Network Working Group
Request for Comments: 4871
Obsoletes: 4870
Category: Standards Track

E. Allman
Sendmail, Inc.
J. Callas
PGP Corporation
M. Delany
M. Libbey
Yahoo! Inc
J. Fenton
M. Thomas
Cisco Systems, Inc.
May 2007

DomainKeys Identified Mail (DKIM) Signatures

Status of This Memo

Copyright Notice

Abstract

DomainKeys Identified Mail (DKIM) defines a domain-level authentication framework for email using public-key cryptography and key server technology to permit verification of the source and contents of messages by either Mail Transfer Agents (MTAs) or Mail User Agents (MUAs). The ultimate goal of this framework is to permit a signing domain to assert responsibility for a message, thus protecting message signer identity and the integrity of the messages they convey while retaining the functionality of Internet email as it is known today. Protection of email identity may assist in the global control of "spam" and "phishing".

Table of Contents

1. Introduction

DomainKeys Identified Mail (DKIM) defines a mechanism by which email messages can be cryptographically signed, permitting a signing domain to claim responsibility for the introduction of a message into the mail stream. Message recipients can verify the signature by querying the signer's domain directly to retrieve the appropriate public key, and thereby confirm that the message was attested to by a party in possession of the private key for the signing domain.

The approach taken by DKIM differs from previous approaches to message signing (e.g., Secure/Multipurpose Internet Mail Extensions (S/MIME) [RFC1847], OpenPGP [RFC2440]) in that:

- the message signature is written as a message header field so that neither human recipients nor existing MUA (Mail User Agent) software is confused by signature-related content appearing in the message body;
- there is no dependency on public and private key pairs being issued by well-known, trusted certificate authorities;
- there is no dependency on the deployment of any new Internet protocols or services for public key distribution or revocation;
- signature verification failure does not force rejection of the message;
- no attempt is made to include encryption as part of the mechanism;
- message archiving is not a design goal.

DKIM:

- is compatible with the existing email infrastructure and transparent to the fullest extent possible;
- requires minimal new infrastructure;
- can be implemented independently of clients in order to reduce deployment time;
- can be deployed incrementally;
- allows delegation of signing to third parties.

1.1. Signing Identity

DKIM separates the question of the identity of the signer of the message from the purported author of the message. In particular, a signature includes the identity of the signer. Verifiers can use the signing information to decide how they want to process the message. The signing identity is included as part of the signature header field.

> INFORMATIVE RATIONALE: The signing identity specified by a DKIM signature is not required to match an address in any particular header field because of the broad methods of interpretation by recipient mail systems, including MUAs.

1.2. Scalability

DKIM is designed to support the extreme scalability requirements that characterize the email identification problem. There are currently over 70 million domains and a much larger number of individual addresses. DKIM seeks to preserve the positive aspects of the current email infrastructure, such as the ability for anyone to communicate with anyone else without introduction.

1.3. Simple Key Management

DKIM differs from traditional hierarchical public-key systems in that no Certificate Authority infrastructure is required; the verifier requests the public key from a repository in the domain of the claimed signer directly rather than from a third party.

The DNS is proposed as the initial mechanism for the public keys. Thus, DKIM currently depends on DNS administration and the security of the DNS system. DKIM is designed to be extensible to other key fetching services as they become available.

2. Terminology and Definitions

This section defines terms used in the rest of the document. Syntax descriptions use the form described in Augmented BNF for Syntax Specifications [RFC4234].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

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 issue is that a message must be signed before it leaves the administrative domain of the signer.

2.2. Verifiers

Elements in the mail system that verify signatures are referred to as verifiers. These may be MTAs, Mail Delivery Agents (MDAs), or MUAs. In most cases it is expected that verifiers will be close to an end user (reader) of the message or some consuming agent such as a mailing list exploder.

2.3. Whitespace

There are three forms of whitespace:

- WSP represents simple whitespace, i.e., a space or a tab character (formal definition in [RFC4234]).

- LWSP is linear whitespace, defined as WSP plus CRLF (formal definition in [RFC4234]).

- 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 and LWSP are given for information only):

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

The definition of FWS is identical to that in [RFC2822] except for the exclusion of obs-FWS.

2.4. Common ABNF Tokens

The following ABNF tokens are used elsewhere in this document:

```
hyphenated-word =  ALPHA [ *(ALPHA / DIGIT / "-") (ALPHA / DIGIT) ]
base64string =     1*(ALPHA / DIGIT / "+" / "/" / [FWS])
                   [ "=" [FWS] [ "=" [FWS] ] ]
```

2.5. 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 [RFC2821]:

- "Local-part" (implementation warning: this permits quoted strings)
- "sub-domain"

The following tokens are imported from [RFC2822]:

- "field-name" (name of a header field)
- "dot-atom-text" (in the Local-part of an email address)

The following tokens are imported from [RFC2045]:

- "qp-section" (a single line of quoted-printable-encoded text)
- "hex-octet" (a quoted-printable encoded octet)

  INFORMATIVE NOTE: Be aware that the ABNF in RFC 2045 does not obey the rules of RFC 4234 and must be interpreted accordingly, particularly as regards case folding.

Other tokens not defined herein are imported from [RFC4234]. These are intuitive primitives such as SP, HTAB, WSP, ALPHA, DIGIT, CRLF, etc.

2.6. DKIM-Quoted-Printable

The DKIM-Quoted-Printable encoding syntax resembles that described in Quoted-Printable [RFC2045], Section 6.7: any character MAY be encoded as an "=" followed by two hexadecimal digits from the alphabet "0123456789ABCDEF" (no lowercase characters permitted) representing the hexadecimal-encoded integer value of that character. All control characters (those with values < %x20), 8-bit characters (values > %x7F), and the characters DEL (%x7F), SPACE (%x20), and semicolon (";", %x3B) MUST be encoded. Note that all whitespace, including SPACE, CR, and LF characters, MUST be encoded. After encoding, FWS MAY be added at arbitrary locations in order to avoid excessively long lines; such whitespace is NOT part of the value, and MUST be removed before decoding.

ABNF:

```
dkim-quoted-printable =
                    *(FWS / hex-octet / dkim-safe-char)
                ; hex-octet is from RFC 2045
dkim-safe-char =   %x21-3A / %x3C / %x3E-7E
                ; '!' - ':', '<', '>' - '~'
                ; Characters not listed as "mail-safe" in
                ; RFC 2049 are also not recommended.
```

INFORMATIVE NOTE: DKIM-Quoted-Printable differs from Quoted-Printable as defined in RFC 2045 in several important ways:

1. Whitespace in the input text, including CR and LF, must be encoded. RFC 2045 does not require such encoding, and does not permit encoding of CR or LF characters that are part of a CRLF line break.

2. Whitespace in the encoded text is ignored. This is to allow tags encoded using DKIM-Quoted-Printable to be wrapped as needed. In particular, RFC 2045 requires that line breaks in the input be represented as physical line breaks; that is not the case here.

3. The "soft line break" syntax ("=" as the last non-whitespace character on the line) does not apply.

4. DKIM-Quoted-Printable does not require that encoded lines be no more than 76 characters long (although there may be other requirements depending on the context in which the encoded text is being 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 5) and Verifier Actions (Section 6)). NOTE: This section must be read in the context of those sections.

3.1. Selectors

To support multiple concurrent public keys per signing domain, the key namespace is subdivided using "selectors". For example, selectors might indicate the names of office locations (e.g., "sanfrancisco", "coolumbeach", and "reykjavik"), the signing date (e.g., "january2005", "february2005", etc.), or even the individual user.

Selectors are needed to support some important use cases. For example:

- Domains that want to delegate signing capability for a specific address for a given duration to a partner, such as an advertising provider or other outsourced function.

- Domains that want to allow frequent travelers to send messages locally without the need to connect with a particular MSA.

- "Affinity" domains (e.g., college alumni associations) that provide forwarding of incoming mail, but that do not operate a mail submission agent for outgoing mail.

Periods are allowed in selectors and are component separators. When keys are retrieved from the DNS, periods in selectors define DNS label boundaries in a manner similar to the conventional use in domain names. Selector components might be used to combine dates with locations, for example, "march2005.reykjavik". In a DNS implementation, this can be used to allow delegation of a portion of the selector namespace.

ABNF:

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

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

Beyond administrative convenience, selectors make it possible to seamlessly replace public keys on a routine basis. If a domain wishes to change from using a public key associated with selector "january2005" to a public key associated with selector "february2005", it merely makes sure that both public keys are advertised in the public-key repository concurrently for the transition period during which email may be in transit prior to verification. At the start of the transition period, the outbound email servers are configured to sign with the "february2005" private key. At the end of the transition period, the "january2005" public key is removed from the public-key repository.

> INFORMATIVE NOTE: A key may also be revoked as described below. The distinction between revoking and removing a key selector record is subtle. When phasing out keys as described above, a signing domain would probably simply remove the key record after

the transition period. However, a signing domain could elect to revoke the key (but maintain the key record) for a further period. There is no defined semantic difference between a revoked key and a removed key.

While some domains may wish to make selector values well known, others will want to take care not to allocate selector names in a way that allows harvesting of data by outside parties. For example, if per-user keys are issued, the domain owner will need to make the decision as to whether to associate this selector directly with the user name, or make it some unassociated random value, such as a fingerprint of the public key.

INFORMATIVE OPERATIONS NOTE: Reusing a selector with a new key (for example, changing the key associated with a user's name) makes it impossible to tell the difference between a message that didn't verify because the key is no longer valid versus a message that is actually forged. For this reason, signers are ill-advised to reuse selectors for new keys. A better strategy is to assign new keys to new selectors.

3.2. Tag=Value Lists

DKIM uses a simple "tag=value" syntax in several contexts, including in messages and domain signature records.

Values are a series of strings containing either plain text, "base64" text (as defined in [RFC2045], Section 6.8), "qp-section" (ibid, Section 6.7), or "dkim-quoted-printable" (as defined in Section 2.6). The name of the tag will determine the encoding of each value. Unencoded semicolon (";") characters MUST NOT occur in the tag value, since that separates tag-specs.

INFORMATIVE IMPLEMENTATION NOTE: Although the "plain text" defined below (as "tag-value") only includes 7-bit characters, an implementation that wished to anticipate future standards would be advised not to preclude the use of UTF8-encoded text in tag=value lists.

Formally, the syntax rules are as follows:

```
tag-list  =  tag-spec 0*( ";" tag-spec ) [ ";" ]
tag-spec  =  [FWS] tag-name [FWS] "=" [FWS] tag-value [FWS]
tag-name  =  ALPHA 0*ALNUMPUNC
tag-value =  [ tval 0*( 1*(WSP / FWS) tval ) ]
                  ; WSP and FWS prohibited at beginning and end
tval      =  1*VALCHAR
VALCHAR   =  %x21-3A / %x3C-7E
                  ; EXCLAMATION to TILDE except SEMICOLON
ALNUMPUNC =  ALPHA / DIGIT / "_"
```

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

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

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

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

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

Unrecognized tags MUST be ignored.

Tags that have an empty value are not the same as omitted tags. An omitted tag is treated as having the default value; a tag with an empty value explicitly designates the empty string as the value. For example, "g=" does not mean "g=*", even though "g=*" is the default for that tag.

3.3. Signing and Verification Algorithms

DKIM supports multiple digital signature algorithms. Two algorithms are defined by this specification at this time: rsa-sha1 and rsa-sha256. The rsa-sha256 algorithm is the default if no algorithm is specified. Verifiers MUST implement both rsa-sha1 and rsa-sha256. Signers MUST implement and SHOULD sign using rsa-sha256.

INFORMATIVE NOTE: Although sha256 is strongly encouraged, some senders of low-security messages (such as routine newsletters) may prefer to use sha1 because of reduced CPU requirements to compute a sha1 hash. In general, sha256 should always be used whenever possible.

3.3.1. The rsa-sha1 Signing Algorithm

The rsa-sha1 Signing Algorithm computes a message hash as described in Section 3.7 below using SHA-1 [FIPS.180-2.2002] as the hash-alg. That hash is then signed by the signer using the RSA algorithm (defined in PKCS#1 version 1.5 [RFC3447]) as the crypt-alg and the signer's private key. The hash MUST NOT be truncated or converted into any form other than the native binary form before being signed. The signing algorithm SHOULD use a public exponent of 65537.

3.3.2. The rsa-sha256 Signing Algorithm

The rsa-sha256 Signing Algorithm computes a message hash as described in Section 3.7 below using SHA-256 [FIPS.180-2.2002] as the hash-alg. That hash is then signed by the signer using the RSA algorithm (defined in PKCS#1 version 1.5 [RFC3447]) 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.

3.3.3. Key Sizes

Selecting appropriate key sizes is a trade-off between cost, performance, and risk. Since short RSA keys more easily succumb to off-line attacks, signers MUST use RSA keys of at least 1024 bits for long-lived keys. Verifiers MUST be able to validate signatures with keys ranging from 512 bits to 2048 bits, and they MAY be able to validate signatures with larger keys. Verifier policies may use the length of the signing key as one metric for determining whether a signature is acceptable.

