Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 27 June 2026

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

24 December 2025

DomainKeys Identified Mail Signatures v2 (DKIM2)
draft-clayton-dkim2-spec-05

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 providing a hash value that has been calculated on the current contents of the message and then applying a cryptographic signature that covers the hash values and other details about the transmission of 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 systems that alter the body or header fields will provide details of their changes and calculate new hash values. Further signatures will be added to provide a validatable "chain". This permits validators to identify the nature of changes made by intermediaries and apply a reputation to the systems that made changed. DKIM2 also allows recipients to detect when messages have been unexpectedly "replayed" and can also 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 will expire on 27 June 2026.

Copyright Notice

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 the email 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. In addition, some features were influenced by experience from (see [CONCLUDEARC]) the experimental ARC protocol ([RFC8617]).

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. 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.

2.2. 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.

2.3. Reviser

As will be seen, a Forwarder may alter the message content or header fields, in such a way that existing signatures on the message will no longer validate. If so, then a record will be made of these changes. We call a Forwarder that makes such changes a Reviser.

2.4. Verifiers

Elements in the mail system that verify signatures are referred to as Verifiers. These may be Forwarders, Revisers, 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.

2.5. 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.

2.6. 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.

2.7. 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.

2.8. Common ABNF Tokens

The following ABNF tokens are used elsewhere in this document:

```
VALCHAR       =  %x21-3A / %x3C-7E
                    ; EXCLAMATION to TILDE except SEMICOLON
ALPHADIGITPS =  (ALPHA / DIGIT / "+" / "/")
ALNUMPUNC     =  ALPHA / DIGIT / "_"

base64string =  ALPHADIGITPS *([FWS] ALPHADIGITPS)
                [ [FWS] "=" [ [FWS] "=" ] ]
textstring    =  VALCHAR * (FWS / VALCHAR)
```

Note that base64strings are defined in [RFC4648], but that document does not contain any ABNF. Note that a base64string MUST be padded with trailing = characters if needed.

Note that the definitions of both textstring and base64string allow for the presence of FWS, which simplifies folding header fields to an allowable line length. FWS within base64strings will be ignored when their value is being used. FWS within a textstring MUST be treated as a single space when this value is used.

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 10) and "Verifier Actions" (Section 11) and 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 use just one selector, whereas administratively distributed organizations can choose to manage disparate selectors and key pairs in different regions or on different email servers. Selectors can also be used to delegate a signing authority, which can be withdraw at any time. Selectors also make it possible to seamlessly replace keys on a routine basis by signing with a new selector, while keeping the key associated with the old selector available.

3.2. Tag=Value Lists

DKIM2 uses a simple "tag=value" syntax in the Message-Instance and DKIM2-Signature header fields, as well as in 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.

Formally, the ABNF syntax rules are as follows:

```
tag-list  =  tag-spec *( ";" tag-spec ) [ ";" ]
tag-spec  =  [FWS] tag-name [FWS] "=" [FWS] tag-value [FWS]
tval      =  1*VALCHAR
tag-name  =  ALPHA *ALNUMPUNC
tag-value =  [ tval *( 1*(WSP / FWS) tval ) ]
                  ; Prohibits WSP and FWS at beginning and end
```

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 hashing and digital signature algorithms. One hash function (SHA256) if specified here and two signing algorithms are defined by this specification: RSA-SHA256 and Ed25519-SHA256. Signers and Verifiers MUST implement SHA256. Signers SHOULD implement both RSA-SHA256 and Ed25519-SHA256. Verifiers MUST implement both RSA-SHA256 and Ed25519-SHA256.

4.1. The SHA256 Hashing Algorithm

The SHA256 hashing algorithm is used to compute body and header hashes as defined in Section 7 and Section 8. The resultant values are stored within Message-Instance header fields.

4.2. The RSA-SHA256 Signing Algorithm

The RSA-SHA256 Signing Algorithm computes a hash over all the Message-Instance and DKIM2-Signature header fields as described in Section 10.4 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 MUST 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 hash over all the Message-Instance and DKIM2-Signature fields as described in Section 10.4 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 hashes or signatures using algorithms that they do not implement.

4.5. 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].

5. The Message-Instance Header Field

The hash values for a message are stored in a Message-Instance header field. This header field documents the current contents of the message and, in the case of a Reviser records any relevant changes that have been made to the incoming message.

The Message-Instance header field is a tag-list as described in Section 3.2.

