — a certificate from a different issuer validates under B as well if that issuer is in the store, which is what the rule is there to notice.
- A held per-address certificate MUST be used, alone or beside the domain certificate according to the policy of Section 7.1 (Section 5.6, requirement 4), and the choice SHOULD be visible in the logs. That includes a per-address certificate the RP already holds when `domain-only` says not to look for one (Section 4.4.1).
- Both ends must consent: the publisher by publishing `/domain`, and the sender by a configured policy permitting it (Section 5.6, requirement 4). The sender's policy may be blanket rather than per-domain; what matters is that the sender set it.
- A domain that will never publish per-address certificates SHOULD publish `domain-only` (Section 4.4.1). Senders reach the same certificate without it, but every distinct recipient then pays for a per-address 404 first, and those negatives do not aggregate the way a per-domain result does.

7.6 Revocation and withdrawal

Removing a file from the directory is not revocation: senders that hold the certificate keep using it until it expires (Section 6.2), and an attacker can suppress the removal anyway. Real revocation requires CRL/OCSP.

Unpublishing has exactly one effect — no sender obtains the certificate from now on — and none on senders that already have it, at any cache lifetime; a publisher who needs a certificate to stop being used must revoke it properly, not merely unpublish it.

Under a publisher's own CA (Section 7.5, option B) this is no different: a compromised leaf is revoked through the CA's CRL and reissued, and every relying party that imported the root follows. A compromised root is what it is for any CA — the publisher creates a new one and every relying party's administrator decides again — which is why a publisher choosing that option should keep a list of its correspondents. The signing key of Section 5.7 is no different in kind: rotating it out of the record stops new

acceptances under it at once, but a certificate a sender already accepted under it is held until it expires or is revoked, so a publisher whose key was used in anger revokes what was published under it.

7.7 Failure behaviour

The most consequential operational decision is what happens when discovery fails for a domain that is normally reachable and nothing is left to encrypt to — a failed per-address lookup falls through to the domain certificate where there is one (Section 5.6, requirement 4), and this policy decides only what happens after that: send in the clear, defer, or bounce. A gateway SHOULD offer the first and the last per domain, and MAY offer deferral, which is more than a queue: a deferred message needs a retry schedule, a maximum age after which it is bounced after all, and a rule for when the sender is told, and a gateway that does not want those decisions bounces at once, which tells the sender immediately and leaves the decision with them. A gateway should default to the conservative option for domains under an explicit encryption policy, and should use the memory of Section 6.2 to tell “never supported SCD” from “supported it last week” in its logs. The policy is reached less often than it looks: a sender that holds a certificate from `D` keeps using it whatever discovery says, so a failure matters only at first contact and when a held certificate has run out without a successor being found.

7.8 Publishing side

For a mail gateway, publishing is nearly free: it already holds certificates for its own users. A conforming directory can be generated by exporting each user’s current certificate plus chain into `<D>/hu/<H>` and syncing that tree to a web root. Worth building in:

- generation as an atomic directory swap, so a partially written tree is never served;
- an opt-in per user, since publication is a disclosure;
- automatic regeneration on certificate import, renewal and revocation;
- a signing key for the domain certificate (Section 5.7): an Ed25519 key pair generated once and kept with the gateway’s other key material, its public half in the TXT record, and `domain.sig` rewritten in the same swap as `domain`. There is no `/domain` without it, and the gateway already does the harder part, since it holds keys and regenerates the tree on every change. A publisher that cannot sign does not publish a domain certificate through SCD, and hands it to its correspondents’

administrators as today;

- for a domain certificate not issued by a public CA, a root of the gateway's own with a validity of decades, a published CRL and an `rfc822Name` name constraint confining it to the domain, the domain certificate issued under it as an ordinary mailbox leaf, and the root exported into the bundle beside the leaf (Section 7.5). A built-in CA which issues internal certificates under a root does most of this already; what it needs beyond that is the name constraint at root creation, and — on the relying side — applying an imported root's own constraints, which a validator built on the platform PKIX classes does only when told to (Section 6.5). Both are ordinary features of a gateway with a built-in CA;
- a self-check that fetches its own directory from the outside, and compares what is served with what was generated, not only that something is served;
- keeping and using the signing key as the gateway's own private keys are kept and used. Whoever holds it chooses which of the organisation's certificates every stranger encrypts first-contact mail to (Section 5.6, requirement 1; Section 5.7; Section 6.6). Signing is the one privileged operation this protocol creates on the publishing side, and it is the step that an insider with an ordinary staff certificate, or with the web root, cannot take alone;
- controlling and logging write access to the published tree, and the credentials of the hosting account behind it, as for any published artefact. The host can no longer substitute a key — nothing it writes carries a signature it can make — but it can still *deny*, by serving 404 for every per-address path and leading senders to a /domain it cannot sign, and *replay*, by serving an older file with its older, still valid signature (Section 5.7). That is an ordinary operational control, not the key-store one, and it should be limited to the publication process rather than shared with the general web-content workflow of a site that happens to live on the same host.

