Individual Submission
Internet-Draft
Intended status: Standards Track
Expires: 30 November 2026

K. Swaminathan
Independent

29 May 2026

Domain Key Authorities (DKA): DNS-Designated Public Key Distribution for Email-Address Identifiers
draft-swaminathan-dka-framework-02

Abstract

This document specifies the Domain Key Authority (DKA) framework, a DNS-anchored public-key distribution mechanism for the email-address namespace. The framework enables an Internet domain to designate an authoritative key service that verifies, stores, and distributes selector-scoped public keys for email-address identifiers under that domain. The result is a decentralized, deterministic, and application-agnostic framework for verified public-key discovery that supports incremental deployment and cryptographic agility.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-swaminathan-dka-framework/.

Status of This Memo

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

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

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

This Internet-Draft will expire on 30 November 2026.

Table of Contents

1. Introduction

Email addresses are widely used as identifiers for individuals and automated agents on the Internet. Beyond email communication, they serve as login identifiers for banking, government services, e-commerce, social media, healthcare portals, and enterprise applications. An email-address identifier is therefore already a practical Internet-wide identifier, but it is not, by itself, a cryptographic identifier: knowing an email address does not provide a means to encrypt to its holder, verify a signature from its holder, or authenticate possession of a corresponding secret.

A framework that associates public keys with email-address identifiers and makes those bindings discoverable in a consistent, interoperable manner could enable end-to-end encrypted email without proprietary infrastructure, digitally signed messages with stronger origin assurance, passwordless authentication in which a relying party verifies possession of a private key rather than a shared secret, and secure key distribution for messaging systems and related protocols.

Over the past three decades, several systems have addressed parts of this problem. Each contributed important ideas, but each also revealed limitations that inform the design requirements for a more deployable framework.

1.1. Prior Approaches and Lessons Learned

1.1.1. OpenPGP and the Web of Trust

OpenPGP [RFC9580] introduced a decentralized model in which users generate their own key pairs and other users may certify bindings between identities and keys through a web of trust. Public keys can be uploaded to key servers for distribution.

This model demonstrated the value of decentralized key generation and distribution, but it has seen limited deployment outside technically sophisticated communities. In particular, user-to-user certification does not scale well for the general Internet population, key servers do not by themselves establish verified bindings between an email address and a submitted key, and key discovery is not deterministic: a relying party may need to consult multiple servers, and the absence of a key at one location does not establish that no key exists.

These observations suggest that a more deployable framework should provide built-in verification of the binding between an email address and its public key, should support deterministic lookup, and should not depend on user-to-user coordination for routine operation.

1.1.2. S/MIME and Certificate Authorities

S/MIME [RFC8551] addressed the verification problem by using Certificate Authorities (CAs) to certify bindings between identities and public keys. In enterprise and managed environments, this approach can provide a workable solution for encrypted and signed email.

However, S/MIME deployment has generally depended on access to CA infrastructure and related operational arrangements, which has limited participation outside managed environments. This suggests that a broadly deployable framework should not depend on managed CA infrastructure for baseline participation.

1.1.3. DANE and DNS-Based Key Distribution

DANE [RFC7671] and OPENPGPKEY [RFC7929] contributed an important architectural insight: the domain namespace is a natural place to begin when looking for keying information associated with addresses under that namespace, and DNS is a natural mechanism for discovering where such information can be obtained.

However, storing per-user key material directly in DNS can create operational challenges at large scale. DNS is well suited to publishing domain-level and service-level information, but per-user key distribution may require storage, update, and retrieval mechanisms that scale more naturally through database-backed services and application-layer APIs.

The lesson is that DNS is well suited to discovery: it can tell a client where to look for key information, while the key material itself may be better served through infrastructure that scales independently of DNS.

1.1.4. Web Key Directory

Web Key Directory (WKD) [I-D.koch-openpgp-webkey-service] partially addressed this issue by separating publication from DNS storage and delivering keys over HTTPS. This demonstrated the practical value of using existing web infrastructure to distribute per-address key material.

WKD defines a conventional domain-hosted location at which an OpenPGP key may be published for an email address. Because OpenPGP keys may also be distributed through other mechanisms, failure to find a key via WKD does not imply that no OpenPGP key exists for that address elsewhere. In addition, WKD is specific to OpenPGP and email encryption, whereas DKA is intended as an application-agnostic framework for distributing selector-scoped public keys for email-address identifiers.

1.2. Why Now: Drivers

Several developments make this a practical time to standardize a framework for verified public-key distribution for email-address identifiers:

* *Passwordless authentication adoption*: FIDO2/WebAuthn and related public-key login models are increasing demand for cryptographic bindings between user identifiers and public keys without reliance on shared secrets.

* *End-to-end encryption at scale*: Messaging and collaboration systems increasingly need scalable, interoperable authentication of public keys that does not depend on proprietary directories.

* *Infrastructure readiness*: DNSSEC deployment, DNS-over-HTTPS [RFC8484], and ubiquitous HTTPS provide the discovery integrity and transport security needed for DNS-designated key services.

* *Cryptographic transition*: Migration to post-quantum algorithms (e.g., ML-KEM, ML-DSA) benefits from a distribution framework that is algorithm-agnostic and can evolve without requiring protocol redesign.

These factors do not create the need for verified public-key distribution for email-address identifiers; that need has long existed. Rather, they create conditions under which a framework like DKA can achieve practical, incremental deployment.

1.3. Design Principles

The preceding discussion motivates the following design principles for the DKA framework:

* *Domain-designated authority.* A domain designates, using DNS, an authoritative key service for identifiers under its namespace. Authority over key records for a given identifier derives from that domain’s DNS designation.

* *Scalable key distribution.* Per-identifier key material is stored and served through infrastructure that scales independently of DNS.

* *Verified binding.* The framework provides mechanisms for verifying the association between an email-address identifier and its public key, and communicates which verification methods were performed so that relying applications can apply local trust policy.

* *Deterministic lookup.* Given an email-address identifier and a selector, a conforming client obtains a single definitive result by following a fixed lookup order: a matching key record, an indication that no record exists, or an error.

* *Application agnosticism.* The framework distributes keys without prescribing or constraining what applications do with them.

* *Cryptographic agility.* The framework does not prescribe a single key type or algorithm and supports future cryptographic schemes.

* *Incremental deployment.* The framework operates over existing DNS, email, and HTTPS infrastructure and does not require coordinated ecosystem-wide upgrades.