Tags on the Message-Instance header field along with their type, encoding and requirement status are shown below.

v= The revision number of the Message-Instance header field.

plain-text unsigned decimal integer; REQUIRED

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

ABNF:

```
mi-v-tag    = %x76 [FWS] "=" [FWS] 1*DIGIT
```

a1= The algorithm used to generate the body hash.

plain-text; REQUIRED

ABNF:

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

b1= 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 hash. In particular, FWS can be safely inserted this value in arbitrary places to conform to line-length limits. See Section 7 for how the body hash is computed.

ABNF:

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

h1= The hash of the canonicalized headers of the message.

base64; REQUIRED

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

ABNF:

```
mi-h-tag      = %x68 %x31 [FWS] "=" [FWS] mi-h-tag-data
mi-h-tag-data = base64string
```

a2=, b2=, h2= Further hashes (equivalent to a1, b1 and h1)

plain text / base64; OPTIONAL

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

r= A "recipe" to recreate the previous version of the message body.

plain text; OPTIONAL

A Body Recipe is a comma separated list of instructions. Each instruction starts with a prefix. Commas can be followed by optional whitespace.

The idea is that you take the message which has been received and apply the Body Recipes so as to recreate the message as it was before modifications were made. Hence it is necessary to cope with body lines being added (c: is used to indicate which lines were original) or removed/altered (b: is used to indicate what the body line was before the removal/alteration).

c: startnumber-endnumber

Copy the lines numbered startnumber up to and including the line numbered endnumber. The first line of the body (immediately after the blank line that indicates that there are no more header fields) is numbered 1. If the endnumber is omitted then all lines (from the startnumber line onwards) are to be copied.

The startnumber of every "c" instruction MUST be in ascending order and MUST be greater than the endnumber of all preceding "c" instructions. There MUST NOT be any further "c" instructions after a "c" instruction with an omitted endnumber. In other words a recipe is not allowed to use "c" instructions to duplicate or re-order lines.

b: base64string

Decode the base64string to get the value of a line to be inserted. The inserted line will have a CRLF appended. The decoded value MUST NOT contain CR or LF characters. If the base64string is absent then a blank line is being added.

z

If a z is present then it MUST be the only "recipe". It indicates that the changes that have been made to the body cannot be undone. For example, a malware attachment may have been removed and it is inappropriate to enable the malicious content to be recreated. Verifiers of the message may be able to inspect the first signer of this Message-Instance header field and determine that the presence of z is acceptable to them because, for example, that signer is providing a contractually arranged service.

ABNF:

```
mi-r-tag       = %x72 [FWS] "=" [FWS] mi-r-tag-data
mi-r-tag-data  = mi-r-recipes / %x7a
mi-r-recipes   = mi-r-recipe * ("," [FWS] mi-r-recipe)
mi-r-recipe    = %x63 ":" 1*DIGIT "-" [ 1*DIGIT ] /
                   %x62 ":" [ base64string ]
```

r_header= A "recipe" to replicate the previous version of a header field.

plain text; OPTIONAL

A Header Recipe is a comma separated list of instructions. Commas can be followed by optional whitespace. Each set of instructions refers to a particular header field name. There MUST be only one set of instructions for any given header field name

The idea is that you take the message which has been received and apply the Header Recipes so as to recreate the relevant header fields as they were before modifications were made. Hence it is necessary to cope with header fields being added (c: is used to indicate how many header fields, if any, should be kept so that the addition is undone) or removed/altered (b: is used to provide the contents of the the header field was before the removal/alteration).

Header fields are numbered "bottom up" (the opposite direction to the body lines). That is to say, when walking the header from top to bottom the last header field instance encountered with any particular header field name is numbered 1, the header field (with the same header field name) before that is numbered 2, and so on. The header fields should be treated as being unwrapped (in the normal [RFC5321] manner). That is, all of the physical lines that form a single header field are processed under the same logical number.

If there are no "recipes" for a specified header field name that means that all instances of that header field should be removed to reinstate the previous state of the message. If a header field name is not present at all then all of header fields with that header field name are to be retained.

c: startnumber-endnumber

Keep the instances of the header fields (with the relevant header field name) which are at position startnumber to endnumber.

The startnumber of every "c" instruction MUST be in ascending order and MUST be greater than
the endnumber of all preceding "c" instructions.

b: base64string

Decode the base64string value to get the header field to be inserted. The inserted header field will start with the field name and a colon followed by the decoded value and then a CRLF. When the base64string is decoded it MUST NOT contain CR or LF characters.
If the base64string is absent then the CRLF will immediately follow the header field name and colon.