Factors that should influence the key size choice include the following:

- The practical constraint that large (e.g., 4096 bit) keys may not fit within a 512-byte DNS UDP response packet
- The security constraint that keys smaller than 1024 bits are subject to off-line attacks
- Larger keys impose higher CPU costs to verify and sign email

- Keys can be replaced on a regular basis, thus their lifetime can be relatively short

- The security goals of this specification are modest compared to typical goals of other systems that employ digital signatures

See [RFC3766] for further discussion on selecting key sizes.

3.3.4. Other Algorithms

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

3.4. Canonicalization

Empirical evidence demonstrates that some mail servers and relay systems modify email in transit, potentially invalidating a signature. There are two competing perspectives on such modifications. For most signers, mild modification of email is immaterial to the authentication status of the email. For such signers, a canonicalization algorithm that survives modest in-transit modification is preferred.

Other signers demand that any modification of the email, however minor, result in a signature verification failure. These signers prefer a canonicalization algorithm that does not tolerate in-transit modification of the signed email.

Some signers may be willing to accept modifications to header fields that are within the bounds of email standards such as [RFC2822], but are unwilling to accept any modification to the body of messages.

To satisfy all requirements, two canonicalization algorithms are defined for each of the header and the body: a "simple" algorithm that tolerates almost no modification and a "relaxed" algorithm that tolerates common modifications such as whitespace replacement and header field line rewrapping. A signer MAY specify either algorithm for header or body when signing an email. If no canonicalization algorithm is specified by the signer, the "simple" algorithm defaults for both header and body. Verifiers MUST implement both canonicalization algorithms. Note that the header and body may use different canonicalization algorithms. Further canonicalization algorithms MAY be defined in the future; verifiers MUST ignore any signatures that use unrecognized canonicalization algorithms.

Canonicalization simply prepares the email for presentation to the signing or verification algorithm. It MUST NOT change the

transmitted data in any way. Canonicalization of header fields and body are described below.

NOTE: This section assumes that the message is already in "network normal" format (text is ASCII encoded, lines are separated with CRLF characters, etc.). See also Section 5.3 for information about normalizing the message.

3.4.1. The "simple" Header Canonicalization Algorithm

The "simple" header canonicalization algorithm does not change header fields in any way. Header fields MUST be presented to the signing or verification algorithm exactly as they are in the message being signed or verified. In particular, header field names MUST NOT be case folded and whitespace MUST NOT be changed.

3.4.2. The "relaxed" Header Canonicalization Algorithm

The "relaxed" header canonicalization algorithm MUST apply the following steps in 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 [RFC2822]; 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.

3.4.3. The "simple" Body Canonicalization Algorithm

The "simple" body canonicalization algorithm ignores all empty lines at the end of the message body. An empty line is a line of zero length after removal of the line terminator. If there is no body or no trailing CRLF on the message body, a CRLF is added. It makes no

other changes to the message body. In more formal terms, the "simple" body canonicalization algorithm converts "0*CRLF" at the end of the body to a single "CRLF".

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

3.4.4. The "relaxed" Body Canonicalization Algorithm

The "relaxed" body canonicalization algorithm:

- Ignores all whitespace at the end of lines. Implementations MUST NOT remove the CRLF at the end of the line.

- Reduces all sequences of WSP within a line to a single SP character.

- Ignores all empty lines at the end of the message body. "Empty line" is defined in Section 3.4.3.

  INFORMATIVE NOTE: It should be noted that the relaxed body canonicalization algorithm may enable certain types of extremely crude "ASCII Art" attacks where a message may be conveyed by adjusting the spacing between words. If this is a concern, the "simple" body canonicalization algorithm should be used instead.

3.4.5. Body Length Limits

A body length count MAY be specified to limit the signature calculation to an initial prefix of the body text, measured in octets. If the body length count is not specified, the entire message body is signed.

INFORMATIVE RATIONALE: This capability is provided because it is very common for mailing lists to add trailers to messages (e.g., instructions how to get off the list). Until those messages are also signed, the body length count is a useful tool for the verifier since it may as a matter of policy accept messages having valid signatures with extraneous data.

INFORMATIVE IMPLEMENTATION NOTE: Using body length limits enables an attack in which an attacker modifies a message to include content that solely benefits the attacker. It is possible for the appended content to completely replace the original content in the end recipient's eyes and to defeat duplicate message detection algorithms. To avoid this attack, signers should be wary of using

this tag, and verifiers might wish to ignore the tag or remove text that appears after the specified content length, perhaps based on other criteria.

The body length count allows the signer of a message to permit data to be appended to the end of the body of a signed message. The body length count MUST be calculated following the canonicalization algorithm; for example, any whitespace ignored by a canonicalization algorithm is not included as part of the body length count. Signers of MIME messages that include a body length count SHOULD be sure that the length extends to the closing MIME boundary string.

INFORMATIVE IMPLEMENTATION NOTE: A signer wishing to ensure that the only acceptable modifications are to add to the MIME postlude would use a body length count encompassing the entire final MIME boundary string, including the final "--CRLF". A signer wishing to allow additional MIME parts but not modification of existing parts would use a body length count extending through the final MIME boundary string, omitting the final "--CRLF". Note that this only works for some MIME types, e.g., multipart/mixed but not multipart/signed.

A body length count of zero means that the body is completely unsigned.

Signers wishing to ensure that no modification of any sort can occur should specify the "simple" canonicalization algorithm for both header and body and omit the body length count.

3.4.6. Canonicalization Examples (INFORMATIVE)

In the following examples, actual whitespace is used only for clarity. The actual input and output text is designated using bracketed descriptors: "<SP>" for a space character, "<HTAB>" for a tab character, and "<CRLF>" for a carriage-return/line-feed sequence. For example, "X <SP> Y" and "X<SP>Y" represent the same three characters.

Example 1: A message reading:

```
A: <SP> X <CRLF>
B <SP> : <SP> Y <HTAB><CRLF>
<HTAB> Z <SP><SP><CRLF>
<CRLF>
<SP> C <SP><CRLF>
D <SP><HTAB><SP> E <CRLF>
<CRLF>
<CRLF>
```

when canonicalized using relaxed canonicalization for both header and body results in a header reading:

```
a:X <CRLF>
b:Y <SP> Z <CRLF>
```

and a body reading:

```
<SP> C <CRLF>
D <SP> E <CRLF>
```

Example 2: The same message canonicalized using simple canonicalization for both header and body results in a header reading:

```
A: <SP> X <CRLF>
B <SP> : <SP> Y <HTAB><CRLF>
<HTAB> Z <SP><SP><CRLF>
```

and a body reading:

```
<SP> C <SP><CRLF>
D <SP><HTAB><SP> E <CRLF>
```

Example 3: When processed using relaxed header canonicalization and simple body canonicalization, the canonicalized version has a header of:

```
a:X <CRLF>
b:Y <SP> Z <CRLF>
```

and a body reading:

```
<SP> C <SP><CRLF>
D <SP><HTAB><SP> E <CRLF>
```

3.5. The DKIM-Signature Header Field

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

The DKIM-Signature header field SHOULD be treated as though it were a trace header field as defined in Section 3.6 of [RFC2822], and hence SHOULD NOT be reordered and SHOULD be prepended to the message.

The DKIM-Signature header field being created or verified is always included in the signature calculation, after the rest of the header fields being signed; however, when calculating or verifying the signature, the value of the "b=" tag (signature value) of that DKIM-Signature header field MUST be treated as though it were an empty string. Unknown tags in the DKIM-Signature header field MUST be included in the signature calculation but MUST be otherwise ignored by verifiers. Other DKIM-Signature header fields that are included in the signature should be treated as normal header fields; in particular, the "b=" tag is not treated specially.

The encodings for each field type are listed below. Tags described as qp-section are encoded as described in Section 6.7 of MIME Part One [RFC2045], with the additional conversion of semicolon characters to "=3B"; intuitively, this is one line of quoted-printable encoded text. The dkim-quoted-printable syntax is defined in Section 2.6.

Tags on the DKIM-Signature header field along with their type and requirement status are shown below. Unrecognized tags MUST be ignored.

v= Version (MUST be included). This tag defines the version of this specification that applies to the signature record. It MUST have the value "1". Note that verifiers must do a string comparison on this value; for example, "1" is not the same as "1.0".

ABNF:

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

INFORMATIVE NOTE: DKIM-Signature version numbers are expected to increase arithmetically as new versions of this specification are released.

a= The algorithm used to generate the signature (plain-text; REQUIRED). Verifiers MUST support "rsa-sha1" and "rsa-sha256"; signers SHOULD sign using "rsa-sha256". See Section 3.3 for a description of algorithms.

ABNF:

```
sig-a-tag       = %x61 [FWS] "=" [FWS] sig-a-tag-alg
sig-a-tag-alg   = sig-a-tag-k "-" sig-a-tag-h
sig-a-tag-k     = "rsa" / x-sig-a-tag-k
sig-a-tag-h     = "sha1" / "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
```

b= 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 5) for how the signature is computed.

ABNF:

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

bh= The hash of the canonicalized body part of the message as limited by the "l=" tag (base64; REQUIRED). Whitespace is ignored in this value and MUST be ignored when reassembling the original signature. In particular, the signing process can safely insert FWS in this value in arbitrary places to conform to line-length limits. See Section 3.7 for how the body hash is computed.

ABNF:

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

c= Message canonicalization (plain-text; OPTIONAL, default is "simple/simple"). This tag informs the verifier of the type of canonicalization used to prepare the message for signing. It consists of two names separated by a "slash" (%d47) character, corresponding to the header and body canonicalization algorithms respectively. These algorithms are described in Section 3.4. If only one algorithm is named, that algorithm is used for the header and "simple" is used for the body. For example, "c=relaxed" is treated the same as "c=relaxed/simple".

ABNF:

```
sig-c-tag       = %x63 [FWS] "=" [FWS] sig-c-tag-alg
                ["/" sig-c-tag-alg]
sig-c-tag-alg   = "simple" / "relaxed" / x-sig-c-tag-alg
x-sig-c-tag-alg = hyphenated-word    ; for later extension
```

d= The domain of the signing entity (plain-text; REQUIRED). This is the domain that will be queried for the public key. This domain MUST be the same as or a parent domain of the "i=" tag (the signing identity, as described below), or it MUST meet the requirements for parent domain signing described in Section 3.8. When presented with a signature that does not meet these requirement, verifiers MUST consider the signature invalid.

Internationalized domain names MUST be encoded as described in [RFC3490].

ABNF:

```
sig-d-tag       = %x64 [FWS] "=" [FWS] domain-name
domain-name     = sub-domain 1*("." sub-domain)
          ; from RFC 2821 Domain, but excluding address-literal
```

h= Signed header fields (plain-text, but see description; REQUIRED). A colon-separated list of header field names that identify the header fields presented to the signing algorithm. The field MUST contain the complete list of header fields in the order presented to the signing algorithm. The field MAY contain names of header fields that do not exist when signed; nonexistent header fields do not contribute to the signature computation (that is, they are treated as the null input, including the header field name, the separating colon, the header field value, and any CRLF terminator). The field MUST NOT include the DKIM-Signature header field that is being created or verified, but may include others. Folding whitespace (FWS) MAY be included on either side of the colon separator. Header field names MUST be compared against actual header field names in a case-insensitive manner. This list MUST NOT be empty. See Section 5.4 for a discussion of choosing header fields to sign.

ABNF:

```
sig-h-tag       = %x68 [FWS] "=" [FWS] hdr-name
                0*( *FWS ":" *FWS hdr-name )
hdr-name        = field-name
```

INFORMATIVE EXPLANATION: By "signing" header fields that do not actually exist, a signer can prevent insertion of those header fields before verification. However, since a signer cannot possibly know what header fields might be created in the future, and that some MUAs might present header fields that are embedded inside a message (e.g., as a message/rfc822 content type), the security of this solution is not total.

INFORMATIVE EXPLANATION: The exclusion of the header field name and colon as well as the header field value for non-existent header fields prevents an attacker from inserting an actual header field with a null value.

i= Identity of the user or agent (e.g., a mailing list manager) on behalf of which this message is signed (dkim-quoted-printable; OPTIONAL, default is an empty Local-part followed by an "@" followed by the domain from the "d=" tag). The syntax is a standard email address where the Local-part MAY be omitted. The domain part of the address MUST be the same as or a subdomain of the value of the "d=" tag.

Internationalized domain names MUST be converted using the steps listed in Section 4 of [RFC3490] using the "ToASCII" function.