1.4. Scope

This document specifies the DKA framework: DNS-based designation of Domain Key Authorities, the key lookup protocol, the key registration mechanism and selector-scoped key management.

The framework is application-agnostic. Applications that consume DKA-distributed keys (such as encrypted email, passwordless authentication, or cryptocurrency wallet addressing) are outside the scope of this document.

2. Requirements Language

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

3. Terminology

* *Email-Address Identifier:* An identifier having the syntactic form of an email address (local-part@domain), used as the name to which one or more public keys can be bound. The syntax of email-address identifiers follows [RFC5321].

* *Domain Portion:* The portion of an email-address identifier following the "@" character.

* *Domain Key Authority (DKA):* A network-accessible service designated by an Internet domain to collect, verify, store, and distribute public keys associated with email-address identifiers under that domain.

* *Designating Domain:* An Internet domain that publishes one or more DKA Discovery Records in order to designate a DKA as authoritative for public keys associated with email-address identifiers under that domain.

* *DKA Discovery Record:* A DNS record by which a designating domain designates its DKA. In this specification, a DKA Discovery Record is published at the DNS name formed by prepending "_dka." to the designating domain. A DKA Discovery Record MAY be expressed as an HTTPS record or as a TXT record using the dka= tag-value form.

* *Selector:* A string value that distinguishes one public key from another for the same email-address identifier, enabling different keys for different application contexts.

* *Default Selector:* The selector value default, used when no selector is explicitly specified.

* *Public Key Record:* A data object maintained by a DKA that associates a public key with an email-address identifier and selector, together with verification metadata and optional application metadata.

* *Verification Methods:* Named methods performed by a DKA to verify the association between an email-address identifier and a submitted public key.

* *Key Lookup Request:* A request sent by a client to a DKA to obtain the Public Key Record associated with a specified email-address identifier and optional selector.

* *Key Lookup Response:* A response returned by a DKA containing the requested Public Key Record, an indication that no matching record exists, or an error.

* *Requesting Client:* A software component that performs DKA discovery and sends Key Lookup Requests.

* *Registrant:* An entity that submits a public key for association with an email-address identifier.

4. Architecture Overview

The DKA framework is a distributed framework in which Internet domains designate Domain Key Authorities to act as key services for email-address identifiers under those domains. The framework separates four functions that have been conflated, underspecified, or application-bound in earlier approaches:

* *Designation of authority.* DNS provides the designation function at domain granularity, identifying which service is authoritative for key records under a given domain.

* *Storage of key material.* The DKA service stores per-identifier key records at identifier-level granularity using infrastructure that scales independently of DNS.

* *Retrieval of key records.* The DKA service is discovered by applications via DNS and accessed via HTTPS to retrieve structured key records with metadata.

* *Identifier-to-key binding.* The framework treats an email-address identifier as an Internet identifier to which one or more public keys may be bound, without limiting use of those keys to email transport. This separates the identifier’s role as a routable email address from its broader role as a name for a person, agent, service, or account in applications that consume DKA-distributed keys.

This separation preserves domain-level authority while permitting key storage and lookup to be implemented using databases, caches, or other application-layer infrastructure.

DKA Framework Architecture

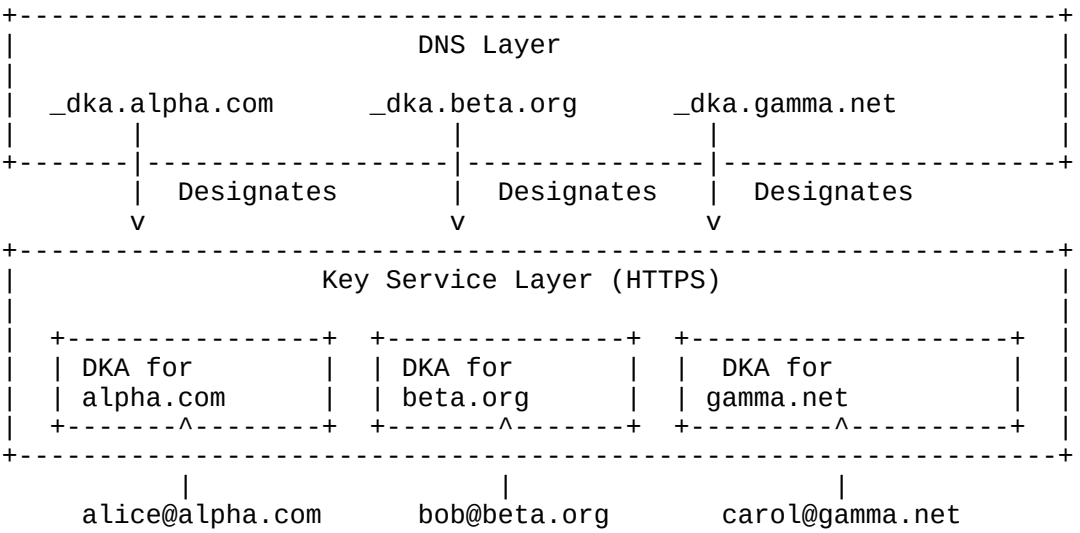

Figure 1: DKA Framework Architecture

4.1. Selector-Scoped Keys

An email-address identifier may be associated with multiple public keys, each distinguished by a selector. Selectors allow different applications to use different keys without the DKA interpreting what the selectors mean or what applications consume them.

For example, an identifier might have one key under selector default for general use, another under auth for authentication, and another under signing for digital signatures. The DKA stores and serves keys by identifier and selector without assigning semantic meaning to selector values. Applications define their own selector conventions independently.

4.2. Cryptographic Agility

The DKA framework does not prescribe a key type or algorithm. Selector-scoped Public Key Records MAY maintain metadata associated with each public key to indicate its cryptographic properties. This cryptographic agnosticism enables the DKA framework to support future cryptographic schemes, including post-quantum schemes.

4.3. Deterministic Lookup

For any given (email-address identifier, selector) pair, a conforming client queries the DKA designated by the identifier's domain. If that DKA returns a matching Public Key Record, that record is the lookup result. If the domain DKA returns a definitive not-found result, or if no domain DKA is designated, the lookup result is not found, meaning that the framework defines no public key for that (email-address identifier, selector) pair. Because all conforming clients follow this same lookup procedure, they obtain the same result for the same (email-address identifier, selector) pair, assuming the underlying DKA state is unchanged.