Acquisition. In practice a gateway obtains certificates from a managed PKI over the CA's own API, usually REST. The gateway generates the key pair locally, builds a CSR, submits it, and receives the signed certificate and its chain; renewal is the same call on a schedule. Domain validation is typically done once, at account level, so issuance for an individual mailbox needs no per-mailbox challenge — which is also what makes the domain-authority validation described in Section 7.5, option A, the normal case rather than an unusual one for organisations using a managed PKI.

This matters for publication because it is an event the gateway already observes. Issuance, renewal and revocation all pass through the same code path, so regenerating the affected directory entry is a hook on an existing flow rather than new machinery.

ACME's email challenge [RFC8823] standardises an alternative — identifier type `email`, challenge type `email-reply-00`, in which the CA sends a signed challenge message carrying half a token and the client replies by mail. A gateway is well placed to answer it, being already in the mail path. But it is Informational, availability from CAs whose anchors senders already trust is limited (Let's Encrypt does not issue S/MIME certificates at all), and the established Java ACME library describes its support as experimental. It is worth tracking rather than building on today.

Who holds the key. A gateway that generates the key pair for a mailbox certificate holds its private key, and decryption then happens at the perimeter, not by the addressee. That is one deployment, not a consequence of publishing: a gateway can equally publish certificates whose private keys exist only with the mailbox holder — issued to the user and handed over, or collected from the user's own — and then the mail is end-to-end. The published certificate looks the same either way. Where the gateway holds the key, a per-address certificate is to a sending RP indistinguishable from a domain certificate — same perimeter decryption, same blast radius if the key is compromised — except that none of Section 5.6 applies: no consent step, no sender-side policy. This is not a defect; key custody is the receiving domain's business. What a sender may not do is infer from Section 5.6 that a per-address certificate implies delivery to the person named: it may, and it may not. A domain certificate, by contrast, always means a key at the perimeter. As with shared mailboxes (Section 7.3), publishers should be deliberate about this, and relying parties cannot detect it.

7.9 Hosting realities

- Static hosting is a feature: object storage, a static site host or a plain `nginx location block` is sufficient, and there is no dynamic attack surface.
- Delegation is a CNAME on `smimecert.<D>` plus a certificate for that name, obtained by the provider (Section 4.3). A provider therefore holds one certificate per customer domain, as any static host with custom domains already does; it cannot serve every customer from a single name of its own.

- A name that keeps resolving after the publisher has moved on is a takeover. A `smimecert.<D>` CNAME left pointing at a provider the domain no longer uses, or an address record pointing at a reallocated address, lets whoever claims the far end answer for the name, obtain a certificate for it, and serve files under `D`'s directory (Section 5.6, requirement 2). For address lookups that gains nothing. For `/domain` the taker can serve a file but cannot sign one (Section 5.7), so what it gains is denial — a per-address 404 or error that leads senders to a `/domain` they cannot verify, which under `domain-only` sends them back to the per-address lookup and only then to their failure policy — and replay of older files with their older signatures; not a key of its own choosing, and nothing against a sender that already holds `D`'s certificates (Section 6.2). A publisher leaving a provider MUST remove the CNAME, and a publisher leaving SCD MUST remove the TXT record, which is the switch: with no record no RP connects, a sender that remembers `D` logs the disappearance, and what senders hold is used until it expires (Section 6.2). A certificate issued for `smimecert.<D>` by a CA that logs to Certificate Transparency [RFC6962] — in practice every CA in the browser root programs, though no Baseline Requirement demands it and Section 4.5 asks a relying party for no SCT — is visible to a publisher that monitors its domain there; the self-check of Section 7.8 is the detection that depends on no CA.
- A CDN in front of the directory sees every lookup and holds the TLS key. This is the same trust concession the whole Web PKI ecosystem makes, but it should be a conscious one for a certificate directory. The signed domain certificate of Section 5.7 exists mainly to address it: a CDN that holds the TLS key still cannot sign.
- Case-insensitive filesystems are harmless because the hash encoding is lower-case hex only.
- Content-Type: serve `application/pem-certificate-chain`. RPs should accept anything and parse defensively, since misconfigured `text/html` error pages are the most common real-world failure.
- Content-Encoding: serve the file compressed or not, as the platform prefers; nothing is gained either way on a file this size. If the platform stores objects pre-compressed and serves them encoded whether or not the client asked, `gzip` is the one coding a relying party will decode (Section 4.5), so use that one.
- Cache-Control on a 404 : several static hosts cannot set headers on error responses. That is harmless — a relying party then caches the negative answer for its default hour (Section 5.5) — but a publisher that wants a newly published certificate found later than that has to be on a

platform that lets it set `max-age` on the 404 .