Note that the hashing algorithm for processing the header fields will work on unwrapped lines -- so there is no need to wrap the header field created by this recipe because it will never appear "on the wire".

z

If a z is present then it MUST be the only "recipe". It indicates that the changes that have been made to the header field cannot be undone and/or that it is inappropriate to provide the original value.
Verifiers of the message may be able to inspect the first signer of this Message-Instance header field and determine that the presence of z is acceptable to them because, for example, that signer is providing a contractually arranged service.

ABNF:

```
mi-rh-tag       = %x72 "_" field-name [FWS] "=" [FWS] mi-rh-tag-data
mi-rh-tag-data  = mi-rh-recipes / %07A
mi-rh-recipes   = mi-rh-recipe * ("," [FWS] mi-rh-recipe)
mi-rh-recipe    = %x63 ":" 1*DIGIT /
                  %x62 ":" base64string
```

6. 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*DIGIT
```

v= The highest numbered Message-Instance header field

plain-text unsigned decimal integer; REQUIRED

This value allows verifiers to determine which entity made a particular revision to the message body or header fields.

ABNF:

```
sig-v-tag    = %x76 [FWS] "=" [FWS] 1*DIGIT
```

n= Nonce value

textstring; 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.

To discourage use of the tag field as an alternative to the use of more appropriate header fields, the length of the textstring MUST NOT exceed 64 characters and implementations SHOULD reject messages where this limit has been ignored.

ABNF:

```
sig-n-tag    = %x6e [FWS] "=" [FWS] textstring
```

t= Signature Timestamp

plain-text date-time; REQUIRED

The time that this header field was created. The format is the number of seconds since 00:00:00 on January 1, 1970 in the UTC time zone. The value is expressed as an unsigned integer in decimal ASCII. 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] dat
```

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.

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

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-SHA256"; See Section 4 for a description of these 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 10) for how the signature is computed.

ABNF:

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

s2=, a2= b2= Second signature (equivalent to s1, a1 and b1)

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.

Since the DNS lookup for the public key will check that the algorithm is correct a different MUST necessarily be used for the second signature.

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;

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

7. Computing the Body Hash

Although the body hash value will be incorporated into a Message-Instance header field, these header fields are ignored when calculating the header hash value and so the body hash and header hash may be calculated 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 DKIM2 body hash is calculated in the same manner as DKIM1's "simple" scheme:

All empty lines at the end of the message body are ignored. 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. That is "*CRLF" at the end of the body is converted to "CRLF".

No other changes are made to the body, which is then processed by the relevant hash algorithm(s). The hash value is converted to base64 form and inserted into (Signers) or compared to (Verifiers) the "bh1=" tag of the Message-Instance header field that is being created. If a second hash is calculated then its base64 representation will be included in the "bh2=" tag.

7.1. Preventing Transport Conversions

DKIM2's design is predicated on valid input.

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], [RFC2047] and other relevant message format standards 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.

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]).

Further, 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.

8. Computing the Header Fields Hash

The header fields hash calculation 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.

* Ignore some header fields

  When calculating the header field hash "Received" or "Return-Path" header fields MUST be ignored. These are Trace headers as described in [RFC5321] and serve only to document details of the SMTP transmission process.

  When calculating the header field hash any header field with a header field name starting with "X-" MUST be ignored. Currently deployed email systems use these fields as proprietary Trace headers which have no defined meaning for other systems and it considerably simplifies reporting on changes to header fields to ignore them.

  When calculating the header field hash any "Message-Instance" or "DKIM2-Signature" header fields MUST be ignored. These header fields will be included in the hash value that will be signed by a DKIM2-Signature header field and it simplifies implementations if they are not included twice, especially when determining whether all modifications to a message have been correctly declared.

  When calculating the header field hash any "DKIM-Signature" header fields and any header fields whose name starts with "ARC-" MUST be ignored. Not including DKIM1 and ARC signatures means that systems that wish to add other types of signature are free to do this in any convenient order.

* 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.

* 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, from the top downwards.

  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 hash(es) of the concatenated header fields are calculated.

The hash value is converted to base64 form and inserted into (Signers) or compared to (Verifiers) the "h1=" tag of the Message-Instance header field that is being created. If a second hash is calculated then its base64 representation will be included in the "h2=" tag.

9. The Relaxed Domain Match Algorithm

To assist in addressing the "DKIM replay" problem DKIM2 provides a "chain of custody" for every message. This is established by checking that the "mf= value of every DKIM2-Signature header field (except of course the i=1 instance) can be matched with the rt= value of the next lower numbered DKIM2-Signature header field.

