Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 25 September 2026

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

24 March 2026

DomainKeys Identified Mail Signatures v2 (DKIM2)
draft-ietf-dkim-dkim2-spec-00

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 will 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 25 September 2026.

Table of Contents

1. Introduction 4
1.1. DKIM2 Architecture Documents 4
2. Terminology and Definitions 4
2.1. Signer 5
2.2. Forwarder 5
2.3. Reviser 5
2.4. Verifier 6
2.5. Signing Domain 6
2.6. Whitespace 6
2.7. Imported ABNF Tokens 6
2.8. Common ABNF Tokens 7
3. Signing and Verification Cryptographic Algorithms 7
3.1. The SHA256 Hashing Algorithm 7
3.2. The RSA-SHA256 Signing Algorithm 8
3.3. The Ed25519-SHA256 Signing Algorithm 8
3.4. Other Algorithms 8
3.5. Selectors 8
3.6. Key Management 9
4. Recipes 9
4.1. Header Recipes 11
4.2. Body Recipes 12
5. Message Hash Values 13
5.1. Computing the Body Hash 14
5.2. Computing the Header Fields Hash 14
6. Tag=Value Lists 16
7. The Message-Instance Header Field 17
7.1. m= the revision number of the Message-Instance header field 17
7.2. r= recipes to recreate the previous instance of the message 17

7.3. h= the hash values for the message 17
8. The DKIM2-Signature Header Field 18
8.1. i= the sequence number of the DKIM2-Signature header field 18
8.2. m= the highest numbered Message-Instance header field . . 18
8.3. n= nonce value 19
8.4. t= signature timestamp 19
8.5. mf= the MAIL FROM used when the message was sent 19
8.6. rt= the RCPT TO value(s) used when the message was sent 20
8.7. d= the domain associated with this signature. 20
8.8. s= the signature value(s) for the message 21
8.9. f= flags 21
9. Signer Actions 22
9.1. Add any Necessary Message-Instance Header Fields 22
9.2. Provide a "Chain of Custody" for the Message 23
9.3. The Relaxed Domain Match Algorithm 24
9.4. Select a Private Key and Corresponding Selector Value . . 24
9.5. Calculate a Signature Value 25
10. Verifier Actions 26
10.1. Output States 26
10.2. Validation of Tag Fields 26
10.3. Fetching the Public Key 27
10.4. Perform the Signature Verification Calculation 28
10.5. Check Most Recent Signature and Hashes for the Message 29
10.6. Checking the Message-Instance Header Fields 30
10.7. Checking the DKIM2-Signature Header Fields 30
10.8. Interpret Results/Apply Local Policy 30
11. Delivery Status Notifications in the DKIM2 ecosystem 31
11.1. DSN contents 31
11.1.1. Bounce Propagation 31
11.1.2. Authentication of Inbound Bounce Notifications . . . 32
12. Preventing Transport Conversions 32
13. EAI (RFC6530) Considerations for DKIM2 33
14. IANA Considerations 33
15. Security Considerations 33
16. Changes from Earlier Versions 33
17. References 34
17.1. Normative References 34
17.2. Informative References 35
Authors' Addresses 36

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.

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.

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 with (see [CONCLUDEARC]) the experimental ARC protocol ([RFC8617]).

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

This document uses JSON [RFC8259] for collections of related information. The JSON objects are then base64 encoded and placed into Tag=Value lists. This is done to avoid significant complexities in parsing Tag=Value lists (see Section 6) and to simplify the way in which algorithmic agility is provided. Unrecognised fields within JSON objects MUST be ignored.

2.1. Signer

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, as an alternative to delivering it into a destination mailbox, can forward 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. Verifier

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]:

* "Domain"

* "Forward-path"

* "reverse-path"

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:

```
ALPHADIGITD  = (ALPHA / DIGIT / "-" / "_")

textstring   =  [FWS] ALPHADIGITD *(ALPHADIGITD) [FWS]

ALPHADIGITPS =  (FWS / ALPHA / DIGIT / "+" / "/")

base64string =  ALPHADIGITPS *(ALPHADIGITPS) [[FWS] "=" [[FWS] "="]]

rcptmailbox  =  ( "<Postmaster@" Domain ">" /
                  "<Postmaster>" /
                  Forward-path )
```

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 definition of base64string allows 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.

3. 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.

