Internet-Draft
Expires: January 10, 2006

E. Allman
Sendmail, Inc.
J. Callas
PGP Corporation
M. Delany
M. Libbey
Yahoo! Inc
J. Fenton
M. Thomas
Cisco Systems, Inc.
July 9, 2005

DomainKeys Identified Mail (DKIM)
draft-allman-dkim-base-00

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 Transport Agents (MTAs) or Mail User Agents (MUAs). The ultimate goal of this framework is to prove and protect message sender identity and the integrity of the messages they convey while retaining the functionality of Internet email as it is known today. Proof and protection of email identity, including repudiation and non-repudiation, may assist in the global control of "spam" and "phishing".

Requirements Language

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].

(Unresolved Issues/To Be Done)

Security Considerations needs further work.

Need to add new and check existing ABNF.

Need to resolve remaining cross references (XINDX)

CONVERSION DISCLAIMER

This initial version that is being submitted as an IETF Internet-Draft has been converted over to RFC format by Dave Crocker. Besides the many rough edges to the resulting format of the document, he suspects there also might be some more serious errors, such as sub-sections being at the wrong level. These errors will be repaired as soon as they are reported.

Table of Contents

1. Introduction

1.1 Overview

DomainKeys Identified Mail (DKIM) defines a simple, low cost, and effective mechanism by which cryptographic signatures can be applied to email messages, to demonstrate that the sender of the message was authorized to use a given email address. Message recipients can verify the signature by querying the signer's domain directly to determine whether the key that was used to sign the message was authorized by that domain for that address. This confirms that the message was sent by a party authorized to use the signer's email address.

The approach taken by DKIM differs from previous approaches to message signing (e.g. S/MIME, OpenPGP) in that:

- the message signature is written to the message header fields so that neither human recipients nor existing MUA software are 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.

DKIM:

- is transparent and compatible with the existing email infrastructure

- requires minimal new infrastructure

- can be implemented independently of clients in order to reduce deployment time

- does not require the use of a trusted third party (such as a certificate authority or other entity) which might impose significant costs or introduce delays to deployment

- can be deployed incrementally.

Just as an email administrator, by creating an email account, gives a user the ability to receive mail sent to a given address, DKIM allows that administrator to constrain the use of that account when sending email. The administrator can delegate the use of an address in

several ways. The administrator could operate a mail transfer agent (MTA) or mail submission agent (MSA) for the domain that authenticates the signer when accepting a message. This MTA or MSA would typically have a private key that is authorized to send mail for anyone in the domain. Alternatively, the administrator could register a public key for the signer with which, assuming the sender has an MTA or MUA with the appropriate software and the corresponding private key, the sender could sign their own outgoing messages. In the latter case, the private key would typically be authorized for one or more specific email addresses that the sender is authorized to use.

A "selector" mechanism allows multiple keys per domain, including delegation of the right to authenticate a portion of the namespace to a trusted third party.

1.2 Signing Identity

DKIM separates the question of the signer of the message from the purported author of the message. In particular, a signature includes the identity of the signer. Recipients can use the signing information to decide how they want to process the message.

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

1.3 Scalability

The email identification problem is characterized by extreme scalability requirements. There are currently on the order of 70 million domains and a much larger number of individual addresses. It is important to preserve the positive aspects of the current email infrastructure, such as the ability for anyone to communicate with anyone else without introduction.

1.4 Simple Key Management

DKIM differs from traditional hierarchical public-key systems in that no key signing infrastructure is required; the verifier requests the public key from the claimed signer directly.

The DNS is proposed as the initial mechanism for publishing public keys. DKIM is specifically designed to be extensible to other key fetching services as they become available.

2. Terminology and Definitions

2.1 Signers

Elements in the mail system that sign messages are referred to as signers. These may be MUAs, MSAs, MTAs, 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, Messages Delivery Agents (MDA), 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 Imported ABNF tokens

The following tokens are imported from other RFCs as noted. Those RFCs should be considered definitive. However, all tokens having names beginning with "obs-" should be excluded from this import.

The following tokens are imported from RFC 2821:

- o Local-part (implementation warning: this permits quoted strings)
- o Domain (implementation warning: this permits address-literals)
- o sub-domain

The following definitions are imported from RFC 2822:

- o WSP (space or tab)
- o FWS (folding white space)
- o field-name (name of a header field)

Other tokens not defined herein are imported from RFC 2234. These are mostly intuitive primitives such as SP, ALPHA, CRLF, etc.