It is also necessary to check DKIM2-Signature header fields for a match between the signing domain (specified in the d= tag) and the MAIL FROM domain (specified in the mf= tag).

To allow systems to use existing "bounce-handling" schemes with special subdomains in their MAIL FROM values a "relaxed" approach is taken to the matches between mf= and rt= and mf= and d=.

* Only the domain part of the rt= and mf= values is used for these matches The local part (and the @) are ignored.

* If there is not an exact match between the domain names then labels are removed, one by one from the left hand side of the mf= domain name and the comparison is repeated.

* If no labels remain then there is no match.

10. Signer Actions

This section gives the actions that need to be undertaken by the signer of a message. They may be done in any appropriate order.

10.1. Add any Necessary Message-Instance Header Fields

If a system is generating the initial form of a message or if it a Reviser that has made to changes to the message body and/or header fields then it MUST compute the body hash as described in Section 7 and the hash of the header fields as described in Section 8.

If the message does not contain a Message-Instance header field then one MUST be added. This MUST NOT contain any "recipes" (r=, r.header=).

If hashing the message body or relevant header fields does not give the same hash values as those recorded in the highest version (v=) Message-Instance header field then a new Message-Instance heade field MUST be added. This Message-Instance header field MUST contain "recipes" to be able to recreate the message corresponding to the hash values in the currently highest numbered Message-Instance header field, or a "recipe" of "z" to indicate that recreating the previous version of the message will not be possible.

A system may add more than one Message-Instance header field if it wishes to do so, but the DKIM2 design allows all modifications made by any single system to be documented in a single Message-Instance header field. Note that the v=1 Message-Instance header field MUST NOT contain any "recipes" and any other Message-Instance field MUST contain at least one "recipe".

10.2. Provide a "chain of custody" 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 (the [RFC5321] "envelope" values) MUST be available to (or deducible by) a Signer.

The receiver of a message will check for an exact match (including the local parts of the email addresses) between the MAIL FROM / RCPT TO [RFC5321] protocol values and the mf=, rt= values in the highest numbered (most recent) DKIM2-Signature header field. It is acceptable for there to be more email addresses in the rt= value than occur in the RCPT TO values, but any address used in the SMTP conversation MUST be present in the rt= value.

Verifiers will check for a relaxed domain match (see {relaxed-domain-match}) between the signing domain (d=) and the domain in the MAIL FROM value (mf=).

When the message being signed already has a DKIM2-Signature header field (i.e. it has already been transmitted at least once) then a valid "chain of custody" MUST be apparent when all of the DKIM2-Signature header fields are considered. This "chain of custody" contributes to the way in which DKIM2 tackles "DKIM replay" attacks. The "chain of custody" uses a relaxed domain match (see {relaxed-domain-match}).

In any situation where a messages will be forwarded in such a way that the mf= on the outgoing message is such that the "chain of custody" would be broken then the Signer MUST generate an extra DKIM2-Signature header field that causes values to match, i.e. a record must be fabricated that documents 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 a DKIM2 private key associated with a domain in the rt= entry. This is often achieved by the Signer creating the private key and never sharing it. The Signer gives the public key (and selector value) to the domain owner who creates an appropriate DNS entry. Alternatively, the Signer creates a public key DNS entry within a part of the DNS that they control and the domain owner merely needs to publish a CNAME pointing at this.

10.3. 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 -- this will be a matter of administrative convenience. Distribution and management of private keys is also outside the scope of this document.

10.4. Calculate a Signature Value

A signer calculates a hash solely over the Message-Instance and DKIM2-Signature header fields of the message and then signs this. The hashes of the body and other header fields are covered by the hashes in the highest version (v=) Message-Instance header field.

Note that the DKIM2-Signature header field contains a signature but does not give the hash value that was signed. This permits flexibility for any future signature schemes where a relevant hash value may not be readily available (or may be inordinately long).

The signature algorithm MUST apply the following steps in the order given.

* Convert all relevant header field names (not the header field values) to lowercase. For example, convert "DKIM2-signature" to "dkim2-signature".

* 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 order. First come the Message-Instance header fields in ascending version (v=) order. Second are the DKIM2-Signature header fields in ascending sequence (i=) order. Last of all is an incomplete DKIM2-Signature header field (the one that this system is creating) with all tags present except that

the signature value (b1=) is null (that is the base64 value is absent. If there will be a second signature then the b2= tag must be present, again with a null base64 value.