4.4. Incremental Deployment

The framework can be deployed by a single domain or multiple domains, serving the email-identifiers of those domains. It is not dependent on universal adoption.

The framework operates over existing DNS, email, and HTTPS infrastructure and does not depend on yet-to-be invented technology or protocols.

4.5. Discovery and Retrieval Flow

A Requesting Client begins with an email-address identifier for which it seeks a public key. The flow proceeds as follows:

* The client performs DKA discovery for the identifier's domain by querying DNS for DKA Discovery Records at _dka.<domain>.

* If discovery succeeds, the client obtains a single DKA hostname and queries that DKA over HTTPS.

* If the DKA returns a matching Public Key Record, the lookup is complete.

* If no DKA Discovery Record exists for the domain, or if the DKA returns a definitive not-found result for the queried (email-address identifier, selector) pair, the lookup result is not found.

* If DKA discovery fails because of inconsistent DKA Discovery Records, or if DKA lookup fails with a non-404 error, the lookup fails with an error.

The normative client discovery procedure is specified in Section 5.4 and the normative lookup procedure is specified in Section 7.5.

5. DNS Designation

5.1. DKA Discovery Record

A domain designates its DKA by publishing one or more DKA Discovery Records in DNS at the name formed by prepending "_dka." to the domain.

A DKA Discovery Record MAY be expressed as:

* an HTTPS record [RFC9460], and/or

* a TXT record using the DKA tag-value syntax defined by this specification.

The TXT form of the DKA Discovery Record identifies the designated DKA hostname using the dka tag:

```
_dka.example.com.  IN  TXT  "dka=dka.example.com"
```

The HTTPS form of the DKA Discovery Record identifies the same DKA hostname and MAY provide standard HTTPS RR parameters for connection establishment:

```
_dka.example.com.  IN  HTTPS  1  dka.example.com.  alpn=h2,h3
```

The HTTPS form is used to convey standard HTTPS RR information such as protocol negotiation and other connection metadata defined by [RFC9460]. This specification defines no DKA-specific HTTPS RR parameters.

For interoperability and backward compatibility, a domain that publishes an HTTPS DKA Discovery Record MUST also publish a TXT DKA Discovery Record at the same owner name.

Determinism is a design goal of the DKA framework and applies both to DKA discovery and to key lookup. For a given designating domain, all valid DKA Discovery Records present at _dka.<domain> MUST designate the same DKA hostname. A client that encounters HTTPS and TXT DKA Discovery Records designating different hostnames MUST treat the condition as a configuration error and abort discovery.

In this version of the specification, the TXT form contains the DKA hostname and the HTTPS records MAY provide connection metadata for reaching that hostname, but MUST NOT change the identity of the designated DKA. In this version of the specification, HTTPS AliasMode is not used. A client that encounters an HTTPS DKA Discovery Record in AliasMode MUST treat the condition as a configuration error and abort discovery.

The dka tag and DKA hostname are defined by the following ABNF [RFC5234]:

```
dka-discovery-txt = dka-tag *(*WSP ";" *WSP future-tag)
dka-tag           = %s"dka=" dka-hostname
future-tag        = tag-name "=" tag-value
tag-name          = ALPHA *(ALPHA / DIGIT / "-")
tag-value         = *(%x21-3A / %x3C-7E)
                    ; printable ASCII excluding ";"
dka-hostname      = sub-domain *("." sub-domain)
sub-domain        = Let-dig [Ldh-str]
Let-dig           = ALPHA / DIGIT
Ldh-str           = *(ALPHA / DIGIT / "-") Let-dig
```

Figure 2: ABNF for DKA Discovery Record TXT Form and DKA Hostname

The underscore-prefixed subdomain follows the conventions established in [RFC8552] for DNS names used with underscore-scoped service designations.

Future versions of this specification are expected to use distinct underscored DNS labels (e.g., _dka2, _dka3) rather than in-band versioning in the TXT payload.

5.2. DKA Service Endpoint

The DKA hostname obtained through the client discovery procedure (Section 5.4) identifies the DKA service endpoint.

The DKA service MUST expose its service root at the well-known URI path /.well-known/dka/. The lookup operation MUST be performed at /.well-known/dka/lookup. Clients MUST construct the full lookup URI by combining the discovered DKA hostname with the lookup path.

The DKA service MUST be served over HTTPS [RFC9110]. Clients MUST NOT send Key Lookup Requests over unencrypted HTTP.

5.3. Subdomain Scoping

Each domain portion in an email-address identifier is treated as a distinct designating domain. A DKA Discovery Record at _dka.example.com designates a DKA for email-address identifiers whose domain portion is example.com. It does not apply to sub.example.com or any other subdomain. A DKA Discovery Record at _dka.sub.example.com would be required to designate a DKA for sub.example.com. Nothing in this specification requires distinct DKAs for distinct domains or subdomains; example.com and sub.example.com MAY designate the same DKA service.

5.4. Client Discovery

Given an email-address identifier user@<domain>, a Requesting Client discovers the corresponding DKA as follows:

1. The client extracts the domain portion of the email-address identifier.

2. The client queries DNS for HTTPS and TXT records at _dka.<domain>.

3. For each HTTPS record, the client extracts the TargetName as the DKA hostname candidate.

4. For each TXT record, the client parses the record as a DKA TXT Discovery Record and extracts the hostname from the dka= tag. TXT records that do not conform to the DKA TXT syntax defined in this specification MUST be ignored.

5. If the client finds no valid HTTPS or TXT DKA Discovery Record at _dka.<domain>, discovery return not found.

6. All valid DKA Discovery Records present at _dka.<domain> MUST designate the same DKA hostname. If they do not, the client MUST treat the condition as a configuration error and abort discovery.

7. If one or more HTTPS records are present and all valid discovery records agree on the same hostname, the client selects the HTTPS record with the lowest priority value. If multiple HTTPS records share the same lowest priority, the client MAY choose any of them. The client uses the common designated hostname as the DKA hostname and MAY use the selected HTTPS record’s SvcParams for connection establishment as specified in [RFC9460].

8. If no HTTPS record is present but one or more valid TXT records are present, the client uses the common hostname extracted from the TXT records as the DKA hostname.

9. The client constructs the DKA Service Endpoint as https://<dka-hostname>/.well-known/dka/.

Determinism is a design goal of the DKA framework and applies to DKA discovery as well as key lookup. The above procedure ensures that all conforming clients derive the same DKA hostname for a given domain, regardless of whether discovery uses HTTPS records, TXT records, or both.