- **Ports:** a redirect to another port on `smimecert.<D>` is permitted (Section 4.5) and some relying parties will not be able to follow it, because their operators firewall outbound traffic to 443 alone, as Section 4.3 invites them to. Serve the directory on 443.

7.10 Interoperability and adoption

Adoption is the hard part, not the protocol. Deployment is one TXT record, one address record and a tree of PEM files that anyone can produce with `openssl` — plus, for a domain certificate, one key pair and one signature, which `openssl` produces as well — comparable to the effort of deploying MTA-STS, and less than deploying DKIM [RFC6376]. A public test suite, a `dig + curl` recipe in the documentation, and a checker page (as MTA-STS deployments have) do more for adoption than any additional protocol feature.

7.11 What was deliberately left out

- **Certificate submission/update protocol.** WKD specifies a mail-based enrolment protocol. For S/MIME the certificate is issued by a CA, and the publisher is normally the same organisation as the CA's subscriber, so enrolment can be handled by existing processes. Adding a submission protocol would roughly double the specification for little benefit.
- **Search / wildcard queries.** Deliberately absent; they turn a static file tree into a service and an address-harvesting oracle.
- **Non-HTTPS transports.** No benefit that outweighs a second code path.
- **A separate policy file.** WKD serves one, because there such a file is the only way a domain can signal that it participates at all. With a mandatory TXT record (Section 4.2) the file has no job left: its keywords fit in `flags=` , and `domain-only` is only useful if it is known *before* the first lookup, so a file would put a second HTTP round trip on the critical path of every lookup — where the domain certificate's second fetch, its signature, is paid once per domain and cached as one entry. A file would also have been the natural home for a publisher-controlled lifetime, as MTA-STS's `max_age` is; this document has no such field at all (Section 4.4), since what a sender holds lives by its own validity and what it remembers is its own affair (Section 6.2). The cost is that a publisher must change DNS to change policy — no burden for a mail team that already publishes the mandatory record — and that there is

no room for a keyword too large for a TXT record. A future version can introduce a file if it ever needs one; anything that large would need a specification of its own regardless.

- **A consent flag for the domain certificate.** A domain-fallback flag by which a domain consented to its domain certificate being used for its individual mailboxes would put consent in the one place it cannot be authenticated. Publishing `/domain` under `smimecert.<D>` is an act only the domain can perform; a TXT record can be spoofed, so a flag there could be added or stripped by anyone who could spoof the answer, and either way it would have steered senders away from the key the domain had actually published (Section 4.4.1). It would also have saved nothing: the `/domain` fetch it gated happens once per domain and caches as one entry. The domain-only flag stays, as a hint about cost with no authorisation in it. The signing key of Section 5.7 lives in the same record and is not the same thing: it grants nothing on its own, since the file must still come from the domain's host and verify, and its loss or replacement touches nothing a sender already holds. A consent flag could have been neither.
- **Trust on first use for private keys.** A relying party could have been allowed to accept a self-signed or privately issued domain certificate by pinning whatever it saw first, which needs rules for what the pin covers, when it moves, what an expiry means and how a compromised key is replaced. All of that automates a trust decision about a key that no configured anchor vouches for, and it would have been the bulk of the document's difficulty. Instead a certificate either validates to an anchor the relying party configured or it is not used. A publisher with its own CA gets it trusted the way any CA is, by an administrator on the other side importing a root constrained to the publisher's domain (Section 7.5), after which everything is ordinary PKIX. The directory distributes; it never decides.
- **A URL in the TXT record.** MTA-STS derives its policy host from the domain, and WKD removed the DNS-supplied host it once had; this document does the same. A `url=` field would have let a provider serve every customer from one name of its own, at the cost of an unsigned DNS answer choosing where a gateway connects, a rule for when an off-domain host may carry a domain certificate, and the URL hygiene and change signalling that go with it (Section 4.3). Delegation by CNAME covers the real need. A later version could add such a field, but a relying party of this version would ignore it and fetch from `smimecert.<D>` regardless, so a publisher would have to keep that host answering, or the field would need a new version tag.
- **Convention-based discovery without DNS.** A directory could have

been defined at <https://<D>/.well-known/smimecert> with no DNS record, letting a domain deploy without touching DNS. Rejected: it would make “this domain does not support SCD” — by far the most common answer — cost a TCP connection, a TLS handshake and an HTTP request, per domain and potentially per recipient, forever, on domains that will never deploy. A DNS query is orders of magnitude cheaper, is cached by the resolver across all senders behind it, and has well-defined negative caching. The lost convenience is one DNS record.