2.4 White Space

There are three forms of white space:

- o WSP represents simple white space, i.e., a space or a tab character, and is inherited from RFC 2822.
- o SWSP is streaming white space; it adds the CR and LF characters.
- o FWS, also from RFC 2822, is folding white space. It allows multiple lines to be joined separated by CRLF followed by at least one white space.

Using the syntax of RFC 2234, the formal ABNF for SWSP is:

```
SWSP =  CR / LF / WSP   ; streaming white space
```

Other terminology is based on [ID-MAIL-ARCH].

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.

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.

> INFORMATIVE IMPLEMENTERS' 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 authorized versus a message that is actually forged. Signers SHOULD NOT change the key associated with a selector. When creating a new key, signers SHOULD associate it with a new selector.

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

- o Domains which want to authorize a partner, such as an advertising provider or other outsourced function, to use a specific address for a given duration.

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

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

Periods are allowed in selectors and are component separators. If keys are stored in DNS, the period defines sub-domain boundaries. Sub-selectors might be used to combine dates with locations; for example, "march2005.reykjavik". This can be used to allow delegation of a portion of the selector name-space.

ABNF:

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

The number of public keys and corresponding selectors for each domain are 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.

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. E.g., if per-user keys are issued, the domain owner will need to make the decision as to whether to make this selector associated directly with the user name, or make it some unassociated random value, such as the fingerprint of the public key.

3.2 Tag=Value Format for DKIM header fields

DKIM uses a simple "tag=value" syntax in several contexts, including

in messages, domain signature records, and policy records.

Values are a series of strings containing either base64 text, plain text, or quoted printable text, as defined in [RFC2045], section 6.7. The name of the tag will determine the encoding of each value. In general, the plain text type SHOULD be used if it is not permissible to have 8 bit and/or syntactically problematic characters (such as semicolon). Binary forms MUST be encoded as base64, and free form text (e.g., user supplied) MUST be typed as quoted printable.

Formally, the syntax rules are:

```
tag-list  =  tag-spec 0*( ";" tag-spec ) [ ";" ]
tag-spec  =  [FWS] tag-name [FWS] "=" [FWS] tag-value [FWS]
tag-name  =  ALPHA 0*ALNUMPUNC
tag-value =  *VALCHAR    ; SWSP prohibited at beginning and end
VALCHAR   =  %9 / %d32 - %d58 / %d60 - %d126
                          ; HTAB and SP to TILDE except SEMICOLON
ALNUMPUNC =  ALPHA / DIGIT / "-"
                          ; alphanumeric plus hyphen.
```

Note that WSP is allowed anywhere around tags; in particular, WSP between the tag-name and the "=", and any WSP before the terminating ";" is not part of the value.

> INFORMATIVE ADVICE: in some cases space might be limited for storing tag-lists; notably, keys stored in DNS. For this reason, tag-values that have length constraints SHOULD have single-character tag names.

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.

Duplicate tags MUST NOT be specified within a single tag-list.

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 key signing/verification algorithms. The only algorithm defined by this specification at this time is rsa-sha1, which is the default if no algorithm is specified and which MUST be supported by all implementations.

3.3.1 The rsa-sha1 Signing Algorithm

The rsa-sha1 Signing Algorithm computes a SHA-1 hash of the message header field and body as described in section XINDX below. That hash is then encrypted by the signer using the RSA algorithm 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.

More formally, the algorithm for the signature using rsa-sha1 is:

```
RSA(SHA1(canon_message, DKIM-SIG), key)
```

where canon_message is the canonicalized message header and body as defined in Section 3.4 and DKIM-SIG is the canonicalized DKIM-Signature header field sans the signature value itself.

3.3.2 Other algorithms

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

3.3.3 Key sizes

Selecting appropriate key sizes is a trade-off between cost, performance and risk. All implementations MUST support keys of sizes 512, 768, 1024, 1536 and 2048 bits and MAY support larger keys.

Factors that should influence the key size choice include:

- o The practical constraint that a 2048 bit key is the largest key that fits within a 512 byte DNS UDP response packet

- o The security constraint that keys smaller than 1024 bits are subject to brute force attacks.

o Larger keys impose higher CPU costs to verify and sign email

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

o The security goals of this specification are modest compared to typical goals of public-key systems

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 however, demand that any modification of the email -- however minor -- results in an authentication failure. These signers prefer a canonicalization algorithm that does not tolerate in-transit modification of the signed email.

