Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 17 April 2026

R. Clayton
Yahoo
W. Chuang
Google
B. Gondwana
Fastmail Pty Ltd

14 October 2025

DomainKeys Identified Mail Signatures v2 (DKIM2)
draft-clayton-dkim2-spec-01

Abstract

DomainKeys Identified Mail v2 (DKIM2) permits a person, role, or organization that owns a signing domain to document that it has handled an email message by associating their domain with the message. This is achieved by applying a cryptographic signature to the message. Verification is performed by querying an entry within the signing domain's DNS space to retrieve an appropriate public key. As a message is transferred from author to recipient further signatures will be added to provide a validatable "chain". This permits validators to identify when messages have been unexpectedly "replayed" and can ensure that delivery status notifications are only sent to entities that were involved in the transmission of a message.

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 17 April 2026.

Table of Contents

1. Introduction

DomainKeys Identified Mail v2 (DKIM2) permits a person, role, or organization to document that they have handled an email message by associating a domain name [RFC1034] with the message [RFC5322]. A public key signature is used to record that they have been able to read the contents of the message and write to it.

Verification of claims is achieved by fetching a public key stored in the DNS under the relevant domain and then checking the signature.

Message transit from author to recipient is through Forwarders that typically make no substantive change to the message content and thus preserve the DKIM2 signature. Where they do make a change the changes they have made are documented so that these can be "undone" and the original signature validated.

When a message is forwarded from one system to another an additional DKIM2 signature is added on each occasion. This chain of custody assists validators in distinguishing between messages that were intended to be sent to a particular email address and those that are being "replayed" to that address.

The chain of custody can also be used to ensure that delivery status notifications are only sent to entities that were involved in the transmission of a message.

Organizations that process a message can add to their signature a request for feedback as to any opinion (for example, that it was considered to be spam) that the eventual recipient of the message wishes to share.

1.1. DKIM2 Architecture Documents

Readers are advised to be familiar with the material in TBA, TBA and TBA which provide the background for the development of DKIM2, an overview of the service, and deployment and operations guidance and advice.

2. Terminology and Definitions

This section defines terms used in the rest of the document.

DKIM2 is designed to operate within the Internet Mail service, as defined in [RFC5598]. Basic email terminology is taken from that specification.

DKIM2 inherits many ideas from DKIM ([RFC6376]) which, for clarity we refer to in this specification as DKIM1.

Syntax descriptions use Augmented BNF (ABNF) [RFC5234].

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 [RFC2119]. These words take their normative meanings only when they are presented in ALL UPPERCASE.

2.1. Forwarder

[RFC5598] defines a Relay as transmitting or retransmitting a message but states that it will not modify the envelope information or the message content semantics. It also defines a Gateway as a hybrid of User and Relay that connects heterogeneous mail services, In this document we use the concept of a Forwarder which is an MTA that receives a message and then either delivers it into a destination mailbox or forwards it on to another system in an automated, pre-determined, manner.

As will be seen, Forwarders may alter message content or envelopes but will create a signed record of what they have done.

2.2. Signers

Elements in the mail system that sign messages on behalf of a domain are referred to as Signers. These may be MUAs (Mail User Agents), MSAs (Mail Submission Agents), MTAs (Mail Transfer Agents), or other agents such as mailing list "exploders". In general, any Signer will be involved in the injection of a message into the message system in some way. The key point is that a message must be signed before it

leaves the administrative domain of the Signer.

2.3. Verifiers

Elements in the mail system that verify signatures are referred to as Verifiers. These may be Forwarders, MTAs, Mail Delivery Agents (MDAs), or MUAs. It is an expectation of DKIM2 that a recipient of a message will wish to verify some or all signatures before determining whether or not to accept the message or pass it on to another entity.

2.4. Signing Domain

A domain name associated with a signature. This domain may be associated with the author of an email, their organization, a company hired to deliver the email, a mailing list operator, or some other entity that handles email. What they have in common is that at some point they had access to the entire contents of the email and were in a position to add their signature to the email.

2.5. Whitespace

There are two forms of whitespace used in this specification:

* WSP represents simple whitespace, i.e., a space or a tab character (formal definition in [RFC5234]).

* FWS is folding whitespace. It allows multiple lines separated by CRLF followed by at least one whitespace, to be joined.

The formal ABNF for these are (WSP given for information only):

```
WSP =   SP / HTAB
FWS =   [*WSP CRLF] 1*WSP
```

The definition of FWS is identical to that in [RFC5322] except for the exclusion of obs-FWS.

2.6. Imported ABNF Tokens

The following tokens are imported from other RFCs as noted. Those RFCs should be considered definitive.

The following tokens are imported from [RFC5321]:

* "local-part" (implementation warning: this permits quoted strings)

* "sub-domain"

The following tokens are imported from [RFC5322]:

* "field-name" (name of a header field)

Other tokens not defined herein are imported from [RFC5234]. These are intuitive primitives such as SP, HTAB, WSP, ALPHA, DIGIT, CRLF, etc.