8. Implementation

8.1 Lookup algorithm (normative summary)

```
# discover() runs at most once per recipient of a message, the number
# of such runs per message is capped, and runs for one D are
# serialised, one in flight at a time (6.4). path_validates() below
# bounded in effort (6.4) and applies an imported root's own name
# constraints (6.5).
```

```
discover(address):
    r = lookup(address)
    if r in (NO_DOMAIN, NOT_SUPPORTED, RESOLVER_FAILURE):
        d = NONE
    else:
        d = lookup_domain(D, address, r, txt)

    if d is FALL_THROUGH:
        r = NONE if r is NO_FORM else
            fetch_address(base, D, H, address)
        d = NONE

    if r is NO_FORM: r = NONE

    if (r in (NONE, ERROR, NO_DOMAIN, NOT_SUPPORTED,
              RESOLVER_FAILURE)
        and store.has_valid(address)):
        r = HELD

    if (d in (NONE, ERROR) and r is not NO_DOMAIN
        and store.has_valid_domain(D)
        and sender_policy_permits_domain_cert(D)
        and not (domain_policy(D) is INSTEAD
```

7.1: the two
certificate
one recipient
without a
there is no
domain side
H, base and
are as look
left them
4.4.1: the
per-address
lookup, in
own slot;
lookup for
non-ASCII
7.4: an abs
like any o
6.2: what i
is used wh
discovery
for both k
a mailbox
from any s
counts (7.
a held dom

```

        and (r is HELD or r is a certificate))):# cert only
        d = HELD# the policy
# 5.6 req 4
# 7.1, as a
# fetched on
if r is ERROR and d is not in (NONE, ERROR,# 5.6 req 4:
                                NOT_ALLOWED):# fallback on
    log_fallback_on_failure(D, address)# failure is
# logged
return combine(r, d)# 7.1: under
# d is added
# under "ins
# one of the
# used; comb
# logs which
# were chose
# (7.5) and
# a failure
# (ERROR,
# RESOLVER_F
# from an ab
# (NONE, NO_I
# NOT_SUPPOR
# for the po
# of 7.7

lookup(address):
    if not valid_addr_spec(address): return NO_DOMAIN# 3.1, 7.4: no
# never repa
    (L, D) = canonicalise(address)# 3.1
    if is_domain_literal(D): return NO_DOMAIN# 3.1: no D,
# lookup of
# kind
    if negative_cache.contains(D): return NOT_SUPPORTED# 4.2.1
    if failed.active("txt:" + D):# 5.5: not ye
        return RESOLVER_FAILURE# an ERROR,
    txt = dns_txt("_smimecertdir." + D)# 5.6 req 4
    if txt is SERVFAIL or timeout:# fall throug
        failed.note("txt:" + D)# 4.2.1: tra
        return RESOLVER_FAILURE# never a neq
# retried who
# memory lap
    records = [parse_scd1(s) for s in# 4.4: string
                concat_strings(txt) if begins_v_scd1(s)]# joined, the
# recognised
# first fiel
# 4.4.2: fie
# split on "
# unknown fla
# dropped, r

```

```

# fields agree
# drop exception
# which merge
# k = padded
# of 32 octets
# dropped
if len(records) == 0: # NXDOMAIN, No
    if memory.contains(D): log_disappearance_once(D) # other versions
    negative_cache.add(D, ttl=clamp(soa_negative_ttl, # 6.2: one line
                                    1h, 24h)) # the log update
    return NOT_SUPPORTED # next accepted
# nothing else
# held certificates
# in use
if len(records) > 1: # 4.2 step 3:
    log_ambiguous(D) # supported;
    txt = Record(flags=(records[0].flags if all( # kept only
        r.flags == records[0].flags # records agree
        for r in records) else []), # keys merged
        keys=union(r.keys for r in records))
else:
    txt = records[0]
txt.keys = first_eight(txt.keys) # 4.4.2: in order
# appearance
# at most eight
# verification
log_flag_appearance(D, txt, last_record.get(D), # 6.6: domain
                    store.accepted_from_scd_any(D)) # newly on the
last_record.set(D, txt.flags, txt.keys, # record, or
               until=memory.expiry(D) # domain that
               or end_of_this_run) # yielded /held
# certs; 6.2
# record as
# read, kept
# long as the
# memory, when
# acceptance
# may start
base = admin_base_url(D) or # 4.2: fixed length
    "https://smimecert." + D + "/.well-known/smimecert" # or
if not is_ascii(L): return NO_FORM # 7.4: no per
# name form;
# domain certificate
# D and still
# applies
H = hex(sha256(L)) # 3.2
if ("domain-only" in txt.flags and txt.keys
    and sender_policy_permits_domain_cert(D)): # 4.4.1: skip
    return NONE # fetch only
# sender that
# use /domain