ABNF:

```
sig-i-tag =   %x69 [FWS] "=" [FWS] [ Local-part ] "@" domain-name
```

INFORMATIVE NOTE: The Local-part of the "i=" tag is optional because in some cases a signer may not be able to establish a verified individual identity. In such cases, the signer may wish to assert that although it is willing to go as far as signing for the domain, it is unable or unwilling to commit to an individual user name within their domain. It can do so by including the domain part but not the Local-part of the identity.

INFORMATIVE DISCUSSION: This document does not require the value of the "i=" tag to match the identity in any message header fields. This is considered to be a verifier policy issue. Constraints between the value of the "i=" tag and other identities in other header fields seek to apply basic authentication into the semantics of trust associated with a role such as content author. Trust is a broad and complex topic and trust mechanisms are subject to highly creative attacks. The real-world efficacy of any but the most basic bindings between the "i=" value and other identities is not well established, nor is its vulnerability to subversion by an attacker. Hence reliance on the use of these options should be strictly limited. In particular, it is not at all clear to what extent a typical end-user recipient can rely on any assurances that might be made by successful use of the "i=" options.

l= Body length count (plain-text unsigned decimal integer; OPTIONAL, default is entire body). This tag informs the verifier of the number of octets in the body of the email after canonicalization included in the cryptographic hash, starting from 0 immediately following the CRLF preceding the body. This value MUST NOT be larger than the actual number of octets in the canonicalized message body.

INFORMATIVE IMPLEMENTATION WARNING: Use of the "l=" tag might allow display of fraudulent content without appropriate warning to end users. The "l=" tag is intended for increasing signature robustness when sending to mailing lists that both modify their content and do not sign their messages. However, using the "l=" tag enables attacks in which an intermediary with malicious intent modifies a message to include content that solely benefits the attacker. It is possible for the appended content to completely replace the original content in the end recipient's eyes and to defeat duplicate message detection algorithms. Examples are described in Security Considerations (Section 8). To avoid this attack, signers should be extremely wary of using this tag, and verifiers might wish to ignore the tag or remove text that appears after the specified content length.

INFORMATIVE NOTE: The value of the "l=" tag is constrained to 76 decimal digits. This constraint is not intended to predict the size of future messages or to require implementations to use an integer representation large enough to represent the maximum possible value, but is intended to remind the implementer to check the length of this and all other tags during verification and to test for integer overflow when decoding the value. Implementers may need to limit the actual value expressed to a value smaller than 10^76, e.g., to allow a message to fit within the available storage space.

ABNF:

```
sig-l-tag    = %x6c [FWS] "=" [FWS] 1*76DIGIT
```

q= A colon-separated list of query methods used to retrieve the public key (plain-text; OPTIONAL, default is "dns/txt"). Each query method is of the form "type[/options]", where the syntax and semantics of the options depend on the type and specified options. If there are multiple query mechanisms listed, the choice of query mechanism MUST NOT change the interpretation of the signature. Implementations MUST use the recognized query mechanisms in the order presented.

Currently, the only valid value is "dns/txt", which defines the DNS TXT record lookup algorithm described elsewhere in this document. The only option defined for the "dns" query type is "txt", which MUST be included. Verifiers and signers MUST support "dns/txt".

ABNF:

```
    sig-q-tag        = %x71 [FWS] "=" [FWS] sig-q-tag-method
                     *([FWS] ":" [FWS] sig-q-tag-method)
    sig-q-tag-method = "dns/txt" / x-sig-q-tag-type
                     ["/" x-sig-q-tag-args]
    x-sig-q-tag-type = hyphenated-word  ; for future extension
    x-sig-q-tag-args = qp-hdr-value
```

s= The selector subdividing the namespace for the "d=" (domain) tag (plain-text; REQUIRED).

ABNF:

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

t= Signature Timestamp (plain-text unsigned decimal integer; RECOMMENDED, default is an unknown creation time). The time that this signature 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). To avoid denial-of-service attacks, implementations MAY consider any value longer than 12 digits to be infinite. Leap seconds are not counted. Implementations MAY ignore signatures that have a timestamp in the future.

ABNF:

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

x= Signature Expiration (plain-text unsigned decimal integer; RECOMMENDED, default is no expiration). The format is the same as in the "t=" tag, represented as an absolute date, not as a time delta from the signing timestamp. The value is expressed as an unsigned integer in decimal ASCII, with the same constraints on the value in the "t=" tag. Signatures MAY be considered invalid if the verification time at the verifier is past the expiration date. The verification time should be the time that the message was first received at the administrative domain of the verifier if that time is reliably available; otherwise the current time should be used. The value of the "x=" tag MUST be greater than the value of the "t=" tag if both are present.

INFORMATIVE NOTE: The "x=" tag is not intended as an anti-replay defense.

ABNF:

```
sig-x-tag    = %x78 [FWS] "=" [FWS] 1*12DIGIT
```

z= Copied header fields (dkim-quoted-printable, but see description; OPTIONAL, default is null). A vertical-bar-separated list of selected header fields present when the message was signed, including both the field name and value. It is not required to include all header fields present at the time of signing. This field need not contain the same header fields listed in the "h=" tag. The header field text itself must encode the vertical bar ("|", %x7C) character (i.e., vertical bars in the "z=" text are metacharacters, and any actual vertical bar characters in a copied header field must be encoded). Note that all whitespace must be encoded, including whitespace between the colon and the header field value. After encoding, FWS MAY be added at arbitrary locations in order to avoid excessively long lines; such whitespace is NOT part of the value of the header field, and MUST be removed before decoding.

The header fields referenced by the "h=" tag refer to the fields in the RFC 2822 header of the message, not to any copied fields in the "z=" tag. Copied header field values are for diagnostic use.

Header fields with characters requiring conversion (perhaps from legacy MTAs that are not [RFC2822] compliant) SHOULD be converted as described in MIME Part Three [RFC2047].

ABNF:

```
    sig-z-tag      = %x7A [FWS] "=" [FWS] sig-z-tag-copy
                   *( [FWS] "|" sig-z-tag-copy )
sig-z-tag-copy = hdr-name ":" qp-hdr-value
qp-hdr-value   = dkim-quoted-printable    ; with "|" encoded
```

INFORMATIVE EXAMPLE of a signature header field spread across multiple continuation lines:

```
DKIM-Signature: a=rsa-sha256; d=example.net; s=brisbane;
   c=simple; q=dns/txt; i=@eng.example.net;
   t=1117574938; x=1118006938;
   h=from:to:subject:date;
   z=From:foo@eng.example.net|To:joe@example.com|
     Subject:demo=20run|Date:July=205,=202005=203:44:08=20PM=20-0700;
   bh=MTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTI=;
   b=dzdVyOfAKCdLXdJOc9G2q8LoXSlEniSbav+yuU4zGeeruD00lszZ
            VoG4ZHRNiYzR
```

3.6. Key Management and Representation

Signature applications require some level of assurance that the verification public key is associated with the claimed signer. Many applications achieve this by using public key certificates issued by a trusted third party. However, DKIM can achieve a sufficient level of security, with significantly enhanced scalability, by simply having the verifier query the purported signer's DNS entry (or some security-equivalent) in order to retrieve the public key.

DKIM keys can potentially be stored in multiple types of key servers and in multiple formats. The storage and format of keys are irrelevant to the remainder of the DKIM algorithm.

Parameters to the key lookup algorithm are the type of the lookup (the "q=" tag), the domain of the signer (the "d=" tag of the DKIM-Signature header field), and the selector (the "s=" tag).

```
    public_key = dkim_find_key(q_val, d_val, s_val)
```

This document defines a single binding, using DNS TXT records to distribute the keys. Other bindings may be defined in the future.

3.6.1. Textual Representation

It is expected that many key servers will choose to present the keys in an otherwise unstructured text format (for example, an XML form would not be considered to be unstructured text for this purpose). The following definition MUST be used for any DKIM key represented in an otherwise unstructured textual form.

The overall syntax is a tag-list as described in Section 3.2. The current valid tags are described below. Other tags MAY be present and MUST be ignored by any implementation that does not understand them.

v= Version of the DKIM key record (plain-text; RECOMMENDED, default is "DKIM1"). If specified, this tag MUST be set to "DKIM1" (without the quotes). This tag MUST be the first tag in the record. Records beginning with a "v=" tag with any other value MUST be discarded. Note that verifiers must do a string comparison on this value; for example, "DKIM1" is not the same as "DKIM1.0".

ABNF:

```
key-v-tag    = %x76 [FWS] "=" [FWS] "DKIM1"
```

g= Granularity of the key (plain-text; OPTIONAL, default is "*"). This value MUST match the Local-part of the "i=" tag of the DKIM-Signature header field (or its default value of the empty string if "i=" is not specified), with a single, optional "*" character matching a sequence of zero or more arbitrary characters ("wildcarding"). An email with a signing address that does not match the value of this tag constitutes a failed verification. The intent of this tag is to constrain which signing address can legitimately use this selector, for example, when delegating a key to a third party that should only be used for special purposes. Wildcarding allows matching for addresses such as "user+*" or "*-offer". An empty "g=" value never matches any addresses.

ABNF:

```
key-g-tag       = %x67 [FWS] "=" [FWS] key-g-tag-lpart
key-g-tag-lpart = [dot-atom-text] ["*" [dot-atom-text] ]
```

h= Acceptable hash algorithms (plain-text; OPTIONAL, defaults to allowing all algorithms). A colon-separated list of hash algorithms that might be used. Signers and Verifiers MUST support the "sha256" hash algorithm. Verifiers MUST also support the "sha1" hash algorithm.

ABNF:

```
key-h-tag       = %x68 [FWS] "=" [FWS] key-h-tag-alg
                0*( [FWS] ":" [FWS] key-h-tag-alg )
key-h-tag-alg   = "sha1" / "sha256" / x-key-h-tag-alg
x-key-h-tag-alg = hyphenated-word   ; for future extension
```

k= Key type (plain-text; OPTIONAL, default is "rsa"). Signers and verifiers MUST support the "rsa" key type. The "rsa" key type indicates that an ASN.1 DER-encoded [ITU.X660.1997] RSAPublicKey [RFC3447] (see Sections 3.1 and A.1.1) is being used in the "p=" tag. (Note: the "p=" tag further encodes the value using the base64 algorithm.)

ABNF:

```
key-k-tag        = %x76 [FWS] "=" [FWS] key-k-tag-type
key-k-tag-type   = "rsa" / x-key-k-tag-type
x-key-k-tag-type = hyphenated-word   ; for future extension
```

n= Notes that might be of interest to a human (qp-section; OPTIONAL, default is empty). No interpretation is made by any program. This tag should be used sparingly in any key server mechanism that has space limitations (notably DNS). This is intended for use by administrators, not end users.

ABNF:

```
key-n-tag    = %x6e [FWS] "=" [FWS] qp-section
```

p= Public-key data (base64; REQUIRED). An empty value means that this public key has been revoked. The syntax and semantics of this tag value before being encoded in base64 are defined by the "k=" tag.

INFORMATIVE RATIONALE: If a private key has been compromised or otherwise disabled (e.g., an outsourcing contract has been terminated), a signer might want to explicitly state that it knows about the selector, but all messages using that selector should fail verification. Verifiers should ignore any DKIM-Signature header fields with a selector referencing a revoked key.

ABNF:

```
key-p-tag    = %x70 [FWS] "=" [ [FWS] base64string ]
```

INFORMATIVE NOTE: A base64string is permitted to include white space (FWS) at arbitrary places; however, any CRLFs must be followed by at least one WSP character. Implementors and administrators are cautioned to ensure that selector TXT records conform to this specification.

s= Service Type (plain-text; OPTIONAL; default is "*"). A colon-separated list of service types to which this record applies. Verifiers for a given service type MUST ignore this record if the appropriate type is not listed. Currently defined service types are as follows:

* matches all service types

email electronic mail (not necessarily limited to SMTP)

This tag is intended to constrain the use of keys for other purposes, should use of DKIM be defined by other services in the future.

ABNF:

```
key-s-tag        = %x73 [FWS] "=" [FWS] key-s-tag-type
                 0*( [FWS] ":" [FWS] key-s-tag-type )
key-s-tag-type   = "email" / "*" / x-key-s-tag-type
x-key-s-tag-type = hyphenated-word   ; for future extension
```

t= Flags, represented as a colon-separated list of names (plain-text; OPTIONAL, default is no flags set). The defined flags are as follows:

y This domain is testing DKIM. Verifiers MUST NOT treat messages from signers in testing mode differently from unsigned email, even should the signature fail to verify. Verifiers MAY wish to track testing mode results to assist the signer.

s Any DKIM-Signature header fields using the "i=" tag MUST have the same domain value on the right-hand side of the "@" in the "i=" tag and the value of the "d=" tag. That is, the "i=" domain MUST NOT be a subdomain of "d=". Use of this flag is RECOMMENDED unless subdomaining is required.