2.7. Common ABNF Tokens

The following ABNF tokens are used elsewhere in this document:

```
ALPHADIGITPS =  (ALPHA / DIGIT / "+" / "/")
base64string =  ALPHADIGITPS *([FWS] ALPHADIGITPS)
                [ [FWS] "=" [ [FWS] "=" ] ]
```

3. Protocol Elements

Protocol Elements are conceptual parts of the protocol that are not specific to either Signers or Verifiers. The protocol descriptions for Signers and Verifiers are described in later sections ("Signer Actions" (Section 6) and "Verifier Actions" (Section 7). NOTE: This section must be read in the context of those sections.

3.1. Selectors

To support multiple concurrent public keys per signing domain, the key namespace is subdivided using "selectors".

Periods are allowed in selectors and are component separators. Periods in selectors define DNS label boundaries in a manner similar to the conventional use in domain names. This will allow portions of the selector namespace to be delegated.

ABNF:

```
selector = sub-domain *( "." sub-domain )
```

The number of public keys and corresponding selectors for each domain is determined by the domain owner. Many domain owners will be satisfied with just one selector, whereas administratively distributed organizations can choose to manage disparate selectors and key pairs in different regions or on different email servers.

Beyond administrative convenience, selectors make it possible to seamlessly replace public keys on a routine basis. If a domain wishes to change from using a public key associated with selector "january2026" to a public key associated with selector

"february2026", it merely makes sure that both public keys are advertised in the public key repository concurrently for the transition period during which email may be in transit prior to verification. At the start of the transition period, the outbound email servers are configured to sign with the "february2026" private key. At the end of the transition period, the "january2026" public key is removed from the public key repository.

INFORMATIVE NOTE: A key may also be revoked as described below. The distinction between revoking and removing a key selector record is subtle. When phasing out keys as described above, a signing domain would probably simply remove the key record after the transition period. However, a signing domain could elect to revoke the key (but maintain the key record) for a further period. There is no defined semantic difference between a revoked key and a removed key.

3.2. Tag=Value Lists

DKIM2 uses a simple "tag=value" syntax in several contexts, including in messages and domain signature records (see [DKIMKEYS]).

Values are a series of strings containing either plain text or "base64" text (as defined in [RFC2045], Section 6.8). The name of the tag will determine the encoding of each value. Unencoded semicolon (";") characters MUST NOT occur in the tag value, since that separates tag-specs.

INFORMATIVE IMPLEMENTATION NOTE: Although the "plain text" defined below (as "tag-value") only includes 7-bit characters, an implementation that wished to anticipate future standards would be advised not to preclude the use of UTF-8-encoded ([RFC3629]) text in tag=value lists.

Formally, the ABNF syntax rules are as follows:

```
tag-list  =  tag-spec *( ";" tag-spec ) [ ";" ]
tag-spec  =  [FWS] tag-name [FWS] "=" [FWS] tag-value [FWS]
tag-name  =  ALPHA *ALNUMPUNC
tag-value =  [ tval *( 1*(WSP / FWS) tval ) ]
                  ; Prohibits WSP and FWS at beginning and end
tval      =  1*VALCHAR
VALCHAR   =  %x21-3A / %x3C-7E
                  ; EXCLAMATION to TILDE except SEMICOLON
ALNUMPUNC =  ALPHA / DIGIT / "_"
```

Note that WSP is allowed anywhere around tags. In particular, any WSP after the "=" and any WSP before the terminating ";" is not part of the value; however, WSP inside the value is significant.

Tags MUST be interpreted in a case-sensitive manner. Values MUST be processed as case sensitive unless the specific tag description of semantics specifies case insensitivity.

Tags with duplicate names MUST NOT occur within a single tag-list; if a tag name does occur more than once, the entire tag-list is invalid.

Whitespace within a value MUST be retained unless explicitly excluded by the specific tag description.

Tag=value pairs that represent the default value MAY be included to aid legibility.

Unrecognized tags MUST be ignored.

Tags that have an empty value are not the same as omitted tags. An omitted tag is treated as having the default value; a tag with an empty value explicitly designates the empty string as the value.

4. Signing and Verification Cryptographic Algorithms

DKIM2 supports multiple digital signature algorithms. Two algorithms are defined by this specification: RSA-SHA256 and Ed25519-SHA256. Signers SHOULD implement both RSA-SHA256 and Ed25519-SHA256. Verifiers MUST implement both RSA-SHA256 and Ed25519-SHA256.

4.1. Key Management

Some level of assurance is required that a public key is associated with the claimed Signer. DKIM2 does this by fetching the key from the DNS for the domain specified in the d= field.

DKIM2 keys are stored in a subdomain named "_domainkey". Given a DKIM2-Signature field with a "d=" tag of "example.com" and an "s1=" tag of "foo.bar", the DNS query will be for "foo.bar._domainkey.example.com".

NOTE: these keys are no different, and are stored in the same locations as those for DKIM1 ([RFC6376]).

Further details can be found in [DKIMKEYS].