* The hash of the concatenated header fields is calculated and this value is then signed using the algorithm specified in the "a1=" tag of the DKIM2- Signature header field and using the private key corresponding to the selector given in the "s1=" tag of the DKIM2-Signature header field, as chosen above in Section 10.3}.

* If a second signature is to be generated then the process if repeated with the a2= and s2= settings.

The signature value(s) are converted to base64 form and inserted into the "b1=" tag (and "b2=") tags of the DKIM2-Signature header field which MUST then be placed into the message.

11. Verifier Actions

This section discusses the actions taken by a Verifier. In essence this will involve repeating all the actions taken by a Signer to produce a Message-Instance or DKIM2-Signature header field. To avoid a lot of repetition these actions will not be spelled out in detail. Once a hash value has been calculated it is then compared with the value reported by the Signer, or the Signer’s public key is used to determine whether the signature that has been provided is correct.

11.1. Output States

A verification 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 or mismatched hash value

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

A verifier MAY cease verifying once a single failure is detected.

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. It should be noted that any "Authentication-Results" header field will count as a modification to the email if any further DKIM2-Signature header fields are to be generated.

11.2. Validation of Tag Fields

Implementers MUST meticulously validate the format and values of Message-Instance and DKIM2-Signature header fields. Errors SHOULD be treated as a PERMFAIL (signature syntax error). Being "liberal in what you accept" is an inappropriate strategy.

Note, however, that the presence of unknown tags in a DKIM2-Signature header field (or a Message-Instance header field), MUST NOT cause a verification to fail.

Verifiers MAY return PERMFAIL ("signature expired") if it is more than 14 days since the timestamp recorded in the "t=" tag of a DKIM2-Signature header field.

11.3. Fetching the Public Key

The public key of a signature is needed to complete the verification process. Details of key management and representation are described in Section 4.5 and [DKIMKEYS]. The Verifier MUST validate the key record and MUST ignore any public key records that are malformed.

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 10.3 using the algorithm in the "a1=" tag, the domain from the "d=" tag, and the selector from the "s1=" tag. If a2= and s2 tags are present, subsequent steps are repeated for the second signature.

2. If the query for the public key fails to complete, 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, then the return PERMFAIL ("more than one key returned" since this is not permitted by [DKIMKEYS].

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 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 specified in the DKIM-Signature header field, the Verifier MAY immediately return PERMFAIL ("inappropriate key algorithm"). However, the tag fields in the public key record that would cause this to occur are now deprecated so DKIM2 implementations MAY ignore these tag fields altogether.

8. If the "h=" tag exists in the public key record and the hash algorithm implied by the "a1=" (or "a2=") 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").

11.4. Perform the Signature Verification Calculation

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-Instance and DKIM2-Signature header fields as described in Section 10.4. Note that this canonicalized version does not actually replace the original content.

2. Based on the algorithm indicated in the "a1=" tag, compute the hashes of the canonical copy. Then verify that the the signature conveyed in the "b1=" tag is correct for this hash value 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").

3. Otherwise, the signature has correctly verified.

4. If there is more than one signature provided then the second signature MUST be checked if the verifier is able to do so, using a2= and b2= as appropriate.

5. If either signature fails then an error SHOULD be reported.

6. If one signature fails and the other passes then the error that is reported should provide that information (e.g. PERMFAIL "RSA signature passed, elliptic curve signature failed")

The reasoning for requiring that both signatures pass is that if a signature scheme has recently become deprecated because it is known to be cryptographically flawed then Signers will use a second (unbroken) signature scheme. However, such a Signer may still provide the other signature for the benefit of Verifiers that have yet to upgrade -- reasoning perhaps that attacks are too expensive to be a very significant security issue. A Verifier that determines that one signature passes whilst the other fails may well be in a position to prevent an attack.

11.5. Check Most Recent Signature and Hashes for the Message

A Verifier SHOULD check the validity of the most recently applied (highest numbered i= value) DKIM2-Signature header field and the associated (v=) Message-Instance 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. If the check gives a TEMPFAIL result then a 4xx error code SHOULD be used to allow the sending MTA to understand the situation.

A Verifier SHOULD check that the mf= value in the most recent DKIM2-Signature header field is identical to the [RFC5321] MAIL FROM values of the SMTP protocol interaction that delivered the email to the Verifier. A Verifier SHOULD also check that all of the [RFC5321] RCPT TO values from the SMTP protocol occur in the most recent DKIM2-Signature header field. The values MUST BE put into lower-case before doing these checks. Note that these check MUST NOT use the relaxed domain match algorithm.