ABNF:

```
key-t-tag        = %x74 [FWS] "=" [FWS] key-t-tag-flag
                 0*( [FWS] ":" [FWS] key-t-tag-flag )
key-t-tag-flag   = "y" / "s" / x-key-t-tag-flag
x-key-t-tag-flag = hyphenated-word   ; for future extension
```

Unrecognized flags MUST be ignored.

3.6.2. DNS Binding

A binding using DNS TXT records as a key service is hereby defined. All implementations MUST support this binding.

3.6.2.1. Namespace

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

> INFORMATIVE OPERATIONAL NOTE: Wildcard DNS records (e.g., *.bar._domainkey.example.com) do not make sense in this context and should not be used. Note also that wildcards within domains (e.g., s._domainkey.*.example.com) are not supported by the DNS.

3.6.2.2. Resource Record Types for Key Storage

The DNS Resource Record type used is specified by an option to the query-type ("q=") tag. The only option defined in this base specification is "txt", indicating the use of a TXT Resource Record (RR). A later extension of this standard may define another RR type.

Strings in a TXT RR MUST be concatenated together before use with no intervening whitespace. TXT RRs MUST be unique for a particular selector name; that is, if there are multiple records in an RRset, the results are undefined.

TXT RRs are encoded as described in Section 3.6.1.

3.7. Computing the Message Hashes

Both signing and verifying message signatures start with a step of computing two cryptographic hashes over the message. Signers will choose the parameters of the signature as described in Signer Actions (Section 5); verifiers will use the parameters specified in the DKIM-Signature header field being verified. In the following discussion, the names of the tags in the DKIM-Signature header field that either exists (when verifying) or will be created (when signing) are used. Note that canonicalization (Section 3.4) is only used to prepare the email for signing or verifying; it does not affect the transmitted email in any way.

The signer/verifier MUST compute two hashes, one over the body of the message and one over the selected header fields of the message.

Signers MUST compute them in the order shown. Verifiers MAY compute them in any order convenient to the verifier, provided that the result is semantically identical to the semantics that would be the case had they been computed in this order.

In hash step 1, the signer/verifier MUST hash the message body, canonicalized using the body canonicalization algorithm specified in the "c=" tag and then truncated to the length specified in the "l=" tag. That hash value is then converted to base64 form and inserted into (signers) or compared to (verifiers) the "bh=" tag of the DKIM-Signature header field.

In hash step 2, the signer/verifier MUST pass the following to the hash algorithm in the indicated order.

1. The header fields specified by the "h=" tag, in the order specified in that tag, and canonicalized using the header canonicalization algorithm specified in the "c=" tag. Each header field MUST be terminated with a single CRLF.

2. The DKIM-Signature header field that exists (verifying) or will be inserted (signing) in the message, with the value of the "b=" tag deleted (i.e., treated as the empty string), canonicalized using the header canonicalization algorithm specified in the "c=" tag, and without a trailing CRLF.

All tags and their values in the DKIM-Signature header field are included in the cryptographic hash with the sole exception of the value portion of the "b=" (signature) tag, which MUST be treated as the null string. All tags MUST be included even if they might not be understood by the verifier. The header field MUST be presented to the hash algorithm after the body of the message rather than with the rest of the header fields and MUST be canonicalized as specified in the "c=" (canonicalization) tag. The DKIM-Signature header field MUST NOT be included in its own h= tag, although other DKIM-Signature header fields MAY be signed (see Section 4).

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 [RFC2821]).

With the exception of the canonicalization procedure described in Section 3.4, the DKIM signing process treats the body of messages as

simply a string of octets. DKIM 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.

More formally, the algorithm for the signature is as follows:

```
body-hash = hash-alg(canon_body)
header-hash = hash-alg(canon_header || DKIM-SIG)
signature = sig-alg(header-hash, key)
```

where "sig-alg" is the signature algorithm specified by the "a=" tag, "hash-alg" is the hash algorithm specified by the "a=" tag, "canon_header" and "canon_body" are the canonicalized message header and body (respectively) as defined in Section 3.4 (excluding the DKIM-Signature header field), and "DKIM-SIG" is the canonicalized DKIM-Signature header field sans the signature value itself, but with "body-hash" included as the "bh=" tag.

> INFORMATIVE IMPLEMENTERS' NOTE: Many digital signature APIs provide both hashing and application of the RSA private key using a single "sign()" primitive. When using such an API, the last two steps in the algorithm would probably be combined into a single call that would perform both the "hash-alg" and the "sig-alg".

3.8. Signing by Parent Domains

In some circumstances, it is desirable for a domain to apply a signature on behalf of any of its subdomains without the need to maintain separate selectors (key records) in each subdomain. By default, private keys corresponding to key records can be used to sign messages for any subdomain of the domain in which they reside; e.g., a key record for the domain example.com can be used to verify messages where the signing identity ("i=" tag of the signature) is sub.example.com, or even sub1.sub2.example.com. In order to limit the capability of such keys when this is not intended, the "s" flag may be set in the "t=" tag of the key record to constrain the validity of the record to exactly the domain of the signing identity. If the referenced key record contains the "s" flag as part of the "t=" tag, the domain of the signing identity ("i=" flag) MUST be the same as that of the d= domain. If this flag is absent, the domain of the signing identity MUST be the same as, or a subdomain of, the d= domain. Key records that are not intended for use with subdomains SHOULD specify the "s" flag in the "t=" tag.

4. Semantics of Multiple Signatures

4.1. Example Scenarios

There are many reasons why a message might have multiple signatures. For example, a given signer might sign multiple times, perhaps with different hashing or signing algorithms during a transition phase.

> INFORMATIVE EXAMPLE: Suppose SHA-256 is in the future found to be insufficiently strong, and DKIM usage transitions to SHA-1024. A signer might immediately sign using the newer algorithm, but continue to sign using the older algorithm for interoperability with verifiers that had not yet upgraded. The signer would do this by adding two DKIM-Signature header fields, one using each algorithm. Older verifiers that did not recognize SHA-1024 as an acceptable algorithm would skip that signature and use the older algorithm; newer verifiers could use either signature at their option, and all other things being equal might not even attempt to verify the other signature.

Similarly, a signer might sign a message including all headers and no "l=" tag (to satisfy strict verifiers) and a second time with a limited set of headers and an "l=" tag (in anticipation of possible message modifications in route to other verifiers). Verifiers could then choose which signature they preferred.

> INFORMATIVE EXAMPLE: A verifier might receive a message with two signatures, one covering more of the message than the other. If the signature covering more of the message verified, then the verifier could make one set of policy decisions; if that signature failed but the signature covering less of the message verified, the verifier might make a different set of policy decisions.

Of course, a message might also have multiple signatures because it passed through multiple signers. A common case is expected to be that of a signed message that passes through a mailing list that also signs all messages. Assuming both of those signatures verify, a recipient might choose to accept the message if either of those signatures were known to come from trusted sources.

> INFORMATIVE EXAMPLE: Recipients might choose to whitelist mailing lists to which they have subscribed and that have acceptable anti-abuse policies so as to accept messages sent to that list even from unknown authors. They might also subscribe to less trusted mailing lists (e.g., those without anti-abuse protection) and be willing to accept all messages from specific authors, but insist on doing additional abuse scanning for other messages.

Another related example of multiple signers might be forwarding services, such as those commonly associated with academic alumni sites.

> INFORMATIVE EXAMPLE: A recipient might have an address at members.example.org, a site that has anti-abuse protection that is somewhat less effective than the recipient would prefer. Such a recipient might have specific authors whose messages would be trusted absolutely, but messages from unknown authors that had passed the forwarder's scrutiny would have only medium trust.

4.2. Interpretation

A signer that is adding a signature to a message merely creates a new DKIM-Signature header, using the usual semantics of the h= option. A signer MAY sign previously existing DKIM-Signature header fields using the method described in Section 5.4 to sign trace header fields.

> INFORMATIVE NOTE: Signers should be cognizant that signing DKIM-Signature header fields may result in signature failures with intermediaries that do not recognize that DKIM-Signature header fields are trace header fields and unwittingly reorder them, thus breaking such signatures. For this reason, signing existing DKIM-Signature header fields is unadvised, albeit legal.

> INFORMATIVE NOTE: If a header field with multiple instances is signed, those header fields are always signed from the bottom up. Thus, it is not possible to sign only specific DKIM-Signature header fields. For example, if the message being signed already contains three DKIM-Signature header fields A, B, and C, it is possible to sign all of them, B and C only, or C only, but not A only, B only, A and B only, or A and C only.

A signer MAY add more than one DKIM-Signature header field using different parameters. For example, during a transition period a signer might want to produce signatures using two different hash algorithms.

Signers SHOULD NOT remove any DKIM-Signature header fields from messages they are signing, even if they know that the signatures cannot be verified.

When evaluating a message with multiple signatures, a verifier SHOULD evaluate signatures independently and on their own merits. For example, a verifier that by policy chooses not to accept signatures with deprecated cryptographic algorithms would consider such signatures invalid. Verifiers MAY process signatures in any order of

their choice; for example, some verifiers might choose to process signatures corresponding to the From field in the message header before other signatures. See Section 6.1 for more information about signature choices.

> INFORMATIVE IMPLEMENTATION NOTE: Verifier attempts to correlate valid signatures with invalid signatures in an attempt to guess why a signature failed are ill-advised. In particular, there is no general way that a verifier can determine that an invalid signature was ever valid.

Verifiers SHOULD ignore failed signatures as though they were not present in the message. Verifiers SHOULD continue to check signatures until a signature successfully verifies to the satisfaction of the verifier. To limit potential denial-of-service attacks, verifiers MAY limit the total number of signatures they will attempt to verify.

5. Signer Actions

The following steps are performed in order by signers.

5.1. Determine Whether the Email Should Be Signed and by Whom

A signer can obviously only sign email for domains for which it has a private key and the necessary knowledge of the corresponding public key and selector information. However, there are a number of other reasons beyond the lack of a private key why a signer could choose not to sign an email.

> INFORMATIVE NOTE: Signing modules may be incorporated into any portion of the mail system as deemed appropriate, including an MUA, a SUBMISSION server, or an MTA. Wherever implemented, signers should beware of signing (and thereby asserting responsibility for) messages that may be problematic. In particular, within a trusted enclave the signing address might be derived from the header according to local policy; SUBMISSION servers might only sign messages from users that are properly authenticated and authorized.
>
> INFORMATIVE IMPLEMENTER ADVICE: SUBMISSION servers should not sign Received header fields if the outgoing gateway MTA obfuscates Received header fields, for example, to hide the details of internal topology.

If an email cannot be signed for some reason, it is a local policy decision as to what to do with that email.

5.2. Select a Private Key and Corresponding Selector Information

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

> INFORMATIVE OPERATIONS ADVICE: A signer should not sign with a private key when the selector containing the corresponding public key is expected to be revoked or removed before the verifier has an opportunity to validate the signature. The signer should anticipate that verifiers may choose to defer validation, perhaps until the message is actually read by the final recipient. In particular, when rotating to a new key pair, signing should immediately commence with the new private key and the old public key should be retained for a reasonable validation interval before being removed from the key server.

5.3. Normalize the Message to Prevent Transport Conversions

Some messages, particularly those using 8-bit characters, are subject to modification during transit, notably conversion to 7-bit form. Such conversions will break DKIM signatures. In order to minimize the chances of such breakage, signers SHOULD convert the message to a suitable MIME content transfer encoding such as quoted-printable or base64 as described in MIME Part One [RFC2045] before signing. Such conversion is outside the scope of DKIM; the actual message SHOULD be converted to 7-bit MIME by an MUA or MSA prior to presentation to the DKIM algorithm.

If the message is submitted to the signer with any local encoding that will be modified before transmission, that modification to canonical [RFC2822] 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.

5.4. Determine the Header Fields to Sign

The From header field MUST be signed (that is, included in the "h=" tag of the resulting DKIM-Signature header field). Signers SHOULD NOT sign an existing header field likely to be legitimately modified or removed in transit. In particular, [RFC2821] explicitly permits modification or removal of the Return-Path header field in transit. Signers MAY include any other header fields present at the time of signing at the discretion of the signer.

> INFORMATIVE OPERATIONS NOTE: The choice of which header fields to sign is non-obvious. One strategy is to sign all existing, non-repeatable header fields. An alternative strategy is to sign only header fields that are likely to be displayed to or otherwise be likely to affect the processing of the message at the receiver. A third strategy is to sign only "well known" headers. Note that verifiers may treat unsigned header fields with extreme skepticism, including refusing to display them to the end user or even ignoring the signature if it does not cover certain header fields. For this reason, signing fields present in the message such as Date, Subject, Reply-To, Sender, and all MIME header fields are highly advised.

The DKIM-Signature header field is always implicitly signed and MUST NOT be included in the "h=" tag except to indicate that other preexisting signatures are also signed.

Signers MAY claim to have signed header fields that do not exist (that is, signers MAY include the header field name in the "h=" tag even if that header field does not exist in the message). When computing the signature, the non-existing header field MUST be treated as the null string (including the header field name, header field value, all punctuation, and the trailing CRLF).