6. Key Registration

6.1. Registration Protocol

A registrant submits a public key to a DKA by email. A DKA MUST accept registration messages at the mailbox register@<dka-hostname>, where <dka-hostname> is the hostname specified in the DKA Discovery Record. The operator MAY implement this mailbox using aliases, forwarding, or other local mail handling arrangements. A domain MAY additionally provide a convenience address such as register-dka@<domain> that forwards to the DKA registration mailbox, but support for such forwarding is outside the scope of this specification.

Public key registration follows a two-step protocol.

Step 1 -- Initiation.
The registrant sends an email from the email address to be associated with the key to the DKA registration mailbox.

Step 2 -- Token exchange and key submission.
The DKA responds to the submission email address with a verification email containing a unique, time-limited verification token and instructions for completing registration. The registrant replies with the verification token and a JSON payload containing the public key and optional registration parameters:

```
{
  "token": "<verification-token>",
  "public_key": "<base64-encoded-key>",
  "selector": "<optional; defaults to 'default'>",
  "metadata": {}
}
```

In this version of the specification, the JSON payload MUST be included either:

1. in a MIME part with Content-Type: application/json, or

2. as the entire content of a text/plain MIME part whose body parses as a complete, syntactically valid JSON value.

If neither condition is met, the DKA MUST reject the submission.

Future versions of this specification may define additional submission mechanisms. It is expected that key submission and verification workflows will typically be performed with the aid of automated assistants.

The reply email MUST originate from the same email address to which the verification token was sent. The DKA MUST verify that the From header on the reply matches the address to which the token was delivered.

6.2. Domain Verification

A DKA MUST verify that the domain portion of the submission email address matches the domain the DKA serves. If the domain portion does not match, the DKA MUST reject the submission.

6.3. Email-Address Verification

The registration protocol described above establishes the registrant’s control of the submission email address through two mechanisms:

* *Mailbox control.*
  The verification token is delivered to the submission email address and must be returned by the registrant. Successful return of the token confirms that the registrant has access to the mailbox associated with the submission email address.

* *DKIM validation.*
  The DKA validates the DKIM signature [RFC6376] on the registration reply and verifies that the domain in the From address is the same as the DKIM signing domain identified by the d= tag. If these checks succeed, the DKA records the verification method dkim-validation.

The DKA records which verification methods were successfully performed. These are reported in the verification_methods field of the Key Lookup Response.

The framework MAY accommodate additional verification methods in future specifications. The initial verification method identifiers are:

* mailbox-control: The registrant demonstrated control of the mailbox.

* dkim-validation: A DKA-defined verification outcome indicating that the registration reply passed DKIM signature validation and that the domain in the From address matched the DKIM signing domain. This identifier documents what checks were performed by the DKA; it does not by itself imply any broader security or policy meaning beyond those checks.

6.4. Verification Tokens

Verification tokens MUST be generated with sufficient entropy to resist brute-force guessing. Tokens SHOULD contain at least 128 bits of entropy generated using a cryptographically secure random source.

Verification tokens MUST have a finite expiration period. The DKA MUST communicate the expiration period to the registrant in the verification email. The DKA MUST reject registration replies containing expired tokens.

6.5. Key Representation and Storage

The public_key field MUST be a base64-encoded value and the DKA MUST verify that the public_key field is syntactically valid base64 before storing the Public Key Record.

The use of base64 is a representation requirement, not a cryptographic one. It provides a uniform way to carry arbitrary public-key material in JSON, email, and HTTP without requiring the DKA to understand the underlying key format. The framework remains cryptographically agnostic because it does not prescribe the algorithm, structure, or semantic interpretation of the decoded key material.

Upon successful verification, the DKA stores the public key as a Public Key Record associated with the submission email address and selector. If a Public Key Record already exists for the same email-address identifier and selector, the new record replaces it, and the record’s version number is incremented.

6.5.1. Identifier Normalization

For storage and matching, the DKA MUST normalize the domain portion of the email-address identifier to lowercase. The local part MUST be preserved exactly as submitted and MUST NOT be case-normalized or otherwise rewritten.

> *Design Note:* Email-address identifiers are used beyond email transport (e.g., as login identifiers, cryptographic identities, or wallet addresses). Provider-specific email canonicalization rules (such as dot removal or plus-tag stripping) are not appropriate for these broader use cases and would break deterministic lookup.

The DKA MUST NOT apply provider-specific transformations such as dot removal, plus-tag stripping, or other canonicalization rules not defined by this specification.

6.6. Registration Confirmation and Rejection Messages

Upon successful registration, update, or deletion of a Public Key Record, the DKA SHOULD send a confirmation email to the submission email address indicating the outcome and the selector affected.

If a registration or deletion request fails (due to an expired or invalid token, invalid base64 in the public_key field, a domain mismatch, a payload containing both public_key and "delete": true, or any other reason) the DKA SHOULD send a rejection email to the submission email address indicating the reason for rejection. The DKA MUST NOT store or modify any Public Key Record as a result of a failed request.

6.7. Key Deletion

A registrant may request deletion of a Public Key Record by initiating the registration protocol and replying to the verification token with a JSON payload that identifies the selector to be deleted:

```
{
  "token": "<verification-token>",
  "selector": "<optional; defaults to 'default'>",
  "delete": true
}
```

A deletion payload MUST NOT contain a public_key field. If a payload contains both a public_key field and "delete": true, the DKA MUST reject the request.

A registration payload containing a valid public_key field and no "delete": true is a create-or-replace operation for the Public Key Record associated with the submission email address and selector. This operation is authorized by mailbox-control verification rather than by signature from a previously registered key; see Section 9.3.1.

The DKA performs the same verification as for registration before deleting the record identified by the submission email address and selector. If no selector field is present, the DKA deletes the Public Key Record associated with the default selector. If no matching Public Key Record exists, the DKA SHOULD treat the request as successfully completed.

