Individual Submission
Internet-Draft
Intended status: Standards Track
Expires: 22 October 2026

K. Swaminathan
Independent

20 April 2026

Domain Key Authorities (DKA): DNS-Designated Public Key Distribution for Email-Address Identifiers
draft-swaminathan-dka-framework-00

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. A Fallback DKA (fDKA) provides coverage for identifiers whose domains have not deployed a DKA, addressing the bootstrapping problem that has hindered comparable proposals. 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 22 October 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. Despite this ubiquity, there is no commonly adopted mechanism by which a public key can be associated with an email address and discovered in a consistent, interoperable manner.

Such a mechanism 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.

Infrastructure frameworks that depend on per-domain deployment for initial utility face a well-known bootstrapping challenge: applications lack incentive to implement the framework until enough domains have deployed, and domains lack incentive to deploy until enough applications create demand. A successful framework must address this bootstrapping problem to achieve practical deployment.

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 support participation by any email address, including addresses under domains that have not deployed specialized PKI infrastructure of their own.

1.1.3. DANE and DNS-Based Key Distribution

DANE [RFC7671] and OPENPGPKEY [RFC7929] contributed an important architectural insight: the domain is a natural authority for information about addresses under its namespace, and DNS is a natural mechanism for designating that authority.

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 for designating where authoritative key information can be obtained, 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.

However, WKD relies on domain-hosted publication without defining a broader framework for verified key registration and universal lookup coverage. Further, WKD is limited to a particular application context and does not define a general mechanism for distributing multiple keys for the same identifier for different uses.

1.2. 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.

* *Universal coverage.* Via the Fallback DKA (fDKA), the framework provides immediate coverage for every email-address identifier regardless of whether that identifier’s domain has deployed a DKA, addressing the bootstrapping problem that has limited adoption of comparable proposals.

1.3. Scope

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

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.

* *Fallback DKA (fDKA):* A DKA designated by a well-known domain that accepts key registrations from email-address identifiers belonging to any domain. The fDKA is operationally identical to a domain DKA except that it does not restrict registrations to a single domain.

* *Designating Domain:* An Internet domain that publishes a DKA Locator Record designating a DKA as authoritative for public keys associated with email-address identifiers under that domain.

* *DKA Locator Record:* A DNS TXT record at a well-known subdomain by which a designating domain designates its DKA.