To satisfy both requirements, two canonicalization algorithms are defined: a "simple" algorithm that tolerates almost no modification and a "nowsp" algorithm that tolerates common modifications as white-space replacement and header field line re-wrapping. A signer MAY specify either algorithm when signing an email. If no canonicalization algorithm is specified by the signer, the "simple" algorithm is used. A verifier MUST be able to process email using either canonicalization algorithm. 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.

In all cases, the header field of the message is presented to the signing algorithm first in the order indicated by the signature header field. Only header fields listed as signed in the signature header field are included. The CRLF separating the header field from the body is then presented. Canonicalization of header fields and body are described below.

3.4.1 The "simple" canonicalization algorithm

- o Ignore all empty lines at the end of the message body. An empty line is a line of zero length after removal of the line terminator.

- o Make no further changes to either the header field or the body. In particular, no white space should be ignored other than as described above.

3.4.2 The "nowsp" canonicalization algorithm

The "nowsp" algorithm ignores all white space from all lines and unwraps header field continuation lines. These rules MUST be applied in order.

- o Unwrap header field continuation lines so that individual header fields are processed as a single line. Only folding line terminators (CRLF followed by white space) should be removed during this step; in particular, implementations MUST NOT remove the colon between the header field name and header field value and MUST NOT remove the terminating CRLF on individual header fields.

- o Map all header field names (not the header field contents) to lower case. For example, convert "SUBJect: AbC" to "subject: AbC".

- o Ignore all SWSP in the body. SWSP is defined in [XINDX]. Line terminators have no special significance here; in particular, CR and LF MUST be ignored when computing the signature.

- o Ignore all SWSP in header fields except for the trailing CRLF. That is, the signing algorithm processes a single CRLF between each header field and two CRLFs between the end of the last header field signed and the body.

  INFORMATIVE RATIONALE: header fields are often rewrapped during transmission, especially at gateways. Some bodies will get wrapped to obey line length limits; eliminating all SWSP allows wrapping even at arbitrary points.

  INFORMATIVE EXAMPLE: A message reading:

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

is canonicalized to:

```
a:X<CRLF>b:YZ<CRLF><CRLF>CDE
```

After the body is processed, a single CRLF followed by the "DKIM-Signature" header field being created or verified is presented to the algorithm. The contents of the "b=" tag, if any, MUST be deleted, and the header field must be canonicalized according to the algorithm that is specified in the "c=" tag. Any final CRLF on the "DKIM-Signature" header field MUST NOT be included in the signature computation.

INFORMATIVE DISCUSSION: Some parties have proposed extending the "nowsp" algorithm to also remove the leading ">" on all lines beginning ">From " (six characters, including a trailing space). This is not included in this specification because of (a) the additional complexity required in the algorithm and (b) a lack of clear understanding of whether this transformation happens primarily during message transmission or primarily during storage in a UNIX V7-style local mailbox. Evidence indicates that such munging during transmission is rare at this time.

3.4.3 Body Length Limits

A body length count MAY be specified to limit the signature calculation to an initial prefix of the body text. If the body length count is not specified then the entire message body is signed and verified.

INFORMATIVE IMPLEMENTATION NOTE: The l= tag could be useful in increasing signature robustness when sending to a mailing list that both modifies content sent to it and does not sign its messages. However, using the l= tag enables a replay attack in which a sender 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. 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.

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 is made following the canonicalization algorithm; for example whitespace characters MUST NOT be included in the count when using the "nowsp" algorithm.

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.

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".

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

Note that verifiers MAY choose to reject or truncate messages that have body content beyond that specified by the body length count.

INFORMATIVE IMPLEMENTATION ADVICE: Signers wishing to ensure that no modification of any sort can occur SHOULD specify the "simple" algorithm and no body length count.

Despite the measures described above, some messages, particularly those containing 8-bit data, could be subject to modification in transit invalidating the signature. Messages containing 8-bit data SHOULD be converted to MIME format prior to signing, using a suitable content transfer-encoding such as quoted-printable or base64. 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.

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 XINDX.

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 kept in blocks prepended to the message. In particular, the "DKIM-Signature" header field SHOULD precede the original email header fields presented to the canonicalization and signature algorithms.