> INFORMATIVE RATIONALE: This allows signers to explicitly assert the absence of a header field; if that header field is added later the signature will fail.

> INFORMATIVE NOTE: A header field name need only be listed once more than the actual number of that header field in a message at the time of signing in order to prevent any further additions. For example, if there is a single Comments header field at the time of signing, listing Comments twice in the "h=" tag is sufficient to prevent any number of Comments header fields from being appended; it is not necessary (but is legal) to list Comments three or more times in the "h=" tag.

Signers choosing to sign an existing header field that occurs more than once in the message (such as Received) MUST sign the physically last instance of that header field in the header block. Signers wishing to sign multiple instances of such a header field MUST include the header field name multiple times in the h= tag of the DKIM-Signature header field, and MUST sign such header fields in order from the bottom of the header field block to the top. The signer MAY include more instances of a header field name in h= than there are actual corresponding header fields to indicate that additional header fields of that name SHOULD NOT be added.

INFORMATIVE EXAMPLE:

If the signer wishes to sign two existing Received header fields, and the existing header contains:

```
Received: <A>
Received: <B>
Received: <C>
```

then the resulting DKIM-Signature header field should read:

```
DKIM-Signature: ... h=Received : Received : ...
```

and Received header fields <C> and <B> will be signed in that order.

Signers should be careful of signing header fields that might have additional instances added later in the delivery process, since such header fields might be inserted after the signed instance or otherwise reordered. Trace header fields (such as Received) and Resent-* blocks are the only fields prohibited by [RFC2822] from being reordered. In particular, since DKIM-Signature header fields may be reordered by some intermediate MTAs, signing existing DKIM-Signature header fields is error-prone.

INFORMATIVE ADMONITION: Despite the fact that [RFC2822] permits header fields to be reordered (with the exception of Received header fields), reordering of signed header fields with multiple instances by intermediate MTAs will cause DKIM signatures to be broken; such anti-social behavior should be avoided.

INFORMATIVE IMPLEMENTER'S NOTE: Although not required by this specification, all end-user visible header fields should be signed to avoid possible "indirect spamming". For example, if the Subject header field is not signed, a spammer can resend a previously signed mail, replacing the legitimate subject with a one-line spam.

5.5. Recommended Signature Content

In order to maximize compatibility with a variety of verifiers, it is recommended that signers follow the practices outlined in this section when signing a message. However, these are generic recommendations applying to the general case; specific senders may wish to modify these guidelines as required by their unique situations. Verifiers MUST be capable of verifying signatures even if one or more of the recommended header fields is not signed (with the exception of From, which must always be signed) or if one or more of the disrecommended header fields is signed. Note that verifiers do have the option of ignoring signatures that do not cover a sufficient portion of the header or body, just as they may ignore signatures from an identity they do not trust.

The following header fields SHOULD be included in the signature, if they are present in the message being signed:

- From (REQUIRED in all signatures)
- Sender, Reply-To
- Subject
- Date, Message-ID
- To, Cc
- MIME-Version
- Content-Type, Content-Transfer-Encoding, Content-ID, Content-Description
- Resent-Date, Resent-From, Resent-Sender, Resent-To, Resent-Cc, Resent-Message-ID
- In-Reply-To, References
- List-Id, List-Help, List-Unsubscribe, List-Subscribe, List-Post, List-Owner, List-Archive

The following header fields SHOULD NOT be included in the signature:

- Return-Path
- Received
- Comments, Keywords

- Bcc, Resent-Bcc
- DKIM-Signature

Optional header fields (those not mentioned above) normally SHOULD NOT be included in the signature, because of the potential for additional header fields of the same name to be legitimately added or reordered prior to verification. There are likely to be legitimate exceptions to this rule, because of the wide variety of application-specific header fields that may be applied to a message, some of which are unlikely to be duplicated, modified, or reordered.

Signers SHOULD choose canonicalization algorithms based on the types of messages they process and their aversion to risk. For example, e-commerce sites sending primarily purchase receipts, which are not expected to be processed by mailing lists or other software likely to modify messages, will generally prefer "simple" canonicalization. Sites sending primarily person-to-person email will likely prefer to be more resilient to modification during transport by using "relaxed" canonicalization.

Signers SHOULD NOT use "l=" unless they intend to accommodate intermediate mail processors that append text to a message. For example, many mailing list processors append "unsubscribe" information to message bodies. If signers use "l=", they SHOULD include the entire message body existing at the time of signing in computing the count. In particular, signers SHOULD NOT specify a body length of 0 since this may be interpreted as a meaningless signature by some verifiers.

5.6. Compute the Message Hash and Signature

The signer MUST compute the message hash as described in Section 3.7 and then sign it using the selected public-key algorithm. This will result in a DKIM-Signature header field that will include the body hash and a signature of the header hash, where that header includes the DKIM-Signature header field itself.

Entities such as mailing list managers that implement DKIM and that modify the message or a header field (for example, inserting unsubscribe information) before retransmitting the message SHOULD check any existing signature on input and MUST make such modifications before re-signing the message.

The signer MAY elect to limit the number of bytes of the body that will be included in the hash and hence signed. The length actually hashed should be inserted in the "l=" tag of the DKIM-Signature header field.

5.7. Insert the DKIM-Signature Header Field

Finally, the signer MUST insert the DKIM-Signature header field created in the previous step prior to transmitting the email. The DKIM-Signature header field MUST be the same as used to compute the hash as described above, except that the value of the "b=" tag MUST be the appropriately signed hash computed in the previous step, signed using the algorithm specified in the "a=" tag of the DKIM-Signature header field and using the private key corresponding to the selector given in the "s=" tag of the DKIM-Signature header field, as chosen above in Section 5.2

The DKIM-Signature header field MUST be inserted before any other DKIM-Signature fields in the header block.

> INFORMATIVE IMPLEMENTATION NOTE: The easiest way to achieve this is to insert the DKIM-Signature header field at the beginning of the header block. In particular, it may be placed before any existing Received header fields. This is consistent with treating DKIM-Signature as a trace header field.

6. Verifier Actions

Since a signer MAY remove or revoke a public key at any time, it is recommended that verification occur in a timely manner. In many configurations, the most timely place is during acceptance by the border MTA or shortly thereafter. In particular, deferring verification until the message is accessed by the end user is discouraged.

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

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

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

6.1. Extract Signatures from the Message

The order in which verifiers try DKIM-Signature header fields is not defined; verifiers MAY try signatures in any order they like. For example, one implementation might try the signatures in textual order, whereas another might try signatures by identities that match the contents of the From header field before trying other signatures. Verifiers MUST NOT attribute ultimate meaning to the order of multiple DKIM-Signature header fields. In particular, there is reason to believe that some relays will reorder the header fields in potentially arbitrary ways.

> INFORMATIVE IMPLEMENTATION NOTE: Verifiers might use the order as a clue to signing order in the absence of any other information. However, other clues as to the semantics of multiple signatures (such as correlating the signing host with Received header fields) may also be considered.

A verifier SHOULD NOT treat a message that has one or more bad signatures and no good signatures differently from a message with no signature at all; such treatment is a matter of local policy and is beyond the scope of this document.

When a signature successfully verifies, a verifier will either stop processing or attempt to verify any other signatures, at the discretion of the implementation. A verifier MAY limit the number of signatures it tries to avoid denial-of-service attacks.

> INFORMATIVE NOTE: An attacker could send messages with large numbers of faulty signatures, each of which would require a DNS lookup and corresponding CPU time to verify the message. This could be an attack on the domain that receives the message, by slowing down the verifier by requiring it to do a large number of DNS lookups and/or signature verifications. It could also be an attack against the domains listed in the signatures, essentially by enlisting innocent verifiers in launching an attack against the DNS servers of the actual victim.

In the following description, text reading "return status (explanation)" (where "status" is one of "PERMFAIL" or "TEMPFAIL") means that the verifier MUST immediately cease processing that signature. The verifier SHOULD proceed to the next signature, if any is present, and completely ignore the bad signature. If the status is "PERMFAIL", the signature failed and should not be reconsidered. If the status is "TEMPFAIL", the signature could not be verified at this time but may be tried again later. A verifier MAY either defer the message for later processing, perhaps by queueing it locally or issuing a 451/4.7.5 SMTP reply, or try another signature; if no good

signature is found and any of the signatures resulted in a TEMPFAIL status, the verifier MAY save the message for later processing. The "(explanation)" is not normative text; it is provided solely for clarification.

Verifiers SHOULD ignore any DKIM-Signature header fields where the signature does not validate. Verifiers that are prepared to validate multiple signature header fields SHOULD proceed to the next signature header field, should it exist. However, verifiers MAY make note of the fact that an invalid signature was present for consideration at a later step.

> INFORMATIVE NOTE: The rationale of this requirement is to permit messages that have invalid signatures but also a valid signature to work. For example, a mailing list exploder might opt to leave the original submitter signature in place even though the exploder knows that it is modifying the message in some way that will break that signature, and the exploder inserts its own signature. In this case, the message should succeed even in the presence of the known-broken signature.

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

6.1.1. Validate the Signature Header Field

Implementers MUST meticulously validate the format and values in the DKIM-Signature header field; any inconsistency or unexpected values MUST cause the header field to be completely ignored and the verifier to return PERMFAIL (signature syntax error). Being "liberal in what you accept" is definitely a bad strategy in this security context. Note however that this does not include the existence of unknown tags in a DKIM-Signature header field, which are explicitly permitted.

Verifiers MUST ignore DKIM-Signature header fields with a "v=" tag that is inconsistent with this specification and return PERMFAIL (incompatible version).

> INFORMATIVE IMPLEMENTATION NOTE: An implementation may, of course, choose to also verify signatures generated by older versions of this specification.

If any tag listed as "required" in Section 3.5 is omitted from the DKIM-Signature header field, the verifier MUST ignore the DKIM-Signature header field and return PERMFAIL (signature missing required tag).

INFORMATIONAL NOTE: The tags listed as required in Section 3.5 are "v=", "a=", "b=", "bh=", "d=", "h=", and "s=". Should there be a conflict between this note and Section 3.5, Section 3.5 is normative.

If the DKIM-Signature header field does not contain the "i=" tag, the verifier MUST behave as though the value of that tag were "@d", where "d" is the value from the "d=" tag.

Verifiers MUST confirm that the domain specified in the "d=" tag is the same as or a parent domain of the domain part of the "i=" tag. If not, the DKIM-Signature header field MUST be ignored and the verifier should return PERMFAIL (domain mismatch).

If the "h=" tag does not include the From header field, the verifier MUST ignore the DKIM-Signature header field and return PERMFAIL (From field not signed).

Verifiers MAY ignore the DKIM-Signature header field and return PERMFAIL (signature expired) if it contains an "x=" tag and the signature has expired.

Verifiers MAY ignore the DKIM-Signature header field if the domain used by the signer in the "d=" tag is not associated with a valid signing entity. For example, signatures with "d=" values such as "com" and "co.uk" may be ignored. The list of unacceptable domains SHOULD be configurable.

Verifiers MAY ignore the DKIM-Signature header field and return PERMFAIL (unacceptable signature header) for any other reason, for example, if the signature does not sign header fields that the verifier views to be essential. As a case in point, if MIME header fields are not signed, certain attacks may be possible that the verifier would prefer to avoid.

6.1.2. Get the Public Key

The public key for a signature is needed to complete the verification process. The process of retrieving the public key depends on the query type as defined by the "q=" tag in the DKIM-Signature header field. Obviously, a public key need only be retrieved if the process of extracting the signature information is completely successful. Details of key management and representation are described in Section 3.6. 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. Retrieve the public key as described in Section 3.6 using the algorithm in the "q=" tag, the domain from the "d=" tag, and the selector from the "s=" tag.

2. If the query for the public key fails to respond, the verifier MAY defer acceptance of this email and return TEMPFAIL (key unavailable). If verification is occurring during the incoming SMTP session, this MAY be achieved with a 451/4.7.5 SMTP reply code. Alternatively, the verifier MAY store the message in the local queue for later trial or ignore the signature. Note that storing a message in the local queue is subject to denial-of-service attacks.

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

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

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

6. If the "g=" tag in the public key does not match the Local-part of the "i=" tag in the message signature header field, the verifier MUST ignore the key record and return PERMFAIL (inapplicable key). If the Local-part of the "i=" tag on the message signature is not present, the "g=" tag must be "*" (valid for all addresses in the domain) or the entire g= tag must be omitted (which defaults to "g=*"), otherwise the verifier MUST ignore the key record and return PERMFAIL (inapplicable key). Other than this test, verifiers SHOULD NOT treat a message signed with a key record having a "g=" tag any differently than one without; in particular, verifiers SHOULD NOT prefer messages that

seem to have an individual signature by virtue of a "g=" tag versus a domain signature.

7. If the "h=" tag exists in the public key record and the hash algorithm implied by the a= tag in the DKIM-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).