4.2. The RSA-SHA256 Signing Algorithm

The RSA-SHA256 Signing Algorithm computes a message hash as described in Section 8 using SHA-256 (FIPS-180-4-2015) as the hash-alg. That hash is then signed by the Signer using the RSA algorithm (defined in PKCS#1 version 1.5 [RFC8017]) as the crypt-alg and the Signer's private key. The hash MUST NOT be truncated or converted into any form other than the native binary form before being signed. The signing algorithm SHOULD use a public exponent of 65537.

Signers MUST use RSA keys of at least 1024 bits. Verifiers MUST be able to validate signatures with keys ranging from 1024 bits to 2048 bits, and they MAY be able to validate signatures with larger keys.

4.3. The Ed25519-SHA256 Signing Algorithm

The Ed25519-SHA256 signing algorithm computes a message hash as defined in Section 3 of [RFC6376] using SHA-256 (FIPS-180-4-2015) as the hash-alg. It signs the hash with the PureEdDSA variant Ed25519, as defined in Section 5.1 of [RFC8032].

4.4. Other Algorithms

Other algorithms MAY be defined in the future. Verifiers MUST ignore any signatures using algorithms that they do not implement.

4.5. Semantics of Multiple Signatures

In DKIM2 a new signature (with an incremented i= value) is added by every MTA that handles a message. However, if these MTAs are handling email within an organization then it is only necessary for the MTA that sends the email outside of that organization to apply a DKIM2 signature.

Verifiers may check the entire chain of signatures to establish that there is a valid chain from start to finish; however where the chain shows that the email has not been modified during its travels it will generally be sufficient to check only the first and last signature.

When there is evidence of "DKIM replay" inspection of the entire chain will enable the location of the replay to be determined and appropriate action can then be taken.

Further information about multiple signatures can be found in TBA.

5. The DKIM2-Signature Header Field

The signature of the email is stored in a DKIM2-Signature header field. This header field contains all of the signature and key-fetching data. The DKIM2-Signature value is a tag-list as described in Section 3.2.

Tags on the DKIM2-Signature header field along with their type, encoding and requirement status are shown below. It will be noted that we have not included a version number. Experience from IMF onwards shows that it is essentially impossible to change version numbers. If it becomes necessary to change DKIM2 in the sort of incompatible way that a v=2 / v=3 version number would support, it is expected that header fields will be labelled as DKIM3 instead.

i= The sequence number of the DKIM2-Signature header field.

plain-text unsigned decimal integer; REQUIRED

The originator of a message uses the value 1. Further DKIM2-Signature header fields are added with a value one more than the current highest numbered DKIM2-Signature header field. Gaps in the numbering MUST be treated as making the whole message unsigned.

ABNF:

```
sig-i-tag    = %x69 [FWS] "=" [FWS] 1*2DIGIT
```

n= Nonce value

plain-text unsigned decimal integer; OPTIONAL

This value, if present, has a meaning to the creator of the signature but MUST NOT be assumed to have any meaning to any other entity. It MAY be used as an index into a database to assist in handling Delivery Status Notifications or for any other purpose.

ABNF:

```
sig-n-tag    = %x6e [FWS] "=" [FWS] 1*(DIGIT)
```

t= Signature Timestamp

plain-text unsigned decimal integer; REQUIRED

The time that this header field was created. The format is the number of seconds since 1970-01-01T00:00:00 UTC. This value is not constrained to fit into a 31- or 32-bit integer. Implementations

SHOULD be prepared to handle values up to at least 10^12 (until approximately AD 200,000; this fits into 40 bits). Implementations MAY ignore signatures that have a timestamp in the future. Implementations MAY ignore signatures that are more than 14 days old.

ABNF:

```
sig-t-tag    = %x74 [FWS] "=" [FWS] 1*12DIGIT
```

mf= The [RFC5321] MAIL FROM value to be used when this message is transmitted over an SMTP link from the signing MTA.

plain-text; REQUIRED

Note that MAIL FROM may be just "<>", for example for a Delivery Status Notification.

ABNF:

```
sig-mf-tag = %x6d %65 [FWS] "="
             [FWS] "<" [ local-part "@" domain-name ] ">"
```

rt= The [RFC5321] RCPT TO value(s) to be used when this message is transmitted over an SMTP link from the signing MTA.

plain-text; REQUIRED

When a message is intended for more than one recipient then this field MAY include all of the recipients so that a single copy of the email MAY be sent to all of the recipients in a single SMTP transaction. Note that if "bcc:" recipients are involved then in order to meet the requirements of [RFC5322] Section 3.6.3 each bcc recipient MUST be sent a message with just their email address appearing in this tag.

ABNF:

```
sig-rt-tag = %72 %x74 [FWS] "="
             1*( [FWS] "<" local-part "@" domain-name ">" )
```

d= The domain associated with this signature.

plain-text; REQUIRED

This domain is used to form the query for the public key. The domain MUST be a valid DNS name under which the DKIM2 key record is published. Internationalized domain names MUST be encoded as A-labels, as described in Section 2.3 of [RFC5890].

The domain in the d= tag MUST exactly match the rightmost labels of the domain-name part of the mf= tag. That is to say, the domain-name of the mf= tag MUST either match the d= domain exactly or be a sub-domain of d= domain.