```

```

# cert already
# is found by
# lookup_domain
# and discovered
# lookup_domain
# falls through
# a 404; the
# without a
# ignored

return fetch_address(base, D, H, address)

fetch_address(base, D, H, address):
    url = base + "/" + D + "/hu/" + H # 4.1
    resp = https_get(url) # 4.5; 5.5: c
    # then memory
    # the network
    # https_get l
    if resp is NOT_YET: return ERROR # 5.5: remember
    if resp is a transport failure: # 5.5: against
        failed.note(host(url), addr(resp)); return ERROR # host name
        # address tr
    if resp is a resource failure: # 5.5: against
        failed.note(url); return ERROR # resource
    if resp.status == 404: # 5.4: a nega
        if store.accepted_from_scd(address): # 6.2: a hel
            log_disappearance_once(address) # cert stays
            return NONE # is logged
    pool = parse_pem_certificates(resp.body, max=32) # 5.1
    if pool is OVER_BOUND: # 6.4: a resou
        failed.note(url); return ERROR # failure, no
        # truncation
    candids = [c in pool where binds_to(c, address)] # 5.3.1
    valid = [c in candids where path_validates(c, pool) # 5.3 req 2 to
        and not expired(c)
        and usable_for_encryption(c)
        and not revoked(c)]
    log_rejected(url, candids - valid, reasons) # 6.2: reject
    # logged, no
    # of them key
    if not valid: # 5.5: a 200
        negative(url); log_empty(url); return NONE # nothing us
    log_change_if_known(address, valid) # 6.6: a chan
    # known
    # correspond
    # 7.1: source
    # disagree
    store.accept(address, valid, paths(valid), SCD) # 6.2: accept
    memory.touch(D) # held with
    return select(valid) # intermedia
    # their orig

```

```

# https_get(u), here and below, returns the cached answer while it
# is fresh (5.5); else NOT_YET while the failure memory holds u, the
# host name of u, or every address that host resolves to; else
# revalidates a stale entry with a conditional GET, extending it or
# 304 (4.5), or makes the request under 4.5 – within the recipient
# budget, checking every resolved address and every redirect target
# against the address rule, never answering a 401 – and fills the
# cache. failed.note(x) remembers x for the local minimum of 5.5, the
# hour, and for a resource also evicts it from the cache, so a
# rejected 200 is not served again from it. A transport failure is
# a connection that cannot be made or is refused, a TLS failure, a
# timeout, or a name with no address that passes 4.5; a resource
# failure is a status other than 200 or 404, a body over the size
# limit, a coding other than identity or gzip, or a redirect off the
# host name or beyond the fifth (5.5).

```

```

lookup_domain(D, address, address_result, txt):          # 5.6 – extra
    flags = txt.flags                                    # txt as look
                                                         # left it
    fall = FALL_THROUGH if "domain-only" in flags        # 4.4.1: what
                                                         # failure be
                                                         # yields und
                                                         # flag; disc
                                                         # then runs
                                                         # per-address
                                                         # lookup
    if not sender_policy_permits_domain_cert(D):         # 5.6 req 4:
        return NOT_ALLOWED                               # entry for l
                                                         # the default
                                                         # off, so no
                                                         # is fetched
    held = store.has_valid(address)                      # 7.1: from a
                                                         # source
    if ((address_result not in (NONE, ERROR, NO_FORM))   # 7.1: under
        or held)                                         # "instead"
        and domain_policy(D) is INSTEAD):               # or found m
        return NOT_APPLICABLE                           # cert settl
                                                         # under "botl
                                                         # default) c
                                                         # and the ca
                                                         # encrypts to
    if not txt.keys: return NONE                         # 5.7: no key
                                                         # no domain
                                                         # published;
                                                         # already he
                                                         # used regar
                                                         # (discover(
    durl = base + "/" + D + "/domain"                   # 5.6 req 2:
    surl = base + "/" + D + "/domain.sig"               # host and n

```

```

resp = https_get(durl)
if resp is NOT_YET: return fall or ERROR
if resp is a transport failure:
    failed.note(host(durl), addr(resp))
    return fall or ERROR
if resp is a resource failure:
    failed.note(durl); return fall or ERROR

if resp.status == 404:
    if store.accepted_domain_from_scd(D):
        log_disappearance_once(D)
        return fall or NONE

sig = https_get(surl)
if sig is NOT_YET: return fall or ERROR
if sig is a transport failure:
    failed.note(host(surl), addr(sig))
    return fall or ERROR
if sig is a resource failure or sig.status != 200:
    failed.note(surl); return fall or ERROR

pool = parse_pem_certificates(resp.body, max=32)
if pool is OVER_BOUND:
    failed.note(durl); return fall or ERROR
k = (one_line_base64_of_64_octets(sig.body)
     and ed25519_verify_any(txt.keys,
        "SCD1 domain " + D + "\n"
        + concat(sorted(set(pem_bytes(c) for c in pool))),
        base64_decode(sig.body)))
if not k:
    refetch domain and domain.sig once, bypassing
    the RP's own HTTP cache, through the same
    NOT_YET, transport, resource and 404 branches
    as above, and verify again; then
    if still not k:
        failed.note(durl); failed.note(surl)
        return fall or ERROR

if not pool:
    negative(durl); log_empty(durl)
    return fall or NONE