Key Registration Flow

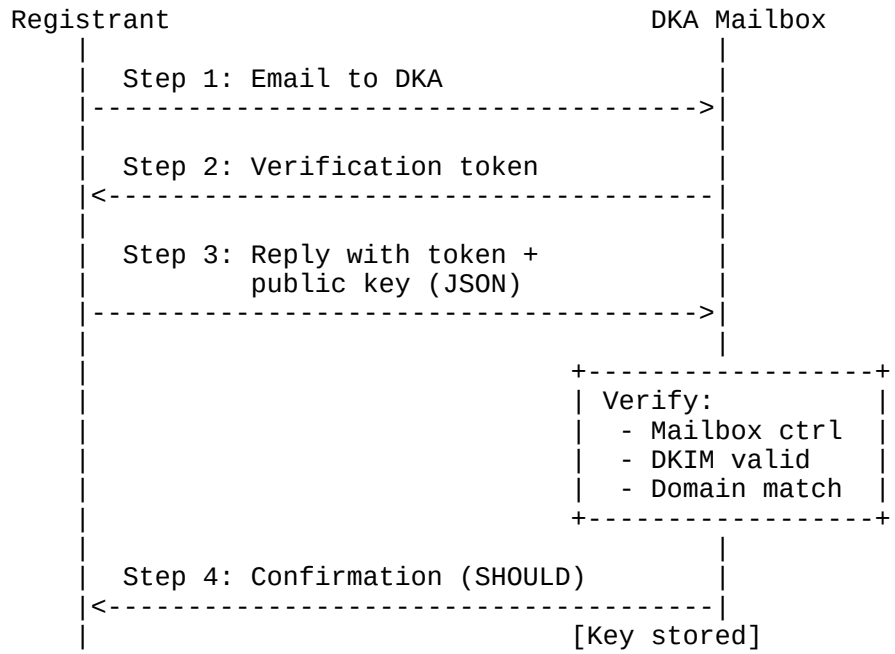

Figure 3: Key Registration Flow

7. Key Lookup

7.1. Lookup Request

A Requesting Client obtains a Public Key Record by sending an HTTPS GET request to the DKA’s lookup path:

```
GET https://dka.example.com/.well-known/dka/lookup
    ?email_address=bob%40example.com&selector=default
```

The email_address parameter is REQUIRED and specifies the email-address identifier to be looked up. The parameter value MUST be percent-encoded as needed for use in a URI query component, as specified in [RFC3986]. For example, the @ character MUST be encoded as %40. For matching purposes, the DKA MUST apply the same identifier normalization rules used during registration and storage.

The selector parameter is OPTIONAL. If omitted, the DKA returns the Public Key Record associated with the default selector.

7.2. Lookup Response

The DKA MUST return all responses with Content-Type: application/json.

A successful lookup returns a JSON object with the following fields:

```
{
  "email_address": "bob@example.com",
  "selector": "default",
  "public_key": "LS0tLS1CRUdJTi...",
  "verification_methods": ["mailbox-control", "dkim-validation"],
  "metadata": {
    "algorithm": "RSA",
    "format": "base64-PEM",
    "expires": "2027-01-31T00:00:00Z"
  },
  "version": 1,
  "updated_at": "2026-03-31T22:31:20+00:00"
}
```

Field definitions:

* *email_address* (REQUIRED): The email-address identifier associated with the Public Key Record.

* *selector* (REQUIRED): The selector value.

* *public_key* (REQUIRED): The base64-encoded public key.

* *verification_methods* (REQUIRED): An array of verification method identifiers indicating which methods the DKA successfully performed when the key was registered. This field provides provenance information for the key-to-identifier binding. It does not assert a trust level; relying applications apply their own trust policies based on the reported methods.

* *metadata* (REQUIRED): A JSON object containing key-value pairs provided by the registrant during registration. MAY be an empty object. The DKA stores and returns metadata without interpreting its contents.

* *version* (REQUIRED): An integer incremented each time the Public Key Record for this email-address identifier and selector is updated. Clients MAY use the version to detect key changes.

* *updated_at* (REQUIRED): An ISO 8601 timestamp indicating the time at which the Public Key Record was created, or if later modified, most recently updated.

If no Public Key Record exists for the specified email-address identifier and selector, the DKA MUST return an HTTP 404 response. The response MUST NOT distinguish between "no keys exist for this identifier" and "keys exist for this identifier but not under the specified selector."

7.3. Error Responses

When a request cannot be fulfilled, the DKA MUST return an appropriate HTTP status code with a JSON body containing at minimum the following fields:

```
{
  "error": "<error-code>",
  "message": "<human-readable description>"
}
```

The following error codes are defined:

* not_found: No matching Public Key Record exists (HTTP 404).

* invalid_request: The request is malformed or missing required parameters (HTTP 400).

* rate_limited: The client has exceeded the DKA's rate limit (HTTP 429).

* server_error: An internal error prevented the DKA from fulfilling the request (HTTP 500).

DKAs MAY define additional error codes. Clients MUST treat unrecognized error codes as equivalent to server_error.

7.4. Caching

DKAs SHOULD include appropriate Cache-Control headers in lookup responses.

Positive lookup responses MAY be cached, but DKAs SHOULD use cache lifetimes that reflect the possibility of key replacement or deletion.

Negative lookup responses (HTTP 404) MAY also be cached for a short duration in order to reduce repeated query load. Short negative-cache lifetimes are RECOMMENDED.

The specific caching policy is an operational decision for each DKA.

7.5. Lookup Order

A Requesting Client looking up a public key for a given (email-address identifier, selector) pair MUST proceed as follows:

1. The client performs DKA discovery for the identifier’s domain as specified in Section 5.4.

2. If DKA discovery returns not found, the lookup result is not found. Within the framework, no public key is defined for the queried (email-address identifier, selector) pair.

3. If DKA discovery fails because of a configuration error, the lookup fails with an error.

4. If DKA discovery succeeds, the client constructs the lookup URL and sends a Key Lookup Request to the discovered DKA, specifying the email-address identifier and optionally a selector.

5. If the DKA returns a matching Public Key Record, the lookup is complete with that record as the result of the lookup.

6. If the DKA returns an HTTP 404 response for the requested (identifier, selector) pair, the lookup result is not found. Within the framework, no public key is defined for that pair.

7. If the DKA returns a non-404 error (such as HTTP 500, HTTP 429, or a network timeout), the client SHOULD treat the condition as a transient failure and MAY retry. A non-404 error is not a not-found result.

This order is deterministic: for any given (email-address identifier, selector) pair, all conforming clients either obtain the same Public Key Record, the same not-found result, or an error.