ABNF:

```
sig-d-tag   = %x64 [FWS] "=" [FWS] domain-name
domain-name = sub-domain 1*("." sub-domain)
                          ; from [RFC5321] Domain,
                          ; excluding address-literal
```

s1= The selector subdividing the namespace for the "d=" (domain) tag.

plain-text; REQUIRED

Internationalized selector names MUST be encoded as A-labels, as described in Section 2.3 of [RFC5890].

ABNF:

```
sig-s-tag = %x73 %x31 [FWS] "=" [FWS] selector
```

a1= The algorithm used to generate the signature.

plain-text; REQUIRED

Verifiers MUST support "rsa-sha256" and "ed25519"; See Section 4 for a description of the algorithms.

ABNF:

```
sig-a-tag     = %x61 %x31 [FWS] "=" [FWS] sig-a-tag-alg
sig-a-tag-alg = sig-a-tag-k "-" sig-a-tag-h
sig-a-tag-k   = "rsa" / "ed25519" / x-sig-a-tag-k
sig-a-tag-h   = "sha256" / x-sig-a-tag-h
x-sig-a-tag-k = ALPHA *(ALPHA / DIGIT)
                          ; for later extension
x-sig-a-tag-h = ALPHA *(ALPHA / DIGIT)
                          ; for later extension
```

b1= The signature data.

base64; REQUIRED

Whitespace is ignored in this value and MUST be ignored when reassembling the original signature. In particular, the signing process can safely insert FWS in this value in arbitrary places to conform to line-length limits. See "Signer Actions" (Section 6) for how the signature is computed.

ABNF:

```
sig-b-tag      = %x62 %x31 [FWS] "=" [FWS] sig-b-tag-data
sig-b-tag-data = base64string
```

bh1= The hash of the canonicalized body part of the message.

base64; REQUIRED

Whitespace is ignored in this value and MUST be ignored when reassembling the original signature. In particular, the signing process can safely insert FWS in this value in arbitrary places to conform to line-length limits. See Section 8 for how the body hash is computed.

ABNF:

```
sig-bh-tag      = %x62 %x68 %x31 [FWS] "=" [FWS] sig-bh-tag-data
sig-bh-tag-data = base64string
```

s2=, a2= b2= bh2= Further signatures (equivalent to s1, a1, b2 and bh1)

plain text / base64; OPTIONAL

To provide for algorithmic dexterity a second signature, using a different algorithm MAY be supplied. A verifier MUST check all signatures that it understands and SHOULD treat any failure as invalidating all signatures.

f= Flags

plain text; OPTIONAL

Flags serve two purposes; they either report what has been done to the message by the system creating the DKIM2-Signature or they make a request to systems that handle the mail thereafter. Flags are separated by commas, and optional white-space allows systems to add several flags without creating long lines.

If a flag value is not recognised it MUST be ignored.

The flag values that report things are:

"exploded": this message (identified by its unique header hash value (recorded in b1= and perhaps b2=)) is being sent to more than one email address. An MTA which receives a message MAY use this information to help it distinguish between malicious "DKIM replay" and legitimate activity performed by mailing list. If this flag is not present in at least one DKIM2-Signature header field then an MTA MAY assume that only one copy of a particular message (identified by relevant cryptographic hash values) is intended to exist;

"modifiedbody": the body of the message has been altered in some way. A corresponding DKIM2-Delta-Body MUST be present with the same i= value as defined in [ALGEBRA]. This flag MUST NOT be present when i=1;

"modifiedheader": the header of the message has been altered in some way. A corresponding DKIM2-Delta-Header must be present with the same i= value as defined in [ALGEBRA]. This flag MUST NOT be present when i=1;

The flags values that make requests are:

"donotexplode": this signer requests that the message not be sent to more than one recipient. A system that, by local policy, ignores this request MUST NOT allow any of the copies it creates to be forwarded on to any MTA outside its control.

"donotmodify": this signer requests that the message not be modified from the form in which it is sent. A system that, by local policy, ignores this request MUST NOT allow the message to be forwarded on to any MTA outside its control.

"feedback": this signer requests feedback about how this message is handled during delivery and thereafter. This document does not describe what such feedback might be or where it might be delivered. If this flag is absent then feedback is explicitly not required.

ABNF:

```
 sig-f-tag = %x66 [FWS] "=" [FWS] sig-f-data *("," [FWS] sig-f-tag-data)
 sig-f-tag-data   = "modifiedbody" | "modifiedheader" | "exploded" |
                    "donotmodify" | "donotexplode" | "feedback"
 x-sig-f-tag-data = ALPHA *(ALPHA / DIGIT)
                          ; for later extension
```

6. Signer Actions

The following steps are performed in order by Signers.

6.1. Document any modifications made to a message

MTAs that accept a DKIM2 signed message and send it onward to another MTA MUST record the changes that they have made to the message. A scheme for generating appropriate DKIM2-Delta-Header header fields to describe such modifications can be found in [ALGEBRA].

Note in particular that adding header fields must be documented, although some header fields (such as Received:) are not signed. Details can be found in {#computing-hashes}.

Failure to record modifications will mean that an MTA that subsequently handles the message SHOULD detect this and this MAY lead to the message being rejected.