3.1. The SHA256 Hashing Algorithm

The SHA256 hashing algorithm is used to compute body and header hashes as defined in Section 5.1 and Section 5.2.

The resultant values are identified by the text string "sha256" and placed into Message-Instance header fields.

3.2. 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 9.5 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.

The signature value (expressed in base64) is placed (with the identifying text string "rsa-sha256") into DKIM2-Signature header fields.

3.3. 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 9.5 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].

The signature value (expressed in base64) is placed (with the identifying text string "ed25519-sha256") into DKIM2-Signature header fields.

3.4. Other Algorithms

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

3.5. Selectors

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

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.

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 = Domain
```

3.6. 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 of the DKIM2-Signature header field.

DKIM2 keys are stored in a subdomain named "_domainkey". Given a DKIM2-Signature field with a "d=" tag of "example.com" and a selector 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. Recipes

A set of "recipes" is used to recreate the previous version of the body and/or header fields of a message. The recipes are provided within a JSON object with the schema:

```
 {
   "$schema": "https://json-schema.org/draft/2020-12/schema",
   "$id": "https://dkim2.org/schemas/recipe-v1",
   "title": "DKIM2 recipes",
   "description": "see draft-dkim-dkim2-spec",
   "type": "object",
   "properties": {
     "h": {
       "description": [ "recipes to recreate header fields",
         "keys are header field names matched case-insensitively",
```

```
        "and there MUST NOT be keys that differ only in case"],
      "oneOf": [
        {
          "description": "per-field-name recipe arrays",
          "type": "object",
          "minProperties": 1,
          "additionalProperties": { "$ref": "#/$defs/recipe-steps" }
        },
        {
          "description": "previous header state cannot be recreated",
          "type": "null"
        }
      ]
    },
    "b": {
      "description": "recipes to recreate the body",
      "oneOf": [
        {
          "description": "body recipes",
          "$ref": "#/$defs/recipe-steps"
        },
        {
          "description": "previous body state cannot be recreated",
          "type": "null"
        },
        {
          "description": "body was truncated (DSN)",
          "type": "object",
          "properties": {
            "z": { "type": "boolean", "const": true }
          },
          "required": ["z"], "additionalProperties": false
        }
      ]
    }
  },
  "anyOf": [
    { "required": ["h"] },
    { "required": ["b"] }
  ],
  "$defs": {
    "recipe-steps": {
      "type": "array",
      "items": {
        "oneOf": [
          {
            "description": "copy lines/fields, start to end inclusive",
            "type": "object",
```