```

... as above, including log_rejected(), but binds_to() uses the

domain-part-only rule of 5.6 req 3 and path_validates() runs again: the RP's configured anchors, as always: an imported private root is one of them (7.5), with its own name constraints applied (6.5), and nothing else is;

```
if nothing passed:                                # 5.5: nothing
    negative(durl); log_empty(durl)                # usable; 4.
    return fall or NONE                            # falls thro
store.accept(D, result, path(result), SCD)          # 6.2: leaf p
memory.touch(D)                                    # its interm
record_continuity(result, D)                        # 5.6 req 5:
return result
```

8.2 Publishing example

A domain publishing per-address certificates only:

```
example.org/
  hu/a418287638a9d71c1a563a47d37aa26207daeb183f6cf720caaa44dfdec31ea
```

served at <https://smimecert.example.org/.well-known/smimecert/> ,
announced by:

```
_smimecertdir.example.org. IN TXT "v=SCD1"
smimecert.example.org.    IN A    198.51.100.10
```

A domain that also publishes a domain certificate adds the file, its signature, and the 44 base64 characters of its Ed25519 public key to the record (Section 5.7):

```
example.org/
  hu/a418287638a9d71c1a563a47d37aa26207daeb183f6cf720caaa44dfdec31ea
  domain
  domain.sig

_smimecertdir.example.org. IN TXT "v=SCD1; k=<base64 key>"
```

or, with hosting delegated to a provider:

```
_smimecertdir.example.org. IN TXT    "v=SCD1"
smimecert.example.org.     IN CNAME  smime.provider.example.
```

The provider serves the name `smimecert.example.org` under a certificate it obtained for that name, and the same tree at the provider can serve every customer domain, each under its own `<D>` segment (Section 4.3). Nothing

changes for a relying party, which connects to `smimecert.example.org` either way.

8.3 Coexistence with SMIMEA

An RP MAY query SMIMEA [RFC8162] in parallel. Both are sources in the sense of Section 7.1, and this document ranks neither: a certificate from either must pass Section 5.3, and one that does is acceptable whichever answered. Where the two return different certificates the RP SHOULD log it. That is often nothing — Section 5.2 expects several valid certificates per address, and the two directories need not carry the same ones — but a certificate in one that fails validation while the other's passes is worth a look, and an RP MAY encrypt to both under separate `RecipientInfo`s rather than choose.

8.4 Notes for implementers

Reasoning that belongs to an implementation rather than to the protocol, collected here so that the normative sections can say what and not how.

- **Why the failure memory is keyed by resource, and transport failures by host (Section 5.5).** The attacks of Section 6.6 are selective: a host serving a bad `/domain` beside good per-address files, or the reverse. A domain-wide backoff would let one failing path deny the other. Keyed by resource, a broken or spoofed `/domain` costs a sender one attempt per hour and leaves the per-address lookups untouched. Transport failures are the exception and are keyed by host, because connect, TLS and timeout failures are host-level by nature: keyed by resource, a host that accepts the connection and stalls would cost one timeout per address the attacker names, and the attacker names them (Section 6.4's port-scan bullet shows how); keyed by host it costs one timeout per hour. The address is a second key for the same reason one level up: a publisher of many names — a wildcard record answers for every sub-domain — can point them all at one stalling address, and keyed by address that costs one timeout too.
- **Why an ambiguous answer is still support (Section 4.2, step 3).** Two copies of one value are not ambiguous and two different values are, and an attacker who adds a record can at most cause the flag's default, which a stripped flag causes anyway (Section 4.4.1). Differing values of a scalar hint are ambiguous where a set of keys merges naturally, and the loss of the key is the one thing a duplicate record must not cause. The union does widen the set of signers the RP accepts, but a key never

suffices alone (Section 5.7): an attacker who can add a record to an answer gains exactly what one who can replace it gains, which Section 5.7 already accepts, and dropping the keys instead would hand the cheaper attacker a denial. MTA-STS discards an ambiguous answer [RFC8461]; it gives no reason, but the one this document infers — two `id` values leave a sender unsure which policy is current — does not carry over to a record that names no policy. Treating the answer as “no support” instead would turn a publisher’s duplicate record into an outage and, worse, let an attacker who can add a record to an answer downgrade a domain without having to remove one (Section 6.2).