6.2. Determine what [RFC5321] "envelope" will be used for the message

The DKIM2-Signature field contains mf= and rt= values, so the MAIL FROM and RCPT TO values that will be used when the message is transmitted MUST be available to (or deducible by) a Signer.

When the message being signed already has a DKIM2-Signature header field (i.e. it has already been transmitted at least once) then the mf= of the message to be signed MUST match with an entry in the rt= of currently highest numbered (i=) DKIM2-Signature header field.

The match algorithm only considers the domain part of the rt= and mf= values, that is the local part is ignored. If there is not an exact match then labels are removed, one by one from the left hand side of the mf= domain and compared with the rt= domain until there is an exact match or no labels remain. This approach allows systems to use existing "bounce-handling" schemes with special subdomains in their MAIL FROM values.

If there will not be a match between the rt= and mf= values then the Signer MUST generate an extra DKIM2-Signature field that causes values to match, i.e. a record of the mail being passed from one domain to another. It will be noted that the creation of this extra header field will require the Signer to have access to the DKIM2 key value associated with a domain in the rt= entry.

6.3. Select a Private Key and corresponding Selector value

This specification does not define the basis by which a Signer should choose which private key and selector value to use. Currently, all selectors are equal as far as this specification is concerned, so the decision should largely be a matter of administrative convenience. Distribution and management of private keys is also outside the scope of this document.

6.4. Normalize the Message to Prevent Transport Conversions

In order to be signed a message will need to be in "network normal" format (text is ASCII encoded, lines are separated with CRLF characters, etc.).

A message that is not compliant with [RFC5322], [RFC2045], and [RFC2047] can be subject to attempts by intermediaries to correct or interpret such content. See Section 8 of [RFC6409] for examples of changes that are commonly made. Such "corrections" may invalidate DKIM2 signatures or have other undesirable effects, including some that involve changes to the way a message is presented to an end user.

Accordingly, DKIM2’s design is predicated on valid input. Therefore, Signers and Verifiers SHOULD take reasonable steps to ensure that the messages they are processing are valid according to [RFC5322], [RFC2045], and any other relevant message format standards.

In particular, messages using 8-bit characters, are subject to modification during transit, notably conversion to 7-bit form. Such conversions will break DKIM2 signatures. Hence the message SHOULD be converted to only contain 7-bit characters, for example by employing a suitable MIME content-transfer encoding such as quoted-printable or base64 as described in [RFC2045]. The way in which this is achieved is outside the scope of this specification.

If the message contains local encoding that will be modified before transmission, that modification to canonical [RFC5322] form MUST be done before signing. In particular, bare CR or LF characters (used by some systems as a local line separator convention) MUST be converted to the SMTP-standard CRLF sequence before the message is signed. Any conversion of this sort SHOULD be applied to the message actually sent to the recipient(s), not just to the version presented to the signing algorithm.

More generally, the Signer MUST sign the message as it is expected to be received by the Verifier rather than in some local or internal form.

6.5. Compute the Message Hash and Signature{#signer-compute}

The Signer MUST compute the message hash as described in Section 8 and then sign it using the selected public key algorithm. This will result in a DKIM2-Signature header field that will include the body hash and a signature of the header hash, where that header includes the DKIM2-Signature header field itself.

6.6. Insert the DKIM2-Signature Header Field

Finally, the Signer MUST insert the DKIM2-Signature header field created in the previous step prior to transmitting the email. The DKIM2-Signature header field MUST be the same as used to compute the hash as described above, except that the value of the "b1=" tag (and "b2=" if present) MUST be the appropriately signed hash computed in the previous step, signed using the algorithm specified in the "a1=" (or "a2=") tag of the DKIM2- Signature header field and using the private key corresponding to the selector given in the "s1=" (or "s2=") tag of the DKIM2-Signature header field, as chosen above in Section 6.3}.

The DKIM2-Signature header field SHOULD be inserted before any other DKIM2-Signature fields in the header block.

INFORMATIVE IMPLEMENTATION NOTE: The easiest way to achieve this is to insert the DKIM2-Signature header field at the beginning of the header block before any existing Received header fields.

7. Verifier Actions

A border or intermediate MTA MAY verify the message signature(s). An MTA that has performed verification MAY communicate the result of that verification by adding a verification header field to incoming messages. This simplifies things considerably for the user, who can now use an existing mail user agent. Most MUAs have the ability to filter messages based on message header fields or content; these filters would be used to implement whatever policy the user wishes with respect to unsigned mail.

A verifying MTA MAY implement a policy with respect to unverifiable mail, regardless of whether or not it applies the verification header field to signed messages.

Verifiers MUST produce a result that is semantically equivalent to applying the steps listed in Section 7.1, Section 7.1.1, and Section 7.1.2 in order. In practice, several of these steps can be performed in parallel in order to improve performance.

7.1. Extract Signatures from the Message

A Verifier SHOULD check the most recently applied (highest numbered i= value) DKIM2-Signature before accepting an email. If this check does not pass then a Delivery Status Notification for the email MUST NOT be generated thereafter -- hence the best strategy, if the email is not wanted, is to reject it (with a 5xx error code) whilst the relevant SMTP conversation is still ongoing.