```
            "properties": {
              "c": { "type": "array",
                "items": { "type": "integer", "minimum": 1 },
                "minItems": 2, "maxItems": 2
              }
            },
            "required": ["c"], "additionalProperties": false
          },
          {
            "description": "data lines/values to emit",
            "type": "object",
            "properties": {
              "d": { "type": "array",
                "items": { "type": "string" },
                "minItems": 1
              }
            },
            "required": ["d"], "additionalProperties": false
          }
        ]
      }
    }
  }
}
```

Note that the specification of JSON schemas is maintained by the JSON Schema organisation, and the relevant specification document is linked to by the $schema field in each JSON schema.

4.1. Header Recipes

A Header Recipe is an array of instructions applied to the specified header fields with the given header field name. These instructions are applied in order to the message which has been received so as to recreate the message as it was before modifications were made.

If there is no "h" field in the JSON object then there was no modification to the header fields.

If the "h" field value is null (there are no recipes for any header field) then the previous state of the header fields cannot be recreated. Verifiers of the message may be able to determine, by seeing which entity makes this declaration, that this is acceptable to them because, for example, that entity is providing a contractually arranged service.

Matching of header field names is always done without regard to case.

If a header field name is not present in the JSON object then all header fields with that header field name are to be retained.

If the recipe array for a header field name that is present in the JSON object is empty then all instances of that header field are to be removed to reinstate the previous state of the message.

Header fields are numbered "bottom up" (the opposite direction to the body lines). That is to say, when walking the header fields from the top of the message to end of the header fields then the last header field instance encountered with any particular header field name is numbered 1, the header field (with the same header field name) above 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.

The recipes are processed in order and the resulting header fields are emitted so that later header field will appear above earlier header fields in the recreated message.

Each recipe step is a JSON object with exactly one key:

A "c" step has the form {"c": [start, end]}. The relevant header field instances numbered from start to end inclusive, are to be emitted. The start value of each "c" step MUST be in ascending order and MUST be greater than the end value of all preceding "c" steps for this header field name.

A "d" step has the form {"d": ["value1", "value2", ...]}. Each string in the array is treated as a value to which the relevant header field name and a colon is prepended and a CRLF is appended and the resultant string is then emitted. Note that the way in which hashes are calculated (see Section 5.2) means that no heed needs to be taken of wrapping or the case of the header field name. The text strings MUST NOT contain CR or LF characters. If a string is empty then the CRLF will immediately follow the header field name and colon.

4.2. Body Recipes

A Body Recipe is an array of instructions applied to the message body which can recreate the message as it was before modifications were made.

If there is no "b" field in the JSON object then there was no modification to the message body. Note that the JSON schema requires either "h" or "b" to be present.

If the "b" field is null (there are no recipes) then the previous state of the message body cannot be recreated. Verifiers of the message may be able to determine, by seeing which entity makes this declaration, that this is acceptable to them because, for example, that entity is providing a contractually arranged service.

Body lines are numbered "top down" (the opposite direction to the header fields). The first line of the body (immediately after the blank line that indicates that there are no more header fields) is numbered 1.

The recipes are processed in order and the resulting body lines fields are emitted so that later lines will appear below earlier lines in the recreated message.

Each recipe step is a JSON object with exactly one key:

A "c" step has the form {"c": [start, end]}. The message body lines from start to end, inclusive, are to be emitted. The start value of each "c" step MUST be in ascending order and MUST be greater than the end value of all preceding "c" steps.

A "d" step has the form {"d": ["line1", "line2", ...]}. Each string in the array has a CRLF appended and the resultant string is emitted. The text strings MUST NOT contain CR or LF characters. If a string is empty then just a CRLF is emitted.

A "z" step has the form {"z": true} and indicates that the body was truncated (see the DSN handling in Section 11).

5. Message Hash Values

A set of cryptographic "hashes" are used to record the current message body and header fields. The hashes are placed into the h= tag of a Message-Instance header field.

To provide for algorithmic dexterity more that one hash value, using a different algorithm MAY be supplied in the same Message-Instance header field.

Since Message-Instance header fields are ignored when calculating the header hash value, the body hash and header hash may be calculated in any convenient order.

5.1. Computing the Body Hash

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 name of the hash and the hash value (converted to base64 form) is then inserted into (Signers) or compared to (Verifiers) the value of the "h=" tag of the Message-Instance header field that is being created/verified. If multiple hashes are calculated then multiple entries within the "h=" value will be inserted/compared.

5.2. Computing the Header Fields Hash

The header fields hash calculation done by a Signer 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 field name starts with "ARC-" MUST be ignored. Not including DKIM1 and ARC signatures means that systems that wish to add other types of signature as well as a DKIM2 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 were likely to have been placed into the message header, that is from the last within the header upwards (the same ordering as is used in the header recipes (see Section 4.1).

  It is sometimes suggested that some MTAs re-order header fields after they receive an email. If an MTA does change the order of header fields with the same header field name (and those header fields will be included in the hash calculation) then it is their responsibility to recover the original order before verifying an existing signature or passing a previously signed message to another MTA that may wish to do such verification.

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

The name of the hash and the hash value (converted to base64 form) is then inserted into (Signers) or compared to (Verifiers) the value of the "h=" tag of the Message-Instance header field that is being created/verified. If multiple hashes are calculated then multiple entries within the "h=" value will be inserted/compared.

6. 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 text or "base64" text (as defined in [RFC2045], Section 6.8). The name of the tag will determine the encoding and structure of each value.

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 *(ALPHA / DIGIT / "_")
tval      =  %x21-3A / %x3C-7E
                  ; not SEMICOLON
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. Semicolon (";") characters MUST NOT occur in the tag value, since that separates tag-specs.

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.

7. The Message-Instance Header Field

A Message-Instance 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 6. The tags are described below.

The m= and h= tags MUST be present. The r= tag is optional.

7.1. m= the revision number of the Message-Instance header field

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-m-tag    = %x6d [FWS] "=" [FWS] 1*DIGIT
```

7.2. r= recipes to recreate the previous instance of the message

The r= tag value is the base64 encoded version of the JSON object that contains the recipes that allow the previous instance of the message to be recreated (see Section 4}.