Key Lookup Flow

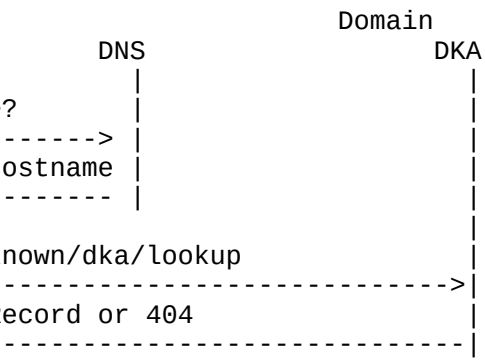

Figure 4: Key Lookup Flow

8. Selectors

8.1. The Default Selector

Every DKA MUST support a selector value of default. A registration that does not specify a selector is stored under the default selector. A lookup that does not specify a selector returns the Public Key Record associated with the default selector.

This ensures that the simplest interaction with the framework --- registering a key without specifying a selector, looking up a key without specifying a selector --- works without the registrant or requesting client being aware of the selector mechanism.

8.2. Selector Naming

Selector values are case-insensitive ASCII strings. The reserved selector value default MUST be supported by every conforming DKA and is used when no selector is explicitly specified. Other selector values MUST be one or more ASCII letters or digits, may include hyphens only in non-initial and non-final positions, and MUST NOT exceed 63 characters. Selector values are compared case-

insensitively.

The syntax of a selector value is defined by the following ABNF [RFC5234]:

```
selector              = default / non-default-selector
default               = %s"default"
non-default-selector  = selector-body
selector-body         = selector-start [selector-middle] selector-end
selector-start        = ALPHA / DIGIT
selector-middle       = 0*61(ALPHA / DIGIT / "-")
selector-end          = ALPHA / DIGIT
```

Figure 5: ABNF for Selector Values

The DKA framework defines one reserved selector value, default, which every conforming DKA MUST support. Apart from default, this specification does not define a registry of selector values or assign semantic meaning to selectors. Applications that consume DKA-distributed keys define their own selector conventions independently of this specification.

8.3. Multiple Keys

An email-address identifier MAY be associated with any number of Public Key Records under different selectors at the same DKA. Each (email-address identifier, selector) pair identifies at most one Public Key Record. Registration, update, and deletion operations act on a single (identifier, selector) pair.

9. Security Considerations

9.1. Security Properties

The DKA framework provides two core security-relevant properties:

* Verified provenance of key-to-identifier bindings. The framework reports what verification methods were successfully performed when a public key was registered for an email-address identifier.

* Deterministic discovery and lookup. For a given (email-address identifier, selector) pair, conforming clients follow a fixed discovery and lookup procedure that yields a single framework-defined result: a matching Public Key Record, an indication that no matching record exists, or an error.

After a successful lookup, a Requesting Client can determine that:

* For the queried (email-address identifier, selector) pair, if the conforming lookup procedure returns a Public Key Record, that record is the framework-defined result for that pair. If the conforming lookup procedure returns a definitive not-found result, then the framework defines no public key for that pair.

* The public key was submitted by a party who demonstrated control of the mailbox associated with the email-address identifier at the time of registration (if mailbox-control is reported).

* The submission satisfied the DKA-defined checks corresponding to dkim validation (if dkim-validation is reported).

The framework does not provide end-to-end authentication of the registrant’s identity, does not attest to the registrant’s possession of the corresponding private key (see Section 9.3.1), and does not by itself provide transparency or historical continuity of keys.

9.2. Verification Provides Provenance, Not Trust

The verification_methods field in a Key Lookup Response reports what the DKA verified when the key was registered. It does not assert a trust level or recommend a level of reliance.

A consuming application that accepts keys verified only by mailbox-control knows that someone with mailbox access registered the key, but has no domain-level endorsement of that binding. An application that additionally requires dkim-validation knows that the registration satisfied the DKA-defined checks associated with that verification method.

Applications SHOULD select verification requirements based on their threat model and the consequences of accepting a fraudulent key.

9.3. Live Key Distribution vs. Certificate PKI

The DKA framework embodies a different security model from traditional certificate PKI. Conventional PKIs are built around long-lived keys, certificate expiration periods, revocation mechanisms, and continuity of trust over time. By contrast, DKA is designed for live discovery of the currently published key for an (email-address identifier, selector) pair at the time of use.

This difference affects lifecycle management. In the DKA framework, registration, replacement, and deletion of key records are governed by control of the mailbox associated with the email-address identifier, rather than by signatures from previously registered keys. This model favors rapid recovery and key agility over cryptographic continuity of record control.

9.3.1. Mailbox Control as the Lifecycle Trust Anchor

The primary threat to the key-to-identifier binding is compromise of the registrant’s email account, which would allow an attacker to register a fraudulent key. This risk is inherent to any system that uses email-based verification and is mitigated by strong email account security practices, including multi-factor authentication.

In the DKA framework, authorization for key registration, replacement, and deletion is based on control of the mailbox associated with the email-address identifier, not on proof of possession of a previously registered private key. This is an intentional design choice. The DKA trust model treats continued mailbox control as the basis for maintaining the key-to-identifier binding over time. As a result, a registrant who retains mailbox control can replace or delete a Public Key Record even if the previously associated private key has been lost, rotated away, or compromised.

Applications that require cryptographic continuity of keys or signed update authorization, or stronger lifecycle assurances MUST layer such mechanisms above the DKA framework.

9.3.2. Proof-of-Possession and Lifecycle Operations

Proof-of-possession (PoP) verification is explicitly out of scope for this specification. The DKA framework is intentionally designed to operate at the key distribution layer, not the cryptographic protocol layer. Incorporating PoP would require the DKA to:

* understand and validate algorithm-specific signature schemes,
* generate and manage cryptographic challenges appropriate to each key type,
* implement replay protection and nonce validation logic, and
* potentially maintain session state for multi-step cryptographic handshakes.

These requirements would bind the distribution infrastructure to specific cryptographic protocols, undermining the framework's cryptographic agility and application agnosticism design principles.

The DKA framework therefore does not require signed updates, signed deletions, or other proof-of-possession checks tied to a previously registered private key. Mailbox control remains the lifecycle trust anchor. Applications that require assurance of private key possession SHOULD perform an application-layer proof-of-possession challenge directly with the key holder, using the cryptographic protocol appropriate to the key type and use case. The DKA provides the verified public key; the application performs the cryptographic validation.

Future specifications may define a standardized, algorithm-agnostic PoP extension for the DKA framework that preserves this separation of concerns. Until then, proof of private key possession remains an application-layer responsibility.

9.4. Metadata Authenticity

The metadata field in a Public Key Record is self-asserted by the registrant (or an application acting on the registrant's behalf) and is stored and returned by the DKA without interpretation, validation, or semantic processing. The DKA treats metadata as opaque key-value pairs communicated by the registrant to consuming applications regarding the public key.