8. 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.

9. If the public key data is not suitable for use with the algorithm and key types defined by the "a=" and "k=" tags in the DKIM-Signature header field, the verifier MUST immediately return PERMFAIL (inappropriate key algorithm).

6.1.3. Compute the Verification

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

1. Based on the algorithm defined in the "c=" tag, the body length specified in the "l=" tag, and the header field names in the "h=" tag, prepare a canonicalized version of the message as is described in Section 3.7 (note that this version does not actually need to be instantiated). When matching header field names in the "h=" tag against the actual message header field, comparisons MUST be case-insensitive.

2. Based on the algorithm indicated in the "a=" tag, compute the message hashes from the canonical copy as described in Section 3.7.

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

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

5. Otherwise, the signature has correctly verified.

INFORMATIVE IMPLEMENTER'S NOTE: Implementations might wish to initiate the public-key query in parallel with calculating the hash as the public key is not needed until the final decryption is calculated. Implementations may also verify the signature on the message header before validating that the message hash listed in the "bh=" tag in the DKIM-Signature header field matches that of the actual message body; however, if the body hash does not match, the entire signature must be considered to have failed.

A body length specified in the "l=" tag of the signature limits the number of bytes of the body passed to the verification algorithm. All data beyond that limit is not validated by DKIM. Hence, verifiers might treat a message that contains bytes beyond the indicated body length with suspicion, such as by truncating the message at the indicated body length, declaring the signature invalid (e.g., by returning PERMFAIL (unsigned content)), or conveying the partial verification to the policy module.

INFORMATIVE IMPLEMENTATION NOTE: Verifiers that truncate the body at the indicated body length might pass on a malformed MIME message if the signer used the "N-4" trick (omitting the final "--CRLF") described in the informative note in Section 3.4.5. Such verifiers may wish to check for this case and include a trailing "--CRLF" to avoid breaking the MIME structure. A simple way to achieve this might be to append "--CRLF" to any "multipart" message with a body length; if the MIME structure is already correctly formed, this will appear in the postlude and will not be displayed to the end user.

6.2. Communicate Verification Results

Verifiers wishing to communicate the results of verification to other parts of the mail system may do so in whatever manner they see fit. For example, implementations might choose to add an email header field to the message before passing it on. Any such header field SHOULD be inserted before any existing DKIM-Signature or preexisting authentication status header fields in the header field block.

INFORMATIVE ADVICE to MUA filter writers: Patterns intended to search for results header fields to visibly mark authenticated mail for end users should verify that such header field was added by the appropriate verifying domain and that the verified identity matches the author identity that will be displayed by the MUA. In particular, MUA filters should not be influenced by bogus results

header fields added by attackers. To circumvent this attack, verifiers may wish to delete existing results header fields after verification and before adding a new header field.

6.3. Interpret Results/Apply Local Policy

It is beyond the scope of this specification to describe what actions a verifier system should make, but an authenticated email presents an opportunity to a receiving system that unauthenticated email cannot. Specifically, an authenticated email creates a predictable identifier by which other decisions can reliably be managed, such as trust and reputation. Conversely, unauthenticated email lacks a reliable identifier that can be used to assign trust and reputation. It is reasonable to treat unauthenticated email as lacking any trust and having no positive reputation.

In general, verifiers SHOULD NOT reject messages solely on the basis of a lack of signature or an unverifiable signature; such rejection would cause severe interoperability problems. However, if the verifier does opt to reject such messages (for example, when communicating with a peer who, by prior agreement, agrees to only send signed messages), and the verifier runs synchronously with the SMTP session and a signature is missing or does not verify, the MTA SHOULD use a 550/5.7.x reply code.

If 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 return 4xx SMTP replies.

Once the signature has been verified, that information MUST be conveyed to higher-level systems (such as explicit allow/whitelists and reputation systems) and/or to the end user. If the message is signed on behalf of any address other than that in the From: header field, the mail system SHOULD take pains to ensure that the actual signing identity is clear to the reader.

The verifier MAY treat unsigned header fields with extreme skepticism, including marking them as untrusted or even deleting them before display to the end user.

While the symptoms of a failed verification are obvious -- the signature doesn't verify -- establishing the exact cause can be more difficult. If a selector cannot be found, is that because the selector has been removed, or was the value changed somehow in transit? If the signature line is missing, is that because it was never there, or was it removed by an overzealous filter? For diagnostic purposes, the exact reason why the verification fails SHOULD be made available to the policy module and possibly recorded in the system logs. If the email cannot be verified, then it SHOULD be rendered the same as all unverified email regardless of whether or not it looks like it was signed.

7. IANA Considerations

DKIM introduces some new namespaces that have been registered with IANA. In all cases, new values are assigned only for values that have been documented in a published RFC that has IETF Consensus [RFC2434].

7.1. DKIM-Signature Tag Specifications

A DKIM-Signature provides for a list of tag specifications. IANA has established the DKIM-Signature Tag Specification Registry for tag specifications that can be used in DKIM-Signature fields.

The initial entries in the registry comprise:

TYPE	REFERENCE
v	(this document)
a	(this document)
b	(this document)
bh	(this document)
c	(this document)
d	(this document)
h	(this document)
i	(this document)
l	(this document)
q	(this document)
s	(this document)
t	(this document)
x	(this document)
z	(this document)

DKIM-Signature Tag Specification Registry Initial Values

7.2. DKIM-Signature Query Method Registry

The "q=" tag-spec (specified in Section 3.5) provides for a list of query methods.

IANA has established the DKIM-Signature Query Method Registry for mechanisms that can be used to retrieve the key that will permit validation processing of a message signed using DKIM.

The initial entry in the registry comprises:

TYPE	OPTION	REFERENCE
dns	txt	(this document)

DKIM-Signature Query Method Registry Initial Values

7.3. DKIM-Signature Canonicalization Registry

The "c=" tag-spec (specified in Section 3.5) provides for a specifier for canonicalization algorithms for the header and body of the message.

IANA has established the DKIM-Signature Canonicalization Algorithm Registry for algorithms for converting a message into a canonical form before signing or verifying using DKIM.

The initial entries in the header registry comprise:

TYPE	REFERENCE
simple	(this document)
relaxed	(this document)

DKIM-Signature Header Canonicalization Algorithm Registry Initial Values

The initial entries in the body registry comprise:

TYPE	REFERENCE
simple	(this document)
relaxed	(this document)

DKIM-Signature Body Canonicalization Algorithm Registry
Initial Values

7.4. _domainkey DNS TXT Record Tag Specifications

A _domainkey DNS TXT record provides for a list of tag specifications. IANA has established the DKIM _domainkey DNS TXT Tag Specification Registry for tag specifications that can be used in DNS TXT Records.

The initial entries in the registry comprise:

TYPE	REFERENCE
v	(this document)
g	(this document)
h	(this document)
k	(this document)
n	(this document)
p	(this document)
s	(this document)
t	(this document)

DKIM _domainkey DNS TXT Record Tag Specification Registry
Initial Values

7.5. DKIM Key Type Registry

The "k=" <key-k-tag> (specified in Section 3.6.1) and the "a=" <sig-a-tag-k> (specified in Section 3.5) tags provide for a list of mechanisms that can be used to decode a DKIM signature.

IANA has established the DKIM Key Type Registry for such mechanisms.

The initial entry in the registry comprises:

TYPE	REFERENCE
rsa	[RFC3447]

DKIM Key Type Initial Values

7.6. DKIM Hash Algorithms Registry

The "h=" <key-h-tag> (specified in Section 3.6.1) and the "a=" <sig-a-tag-h> (specified in Section 3.5) tags provide for a list of mechanisms that can be used to produce a digest of message data.

IANA has established the DKIM Hash Algorithms Registry for such mechanisms.

The initial entries in the registry comprise:

TYPE	REFERENCE
sha1	[FIPS.180-2.2002]
sha256	[FIPS.180-2.2002]

DKIM Hash Algorithms Initial Values

7.7. DKIM Service Types Registry

The "s=" <key-s-tag> tag (specified in Section 3.6.1) provides for a list of service types to which this selector may apply.

IANA has established the DKIM Service Types Registry for service types.

The initial entries in the registry comprise:

TYPE	REFERENCE
email	(this document)
*	(this document)

DKIM Service Types Registry Initial Values

7.8. DKIM Selector Flags Registry

The "t=" <key-t-tag> tag (specified in Section 3.6.1) provides for a list of flags to modify interpretation of the selector.

IANA has established the DKIM Selector Flags Registry for additional flags.

The initial entries in the registry comprise:

TYPE	REFERENCE
y	(this document)
s	(this document)

DKIM Selector Flags Registry Initial Values

7.9. DKIM-Signature Header Field

IANA has added DKIM-Signature to the "Permanent Message Header Fields" registry (see [RFC3864]) for the "mail" protocol, using this document as the reference.

8. Security Considerations

It has been observed that any mechanism that is introduced that attempts to stem the flow of spam is subject to intensive attack. DKIM needs to be carefully scrutinized to identify potential attack vectors and the vulnerability to each. See also [RFC4686].

8.1. Misuse of Body Length Limits ("l=" Tag)

Body length limits (in the form of the "l=" tag) are subject to several potential attacks.

8.1.1. Addition of New MIME Parts to Multipart/*

If the body length limit does not cover a closing MIME multipart section (including the trailing "--CRLF" portion), then it is possible for an attacker to intercept a properly signed multipart message and add a new body part. Depending on the details of the MIME type and the implementation of the verifying MTA and the receiving MUA, this could allow an attacker to change the information displayed to an end user from an apparently trusted source.

For example, if attackers can append information to a "text/html" body part, they may be able to exploit a bug in some MUAs that continue to read after a "</html>" marker, and thus display HTML text on top of already displayed text. If a message has a "multipart/alternative" body part, they might be able to add a new body part that is preferred by the displaying MUA.

8.1.2. Addition of new HTML content to existing content

Several receiving MUA implementations do not cease display after a ""</html>"" tag. In particular, this allows attacks involving overlaying images on top of existing text.

INFORMATIVE EXAMPLE: Appending the following text to an existing, properly closed message will in many MUAs result in inappropriate data being rendered on top of existing, correct data:

```
<div style="position: relative; bottom: 350px; z-index: 2;">
<img src="http://www.ietf.org/images/ietflogo2e.gif"
  width=578 height=370>
</div>
```

8.2. Misappropriated Private Key

If the private key for a user is resident on their computer and is not protected by an appropriately secure mechanism, it is possible for malware to send mail as that user and any other user sharing the same private key. The malware would not, however, be able to generate signed spoofs of other signers' addresses, which would aid in identification of the infected user and would limit the possibilities for certain types of attacks involving socially engineered messages. This threat applies mainly to MUA-based implementations; protection of private keys on servers can be easily achieved through the use of specialized cryptographic hardware.

A larger problem occurs if malware on many users' computers obtains the private keys for those users and transmits them via a covert channel to a site where they can be shared. The compromised users would likely not know of the misappropriation until they receive "bounce" messages from messages they are purported to have sent. Many users might not understand the significance of these bounce messages and would not take action.

One countermeasure is to use a user-entered passphrase to encrypt the private key, although users tend to choose weak passphrases and often reuse them for different purposes, possibly allowing an attack against DKIM to be extended into other domains. Nevertheless, the decoded private key might be briefly available to compromise by malware when it is entered, or might be discovered via keystroke

logging. The added complexity of entering a passphrase each time one sends a message would also tend to discourage the use of a secure passphrase.

A somewhat more effective countermeasure is to send messages through an outgoing MTA that can authenticate the submitter using existing techniques (e.g., SMTP Authentication), possibly validate the message itself (e.g., verify that the header is legitimate and that the content passes a spam content check), and sign the message using a key appropriate for the submitter address. Such an MTA can also apply controls on the volume of outgoing mail each user is permitted to originate in order to further limit the ability of malware to generate bulk email.

8.3. Key Server Denial-of-Service Attacks

Since the key servers are distributed (potentially separate for each domain), the number of servers that would need to be attacked to defeat this mechanism on an Internet-wide basis is very large. Nevertheless, key servers for individual domains could be attacked, impeding the verification of messages from that domain. This is not significantly different from the ability of an attacker to deny service to the mail exchangers for a given domain, although it affects outgoing, not incoming, mail.

A variation on this attack is that if a very large amount of mail were to be sent using spoofed addresses from a given domain, the key servers for that domain could be overwhelmed with requests. However, given the low overhead of verification compared with handling of the email message itself, such an attack would be difficult to mount.

8.4. Attacks Against the DNS

Since the DNS is a required binding for key services, specific attacks against the DNS must be considered.

While the DNS is currently insecure [RFC3833], these security problems are the motivation behind DNS Security (DNSSEC) [RFC4033], and all users of the DNS will reap the benefit of that work.