ABNF:

```
mi-r-tag    = %x72 [FWS] "=" base64string
```

7.3. h= the hash values for the message

The h= tag value contains the hash name, header hash value and body hash value. Calculating the hash values is explained in Section 5.

ABNF:

```
mi-h-tag    = %x68 [FWS] "=" hash-set *("," hash-set )
hash-set    = [FWS] hash-name [FWS] ":" header-hash ":" body-hash
hash-name   = "sha256" / x-hash-name
header-hash = base64string
body-hash   = base64string
x-hash-name = textstring ; for later expansion
```

8. 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 6.

The tags are described 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.

The i=, m=, t=, mf=, rt=, d= and s= tags MUST be present. The other tags are optional.

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

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
```

8.2. m= the highest numbered Message-Instance header field

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

ABNF:

```
sig-m-tag = %x6d [FWS] "=" [FWS] 1*DIGIT
```

8.3. n= nonce value

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

Note the value MUST be simple ASCII and MUST NOT contain semicolon.

ABNF:

```
sig-n-tag = %x6e [FWS] "=" [FWS] *(%x21-3A / %x3C-7E/ FWS)
```

8.4. t= signature timestamp

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] 1*DIGIT
```

8.5. mf= the MAIL FROM used when the message was sent

DKIM2 records the [RFC5321] MAIL FROM value that was used when the message was transmitted over an SMTP link from the signing MTA. Note that MAIL FROM may be just "<>", for example for a Delivery Status Notification.

The value is recorded as the base64 encoding of the [RFC5321] reverse-path because of the complex syntax of reverse-path values (which can include characters which would confuse naive parsers of DKIM2-Signature header fields). The angle brackets MUST be included, but any "Mail-parameters" that were present after the reverse-path MUST NOT be included.

ABNF:

```
sig-mf-tag  = %x6d %x66 [FWS] "=" base64string
```

8.6. rt= the RCPT TO value(s) used when the message was sent

DKIM2 records the [RFC5321] RCPT TO value(s) that were used when the message was transmitted over an SMTP link from the signing MTA.

The value is recorded as the base64 encoding of the [RFC5321] Forward-path because of the complex syntax of Forward-path values (which can include characters which would confuse naive parsers of DKIM2-Signature header fields). The angle brackets MUST be included, but any "Rcpt-parameters" that were present after the Forward-path MUST NOT be included.

When a message is intended for more than one recipient then the RCPT TO values provided 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. Alternatively, multiple copies of the email may be generated so as to not immediately reveal who else received the email.

However, if "bcc:" recipients are involved then in order to meet the requirements of [RFC5322] Section 3.6.3 each and every bcc recipients MUST NOT revealed to any other message recipient.

ABNF:

```
sig-rt-tag = %x72 %x74 [FWS] "=" base64string *("," base64string)
```

8.7. d= the domain associated with this signature.

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 name in the d= tag MUST exactly match the rightmost labels of the domain name 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 the d= domain name.

When the mf= domain is empty ("<>"), as will be the case for Delivery Status Notifications (DSNs), then no match is required.

ABNF:

```
sig-d-tag   = %x64 [FWS] "=" [FWS] Domain
```

8.8. s= the signature value(s) for the message

The s= tag value contains the selector, signature algorithm name and signature value. Calculating the value is explained in Section 9.5.

The selector values subdivides the namespace for the domain being used for signing.

The algorithm value is the one used to generate the signature. Verifiers MUST support "RSA-SHA256" for which the string "rsa-sha256" is used and "Ed25519-SHA256" for which the string "ed25519-sha256" is used. See Section 3 for a description of these algorithms.

To provide for algorithmic dexterity more than one signature, using different algorithms, MAY be supplied. Since the DNS lookup for the public key will check that the k= algorithm value matches, a different selector MUST necessarily be used for each signature.