This design is intentional: the DKA framework is application-agnostic, and different applications may define their own metadata conventions for different selectors. Further, different cryptographic algorithms may require different metadata. The DKA does not define the keys that belong in metadata, nor does it verify their values.

9.5. Key Lifecycle

Key updates and deletions follow the same verification requirements as initial registration. Detailed key lifecycle management, including rotation policies, revocation mechanisms, and multi-device coordination, is expected to be addressed in future specifications.

Future specifications may also define transparency logs or other mechanisms for tracking the public-key history of (email-address identifier, selector) pairs.

9.6. Transport Security

DKA service endpoints MUST be served over HTTPS to protect the integrity of lookup responses and prevent substitution of key material by on-path attackers.

A client discovering a DKA via an HTTPS DKA Discovery Record MAY use standard HTTPS RR connection metadata, as defined by [RFC9460], when establishing its HTTPS connection to the DKA. This specification defines no DKA-specific HTTPS RR parameters or transport modes.

9.7. DNS Security

The DKA framework relies on DNS for discovery of Domain Key Authorities and therefore inherits the known vulnerabilities of the DNS protocol, including cache poisoning, unauthorized zone modification, and man-in-the-middle attacks on queries that are not protected by DNSSEC. Without DNSSEC validation, an attacker capable of poisoning the cache for a _dka.<domain> record could redirect clients to a malicious DKA service that supplies attacker-controlled public keys, thereby undermining the integrity of key-to-identifier bindings.

Domains that publish DKA Discovery Records SHOULD deploy DNSSEC to provide origin authentication and data integrity for those records.

9.8. Non-Enumeration

A DKA MUST NOT provide any interface that enumerates email-address identifiers for which Public Key Records exist.

A DKA MUST NOT provide any interface that enumerates selector values associated with a given email-address identifier.

Lookups are point queries only: given an email-address identifier and a selector, the DKA returns the matching record or indicates that no matching record exists. When no matching record exists, the DKA MUST NOT distinguish between "this identifier has no keys" and "this identifier has keys but not under the specified selector". Both cases produce the same response.

9.9. Rate Limiting

DKAs SHOULD implement rate limiting on Key Lookup Requests to mitigate enumeration attempts through brute-force querying of identifier-selector combinations.

10. Privacy Considerations

The DKA framework publishes public-key bindings for email-address identifiers. As with any public lookup mechanism, this creates privacy considerations related to the exposure of identifier participation, metadata, and lookup behavior. The privacy properties of DKA lookups are those of standard HTTPS requests. The lookup request, including the queried email-address identifier, is protected by TLS. The DKA hostname is disclosed in DNS during discovery and is not considered confidential.

A successful lookup reveals that a given (email-address identifier, selector) pair has a registered Public Key Record at a particular DKA. This may disclose that the identifier participates in the framework and, in some cases, that the registrant uses multiple selectors for different application contexts. Although the DKA framework prohibits enumeration interfaces and recommends rate limiting, a determined attacker may still attempt to infer participation through repeated point queries.

The framework mitigates some privacy risks by decentralizing storage and lookup. Public Key Records are not maintained in a single global repository. Domains MAY operate their own DKAs, thereby limiting centralized aggregation of identifier-to-key bindings.

The DKA framework does not require submission, escrow, or storage of private keys. DKAs store only public-key material and associated metadata. Compromise of a DKA therefore does not, by itself, reveal private keys.

The framework avoids dependence on a separate revocation authority. A registrant who controls the mailbox associated with an email-address identifier can replace or delete a Public Key Record using the same verification process as registration. This allows users to revoke or rotate keys without relying on an external certificate or revocation service.

DKA operators will generally be able to observe lookup requests and registration traffic, including queried identifiers, source addresses, and timing information. Operators SHOULD minimize retention of logs containing personal data, SHOULD protect such logs appropriately, and SHOULD publish clear retention policies where feasible.

Applications using DKA-distributed keys should consider whether a lookup may itself disclose user intent. For example, querying for a recipient’s key may reveal an intention to communicate securely with that recipient. The lookup selector may reveal the purpose or

application context of such communication. Applications with stronger privacy requirements MAY use additional measures such as query minimization, proxying, or privacy-preserving access mechanisms.

The framework supports self-service key recovery, provided the user retains control of the associated mailbox.

11. Manageability Considerations

DKA operators are responsible for maintaining reliable, secure, and scalable key services. This section outlines operational considerations for deploying and managing DKAs.

11.1. Service Availability

DKA services SHOULD be operated with high availability. Operators SHOULD deploy redundant infrastructure, load balancing, and monitoring to ensure consistent responsiveness to lookup and registration requests.

11.2. Logging and Monitoring

Operators SHOULD implement logging sufficient to diagnose operational issues, detect abuse, and support security investigations. Logs SHOULD avoid storing unnecessary personal data and SHOULD be protected appropriately. Retention periods SHOULD be minimized.

11.3. Abuse Mitigation

DKAs SHOULD implement rate limiting, anomaly detection, and other abuse controls to prevent enumeration attempts, denial-of-service attacks, and malicious registration behavior.

11.4. Key Storage Integrity

Operators MUST ensure that stored Public Key Records are protected against unauthorized modification. Storage systems SHOULD support integrity verification, access control, and regular backups.

11.5. DNS Operations

Domains publishing DKA Discovery Records SHOULD maintain DNSSEC signing and monitoring. Operators SHOULD ensure timely updates to DKA hostnames and SHOULD avoid stale or conflicting records.

11.6. Email Deliverability

The DKA operator SHOULD configure the domain's mail infrastructure so that verification messages sent from register@<dka-hostname> to recipients under that domain are reliably accepted and are not rejected or filtered as spam.

11.7. Operational Transparency

Operators SHOULD publish documentation describing service behavior, retention policies, rate limits, and operational practices.

12. IANA Considerations

12.1. Well-Known URI Registration

This document requests registration of the following well-known URI [RFC8615]:

Field	Value
URI suffix	dka
Change controller	IETF
Specification document	This document
Status	permanent

Table 1

12.2. Verification Methods Registry

This document requests establishment of an IANA registry for DKA verification method identifiers. The initial contents of the registry are:

Identifier	Description	Reference
mailbox-control	Registrant demonstrated control of the mailbox for the submission email address	This document
dkim-validation	DKA-defined verification outcome for a registration reply that passes DKIM validation and that the domain in the From address matched the DKIM signing domain	This document

Table 2

New entries may be added through the Specification Required [RFC8126] policy.