A Verifier that wishes to ascertain whether or not a message has been modified since it was first created need only check the first (i=1) DKIM2-Signature.

If the message has been modified since its original creation then the DKIM2-Modification header fields (see [ALGEBRA]) will enable a Verifier to determine whether or not all the changes made are correctly recorded.

For each signature to be validated, the following steps should be performed in such a manner as to produce a result that is semantically equivalent to performing them in the indicated order.

Where the status is TEMPFAIL then it may be possible for the Verifier to determine the validity of the signature at a later time. If the SMTP conversation is still ongoing then a 4xx error code SHOULD be used to allow the sending MTA to understand the situation.

7.1.1. Validate the Signature Header Field

Implementers MUST meticulously validate the format and values in the DKIM2-Signature header field; any inconsistency or unexpected values MUST cause the header field to be completely ignored and the Verifier to return PERMFAIL (signature syntax error). Being "liberal in what you accept" is definitely a bad strategy in this security context.

Note, however, that this does not include the existence of unknown tags in a DKIM2-Signature header field, which are explicitly permitted.

Verifiers MAY return PERMFAIL (signature expired) if it is more than 14 days since the timestamp recorded in the "t=" tag.

7.1.2. Get the Public Key

The public key for a signature is needed to complete the verification process. Details of key management and representation are described in Section 4.1. The Verifier MUST validate the key record and MUST ignore any public key records that are malformed.

NOTE: The use of a wildcard TXT RR that covers a queried DKIM domain name will produce a response to a DKIM query that is unlikely to be a valid DKIM key record. This problem is not specific to DKIM and applies to many other types of queries. Client software that processes DNS responses needs to take this problem into account.

When validating a message, a Verifier MUST perform the following steps in a manner that is semantically the same as performing them in the order indicated; in some cases, the implementation may parallelize or reorder these steps, as long as the semantics remain unchanged:

1. The Verifier retrieves the public key as described in Section 6.3 using the algorithm in the "a=" tag, the domain from the "d=" tag, and the selector from the "s=" tag.

2. If the query for the public key fails to respond, the Verifier MAY seek a later verification attempt by returning TEMPFAIL (key unavailable).

3. If the query for the public key fails because the corresponding key record does not exist, the Verifier MUST return PERMFAIL (no key for signature).

4. If the query for the public key returns multiple key records, the Verifier can choose one of the key records or may cycle through the key records, performing the remainder of these steps on each record at the discretion of the implementer. The order of the key records is unspecified. If the Verifier chooses to cycle through the key records, then the "return ..." wording in the remainder of this section means "try the next key record, if any; if none, return to try another signature in the usual way".

5. If the result returned from the query does not adhere to the format defined in the DKIM key specification [DKIMKEYS], the Verifier MUST ignore the key record and return PERMFAIL (key syntax error). Verifiers are urged to validate the syntax of key records carefully to avoid attempted attacks. In particular, the Verifier MUST ignore keys with a version code ("v=" tag) that they do not implement.

6. If the public key data (the "p=" tag) is empty, then this key has been revoked and the Verifier MUST treat this as a failed signature check and return PERMFAIL (key revoked). There is no defined semantic difference between a key that has been revoked and a key record that has been removed.

7. If the public key data is not suitable for use with the algorithm and key types defined by the "a=" and "k=" tags in the DKIM-Signature header field, the Verifier MAY immediately return PERMFAIL (inappropriate key algorithm). However, these fields are now deprecated so DKIM2 implementations MAY ignore them altogether.

8. If the "h=" tag exists in the public key record and the hash algorithm implied by the "a=" tag in the DKIM2-Signature header field is not included in the contents of the "h=" tag, the Verifier MUST ignore the key record and return PERMFAIL (inappropriate hash algorithm).

7.1.3. Compute the Verification

Given a Signer and a public key, verifying a signature consists of actions semantically equivalent to the following steps.

1. Prepare a canonicalized version of the message as is described in Section 8 (note that this canonicalized version does not actually replace the original content).

2. Based on the algorithm indicated in the "a1=" tag, compute the message hashes from the canonical copy as described in Section 8.

3. Verify that the hash of the canonicalized message body computed in the previous step matches the hash value conveyed in the "bh=" tag. If the hash does not match, the Verifier SHOULD ignore the signature and return PERMFAIL (body hash did not verify).

4. Using the signature conveyed in the "b1=" tag, verify the signature against the header hash using the mechanism appropriate for the public key algorithm described in the "a1=" tag. If the signature does not validate, the Verifier SHOULD ignore the signature and return PERMFAIL (signature did not verify).

5. Otherwise, the signature has correctly verified.

7.2. Output Requirements

The evaluation of each signature ends in one of three states, which this document refers to as follows:

SUCCESS: a successful verification

PERMFAIL: a permanent, non-recoverable error such as a signature verification failure

TEMPFAIL: a temporary, recoverable error such as a DNS query timeout

For each signature that verifies successfully or produces a TEMPFAIL result, output of the DKIM2 algorithm MUST include the set of:

* The domain name, taken from the "d=" signature tag; and

* The result of the verification attempt for that signature.