ABNF:

```
sig-s-tag   = %x73 [FWS] "=" [FWS] sig-set *( "," sig-set )
sig-set     = selector [FWS] ":" [FWS] sig-name [FWS] ":" message-sig
sig-name    = "rsa-sha256" / "ed25519-sha256" / x-sig-name
x-sig-name  = textstring     ; for later extension
message-sig = base64string
```

8.9. f= flags

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 the h= JSON object of the relevant Message-Instance) 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] "," [FWS] sig-f-tag-data)
sig-f-tag-data   = "donotmodify" | "donotexplode" | "feedback" |
                   "exploded" | x-sig-f-tag-data
x-sig-f-tag-data = textstring ; for later extension
```

9. 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.

9.1. 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 5.1 and the hash of the header fields as described in Section 5.2.

If the message does not contain a Message-Instance header field then one MUST be added.

If hashing the message body or relevant header fields does not give the same hash values as those recorded in the highest version (m=) Message-Instance header field then a new Message-Instance header 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 null recipe 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 first (m=1) Message-Instance header field MAY contain "recipes" if it is wished to record any changes made to a message as it enters the DKIM2 ecosystem. All other Message-Instance header fields SHOULD contain at least one "recipe".

9.2. Provide a "Chain of Custody" for the Message

To construct the DKIM2-Signature header field contains the MAIL FROM and RCPT TO values that will be used when the message is transmitted so these [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= and rt= values in the highest numbered (most recent) DKIM2-Signature header field. It is acceptable for there to be more RCPT TO email addresses recorded in rt= than are actually used in the SMTP conversation, but any RCPT TO value which is used MUST be present.

Verifiers will check for a relaxed domain match (see Section 9.3) between the signing domain (d=) and the domain in the MAIL FROM value.

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.

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 RCPT TO 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.

9.3. 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 MAIL FROM value recorded in every DKIM2-Signature header field (except of course the i=1 instance) can be matched with a RCPT TO 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.

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 these values.

* Only the domain part of the MAIL FROM and RCPT TO 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 MAIL FROM domain name and the comparison is repeated.

* If no labels remain then there is no match.

9.4. 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.

9.5. Calculate a Signature Value

A Signer calculates a signature solely over the Message-Instance and DKIM2-Signature header fields of the message. The hashes of the body and other header fields are covered by the hashes in the highest version (m=) Message-Instance header field and hence the signature will in practice be signing the message as a whole.

Most cryptographic schemes proceed by first calculating a hash value and then signing the hash value, but the DKIM2-Signature header field only provides the final signature value. This means that there is no difficulty if the hash value is inordinately long, or is not emitted by the cryptographic routine being used.

The signature algorithm MUST apply the following steps in the order given (which are consistent with the steps undertaken in calculating header hashes).

* 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.

* Delete all WSP characters. This means all WSP characters before and after the colon separating the header field name from the header field value, all WSP characters within the unfolded header field value and all trailing WPS characters before the CRLF. The colon separator and the CRLF MUST be retained.

* Place the header fields in order. First come the Message-Instance header fields in ascending instance (m=) 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(s) within the (s=) value are set to the null string (""). The incomplete header field MUST be unfolded and have spaces removed in just the same way as the complete header fields being processed.

* The concatenated header fields are then fed to the signature algorithm(s). Once all the values are available the null strings are replaced by the base64 values of the signatures.

10. 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 a signature that has been provided is correct.

10.1. 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.

10.2. 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.

10.3. 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 3.6 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 3.6 using the "d=" tag, and the selector from within the JSON object in the "s" tag. If there is more than one signature within the JSON object then these steps are repeated for each one.

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 ("m=" 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 type of signature being checked is not included in the contents of the "h=" tag, the Verifier MUST ignore the key record and return PERMFAIL ("inappropriate hash algorithm").

10.4. Perform the Signature Verification Calculation

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 9.5. Note that this canonicalized version does not actually replace the original content.

2. Based on the algorithm and selector indicated s= tag value determine whether the signature of the highest numbered DKIM2-Signature field validates. The signature value(s) themselves will need to be removed to correspond with what was actually signed. If the signature is incorrect the Verifier SHOULD ignore the signature and return PERMFAIL ("signature did not verify").

3. If there is more than one signature provided then they MUST all be checked if the verifier is able to do so. If any signature fails then an error SHOULD be reported. If all signatures that can be checked fail then PERMFAIL MUST be reported.