* *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

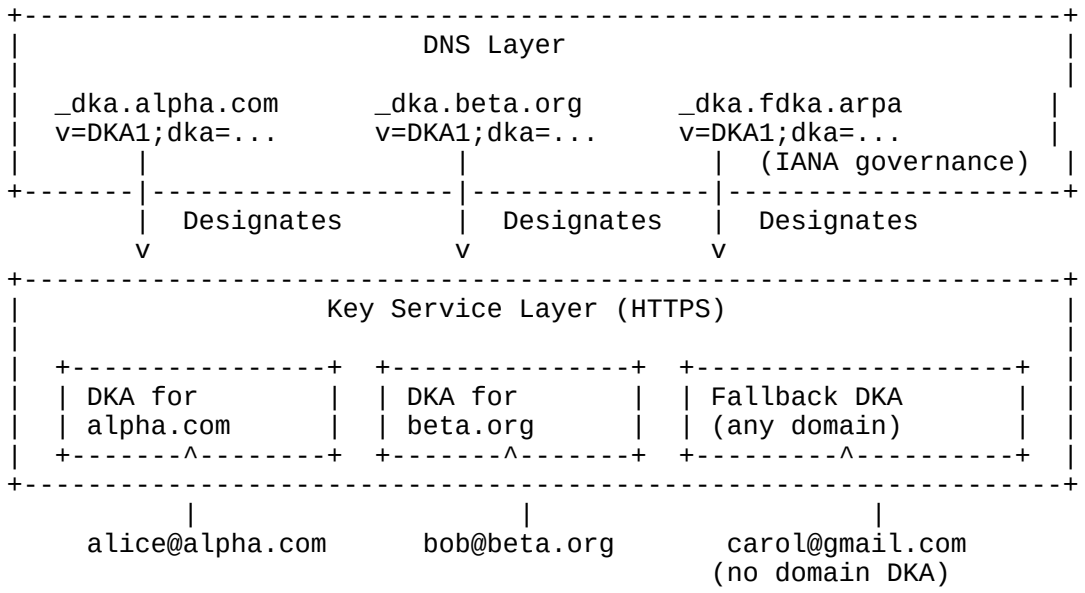

Figure 1: DKA Framework Architecture

4.1. Lookup Priority

The lookup protocol gives priority to the domain DKA for any (email-address identifier, selector) pair it holds. For a given pair, the client queries the domain DKA first. If the domain DKA returns a matching record, the lookup is complete and the fDKA is not queried. If the domain DKA does not hold a record for that pair, or if no domain DKA exists, the client queries the fDKA. The complete procedure is specified in Section 7.5.

4.2. Universal Coverage

The Fallback DKA (fDKA), designated by a well-known domain under IANA governance, accepts registrations from any email-address identifier regardless of its domain. The fDKA addresses the bootstrapping problem: any email address can register a key with the fDKA and any application can query it, creating a usable key-distribution service from initial deployment without requiring per-domain adoption.

As individual domains deploy their own DKAs, the lookup order (Section 7.5) naturally gives priority to domain-designated services. The fDKA continues to serve (identifier, selector) pairs for which the domain DKA has no record, ensuring uninterrupted coverage during the transition.

4.3. 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.

4.4. 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.

4.5. Deterministic Lookup

For any given (email-address identifier, selector) pair, a conforming client follows a fixed lookup order. The client first queries the DKA designated by the identifier's domain. If that DKA returns a matching Public Key Record, that record is the lookup result and the client does not query the fDKA for the same pair. If the domain DKA returns no matching record, or if no domain DKA is designated, the client queries the fDKA, whose response becomes the final result. Because all conforming clients follow this same ordered procedure, they obtain the same result for the same (email-address identifier, selector) pair, assuming the underlying DKA state is unchanged.

4.6. Incremental Deployment

The framework operates over existing DNS, email, and HTTPS infrastructure and is not dependent on universal adoption. Email-address identifiers under domains that have deployed a DKA use their respective DKAs for key distribution. Email-address identifiers under domains that have not deployed a DKA use the fDKA.

4.7. 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:

1. The client extracts the domain portion of the identifier.

2. The client queries DNS for a DKA Locator Record at _dka.<domain>.

3. If a DKA Locator Record is found, the client obtains the DKA hostname.

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

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

6. If no matching record is found at the domain DKA, or if no DKA Locator Record exists for the domain, the client sends a Key Lookup Request to the fDKA.

5. DNS Designation

5.1. DKA Locator Record

A domain designates its DKA by publishing a TXT record at the DNS name formed by prepending "_dka." to the domain. The record format follows the tag-value syntax used by DMARC [RFC7489] and DKIM [RFC6376]:

```
_dka.example.com.  IN  TXT  "v=DKA1;dka=dka.example.com"
```

The record contains the following tags:

* *v=DKA1* (REQUIRED): Version tag identifying this as a DKA designation record and indicating protocol version DKA1.

* *dka=* (REQUIRED): Hostname of the DKA service.

The record format is defined by the following ABNF [RFC5234]:

```
dka-record   = version-tag ";" dka-tag *(";" future-tag)
version-tag  = %s"v=DKA1"
dka-tag      = %s"dka=" dka-hostname
dka-hostname = sub-domain *("." sub-domain)
                 ; sub-domain as defined in RFC 5321, Section 4.1.2
future-tag   = tag-name "=" tag-value
tag-name     = ALPHA *(ALPHA / DIGIT / "-")
tag-value    = *(%x21-3A / %x3C-7E)
                 ; printable ASCII excluding ";"
```

Figure 2: ABNF for DKA Locator Record

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

A domain MUST NOT publish more than one valid DKA Locator Record at the same _dka subdomain. If multiple TXT records exist at that name, a client MUST ignore records that do not contain a syntactically valid v=DKA1 tag. If more than one valid DKA Locator Record remains, the client MUST treat the condition as a configuration error.