The "DKIM-Signature:" header field is included in the signature calculation, after the body of the message; however, when calculating or verifying the signature, the b= (signature value) value MUST be treated as though it were the null string. Unknown tags MUST be signed but MUST be otherwise ignored by verifiers.

The encodings for each field type are listed below. Tags described as quoted-printable are as described in section 6.7 of [RFC2045], with the additional conversion of semicolon and vertical bar characters to =3B and =7C, respectively.

Tags on the DKIM-Signature header field along with their type and requirement status are shown below. Valid tags are:

v= Version (MUST NOT be included). This tag is reserved for future use to indicate a possible new, incompatible version of the specification. It MUST NOT be included in the DKIM-Signature header field.

a= The algorithm used to generate the signature (plain-text; REQUIRED). Signers and verifiers MUST support "rsa-sha1", an RSA-signed SHA-1 digest. See section XINDX for a description of algorithms.

> INFORMATIVE RATIONALE: The authors understand that SHA-1 has been theoretically compromised. However, viable attacks require the attacker to choose both sets of input text; given a preexisting input (a "preimaging" attack), it is still hard to determine another input that produces an SHA-1 collision, and the chance that such input would be of value to an attacker is minimal. Also, there is broad library for SHA-1, whereas alternatives such as SHA-256 are just emerging. Finally, DKIM is not intended to have legal- or military-grade requirements. There is nothing inherent in using SHA-1 here other than

implementer convenience. See <http://www3.ietf.org/proceedings/05mar/slides/saag-3.pdf> for a discussion of the security issues.

b= The signature data (base64; REQUIRED). Whitespace is ignored in this value and MUST be ignored when re-assembling the original signature. This is another way of saying that the signing process can safely insert FWS in this value in arbitrary places to conform to line-length limits. See section [XINDX (Signer Actions)] for how the signature is computed.

c= Body canonicalization (plain-text; OPTIONAL, default is "simple"). This tag informs the verifier of the type of canonicalization used to prepare the message for signing. The semantics of this field is described in section XINDX above.

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. When presented with a signature that does not meet this requirement, verifiers MUST either ignore the signature or reject the message..

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), and when verified non-existent header fields MUST be treated in the same way. The field MUST NOT include the DKIM-Signature header field that is being created or verified. Folding white space (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.

ABNF:

```
sig-h-tag  = "h=" *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 sender 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 (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 is optional. If the signing domain is unable or unwilling to commit to an individual user name within their domain, the local-part of the address MUST be omitted. If the local-part of the address is omitted or the "i=" tag is not present, the key used to sign MUST be valid for any address in the domain. The domain part of the address MUST be the same as or a subdomain of the value of the "d=" tag.

ABNF:

```
sig-i-tag =     "i=" [ Local-part ] "@" Domain
```

INFORMATIVE DISCUSSION: This document does not require the value of the "i=" tag to match the identity in any message header field fields. This is considered to be a verifier policy issue, described in another document [XREF-TBD]. 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 count (plain-text decimal integer; OPTIONAL, default is entire body). This tag informs the verifier of the number of bytes in the body of the email included in the cryptographic hash, starting from 0 immediately following the CRLF preceding the 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 man-in-the-middle 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 [XINDX].

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.

q= A colon-separated list of query methods used to retrieve the public key (plain-text; OPTIONAL, default is "dns"). Each query method is of the form "type[/options]", where the syntax and semantics of the options depends on the type. If there are multiple query mechanisms listed, the choice of query mechanism MUST NOT change the interpretation of the signature. Currently the only valid value is "dns" which defines the DNS lookup algorithm described elsewhere in this document. No options are defined for the "dns" query type, but the string "dns" MAY have a trailing "/" character. Verifiers and signers MUST support "dns".

INFORMATIVE RATIONALE: Explicitly allowing a trailing "/" on "dns" allows for the possibility of adding options later and makes it clear that matching of the query type must terminate on either "/" or end of string.

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

t= Signature Timestamp (plain-text; RECOMMENDED, default is an unknown creation time). The time that this signature was created. The format is the standard Unix seconds-since-1970. The value is expressed as an unsigned integer in decimal ASCII.

INFORMATIVE IMPLEMENTATION NOTE: 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.

x= Signature Expiration (plain-text; RECOMMENDED, default is no expiration). Signature expiration in seconds-since-1970 format as an absolute date, not as a time delta from the signing timestamp. Signatures MUST NOT be considered valid if the current time at the verifier is past the expiration date. The value is expressed as an unsigned integer in decimal ASCII.

INFORMATIVE IMPLEMENTATION NOTE: See above.

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

z= Copied header fields (plain-text, but see description; OPTIONAL, default is null). A vertical-bar-separated list of header field names and copies of header field values 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. Copied header field values MUST immediately follow the header field name with a colon separator (no white space permitted). Header field values MUST be represented as Quoted-Printable [XREF] with vertical bars, colons, semicolons, and white space encoded in addition to the usual requirements.

Verifiers MUST NOT use the copied header field values for verification should they be present in the h= field. Copied header field values are for forensic use only.

header fields with characters requiring conversion (perhaps from legacy MTAs which are not RFC 2822 compliant) SHOULD be converted as described in [RFC2047].

ABNF:

```
sig-z-tag    = "z=" *FWS hdr-copy *( *FWS "|" hdr-copy )
                     *FWS <hdr-copy =   hdr-name ":"
                     *FWS qp-hdr-value
qp-hdr-value = <quoted-printable text with WS,
                 "|", ":", and ";" encoded>
                     ; needs to be updated with real definition
```

```
                        ; (could be messy)
```

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

```
DKIM-Signature: a=rsa-sha1; d=example.net; s=brisbane
   c=simple; q=dns; 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
   b=dzdVyOfAKCdLXdJOc9G2q8LoXSlEniSbav+yuU4zGeeruD00lszZ
            VoG4ZHRNiYzR
```

3.6 The Authentication-Results header field

Verifiers wishing to communicate the results of verification via an email header field SHOULD use the Authentication-Results header field [ID-AUTH-RES].

3.7 Key Management and Representation

DKIM keys do not require third party signatures by Certificate Authorities in order to be trusted, since the public key is retrieved directly from the signer.

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 domain of the responsible signer (the "d=" tag of the DKIM-Signature header field), the selector (the "s=" tag), and the signing identity (the "i=" tag). The "i=" tag value could be ignored by some key services.

This document defines a single binding, using DNS to distribute the keys.

3.7.1 Textual Representation

It is expected that many key servers will choose to present the keys in a text format. The following definition MUST be used for any DKIM key represented in textual form.

The overall syntax is a key-value-list as described above. The current valid tags are:

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 response. Responses beginning with a "v=" tag with any other value MUST be discarded.

g= granularity of the key (plain-text; OPTIONAL, default is "*"). This value MUST match the local part of the signing address, with a "*" character acting as a wildcard. The intent of this tag is to constrain which signing address can legitimately use this selector. An email with a signing address that does not match the value of this tag constitutes a failed verification. Wildcarding allows matching for addresses such as "user+*". An empty "g=" value never matches any addresses.

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 "sha1" hash algorithm.

k= Key type (plain-text; OPTIONAL, default is "rsa"). Signers and verifiers MUST support the 'rsa' key type.

n= Notes that might be of interest to a human (quoted-printable; 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).

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 is defined by the k= tag.

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:

* matches all service types

email electronic mail (not necessarily limited to SMTP)

This tag is intended to permit senders to constrain the use of delegated keys, e.g., where a company is willing to delegate the right to send mail in their name to an outsourcer, but not to send IM or make VoIP calls. (This of course presumes that these keys are used in other services in the future.)

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

y This domain is testing DKIM; unverified email MUST NOT be treated differently from verified email. Verifier systems MAY wish to track testing mode results to assist the signer.

Unrecognized flags MUST be ignored.

3.7.2 DNS binding

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

3.7.2.1 Name Space.

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

The value of the "i=" tag is not used by the DNS binding.

3.7.2.2 Resource Record Types for Key Storage

This section needs to be fleshed out. ACTUALLY: will be addressed in another document.

Two RR types are used: DKK and TXT.

The DKK RR is expected to be a non-text, binary representation intended to allow the largest possible keys to be represented and transmitted in a UDP DNS packet. Details of this RR are described in [ID-DK-RR].

TXT records are encoded as described in section XINDX above.

Verifiers SHOULD search for a DKIM RR first, if possible, followed by a TXT RR. If the verifier is unable to search for a DKK RR or a DKK RR is not found, the verifier MUST search for a TXT RR.

4. Semantics of Multiple Signatures

Considerable energy has been spent discussion the desirability and semantics of multiple DKIM signatures in a single message. On the one hand, discarding existing signature header fields loses information which could prove to be valuable in the future. On the other hand, since header fields are known to be re-ordered in transit

by at least some MTAs, determining the most interesting signature header field is non-trivial.

Further confusion could occur with multiple signatures added at the same logical "depth". For example, a signer could choose to sign using different signing or canonicalization algorithms. However, even this is problematic because some of those signatures will inevitably have to sign some of the others (and at very minimum must be presented to the verification algorithm in the same order as presented to the signature algorithm).

Also, many agents are expected to break existing signatures (e.g., a mailing list that modifies Subject header fields or adds unsubscribe information to the end of the message). Retaining signature information that is known to be bad could create more problems than it solves.

For these reasons, multiple signatures are not prohibited but are left undefined.