A Verifier SHOULD check that there is a relaxed domain match (see {relaxed-domain-match}) between the signing domain of the most recently applied DKIM2-Signature header field and the mf= value in that header field.

A Verifier SHOULD also check the chain of custody for the message (see {chain-of-custody}) is valid, using a relaxed domain match (see {#relaxed-domain-match}).

Should checking these signatures (all but the most recently applied) give the status TEMPFAIL then it may be possible for the Verifier to determine the validity of the signature at a later time. It the TEMPFAIL status continues to occur, or if a PERMFAIL is encountered then this MAY be reported to the sending MTA by means of a Delivery Status Notification. However the non-validating email MUST NOT be forwarded to any MTA outside of the current organisation.

11.6. Checking the Message-Instance Header Fields

The highest numbered (v=) Message-Instance header field SHOULD be checked to determine that the message body has not been altered since the body hash was calculated,

If the message has been modified since its original creation then the Message-Instance header fields will enable a Verifier to determine whether or not all the changes made are correctly recorded by using the "recipes" to construct each preceding version of the message.

Note that if it is only the first form of the message is of interest then all the "recipes" can be applied in turn and only one hash value checked -- the correctness of the intermediate hash values are not relevant to this assessment.

11.7. Checking the DKIM2-Signature Header Fields

However, in order to check the chain of custody, to assess whether the message has been exploded, to pick out "feedback" requests to be honoured or to assign reputation to Revisers then all of the DKIM2-Signature header fields will have to checked for validity. The TBA document explores these issues in more detail.

11.8. 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.

If an MTA wishes to reject messages where signatures are missing or do 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.

12. Delivery Status Notifications in the DKIM2 ecosystem

[BOUNCES] should be incorporated here.

13. EAI ([RFC6530]) considerations for DKIM2

TBA

14. IANA Considerations

TBA

15. Security Considerations

TBA

16. References

16.1. Normative References

[BOUNCES] Robinson, A., "DKIM2 Procedures for bounce processing", Work in Progress, Internet-Draft, draft-robinson-dkim2-bounce-processing-01, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-robinson-dkim2-bounce-processing-01>.

[DKIMKEYS] Chuang, W., "Domain Name Specification for DKIM2", Work in Progress, Internet-Draft, draft-chuang-dkim2-dns-03, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-chuang-dkim2-dns-03>.

[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>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/rfc4648>.

[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>.

[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>.

[RFC6530] Klensin, J. and Y. Ko, "Overview and Framework for Internationalized Email", RFC 6530, DOI 10.17487/RFC6530, February 2012, <https://www.rfc-editor.org/rfc/rfc6530>.

[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>.

16.2. Informative References

[CONCLUDEARC] Adams, J. T. and J. R. Levine, "Concluding the ARC Experiment", Work in Progress, Internet-Draft, draft-adams-arc-experiment-conclusion-01, 4 December 2025, <https://datatracker.ietf.org/doc/html/draft-adams-arc-experiment-conclusion-01>.

[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>.

[RFC8617] Andersen, K., Long, B., Ed., Blank, S., Ed., and M. Kucherawy, Ed., "The Authenticated Received Chain (ARC) Protocol", RFC 8617, DOI 10.17487/RFC8617, July 2019, <https://www.rfc-editor.org/rfc/rfc8617>.

Appendix A. Changes from Earlier Versions

draft-clayton-dkim2-spec-05

Assorted fixes, with particular thanks to Hannah Stern. Clarified that there are two types of rt/mt matches performed. Specified that recipes must not contain overlaps. Set out the need for matching mf= and d= and put the relaxed domain match algorithm into its own section. Set out that in practice all DKIM2-Signature header fields will need to be checked.

draft-clayton-dkim2-spec-04

Added a definition of a Reviser. Incorporate the Message-Instance scheme previously found in ALGEBRA. Recast the text relating to hashes and signatures accordingly. Changed t= back to just digits.

draft-clayton-dkim2-spec-03

Removed the pp= proposal, and briefly explained how signers often handle private keys on behalf of domain owners. Changed t= to be human-readable. Fixed description of body canonicalisation to match DKIM1/relaxed.

draft-clayton-dkim2-spec-02

Further clarifications and tidying up; alignment of ALGEBRA description with the new MailVersion header field-name; addition of h= tag field. Also added the pp= mechanism to address forwarders who do not have private keys to hand to make the rt/mf/rt chains validate.

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