Clients MUST ignore unrecognized tags in the DKA Locator Record. This permits future versions to introduce additional tags without breaking existing clients.

5.2. DKA Service Endpoint

The DKA service identified by the DKA Locator Record MUST expose its lookup interface at the well-known URI path /.well-known/dka/ as defined by [RFC8615].

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

5.3. fDKA Designation

The fDKA is designated by a well-known domain managed under IANA governance. The designation uses the same DKA Locator Record format as domain DKAs. The recommended designation is fdka.arpa. The corresponding DKA Locator Record would be:

```
_dka.fdka.arpa.  IN  TXT  "v=DKA1;dka=dka.fdka.arpa"
```

The fDKA is intended to operate as a shared community infrastructure service, analogous to other IANA-managed infrastructure zones such as those under the .arpa top-level domain (see RFC 3172). IANA will designate the domain, and the IETF community will determine the detailed operational arrangements including selection of operator(s), service expectations, abuse mitigation policies, and transparency requirements through the normal IETF process.

The fDKA implements the same registration, verification, lookup, and response protocols as a domain DKA, with the sole difference that it does not enforce domain verification (Section 6.2). Operators of the fDKA zone MUST publish the DKA Locator Record with DNSSEC signing and are expected to follow established best practices for high-availability infrastructure services, including robust rate limiting, monitoring, and minimal logging of personal data.

5.4. Subdomain Scoping

Each domain portion in an email-address identifier is treated as a distinct designating domain. A DKA Locator 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 Locator Record at _dka.sub.example.com would be required to designate a DKA for sub.example.com.

5.5. Client Discovery

Given an email-address identifier user@example.com, a Requesting Client:

1. Extracts the domain portion: example.com.

2. Queries DNS for TXT records at _dka.example.com.

3. Parses the response for a record containing v=DKA1.

4. Extracts the DKA hostname from the dka= tag.

5. Constructs the DKA Service Endpoint as https://<dka-hostname>/.well-known/dka/.

6. Key Registration

6.1. Registration Protocol

A registrant submits a public key to a DKA by email. The registration follows a two-step protocol.

Step 1 -- Initiation.
The registrant sends an email from the email address to be associated with the key (the submission email address) to the DKA's designated mailbox. The email need not contain any specific content.

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 version DKA1, the JSON payload MUST be included in the body of the reply email. The DKA MUST extract the JSON from the first MIME part with Content-Type: application/json, or, if no such part exists, from the first text/plain MIME part whose content parses as valid JSON. Future versions of this specification may define additional submission mechanisms. It is expected that key submission and verification workflow will be performed through the use 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 sender address on the reply matches the address to which the token was delivered.

6.2. Domain Verification

A domain-scoped 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.

An fDKA does not perform this domain check and accepts public keys from any email address. This is the sole protocol-level difference between a domain DKA and the fDKA.

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 is designed to 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. Token Expiration

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 Storage

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.

The public_key field contains a base64-encoded value. The DKA MUST verify that the public_key field is syntactically valid base64 before storing the Public Key Record. The DKA stores and serves this value without interpreting its decoded contents. The framework does not prescribe the key type, algorithm, or format of the encoded key material. Interpretation of the decoded key material is the responsibility of the consuming application.

6.6. Registration Confirmation

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.

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.