> INFORMATIVE IMPLEMENTATION GUIDANCE: Agents that forward mail without modification could decide whether to add another signature or simply retain an existing signatures. Agents that are known to break existing signatures MAY leave the existing signature or delete it. Agents that re-sign messages that are already signed SHOULD verify the previous signature and should probably refuse to sign any critical information that failed a signature verification.

5. Signer Actions

5.1 Determine if 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.

A SUBMISSION server MAY sign if the sender is authenticated by some secure means, e.g., SMTP AUTH. Within a trusted enclave the signing address MAY be derived from the header field according to local signer policy. Within a trusted enclave an MTA MAY do the signing.

> 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.

A signer MUST NOT sign an email if the submitter is not authorized to use the signing address.

A signer SHOULD NOT remove an existing "DKIM-Signature:" header field unless that signature was added by an entity under the same domain. That is, DKIM-Signature header fields SHOULD NOT be removed unless the d= tag of that existing DKIM-Signature header field is the same as or a subdomain of the d= tag of the new DKIM-Signature header field that is being added.

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.

A signer SHOULD NOT sign with a key that is expected to expire within seven days; that is, when rotating to a new key, signing should immediately commence with the new key and the old key SHOULD be retained for at least seven days before being removed from the key server.

5.2.1 Normalize the Message to Prevent Transport Conversions

Some messages, notably those using 8-bit characters, are subject to conversion to 7-bit during transmission. Such conversions will break DKIM signatures. In order to minimize the chances of such breakage, signers SHOULD convert the message to MIME-encoded 7-bit form as described in [RFC2045] before signing.

Should the message be submitted to the signer with any local encoding that will be modified before transmission, such conversion to canonical form MUST be done before signing. In particular, some systems use local line separator conventions (such as the Unix newline character) internally rather than the SMTP-standard CRLF sequence. All such local conventions MUST be converted to canonical format before signing.

More generally, the signer MUST sign the message as it will be emitted on the wire rather than in some local or internal form.

5.2.2 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); any header field that describes the role of the signer (for example, the Sender or Resent-From header field if the signature is on behalf of the corresponding address and that address is different from the From address) MUST also be included. The signed header fields SHOULD also include the Subject and Date header fields as well as all MIME header fields. Signers SHOULD NOT sign an existing header field likely to be legitimately modified or removed in transit. In particular, RFC 2821 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. It is RECOMMENDED that all other existing, non-repeatable header fields be signed.

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 MUST sign any header fields that the signers wish to have the verifiers treat as trusted. Put another way, verifiers MAY treat unsigned header fields with extreme skepticism, up to and including refusing to display them to the end user.

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 should be added later the signature will fail.

Signers choosing to sign an existing replicated header field (such as Received) MUST sign the physically last instance of that header field in the header field block. Signers wishing to sign multiple instances of an existing replicated 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 header field names than there are actual corresponding header fields to indicate that additional header fields of that name SHOULD NOT be added. (However, header fields that can be replicated should not be signed; see below.)

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 NOT sign header fields that might be replicated (either at the time of signing or potentially in the future), with the exception of trace header fields such as Received. Comment and non standard header fields (including X-* header fields) are permitted by [RFC2822] to be replicated; however, many such header fields are, by convention, not replicated. Signers need to understand the implications of signing header field fields that might later be replicated, especially in the face of header field reordering. In particular, [RFC2822] only requires that trace header fields retain the original order.

INFORMATIVE RATIONALE: Received: is allowed because these header fields, as well as Resent-* header fields, are already order-sensitive.

INFORMATIVE ADMONITION: Despite the fact that [RFC2822] permits header field blocks to be reordered (with the exception of Received header fields), reordering of signed replicated header fields 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.

INFORMATIVE NOTE: There has been some discussion that a Sender Signing Policy include the list of header fields that the signer always signs. N.B. In theory this is unnecessary, since as long as the signer really always signs the indicated header fields there is no possibility of an attacker replaying an existing message that has such an unsigned header field.

5.2.3 Compute the Signature