4. If some signatures fail and other pass then the error that is reported should provide that information (e.g. PERMFAIL "rsa-sha256 signature passed, ed25519-sha256 signature failed").

The reasoning for requiring that all 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.

10.5. 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 (m=) 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 MAIL FROM 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 Section 9.3).

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.

10.6. Checking the Message-Instance Header Fields

The highest numbered (m=) 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.

10.7. 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.

10.8. 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.

11. Delivery Status Notifications in the DKIM2 ecosystem

In the DKIM2 ecosystem, when a message cannot be delivered then this is reported to the sending machine by means of an [RFC5321] return code or, if the SMTP session has completed, by generating a Delivery Status Notification (DSN, as defined in [RFC3461].

A DSN MUST be addressed to the MTA that sent the message. This prevents "backscatter" by passing failures back along the chain of MTAs that were in involved in passing the message forwards. This is achieved by using the mf= tag from the highest numbered DKIM2-Signature field. If this field is null ("mf=<>") then a DSN MUST NOT be sent.

11.1. DSN contents

As set out in [RFC3461], the DSN has a top-level MIME part of type multipart/report. Among other things, that MIME part must contain a MIME part of type message/rfc822 that holds either the original message exactly as it was submitted by the sending system or just the header fields of that message.

All relevant DKIM2-Signature header fields (and Message-Instance header fields if the message body is supplied) MUST verify. The DSN itself MUST have appropriate Message-Instance and DKIM2-Signature fields, noting that the MAIL FROM to be used will be null ("<>").

If the message body has been truncated (rather than omitted altogether) then in order to allow verification of the DNS contents a Message-Instance header field MUST be added to the message with a body recipe containing a {"z": true} step.

11.1.1. Bounce Propagation

A Forwarder which receives a DSN MAY decide to propagate this DSN to the MAIL FROM address used to deliver the message to it (which can be found in the relevant DKIM2-Signature header field). The DSN SHOULD be handled in the usual way, with Message-Instance header fields documenting any changes and a DKIM2-Signature field with an incremented hop count value added.

The Forwarder MAY alternatively decide to reconstruct the message (or just the message header fields) as they were when the message was delivered to the Forwarder and construct a DSN using that information. The information in Message-Instance header fields can be used to achieve this. The resultant DSN is sent to the MAIL FROM address from the now highest numbered DKIM2-Signature header field. Doing this will ensure that details of where the message was forwarded to will not be revealed to the previous hop.

11.1.2. Authentication of Inbound Bounce Notifications

When a system receives a DKIM2 signed bounce notification, and the included original message is also DKIM2 signed, it SHOULD verify that this message (or just the header fields if the body is not present) has not been altered.

This means:

1. The DSN's DKIM2-Signature will have a signing domain that is aligned with the recipient of the message that is being returned. The recipient's address is located in the rt= tag of the last (highest i= tag) DKIM2-Signature in the returned message.

2. The last (highest i= tag) DKIM2-Signature header field of the returned message will be one that was generated by the system receiving the bounce notification, determined by examining the d= and mf= tags of that DKIM2-Signature header field.

3. The header fields of the embedded message (in the message/rfc822 MIME part) can be verified. If the message body is present then that can also be verified by inspecting the Message-Instance header field(s).

If the verification fails then the DSN MUST NOT be propagated any further. If verification has been performed prior to accepting the DSN from the sender the DSN SHOULD be rejected with a 550/5.7.x return code. If the verification cannot be completed because of a temporary issue (with DNS lookups) then a 4xx return code should be used.

12. 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.

13. EAI ([RFC6530]) Considerations for DKIM2

TBA

14. IANA Considerations

TBA

15. Security Considerations

TBA

16. Changes from Earlier Versions

draft-ietf-dkim-dkim2-spec-00

Removed JSON for hashes, signatures and SMTP parameters. Provided valid JSON for recipes and added "z" for truncated body. Changed algorithm names for signing. Simplified the canonicalisation performed for the header fields signed by DKIM2-Signature. Changed v= to m= for message instance numbering.

General tidying up of specifying tag=value specifications and associated ABNF. Various other fixes for issues flagged in WG.

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

17. References

17.1. Normative References

[DKIMKEYS] Chuang, W., "Domain Name Specification for DKIM2", Work in Progress, Internet-Draft, draft-chuang-dkim2-dns-04, 18 March 2026, <https://datatracker.ietf.org/doc/html/draft-chuang-dkim2-dns-04>.

[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>.

[RFC3461] Moore, K., "Simple Mail Transfer Protocol (SMTP) Service Extension for Delivery Status Notifications (DSNs)", RFC 3461, DOI 10.17487/RFC3461, January 2003, <https://www.rfc-editor.org/rfc/rfc3461>.

[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>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/rfc/rfc8259>.

[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>.

17.2. 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>.

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