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

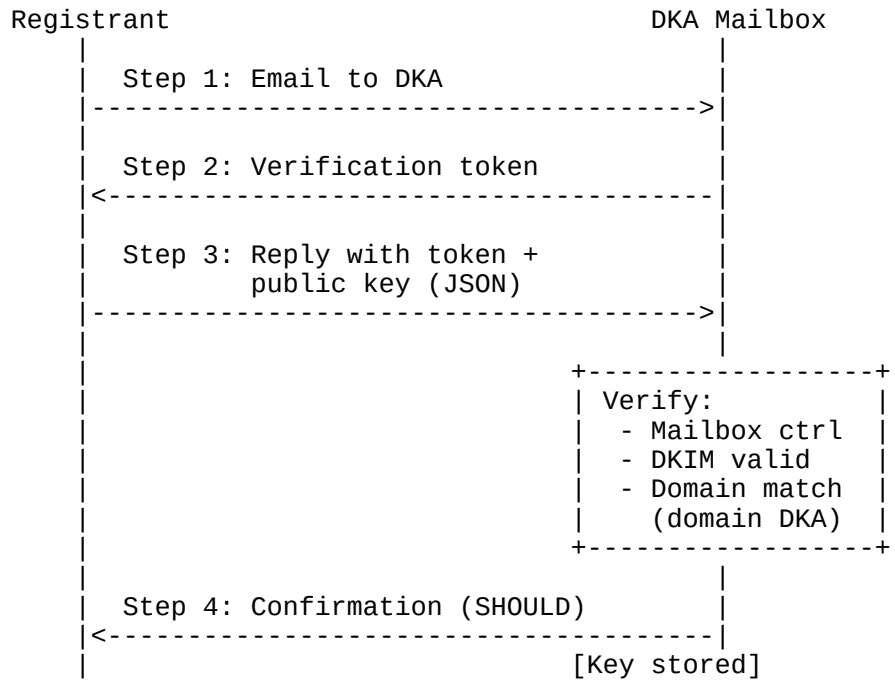

Figure 3: Key Registration Flow

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 endpoint:

```
GET https://<dka-host>/.well-known/dka/lookup
    ?email_address=<percent-encoded-email>&selector=<selector>
```

The email_address parameter is REQUIRED and specifies the email-address identifier. The value MUST be percent-encoded per [RFC3986]; in particular, the "@" character MUST be encoded as %40.

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 when the Public Key Record was last modified.

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. 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 queries DNS for a DKA Locator Record for the identifier's domain.

2. If a DKA Locator Record is found, the client sends a Key Lookup Request to the designated domain DKA.

3. If the domain DKA returns a matching Public Key Record, the lookup is complete. The client MUST NOT query the fDKA for the same (identifier, selector) pair.

4. If the domain DKA returns a 404 response for the requested (identifier, selector) pair, or if no DKA Locator Record exists for the domain, the client sends a Key Lookup Request to the fDKA.

5. The result from the fDKA (a matching record or 404) is the final result.

This order is deterministic: for any given (identifier, selector) pair, a conforming client always obtains the same result regardless of implementation. The domain DKA has priority; the fDKA serves as the fallback for (identifier, selector) pairs that the domain DKA does not hold.

A user MAY register keys at both their domain DKA and the fDKA. For a given email-address identifier and selector, a conforming client first queries the domain DKA. If the domain DKA returns a key for that selector, that key is the lookup result. Otherwise, the client queries the fDKA, and a key returned by the fDKA becomes the lookup result.