- **Why a rejected bundle is not kept for diagnosis (Section 6.2).** A bundle is content the publisher chooses and can make a sender fetch at will (Section 6.4), so storing what it contains would let a hostile domain add rows to the store on every lookup. And unlike a certificate harvested from an inbound message, which the gateway keeps for diagnosis because the sender will not send it again, a bundle can be fetched again whenever an administrator wants to look at it, so there is nothing to keep (Section 7.1).
- **Why `gzip` must be accepted whether or not it was asked for (Section 4.5).** [RFC9110] lets a server send any content coding when none was requested, and a host that serves a pre-compressed object does. Requesting compression gains nothing on a few kilobytes of PEM, and an RP that sends no `Accept-Encoding` receives an identity body from common servers in practice; an RP that does send it exercises the same bounded decoder on every fetch, which is a way of keeping that decoder tested.
- **Why local parts are compared case-insensitively (Section 5.3.1).** Outlook compares addresses without regard to case, and NSS lower-cases every address before it stores or compares one (its `CERT_FixupEmailAddr`), so a case-sensitive RP would reject matches that every common client accepts.
- **Where the request bound of Section 4.5 comes from.** One lookup makes one TXT query, at most one `/hu/` fetch, at most one `/domain` and one `/domain.sig`, and at most one refetch of those two on a verification failure (Section 5.7): five resource fetches at the outside, each of up to five redirects — under `domain-only` the `/hu/` fetch comes last instead of first, which is the same count — and one or two in the ordinary case, plus the address resolution each connection needs. With the failure memory, a publisher’s broken signature then costs a sender one attempt per hour and not one per message, and leaves the per-address lookups of the same domain untouched.

- **Why the signature covers the DER and not the file (Section 5.7).** The threat is not transit, which HTTPS leaves untouched, but deployment, for the reason DKIM canonicalises mail. A publisher signs the file on its gateway and then a sync tool with line-ending conversion, an FTP client in text mode, an editor that adds a final newline or a CDN that rewrites text alters it on the way to the web root; the result is the same certificates in different bytes, and a signature over the bytes would fail for every sender with nothing in the file to say why. A text canonicalisation in DKIM's style would have handled line endings only.
- **What the one-shot refetch can and cannot do (Section 5.7).** A relying party can bypass its own HTTP cache to fetch `domain` and `domain.sig` from the same moment; it cannot bypass its resolver's cache, so a record that is still the old one stays the old one. That is why the publisher waits a TTL between adding a key and re-signing, and why the refetch is for file skew, not for record skew.
- **Why the memory of Section 6.2 is safe to keep.** It takes no value from the record and is set only when a certificate was accepted. A spoofed record can therefore neither plant it, lengthen it nor shorten it, and a spoofed record that leads a sender to a wildcard host — one that answers 404 for `smimecert.<D>` without ever having heard of SCD — installs nothing, since nothing was accepted.
- **Why there is no publisher-controlled lifetime (Sections 4.4 and 6.2).** MTA-STX can afford a `max_age` because its value arrives authenticated over HTTPS; in an unauthenticated TXT record the value would be set by whoever can spoof the record, and every repair — a local clamp, a rule that a refresh never shortens, a pin that only a certificate could install — is a patch on that. Holding what was accepted gives the availability such a lifetime would be for without any of it, and a pinned key would have the further defect of being impossible to rotate out of senders that had pinned it.
- **Why the fallback on a failed per-address fetch is not refused (Section 5.6, requirement 4).** Refusing it would stop an attacker who can break the fetch, which takes no more than dropping packets, from steering mail onto the domain key. But “no certificate” sends the message to the failure policy of Section 7.7, and every outcome of that policy is worse for the sender: in the clear hands the attacker plaintext where the fallback leaves them needing the perimeter key, and defer or bounce lets the attacker delay or block mail at will. The steering gains an attacker something only if it already holds the perimeter key, which is the compromise a sender accepted when it set a domain policy at all, and under the “both” policy the message is encrypted to the domain key regardless.

- **Library primitives for Section 5.1.** In Java the pool is what `CertificateFactory.generateCertificates()` returns, fed to a `CollectionCertStore` used by a `PKIXBuilderParameters` path builder; in OpenSSL it is an untrusted stack handed to `X509_STORE_CTX`. Either way the relying party does its own path building and depends on nothing in the file's order.
- **Logging once.** Section 6.2 asks that a disappearance be logged once, of the record and of a per-address certificate alike. How that is achieved — a flag on the memory or on the held certificate, cleared by the next acceptance, or a de-duplicating logger — is the implementation's business; what matters is that an operator sees one line per event and not one per negative-cache lapse.

9. IANA considerations

- Registration of the well-known URI suffix `smimecert` in the “Well-Known URIs” registry [RFC8615].
- Registration of the underscored node name `_smimecertdir` for the TXT RRTYPE in the “Underscored and Globally Scoped DNS Node Names” registry [RFC8552].
- A registry for TXT record fields and for `flags` values would be needed for a real standards-track document.