The signer MUST use one of the defined canonicalization algorithms as described in section XINDX to present the email to the signing algorithm. Canonicalization is only used to prepare the email for signing; it does not affect the transmitted email in any way.

To avoid possible ambiguity, a signer SHOULD either sign or remove any preexisting "Authentication-Results:" header fields from the email prior to preparation for signing and transmission. "Authentication-Results" header fields MUST only be signed if the signer is certain of the authenticity of the preexisting header field, for example, if it is locally generated or signed by a previous DKIM-Signature line that the current signer has verified. Signers MUST NOT sign Authentication-Results header fields that could be forgeries.

Entities such as mailing list managers that implement DKIM and which modify the message or the 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; such signing SHOULD include the Authentication-Results header field, if any, inserted upon message receipt.

All tags and their values in the DKIM-Signature header field are included in the cryptographic hash with the sole exception of the value 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.

When calculating the hash on values that will be base64 or quoted-printable encoded, the hash MUST be computed after the encoding. Likewise, the verifier MUST incorporate the values into the hash

before decoding the base64 or quoted-printable text.

With the exception of the canonicalization procedure described in section XINDX, the DKIM signing process treats the body of messages as simply a string of characters. 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 which is signed.

5.2.4 Insert the DKIM-Signature header field

The final step in the signing process is that the signer MUST insert the "DKIM-Signature:" header field as defined in section XINDX prior to transmitting the email. The "DKIM-Signature" SHOULD be inserted before any header fields that it signs in the header field block.

INFORMATIVE IMPLEMENTATION NOTE: The easiest way to achieve this is to insert the "DKIM-Signature" header field at the beginning of the header field block.

6. Verifier Actions

6.1 Introduction

Since a signer MAY expire a public key at any time, it is recommended that verification occur in a timely manner with the most timely place being during acceptance by the border MTA.

A border or intermediate MTA MAY verify the message signatures and add 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. Separate policies MAY be defined for unsigned messages, messages with incorrect signatures, and when the signature cannot be verified. Treatment of unsigned messages MUST be based on the results of the Sender Signing Policy check described in [ID-DKPOLICY].

6.2 Extract the Signature from the Message

The signature and associated signing identity is included in the

value of the DKIM-Signature header field.

Verifiers MUST ignore DKIM-Signature header fields with a "v=" tag. Existence of such a tag indicates a new, incompatible version of the DKIM-Signature header field.

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 (which MUST exist).

Verifiers MUST confirm that the domain specified in the "d=" tag is the same as or a superdomain of the domain part of the "i=" tag. If not, the DKIM-Signature header field MUST be ignored.

Implementers MUST meticulously validate the format and values in the "DKIM-Signature:" header field; any inconsistency or unexpected values MUST result in an unverified email. 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 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 field 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 must be considered before using ordering.

Since there can be multiple signatures in a message, a verifier SHOULD ignore an invalid signature (regardless if caused by a syntactic or semantic problem) and try other signatures. A verifier MAY choose to treat a message with one or more invalid signatures with more suspicion than a message with no signature at all.

6.3 Get the Public Key

The public key 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 line. Obviously, a public key should only be retrieved if the process of extracting the signature information is completely successful. Details of key management are described in section XINDX. The verifier MUST validate the key record and MUST ignore any public key records that are malformed.

When validating a message, a verifier MUST:

1. Retrieve the public key as described under Key Management (XINDX) using 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 SHOULD defer acceptance of this email (normally this will be achieved with a 451/4.7.5 SMTP response code).

3. If the query for the public key fails because the corresponding RR does not exist, the verifier MUST ignore the signature.

4. If the result returned from the query does not adhere to the format defined in this specification, the verifier MUST ignore the signature.

5. If the "g=" tag in the public key does not match the local part of the "i=" tag on the message signature, the verifier MUST treat the signature as invalid. 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 not present (which defaults to *), otherwise the verifier MUST ignore the signature. Other than this test, verifiers MUST NOT treat a message signed with a key record having a g= tag any differently than one without; in particular, verifiers MUST NOT prefer messages that seem to have an individual signature by virtue of a g= tag vs. a domain signature.