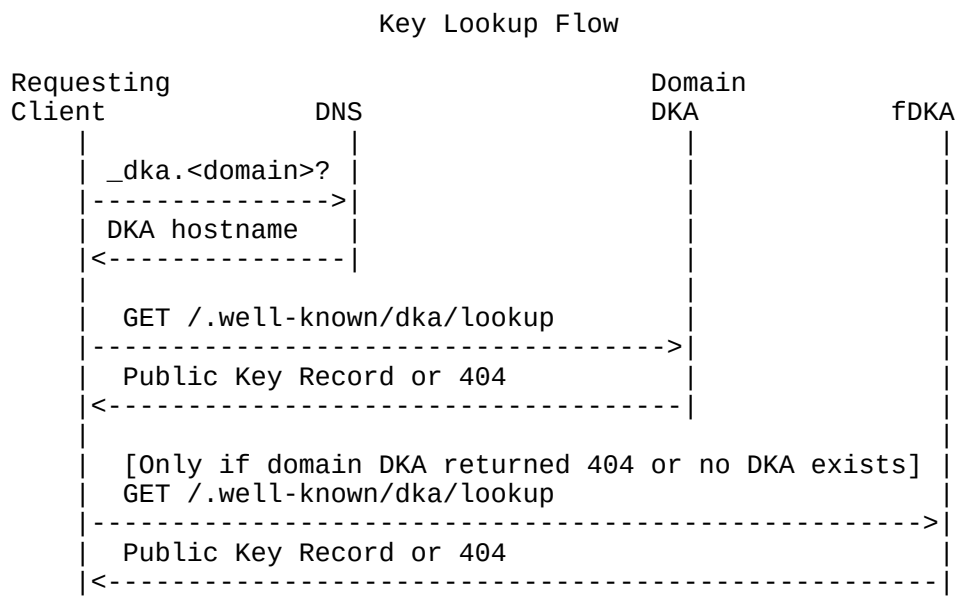

Figure 4: Key Lookup Flow

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 consisting of letters, digits, and hyphens. Selector values MUST NOT begin or end with a hyphen. Selector values MUST NOT exceed 63 characters in length.

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

```
selector        = selector-char *(selector-char / "-") selector-char /
                  selector-char
selector-char   = ALPHA / DIGIT
                 ; maximum 63 characters total
                 ; "-" may appear only between selector-char elements
```

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. The Fallback DKA

9.1. Purpose

The fDKA is necessary to bootstrap the DKA framework. Without the fDKA, the framework would provide service only for identifiers under domains that have already deployed a DKA, creating a deployment deadlock in which applications have limited incentive to implement the framework until enough domains participate, and domains have limited incentive to deploy until enough applications consume it. The fDKA breaks this deadlock by providing an immediately available location at which any email-address identifier can register a key and from which clients can retrieve keys using the same lookup model.

9.2. Designation

The fDKA is designated by a well-known domain managed under IANA governance. The designation uses the same DKA Locator Record format as domain DKAs.

9.3. Operational Equivalence

The fDKA implements the same registration protocol, verification methods, lookup protocol, and response format as a domain DKA. The sole operational difference is that the fDKA does not perform domain verification (Section 6.2): it accepts registrations from email-address identifiers belonging to any domain.

A conforming fDKA implementation is identical to a conforming domain DKA implementation with the domain restriction removed.

9.4. Registration Independence

The domain DKA and the fDKA are independent key stores. A user MAY register keys at both their domain DKA and the fDKA. The existence of a domain DKA does not prevent a user from registering at the fDKA, and a domain DKA does not suppress the fDKA. For a given (identifier, selector) pair, the domain DKA takes precedence in the lookup order (Section 7.5). As a result, the domain DKA has priority for selectors it serves, while the fDKA can serve selectors not present at the domain DKA.

Future specifications may define mechanisms by which a user can express a policy regarding which of their registered keys should be preferred for a given purpose.

10. Security Considerations

10.1. Security Property

The DKA framework provides a single security property: verified provenance of the key-to-identifier binding. After a successful lookup, a Requesting Client can determine that:

* 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 10.5), and does not by itself provide transparency or historical continuity of keys.

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

10.2. 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.

10.3. 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 modifications, 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 Locator Records SHOULD deploy DNSSEC to provide origin authentication and data integrity for these records. The DKA Locator Record designating the Fallback DKA (fDKA) MUST be signed with DNSSEC, as the fDKA might serve a large number of email address identifiers. Operators of the fDKA zone are expected to follow established DNSSEC operational practices, including regular key rollover and monitoring.

10.4. 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. Neither method verifies that the registrant possesses the corresponding private key.

Applications SHOULD select verification requirements based on their threat model and the consequences of accepting a fraudulent key. The framework does not impose policy; it provides the provenance data from which applications can make informed decisions.

10.5. Mailbox Compromise

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.