The output MAY include other signature properties or result meta-data, including PERMFAILed or otherwise ignored signatures, for use by modules that consume those results.

See Section 7.1 for discussion of signature validation result codes.

7.3. Communicate Verification Results

Verifiers wishing to communicate the results of verification to other parts of the mail system may do so in whatever manner they see fit. For example, implementations might choose to add an email header field to the message before passing it on. Any such header field SHOULD be inserted before any existing DKIM2-Signature or pre-existing authentication status header fields in the header field block. The Authentication-Results: header field ([RFC8601]) MAY be used for this purpose.

7.4. Interpret Results/Apply Local Policy

It is beyond the scope of this specification to describe what actions the recipient of an email performs, but mail carrying valid DKIM2 signatures gives the recipient opportunities that unauthenticated email would not. Specifically, an authenticated email provides predictable information by which other decisions can reliably be managed, such as trust and reputation. Conversely, it is hard to assign trust or reputation to unauthenticated email.

In general, modules that consume DKIM2 verification results SHOULD NOT determine message acceptability based solely on a lack of any signature or on an unverifiable signature; such rejection would cause severe interoperability problems. If an MTA does wish to reject such messages during an SMTP session (for example, when communicating with a peer who, by prior agreement, agrees to only send signed messages), and a signature is missing or does not verify, the handling MTA SHOULD use a 550/5.7.x reply code.

Where the Verifier is integrated within the MTA and it is not possible to fetch the public key, perhaps because the key server is not available, a temporary failure message MAY be generated using a 451/4.7.5 reply code, such as:

451 4.7.5 Unable to verify signature - key server unavailable

Temporary failures such as inability to access the key server or other external service are the only conditions that SHOULD use a 4xx SMTP reply code. In particular, cryptographic signature verification failures MUST NOT provoke 4xx SMTP replies.

If the email cannot be verified, then it SHOULD be treated the same as all unverified email, regardless of whether or not it looks like it was signed.

8. Computing the Message Hashes

Both signing and verifying message signatures start by computing cryptographic hashes over the body and then over selected header fields message. Signers will choose the parameters of the signature as described in "Signer Actions" (Section 6); Verifiers will use the parameters specified in the DKIM2-Signature header field being verified. In the following discussion, the names of the tags in the DKIM2-Signature header field that either exist (when verifying) or will be created (when signing) are used.

Some mail systems modify email in transit, potentially invalidating a signature. DKIM2 provides a method of documenting such changes as they occur. However, in order to reduce the necessity to document minor changes to header fields DKIM2 uses an algorithm for computing header hashes which permits minor changes. This is identical to the "relaxed" algorithm of DKIM1 ([RFC6376]). Minor changes to

Some systems make minor changes to white space and line endings within messages. To avoid having to document such changes DKIM2 canonicalises the body before hashing using a scheme which is identical to the DKIM1 "relaxed" algorithm.

NOTE: canonicalization simply prepares the email for presentation to the signing or verification algorithm. It MUST NOT change the transmitted data in any way.

When calculating the hash on messages that will be transmitted using base64 or quoted-printable encoding, Signers MUST compute the hash after the encoding. Likewise, the Verifier MUST incorporate the values into the hash before decoding the base64 or quoted-printable text. However, the hash MUST be computed before transport-level

encodings such as SMTP "dot-stuffing" (the modification of lines beginning with a "." to avoid confusion with the SMTP end-of-message marker, as specified in [RFC5321]).

For clarity this section will discuss the first pair of signatures (s1=, a1=, h1=, bh1). If a second pair (s2=, a2=, h2=, bh2=) is present then that is calculated or verified in an identical manner except using the "2" values rather than the "1" values.

8.1. Computing the Body Hash

The body hash MUST be computed first by a signer because its value will be included in the subsequent hash of the header fields. A Verifier MAY check the hashes in any convenient order.

The body of messages is treated as merely a string of octets. DKIM2 messages MAY be either in plain-text or in MIME format; no special treatment is afforded to MIME content. Message attachments in MIME format MUST be included in the content that is signed.

The body canonicalization algorithm MUST apply the following steps in order:

* Ignore all whitespace at the end of lines. Implementations MUST NOT remove the CRLF at the end of the line.

* Reduce all sequences of WSP within a line to a single SP character.

* Ignore all empty lines at the end of the message body. An empty line is a line of zero length after removal of the line terminator. If there is no body or no trailing CRLF on the message body, a CRLF is added.

Note that a completely empty or missing body is canonicalized as a single "CRLF"; that is, the canonicalized length will be 2 octets.

The canonicalized version of the body is then hashed by the relevant algorithm(s). The value is converted to base64 form and inserted into (Signers) or compared to (Verifiers) the "bh1=" tag of the DKIM-Signature header field that is being created.

8.2. Construct appropriate DKIM2 header fields

A signer will need to provide appropriate DKIM2-Delta-Header fields if the message arrived with a DKIM2 signature and changes have been made to the message.

A DKIM2-Signature header field should then be constructed. The bh= value(s) should be filled in with the body hash value(s) that have been calculated as above. The value of the "b=" tag(s) (including all surrounding whitespace) should be omitted.