13. References

13.1. Normative References

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

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

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

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

[RFC8552] Crocker, D., "Scoped Interpretation of DNS Resource Records through "Underscored" Naming of Attribute Leaves", BCP 222, RFC 8552, DOI 10.17487/RFC8552, March 2019, <https://www.rfc-editor.org/info/rfc8552>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/info/rfc8615>.

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

[RFC9460] Schwartz, B., Bishop, M., and E. Nygren, "Service Binding and Parameter Specification via the DNS (SVCB and HTTPS Resource Records)", RFC 9460, DOI 10.17487/RFC9460, November 2023, <https://www.rfc-editor.org/info/rfc9460>.

13.2. Informative References

[I-D.koch-openpgp-webkey-service] Koch, W., "OpenPGP Web Key Directory", Work in Progress, Internet-Draft, draft-koch-openpgp-webkey-service-21, March 2024, <https://datatracker.ietf.org/doc/html/draft-koch-openpgp-webkey-service-21>.

[RFC7671] Dukhovni, V. and W. Hardaker, "The DNS-Based Authentication of Named Entities (DANE) Protocol: Updates and Operational Guidance", RFC 7671, DOI 10.17487/RFC7671, October 2015, <https://www.rfc-editor.org/info/rfc7671>.

[RFC7929] Wouters, P., "DNS-Based Authentication of Named Entities (DANE) Bindings for OpenPGP", RFC 7929, DOI 10.17487/RFC7929, August 2016, <https://www.rfc-editor.org/info/rfc7929>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8484] Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", RFC 8484, DOI 10.17487/RFC8484, October 2018, <https://www.rfc-editor.org/info/rfc8484>.

[RFC8551] Schaad, J., Ramsdell, B., and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification", RFC 8551, DOI 10.17487/RFC8551, April 2019, <https://www.rfc-editor.org/info/rfc8551>.

[RFC9580] Wouters, P., Ed., Huigens, D., Winter, J., and Y. Niibe, "OpenPGP", RFC 9580, DOI 10.17487/RFC9580, July 2024, <https://www.rfc-editor.org/info/rfc9580>.

Appendix A. Changes from Previous Version

Difference from version 00 and 01: The major difference between version 00 and 01 is that the latter drops the Fallback DKA aimed at helping the bootstrapping of the framework entirely. The selector syntax, ABNF, and other details have been tightened, and the text has been edited for clarity.

Difference from version 01 and 02: The major difference between version 01 and 02 is that the latter renames DKA Locator Record as DKA Discovery Record and defines an HTTPS DKA Discovery Record in addition to the TXT form. The TXT form is now expressed as a tag-value record using the dka= tag, and the client discovery procedure requires all valid DKA Discovery Records present at _dka.<domain> to designate the same DKA hostname. Mismatches between DKA Discovery Records are treated as a configuration error.

Appendix B. Complete Example

This appendix illustrates the full DKA discovery, registration, and lookup flow using a working deployment.

B.1. DNS Discovery

A Requesting Client seeks the public key for alice@example.com. It queries DNS for HTTPS and TXT records at _dka.example.com and obtains:

```
_dka.example.com.  IN  HTTPS  1  dka.example.com.  alpn=h2,h3
_dka.example.com.  IN  TXT  "dka=dka.example.com"
```

The client parses the TXT record, extracts the hostname from the dka= tag, and confirms that the TXT and HTTPS records designate the same DKA hostname. The client then uses the HTTPS record as the preferred DKA Discovery Record for connection establishment.

B.2. Key Registration

Alice registers a public key with her domain's DKA:

1. Alice sends an email from alice@example.com to register@dka.example.com.

2. The DKA responds with a verification email:

   Subject: DKA: Your Verification Token

   Your DKA verification token:

     XGsQ2qfjXyJy8y6IVsKAUZQgmlaIL0G6fTbhK8KN

   This token expires in 600 seconds.

   To register a public key, reply with:
     Subject: register
     Body (JSON): {"token": "<token>", "public_key": "<base64>",
                   "selector": "<optional>", "metadata": {}}

3. Alice replies from alice@example.com with:

   Subject: register
   Body:
   {
     "token": "XGsQ2qfjXyJy8y6IVsKAUZQgmlaIL0G6fTbhK8KN",
     "public_key": "LS0tLS1CRUdJTi...",
     "selector": "default",
     "metadata": {
       "algorithm": "RSA",
       "format": "base64-PEM",
       "expires": "2027-01-31T00:00:00Z"
     }
   }

4. The DKA verifies mailbox control and applies the checks associated with dkim-validation, then stores the key.

B.3. Key Lookup --- Domain DKA

The Requesting Client sends a lookup request:

GET https://dka.example.com/.well-known/dka/lookup
    ?email_address=alice%40example.com&selector=default

The DKA returns:

```
{
  "email_address": "alice@example.com",
  "selector": "default",
  "public_key": "LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0t...",
  "verification_methods": ["mailbox-control", "dkim-validation"],
  "metadata": {
    "algorithm": "RSA",
    "format": "base64-PEM",
    "expires": "2027-01-31T00:00:00Z"
  },
  "version": 1,
  "updated_at": "2026-03-31T22:31:20+00:00"
}
```


Because the domain DKA returned a matching record, the lookup is complete.

Appendix C. Reference Implementation

An open-source reference implementation of the DKA framework is available. The implementation serves as the DKA for any domain once a configuration variable is set to that domain and the domain’s DNS record is configured as described in Section 5.

Appendix D. Working Demonstration

A working demonstration of the DKA framework, illustrating key registration and key lookup is available at:

* https://keyzero.org

D.1. Appendix: Example Selector Conventions for Early Deployment (Informational)

This appendix provides non-normative examples that early adopters MAY find useful. It does not create a registry, does not define interoperability requirements, and does not constrain future specifications or application-specific selector conventions.

Applications MAY use selector values such as the following as local conventions:

* default -- General-purpose key for the identifier

* encrypt -- Key intended for confidential message encryption

* auth -- Key intended for authentication or challenge-response

* sign -- Key intended for document or message signing

These values are examples only. Applications are free to define additional or alternative selectors.

Author’s Address

Kishore Swaminathan
Independent
United States
Email: k.s.swaminathan@live.com