The current registration protocol verifies that the registrant controls the mailbox associated with the email-address identifier but does not verify that the registrant possesses the private key corresponding to the submitted public key. A proof-of-possession mechanism, in which the registrant signs a challenge with the corresponding private key, would strengthen the key-to-identifier binding by ensuring that only a party holding the private key can register it. Such a mechanism is expected to be addressed in a future specification.

10.6. 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.

10.7. Rate Limiting

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

10.8. Metadata Authenticity

The metadata field in a Public Key Record is self-asserted by the registrant and is not verified by the DKA. Consuming applications MUST NOT treat metadata values (such as algorithm identifiers or expiration dates) as authoritative without independent verification.

10.9. 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. Key transparency logs, which would allow clients to detect unexpected key changes, are a potential future extension.

11. 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.

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 fDKA provides universal coverage without requiring all domains to deploy their own DKAs, but it also concentrates lookup traffic for identifiers whose domains have not deployed a DKA.

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 and fDKA 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.

12. Manageability Considerations

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

12.1. 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.

12.2. 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.

12.3. 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.

12.4. 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.

12.5. DNS Operations

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

12.6. Operational Transparency

Operators SHOULD publish documentation describing service behavior, retention policies, rate limits, and operational practices. For the fDKA, such transparency is especially important due to its role as a shared infrastructure service.

13. IANA Considerations

13.1. 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

13.2. fDKA Domain

This document requests designation of a well-known domain for the Fallback DKA under IANA governance. The operational model for the fDKA is that of a community-operated infrastructure service: IANA designates the domain name, and the IETF community determines the operational arrangements through the normal IETF process, similar to other infrastructure services under the .arpa top-level domain.

The recommended designation is fdka.arpa. The corresponding DKA Locator Record would be:

```
_dka.fdka.arpa.  IN  TXT  "v=DKA1;dka=dka.fdka.arpa"
```

13.3. 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 the additional domain-consistency checks defined by this specification	This document

Table 2

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

14. References

14.1. 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>.

14.2. 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>.

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

[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>.

[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. Complete Example

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

A.1. DNS Discovery

A Requesting Client seeks the public key for alice@example.com. It queries DNS:

```
_dka.example.com.  IN  TXT  "v=DKA1;dka=dka.example.com"
```

The client parses the record and determines that the DKA for example.com is located at dka.example.com.

A.2. Key Registration

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

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

A.3. Key Lookup --- Domain DKA

The Requesting Client sends a lookup request:

```
GET https://dka.example.com/.well-known/dka/lookup
    ?email_address=alice%example.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, a conforming client does not query the fDKA for (alice@example.com, "default").

A.4. Key Lookup --- Fallback DKA

A Gmail user, bob21@gmail.com, has registered a key with the fDKA at dka.fdka.arpa. A Requesting Client seeking bob21@gmail.com's key:

1. Queries _dka.gmail.com --- no DKA Locator Record found.

2. Since no domain DKA exists, queries the fDKA:

```
GET https://dka.fdka.arpa/.well-known/dka/lookup
    ?email_address=bob21%40gmail.com&selector=default
```

3. The fDKA returns:


```
{
  "email_address": "bob21@gmail.com",
  "selector": "default",
  "public_key": "LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0t...",
  "verification_methods": ["mailbox-control", "dkim-validation"],
  "metadata": {},
  "version": 1,
  "updated_at": "2026-03-31T19:17:13+00:00"
}
```


This example demonstrates the fDKA's role as a bootstrap mechanism: a Gmail user participates in the framework without any action by Google. If Google later deploys a DKA for gmail.com, the domain DKA would take priority for any (identifier, selector) pairs it serves, while the fDKA would continue to serve pairs not present at Google's DKA.

Appendix B. Reference Implementation

An open-source reference implementation of the DKA framework is available. The implementation serves as both a domain DKA and an fDKA, with the operational difference controlled by a configuration parameter specifying whether the DKA restricts registrations to a specific domain or accepts registrations from any domain.

Appendix C. Working Demonstration

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

* https://keymail1.com/demo

Author’s Address

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