10. References

- [EFAIL] Poddebniak, Dresen, Müller, Ising, Schinzel, Friedberger, Somorovsky & Schwenk, “Efail: Breaking S/MIME and OpenPGP Email Encryption using Exfiltration Channels”, 27th USENIX Security Symposium, 2018.
- [MS276597] Microsoft, “How to turn off e-mail matching for certificates in Outlook”, support article 276597 — the registry value is spelled `SupressNameChecks` there.
- [RFC2119] Bradner, “Key words for use in RFCs to Indicate Requirement Levels”.
- [RFC2308] Andrews, “Negative Caching of DNS Queries (DNS NCACHE)”.
- [RFC4648] Josefsson, “The Base16, Base32, and Base64 Data Encodings”.

- [RFC5083] Housley, "Cryptographic Message Syntax (CMS) Authenticated-Enveloped-Data Content Type".
- [RFC5084] Housley, "Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS)".
- [RFC5280] Cooper et al., "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile".
- [RFC5321] Klensin, "Simple Mail Transfer Protocol".
- [RFC5322] Resnick, "Internet Message Format".
- [RFC5891] Klensin, "Internationalized Domain Names in Applications (IDNA): Protocol".
- [RFC6376] Crocker, Hansen & Kucherawy, "DomainKeys Identified Mail (DKIM) Signatures", STD 76.
- [RFC6531] Yao & Mao, "SMTP Extension for Internationalized Email".
- [RFC6890] Cotton, Vegoda, Bonica & Haberman, "Special-Purpose IP Address Registries", BCP 153.
- [RFC6962] Laurie, Langley & Kasper, "Certificate Transparency" (Experimental; what deployed logs run).
- [RFC7208] Kitterman, "Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1" — Section 3.3 for multi-string TXT records.
- [RFC7468] Josefsson & Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures".
- [RFC7858] Hu et al., "Specification for DNS over Transport Layer Security (TLS)".
- [RFC8032] Josefsson & Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)" — Ed25519.
- [RFC8162] Hoffman & Schlyter, "Using Secure DNS to Associate Certificates with Domain Names for S/MIME" (SMIMEA, Experimental).
- [RFC8174] Leiba, "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words".
- [RFC8461] Margolis et al., "SMTP MTA Strict Transport Security (MTA-STS)".
- [RFC8463] Levine, "A New Cryptographic Signature Method for DomainKeys Identified Mail (DKIM)" — Ed25519 in a TXT record, the precedent for $k =$.
- [RFC8484] Hoffman & McManus, "DNS Queries over HTTPS (DoH)".
- [RFC8552] Crocker, "Scoped Interpretation of DNS Resource Records through 'Underscored' Naming of Attribute Leaves".
- [RFC8555] Barnes et al., "Automatic Certificate Management

Environment (ACME)” — defines `application/pem-certificate-chain` .

- [RFC8615] Nottingham, “Well-Known Uniform Resource Identifiers (URIs)”.
- [RFC8823] Melnikov, “Extensions to Automatic Certificate Management Environment for End-User S/MIME Certificates” (Informational, April 2021).
- [RFC9110] Fielding, Nottingham & Reschke, “HTTP Semantics”, STD 97 — Section 9.1 already requires GET and HEAD of all general-purpose servers.
- [RFC9111] Fielding, Nottingham & Reschke, “HTTP Caching”, STD 98.
- [RFC9309] Koster, Illyes, Zeller & Sassman, “Robots Exclusion Protocol”.
- [RFC9525] Saint-Andre & Hodges, “Service Identity in TLS” (obsoletes RFC 6125).
- [RFC9598] Melnikov, Chuang & Bonnell, “Internationalized Email Addresses in X.509 Certificates” (May 2024, obsoletes RFC 8398); defines `Smtputf8Mailbox` .
- [RFC9849] Rescorla, Oku, Sullivan & Wood, “TLS Encrypted Client Hello” (March 2026).
- [SBR] CA/Browser Forum, “Baseline Requirements for the Issuance and Management of Publicly-Trusted S/MIME Certificates”, version 1.0.16 at the time of writing; Section 3.2.2 for mailbox validation methods, 7.1.4.2.1 for the `subjectAltName` profile, and the `2.23.140.1.5` certificate policy arc in the CA/Browser Forum object registry for certificate types.
- [TLSBR] CA/Browser Forum, “Baseline Requirements for the Issuance and Management of Publicly-Trusted TLS Server Certificates”, current version; Section 3.2.2.4 for the domain validation methods that [SBR] delegates to.
- [WKD] Koch, “OpenPGP Web Key Directory”, draft-koch-openpgp-webkey-service (revision -22 at the time of writing).