DKIM is only intended as a "sufficient" method of proving authenticity. It is not intended to provide strong cryptographic proof about authorship or contents. Other technologies such as OpenPGP [RFC2440] and S/MIME [RFC3851] address those requirements.

A second security issue related to the DNS revolves around the increased DNS traffic as a consequence of fetching selector-based data as well as fetching signing domain policy. Widespread

deployment of DKIM will result in a significant increase in DNS queries to the claimed signing domain. In the case of forgeries on a large scale, DNS servers could see a substantial increase in queries.

A specific DNS security issue that should be considered by DKIM verifiers is the name chaining attack described in Section 2.3 of the DNS Threat Analysis [RFC3833]. A DKIM verifier, while verifying a DKIM-Signature header field, could be prompted to retrieve a key record of an attacker's choosing. This threat can be minimized by ensuring that name servers, including recursive name servers, used by the verifier enforce strict checking of "glue" and other additional information in DNS responses and are therefore not vulnerable to this attack.

8.5. Replay Attacks

In this attack, a spammer sends a message to be spammed to an accomplice, which results in the message being signed by the originating MTA. The accomplice resends the message, including the original signature, to a large number of recipients, possibly by sending the message to many compromised machines that act as MTAs. The messages, not having been modified by the accomplice, have valid signatures.

Partial solutions to this problem involve the use of reputation services to convey the fact that the specific email address is being used for spam and that messages from that signer are likely to be spam. This requires a real-time detection mechanism in order to react quickly enough. However, such measures might be prone to abuse, if for example an attacker resent a large number of messages received from a victim in order to make them appear to be a spammer.

Large verifiers might be able to detect unusually large volumes of mails with the same signature in a short time period. Smaller verifiers can get substantially the same volume of information via existing collaborative systems.

8.6. Limits on Revoking Keys

When a large domain detects undesirable behavior on the part of one of its users, it might wish to revoke the key used to sign that user's messages in order to disavow responsibility for messages that have not yet been verified or that are the subject of a replay attack. However, the ability of the domain to do so can be limited if the same key, for scalability reasons, is used to sign messages for many other users. Mechanisms for explicitly revoking keys on a per-address basis have been proposed but require further study as to their utility and the DNS load they represent.

8.7. Intentionally Malformed Key Records

It is possible for an attacker to publish key records in DNS that are intentionally malformed, with the intent of causing a denial-of-service attack on a non-robust verifier implementation. The attacker could then cause a verifier to read the malformed key record by sending a message to one of its users referencing the malformed record in a (not necessarily valid) signature. Verifiers MUST thoroughly verify all key records retrieved from the DNS and be robust against intentionally as well as unintentionally malformed key records.

8.8. Intentionally Malformed DKIM-Signature Header Fields

Verifiers MUST be prepared to receive messages with malformed DKIM-Signature header fields, and thoroughly verify the header field before depending on any of its contents.

8.9. Information Leakage

An attacker could determine when a particular signature was verified by using a per-message selector and then monitoring their DNS traffic for the key lookup. This would act as the equivalent of a "web bug" for verification time rather than when the message was read.

8.10. Remote Timing Attacks

In some cases, it may be possible to extract private keys using a remote timing attack [BONEH03]. Implementations should consider obfuscating the timing to prevent such attacks.

8.11. Reordered Header Fields

Existing standards allow intermediate MTAs to reorder header fields. If a signer signs two or more header fields of the same name, this can cause spurious verification errors on otherwise legitimate messages. In particular, signers that sign any existing DKIM-Signature fields run the risk of having messages incorrectly fail to verify.

8.12. RSA Attacks

An attacker could create a large RSA signing key with a small exponent, thus requiring that the verification key have a large exponent. This will force verifiers to use considerable computing resources to verify the signature. Verifiers might avoid this attack by refusing to verify signatures that reference selectors with public keys having unreasonable exponents.

In general, an attacker might try to overwhelm a verifier by flooding it with messages requiring verification. This is similar to other MTA denial-of-service attacks and should be dealt with in a similar fashion.

8.13. Inappropriate Signing by Parent Domains

The trust relationship described in Section 3.8 could conceivably be used by a parent domain to sign messages with identities in a subdomain not administratively related to the parent. For example, the ".com" registry could create messages with signatures using an "i=" value in the example.com domain. There is no general solution to this problem, since the administrative cut could occur anywhere in the domain name. For example, in the domain "example.podunk.ca.us" there are three administrative cuts (podunk.ca.us, ca.us, and us), any of which could create messages with an identity in the full domain.

> INFORMATIVE NOTE: This is considered an acceptable risk for the same reason that it is acceptable for domain delegation. For example, in the example above any of the domains could potentially simply delegate "example.podunk.ca.us" to a server of their choice and completely replace all DNS-served information. Note that a verifier MAY ignore signatures that come from an unlikely domain such as ".com", as discussed in Section 6.1.1.

9. References

9.1. Normative References

[FIPS.180-2.2002] U.S. Department of Commerce, "Secure Hash Standard", FIPS PUB 180-2, August 2002.

[ITU.X660.1997] "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.660, 1997.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message header field Extensions for Non-ASCII Text", RFC 2047, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2821] Klensin, J., "Simple Mail Transfer Protocol", RFC 2821, April 2001.

[RFC2822] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.

[RFC3490] Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003.

[RFC4234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.

9.2. Informative References

[BONEH03] Proc. 12th USENIX Security Symposium, "Remote Timing Attacks are Practical", 2003.

[RFC1847] Galvin, J., Murphy, S., Crocker, S., and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", RFC 1847, October 1995.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[RFC2440] Callas, J., Donnerhacke, L., Finney, H., and R. Thayer, "OpenPGP Message Format", RFC 2440, November 1998.

[RFC3766] Orman, H. and P. Hoffman, "Determining Strengths for Public Keys Used For Exchanging Symmetric Keys", RFC 3766, April 2004.

[RFC3833] Atkins, D. and R. Austein, "Threat Analysis of the Domain Name System (DNS)", RFC 3833, August 2004.

[RFC3851] Ramsdell, B., "S/MIME Version 3 Message Specification", RFC 3851, June 1999.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, September 2004.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC4686] Fenton, J., "Analysis of Threats Motivating DomainKeys Identified Mail (DKIM)", RFC 4686, September 2006.

[RFC4870] Delany, M., "Domain-Based Email Authentication Using Public Keys Advertised in the DNS (DomainKeys)", RFC 4870, May 2007.

Appendix A. Example of Use (INFORMATIVE)

This section shows the complete flow of an email from submission to final delivery, demonstrating how the various components fit together. The key used in this example is shown in Appendix C.

A.1. The User Composes an Email

```
From: Joe SixPack <joe@football.example.com>
To: Suzie Q <suzie@shopping.example.net>
Subject: Is dinner ready?
Date: Fri, 11 Jul 2003 21:00:37 -0700 (PDT)
Message-ID: <20030712040037.46341.5F8J@football.example.com>

Hi.

We lost the game. Are you hungry yet?

Joe.
```

A.2. The Email Is Signed

This email is signed by the example.com outbound email server and now looks like this:

```
DKIM-Signature: v=1; a=rsa-sha256; s=brisbane; d=example.com;
      c=simple/simple; q=dns/txt; i=joe@football.example.com;
      h=Received : From : To : Subject : Date : Message-ID;
      bh=2jUSOH9NhtVGCQWNr9BrIAPreKQjO6Sn7XIkfJVOzv8=;
      b=AuUoFEfDxTDkHlLXSZEpZj79LICEps6eda7W3deTVFOk4yAUoqOB
        4nujc7YopdG5dWLSdNg6xNAZpOPr+kHxt1IrE+NahM6L/LbvaHut
        KVdkLLkpVaVVQPzeRDI009SO2Il5Lu7rDNH6mZckBdrIx0orEtZV
        4bmp/YzhwvcubU4=;
Received: from client1.football.example.com  [192.0.2.1]
      by submitserver.example.com with SUBMISSION;
      Fri, 11 Jul 2003 21:01:54 -0700 (PDT)
From: Joe SixPack <joe@football.example.com>
To: Suzie Q <suzie@shopping.example.net>
Subject: Is dinner ready?
Date: Fri, 11 Jul 2003 21:00:37 -0700 (PDT)
Message-ID: <20030712040037.46341.5F8J@football.example.com>

Hi.

We lost the game. Are you hungry yet?

Joe.
```

The signing email server requires access to the private key associated with the "brisbane" selector to generate this signature.

A.3. The Email Signature Is Verified

The signature is normally verified by an inbound SMTP server or possibly the final delivery agent. However, intervening MTAs can also perform this verification if they choose to do so. The verification process uses the domain "example.com" extracted from the "d=" tag and the selector "brisbane" from the "s=" tag in the DKIM-Signature header field to form the DNS DKIM query for:

```
brisbane._domainkey.example.com
```

Signature verification starts with the physically last Received header field, the From header field, and so forth, in the order listed in the "h=" tag. Verification follows with a single CRLF followed by the body (starting with "Hi."). The email is canonically prepared for verifying with the "simple" method. The result of the query and subsequent verification of the signature is stored (in this

example) in the X-Authentication-Results header field line. After successful verification, the email looks like this:

```
X-Authentication-Results: shopping.example.net
      header.from=joe@football.example.com; dkim=pass
Received: from mout23.football.example.com (192.168.1.1)
      by shopping.example.net with SMTP;
      Fri, 11 Jul 2003 21:01:59 -0700 (PDT)
DKIM-Signature: v=1; a=rsa-sha256; s=brisbane; d=example.com;
      c=simple/simple; q=dns/txt; i=joe@football.example.com;
      h=Received : From : To : Subject : Date : Message-ID;
      bh=2jUSOH9NhtVGCQWNr9BrIAPreKQjO6Sn7XIkfJVOzv8=;
      b=AuUoFEfDxTDkHlLXSZEpZj79LICEps6eda7W3deTVFOk4yAUoqOB
        4nujc7YopdG5dWLSdNg6xNAZpOPr+kHxt1IrE+NahM6L/LbvaHut
        KVdkLLkpVaVVQPzeRDI009SO2Il5Lu7rDNH6mZckBdrIx0orEtZV
        4bmp/YzhwvcubU4=;
Received: from client1.football.example.com  [192.0.2.1]
      by submitserver.example.com with SUBMISSION;
      Fri, 11 Jul 2003 21:01:54 -0700 (PDT)
From: Joe SixPack <joe@football.example.com>
To: Suzie Q <suzie@shopping.example.net>
Subject: Is dinner ready?
Date: Fri, 11 Jul 2003 21:00:37 -0700 (PDT)
Message-ID: <20030712040037.46341.5F8J@football.example.com>

Hi.

We lost the game. Are you hungry yet?

Joe.
```

Appendix B. Usage Examples (INFORMATIVE)

DKIM signing and validating can be used in different ways, for different operational scenarios. This Appendix discusses some common examples.

> NOTE: Descriptions in this Appendix are for informational purposes only. They describe various ways that DKIM can be used, given particular constraints and needs. In no case are these examples intended to be taken as providing explanation or guidance concerning DKIM specification details, when creating an implementation.

B.1. Alternate Submission Scenarios

In the most simple scenario, a user's MUA, MSA, and Internet (boundary) MTA are all within the same administrative environment, using the same domain name. Therefore, all of the components involved in submission and initial transfer are related. However, it is common for two or more of the components to be under independent administrative control. This creates challenges for choosing and administering the domain name to use for signing, and for its relationship to common email identity header fields.

B.1.1. Delegated Business Functions

Some organizations assign specific business functions to discrete groups, inside or outside the organization. The goal, then, is to authorize that group to sign some mail, but to constrain what signatures they can generate. DKIM selectors (the "s=" signature tag) and granularity (the "g=" key tag) facilitate this kind of restricted authorization. Examples of these outsourced business functions are legitimate email marketing providers and corporate benefits providers.

Here, the delegated group needs to be able to send messages that are signed, using the email domain of the client company. At the same time, the client often is reluctant to register a key for the provider that grants the ability to send messages for arbitrary addresses in the domain.

There are multiple ways to administer these usage scenarios. In one case, the client organization provides all of the public query service (for example, DNS) administration, and in another it uses DNS delegation to enable all ongoing administration of the DKIM key record by the delegated group.

If the client organization retains responsibility for all of the DNS administration, the outsourcing company can generate a key pair, supplying the public key to the client company, which then registers it in the query service, using a unique selector that authorizes a specific From header field Local-part. For example, a client with the domain "example.com" could have the selector record specify "g=winter-promotions" so that this signature is only valid for mail with a From address of "winter-promotions@example.com". This would enable the provider to send messages using that specific address and have them verify properly. The client company retains control over the email address because it retains the ability to revoke the key at any time.

If the client wants the delegated group to do the DNS administration, it can have the domain name that is specified with the selector point to the provider's DNS server. The provider then creates and maintains all of the DKIM signature information for that selector. Hence, the client cannot provide constraints on the Local-part of addresses that get signed, but it can revoke the provider's signing rights by removing the DNS delegation record.