The i= values for these header fields MUST be one greater that the highest i= value in any existing DKIM2-Signature header field, or if there is no such header field then i=1 MUST be used.

Unlike DKIM1 these DKIM2 header fields are terminated by a CRLF.

8.3. Computing the Header Hash

The header hash algorithm signing MUST apply the following steps in the order given. A verifier will need to do the equivalent steps in order to check that the hash they have received is correct.

* Construct a DKIM2-Signature header field with the

* Ignore header fields that are never signed:

  DKIM2 implementations MUST NOT sign "Received" or "Return-Path" header fields. These are Trace headers as described in [RFC5321] and serve only to document details of the SMTP transmission process.

  DKIM2 implementations MUST NOT sign "DKIM-Signature" header fields or any header field whose name starts with "ARC". Not over-signing DKIM1 and ARC signatures means that systems that wish to add other types of signature are free to do this in any convenient order.

  DKIM2 implementations MUST NOT sign any header field whose name starts with "X-". Currently deployed email systems use these fields as proprietary Trace headers and it considerably simplifies reporting on changes to header fields not to sign them.

* Convert all header field names (not the header field values) to lowercase. For example, convert "SUBJect: AbC" to "subject: AbC".

* Unfold all header field continuation lines as described in [RFC5322]; in particular, lines with terminators embedded in continued header field values (that is, CRLF sequences followed by WSP) MUST be interpreted without the CRLF. Implementations MUST NOT remove the CRLF at the end of the header field value.

* Convert all sequences of one or more WSP characters to a single SP character. WSP characters here include those before and after a line folding boundary.

* Delete all WSP characters at the end of each unfolded header field value.

* Delete any WSP characters remaining before and after the colon separating the header field name from the header field value. The colon separator MUST be retained.

* Place the header fields in alphabetical order by the header field name, except for any header fields that start "DKIM2" which are ordered specially.

* If there is more than one header with the same header field name then the header fields are placed in the order in which they occur in the email header

  It is sometimes suggested that some MTAs re-order header fields after they receive an email, if they do then it is their responsibility to recover the original order of any header fields with identical header field names (that are part of a signature calculation) before verifying an existing signature or passing a previously signed message to another MTA that is going to wish to verify a signature.

* The DKIM2 header fields are placed at the end of the list of header fields to be signed, ordered first by their "i=" value (in ascending numerical order) and then by the alphabetical order of their header field name.

  NOTE: the special rules for ordering DKIM2 header fields are intended to assist systems that wish to pre-calculate a hash value for all the other header fields and then finish off with the actual DKIM2 headers that they will be adding.

* The hash(es) of the concatenated header fields is calculated.

* The value(s) of the hash(es) are then converted to base64 and added to the b= tags.

9. IANA Considerations

TBA

10. Security Considerations

TBA

11. References

11.1. Normative References

[ALGEBRA] Gondwana, B., "A method for describing changes made to an email", Work in Progress, Internet-Draft, draft-gondwana-dkim2-modification-alegbra-03, 1 October 2025, <https://datatracker.ietf.org/doc/html/draft-gondwana-dkim2-modification-alegbra-03>.

[DKIMKEYS] Chuang, W., "Domain Name Specification for DKIM2", Work in Progress, Internet-Draft, draft-chuang-dkim2-dns-02, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-chuang-dkim2-dns-02>.

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

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <https://www.rfc-editor.org/rfc/rfc2045>.

[RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, DOI 10.17487/RFC2047, November 1996, <https://www.rfc-editor.org/rfc/rfc2047>.

[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/rfc/rfc2119>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/rfc/rfc3629>.

[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/rfc/rfc5234>.

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

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/rfc/rfc5322>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/rfc/rfc5890>.

[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/rfc/rfc6376>.

[RFC6409] Gellens, R. and J. Klensin, "Message Submission for Mail", STD 72, RFC 6409, DOI 10.17487/RFC6409, November 2011, <https://www.rfc-editor.org/rfc/rfc6409>.

[RFC8601] Kucherawy, M., "Message Header Field for Indicating Message Authentication Status", RFC 8601, DOI 10.17487/RFC8601, May 2019, <https://www.rfc-editor.org/rfc/rfc8601>.

11.2. Informative References

[RFC5598] Crocker, D., "Internet Mail Architecture", RFC 5598, DOI 10.17487/RFC5598, July 2009, <https://www.rfc-editor.org/rfc/rfc5598>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/rfc/rfc8017>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/rfc/rfc8032>.

Appendix A. Changes from Earlier Versions

draft-clayton-dkim2-spec-01

Significant re-ordering of sections and removal of repetitious material.

Relax the matching algorithm between rt= and mf=

[[This section to be removed by RFC Editor]]

Authors' Addresses

Richard Clayton
Yahoo
Email: rclayton@yahooinc.com

Wei Chuang
Google
Email: weihaw@google.com

Bron Gondwana
Fastmail Pty Ltd
Level 2, 114 William Street
3000
Australia
Phone: +61 457 416 436
Email: brong@fastmailteam.com