B.1.2. PDAs and Similar Devices

PDAs demonstrate the need for using multiple keys per domain. Suppose that John Doe wanted to be able to send messages using his corporate email address, jdoe@example.com, and his email device did not have the ability to make a Virtual Private Network (VPN) connection to the corporate network, either because the device is limited or because there are restrictions enforced by his Internet access provider. If the device was equipped with a private key registered for jdoe@example.com by the administrator of the example.com domain, and appropriate software to sign messages, John could sign the message on the device itself before transmission through the outgoing network of the access service provider.

B.1.3. Roaming Users

Roaming users often find themselves in circumstances where it is convenient or necessary to use an SMTP server other than their home server; examples are conferences and many hotels. In such circumstances, a signature that is added by the submission service will use an identity that is different from the user's home system.

Ideally, roaming users would connect back to their home server using either a VPN or a SUBMISSION server running with SMTP AUTHentication on port 587. If the signing can be performed on the roaming user's laptop, then they can sign before submission, although the risk of further modification is high. If neither of these are possible, these roaming users will not be able to send mail signed using their own domain key.

B.1.4. Independent (Kiosk) Message Submission

Stand-alone services, such as walk-up kiosks and web-based information services, have no enduring email service relationship with the user, but users occasionally request that mail be sent on their behalf. For example, a website providing news often allows the reader to forward a copy of the article to a friend. This is typically done using the reader's own email address, to indicate who the author is. This is sometimes referred to as the "Evite problem",

named after the website of the same name that allows a user to send invitations to friends.

A common way this is handled is to continue to put the reader's email address in the From header field of the message, but put an address owned by the email posting site into the Sender header field. The posting site can then sign the message, using the domain that is in the Sender field. This provides useful information to the receiving email site, which is able to correlate the signing domain with the initial submission email role.

Receiving sites often wish to provide their end users with information about mail that is mediated in this fashion. Although the real efficacy of different approaches is a subject for human factors usability research, one technique that is used is for the verifying system to rewrite the From header field, to indicate the address that was verified. For example: From: John Doe via news@news-site.com <jdoe@example.com>. (Note that such rewriting will break a signature, unless it is done after the verification pass is complete.)

B.2. Alternate Delivery Scenarios

Email is often received at a mailbox that has an address different from the one used during initial submission. In these cases, an intermediary mechanism operates at the address originally used and it then passes the message on to the final destination. This mediation process presents some challenges for DKIM signatures.

B.2.1. Affinity Addresses

"Affinity addresses" allow a user to have an email address that remains stable, even as the user moves among different email providers. They are typically associated with college alumni associations, professional organizations, and recreational organizations with which they expect to have a long-term relationship. These domains usually provide forwarding of incoming email, and they often have an associated Web application that authenticates the user and allows the forwarding address to be changed. However, these services usually depend on users sending outgoing messages through their own service providers' MTAs. Hence, mail that is signed with the domain of the affinity address is not signed by an entity that is administered by the organization owning that domain.

With DKIM, affinity domains could use the Web application to allow users to register per-user keys to be used to sign messages on behalf of their affinity address. The user would take away the secret half

of the key pair for signing, and the affinity domain would publish the public half in DNS for access by verifiers.

This is another application that takes advantage of user-level keying, and domains used for affinity addresses would typically have a very large number of user-level keys. Alternatively, the affinity domain could handle outgoing mail, operating a mail submission agent that authenticates users before accepting and signing messages for them. This is of course dependent on the user's service provider not blocking the relevant TCP ports used for mail submission.

B.2.2. Simple Address Aliasing (.forward)

In some cases, a recipient is allowed to configure an email address to cause automatic redirection of email messages from the original address to another, such as through the use of a Unix .forward file. In this case, messages are typically redirected by the mail handling service of the recipient's domain, without modification, except for the addition of a Received header field to the message and a change in the envelope recipient address. In this case, the recipient at the final address' mailbox is likely to be able to verify the original signature since the signed content has not changed, and DKIM is able to validate the message signature.

B.2.3. Mailing Lists and Re-Posters

There is a wide range of behaviors in services that take delivery of a message and then resubmit it. A primary example is with mailing lists (collectively called "forwarders" below), ranging from those that make no modification to the message itself, other than to add a Received header field and change the envelope information, to those that add header fields, change the Subject header field, add content to the body (typically at the end), or reformat the body in some manner. The simple ones produce messages that are quite similar to the automated alias services. More elaborate systems essentially create a new message.

A Forwarder that does not modify the body or signed header fields of a message is likely to maintain the validity of the existing signature. It also could choose to add its own signature to the message.

Forwarders which modify a message in a way that could make an existing signature invalid are particularly good candidates for adding their own signatures (e.g., mailing-list-name@example.net). Since (re-)signing is taking responsibility for the content of the message, these signing forwarders are likely to be selective, and forward or re-sign a message only if it is received with a valid

signature or if they have some other basis for knowing that the message is not spoofed.

A common practice among systems that are primarily redistributors of mail is to add a Sender header field to the message, to identify the address being used to sign the message. This practice will remove any preexisting Sender header field as required by [RFC2822]. The forwarder applies a new DKIM-Signature header field with the signature, public key, and related information of the forwarder.

Appendix C. Creating a Public Key (INFORMATIVE)

The default signature is an RSA signed SHA256 digest of the complete email. For ease of explanation, the openssl command is used to describe the mechanism by which keys and signatures are managed. One way to generate a 1024-bit, unencrypted private key suitable for DKIM is to use openssl like this:

```
$ openssl genrsa -out rsa.private 1024
```

For increased security, the "-passin" parameter can also be added to encrypt the private key. Use of this parameter will require entering a password for several of the following steps. Servers may prefer to use hardware cryptographic support.

The "genrsa" step results in the file rsa.private containing the key information similar to this:

```
-----BEGIN RSA PRIVATE KEY-----
MIICXwIBAAKBgQDwIRP/UC3SBsEmGqZ9ZJW3/DkMoGeLnQg1fWn7/zYtIxN2SnFC
jxOCKG9v3b4jYfcTNh5ijSsq631uBItLa7od+v/RtdC2UzJ1lWT947qR+Rcac2gb
to/NMqJ0fzfVjH4OuKhitdY9tf6mcwGjaNBcWToIMmPSPDdQPNUYckcQ2QIDAQAB
AoGBALmn+XwWk7akvkUlqb+dOxyLB9i5VBVfje89Teolwc9YJT36BGN/l4e0l6QX
/1//6DWUTB3KI6wFcm7TWJcxbS0tcKZX7FsJvUz1SbQnkS54DJck1EZO/BLa5ckJ
gAYIaqlA9C0ZwM6i58lLlPadX/rtHb7pWzeNcZHjKrjM461ZAkEA+itss2nRlmyO
n1/5yDyCluST4dQfO8kAB3toSEVc7DeFeDhnC1mZdjASZNvdHS4gbLIA1hUGEF9m
3hKsGUMMPwJBAPW5v/U+AWTADFCS22t72NUurgzeAbzb1HWMqO4y4+9Hpjk5wvL/
eVYizyuce3/fGke7aRYw/ADKygMJdW8H/OcCQQDz5OQb4j2QDpPZc0Nc4QlbvMsj
7p7otWRO5xRa6SzXqqV3+F0VpqvDmshEBkoCydaYwc2o6WQ5EBmExeV8124XAkEA
qZzGsIxVP+sEVRWZmW6KNFSdVUpk3qzK0Tz/WjQMe5z0UunY9Ax9/4PVhp/j61bf
eAYXunajbBSOLlx4D+TunwJBANkPI5S9iylsbLs6NkaMHV6k5ioHBBmgCak95JGX
GMot/L2x0IYyMLAz6oLWh2hm7zwtb0CgOrPo1ke44hFYnfc=
-----END RSA PRIVATE KEY-----
```

To extract the public-key component from the private key, use openssl like this:

```
$ openssl rsa -in rsa.private -out rsa.public -pubout -outform PEM
```

This results in the file rsa.public containing the key information similar to this:

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDwIRP/UC3SBsEmGqZ9ZJW3/DkM
oGeLnQg1fWn7/zYtIxN2SnFCjxOCKG9v3b4jYfcTNh5ijSsq631uBItLa7od+v/R
tdC2UzJ1lWT947qR+Rcac2gbto/NMqJ0fzfVjH4OuKhitdY9tf6mcwGjaNBcWToI
MmPSPDdQPNUYckcQ2QIDAQAB
-----END PUBLIC KEY-----
```

This public-key data (without the BEGIN and END tags) is placed in the DNS:

```
brisbane IN  TXT  ("v=DKIM1; p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQ"
                   "KBgQDwIRP/UC3SBsEmGqZ9ZJW3/DkMoGeLnQg1fWn7/zYt"
                   "IxN2SnFCjxOCKG9v3b4jYfcTNh5ijSsq631uBItLa7od+v"
                   "/RtdC2UzJ1lWT947qR+Rcac2gbto/NMqJ0fzfVjH4OuKhi"
                   "tdY9tf6mcwGjaNBcWToIMmPSPDdQPNUYckcQ2QIDAQAB")
```

Appendix D. MUA Considerations

When a DKIM signature is verified, the processing system sometimes makes the result available to the recipient user's MUA. How to present this information to the user in a way that helps them is a matter of continuing human factors usability research. The tendency is to have the MUA highlight the address associated with this signing identity in some way, in an attempt to show the user the address from which the mail was sent. An MUA might do this with visual cues such as graphics, or it might include the address in an alternate view, or it might even rewrite the original From address using the verified information. Some MUAs might indicate which header fields were protected by the validated DKIM signature. This could be done with a positive indication on the signed header fields, with a negative indication on the unsigned header fields, by visually hiding the unsigned header fields, or some combination of these. If an MUA uses visual indications for signed header fields, the MUA probably needs to be careful not to display unsigned header fields in a way that might be construed by the end user as having been signed. If the message has an l= tag whose value does not extend to the end of the message, the MUA might also hide or mark the portion of the message body that was not signed.

The aforementioned information is not intended to be exhaustive. The MUA may choose to highlight, accentuate, hide, or otherwise display any other information that may, in the opinion of the MUA author, be deemed important to the end user.

Appendix E. Acknowledgements

The authors wish to thank Russ Allbery, Edwin Aoki, Claus Assmann, Steve Atkins, Rob Austein, Fred Baker, Mark Baugher, Steve Bellovin, Nathaniel Borenstein, Dave Crocker, Michael Cudahy, Dennis Dayman, Jutta Degener, Frank Ellermann, Patrik Faeltstroem, Mark Fanto, Stephen Farrell, Duncan Findlay, Elliot Gillum, Olafur Gu[eth]mundsson, Phillip Hallam-Baker, Tony Hansen, Sam Hartman, Arvel Hathcock, Amir Herzberg, Paul Hoffman, Russ Housley, Craig Hughes, Cullen Jennings, Don Johnsen, Harry Katz, Murray S. Kucherawy, Barry Leiba, John Levine, Charles Lindsey, Simon Longsdale, David Margrave, Justin Mason, David Mayne, Thierry Moreau, Steve Murphy, Russell Nelson, Dave Oran, Doug Otis, Shamim Pirzada, Juan Altmayer Pizzorno, Sanjay Pol, Blake Ramsdell, Christian Renaud, Scott Renfro, Neil Rerup, Eric Rescorla, Dave Rossetti, Hector Santos, Jim Schaad, the Spamhaus.org team, Malte S. Stretz, Robert Sanders, Rand Wacker, Sam Weiler, and Dan Wing for their valuable suggestions and constructive criticism.

The DomainKeys specification was a primary source from which this specification has been derived. Further information about DomainKeys is at [RFC4870].

Authors' Addresses

Eric Allman
Sendmail, Inc.
6425 Christie Ave, Suite 400
Emeryville, CA 94608
USA

Phone: +1 510 594 5501
EMail: eric+dkim@sendmail.org
URI:

Jon Callas
PGP Corporation
3460 West Bayshore
Palo Alto, CA 94303
USA

Phone: +1 650 319 9016
EMail: jon@pgp.com

Mark Delany
Yahoo! Inc
701 First Avenue
Sunnyvale, CA 95087
USA

Phone: +1 408 349 6831
EMail: markd+dkim@yahoo-inc.com
URI:

Miles Libbey
Yahoo! Inc
701 First Avenue
Sunnyvale, CA 95087
USA

EMail: mlibbeymail-mailsig@yahoo.com
URI:

Jim Fenton
Cisco Systems, Inc.
MS SJ-9/2
170 W. Tasman Drive
San Jose, CA 95134-1706
USA

Phone: +1 408 526 5914
EMail: fenton@cisco.com
URI:

Michael Thomas
Cisco Systems, Inc.
MS SJ-9/2
170 W. Tasman Drive
San Jose, CA 95134-1706

Phone: +1 408 525 5386
EMail: mat@cisco.com

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.