6. If the "h=" tag exists in the public key record and the hash algorithm implied by the a= tag in the DKIM-Signature header is not included in the "h=" tag, the verifier MUST ignore the signature.

7. If the public key data is not suitable for use with the algorithm type defined by the "a=" tag in the "DKIM-Signature" header field, the verifier MUST ignore the signature.

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.

6.4 Compute the Verification

Given a signer and a public key, verifying a signature consists of the following steps.

- o 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, create a canonicalized copy of the email as is described in section XINDX. When matching header field names in the "h=" tag against the actual message header field, comparisons MUST be case-insensitive.

- o Based on the algorithm indicated in the "a=" tag,

  - * Compute the message hash from the canonical copy as described in section XINDX. Note that this requires presenting the "nowsp" canonicalized DKIM-Signature header field to the hash algorithm after the body of the message, and with the "b=" value treated as the empty string.

  - * Decrypt the signature using the signer's public key.

- o Compare the decrypted signature to the message hash.

  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.

Verifiers MUST 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.

If a body length is specified in the "l=" tag of the signature, verifiers MUST only verify the number of bytes indicated in the body length. Verifiers MAY decide that a message containing bytes beyond the indicated body length MAY still treat such a message as suspicious. Verifiers MAY truncate the message at the indicated body length or reject the message outright. MUAs MAY visually mark the unverified part of the body in a distinctive font or color to the end user.

6.5 Apply Sender Signing Policy

Verifiers MUST consult the Sender Signing Policy as described in [ID-DKPOLICY] and act accordingly. The range of possibilities is up to the verifier, but it MAY include rejecting the email.

6.6 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.

If the verifying MTA is capable of verifying the public key of the signer and check the signature on the message synchronously with the SMTP session and such signature is missing or does not verify, the MTA MAY reject the message with an error such as: 550 5.7.1 Unsigned messages not accepted 550 5.7.5 Message signature incorrect

If it is not possible to verify the authorization of the public key in the message, perhaps because the key server is not available, a temporary failure message MAY be generated, such as: 451 4.7.5 Unable to verify signature - key server unavailable

Once the signature has been verified, that information MUST be conveyed to higher level systems (such as explicit allow/white lists and reputation systems) and/or to the end user. If the authentication status is to be stored in the message header field, the Authentication-Results header field [ID-AUTH-RES] SHOULD be used to convey this information. 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 over-zealous filter? For diagnostic purposes, the exact reason why the verification fails SHOULD be recorded in the "Authentication-Results" header field and possibly the system logs. However in terms of presentation to the end user, the result SHOULD be presented as a simple binary result: either the email is verified or it is not. If the email cannot be verified, then it SHOULD be rendered the same as all unverified email regardless of whether it looks like it was signed or not.

Insert the Authentication-Results header field. That header field is described in [ID-AUTH-RES]. The Authentication-Results header field SHOULD be inserted before any existing DKIM-Signature or Authentication-Results header fields in the header field block.

INFORMATIVE ADVICE to MUA filter writers:

Patterns intended to search for Authentication-Results header fields to visibly mark authenticated mail for end users should verify that the Authentication-Results header field was added by the appropriate verifying domain and that the verified identity matches the sender identity that will be displayed by the MUA. In particular, MUA patterns should not be influenced by bogus Authentication-Results header fields added by attackers.

In order to retain the current semantics and visibility of the From header field, verifying mail agents SHOULD take steps to ensure that the signing address is prominently visible to the user if it is different from the From address. If MUA implementations that highlight the signed address are not available, this MAY be done by the validating MTA or MDA by rewriting the From address in a manner which remains compliant with [RFC2822]. If performed, the rewriting SHOULD include the name of the signer in the address. For example:

```
From: John Q. User <user@example.com>
```

might be converted to

```
From: "John Q. User via <asrg-admin@ietf.org>" <user@example.com>
```

This sort of address inconsistency is expected for mailing lists, but might be otherwise used to mislead the verifier, for example if a message supposedly from administration@your-bank.com had a Sender address of fraud@badguy.com.

Under no circumstances should an unsigned header field be displayed in any context that might be construed by the end user as having been signed. Notably, unsigned header fields SHOULD be hidden from the end user to the extent possible.

7. Compliance

Placeholder for Phillip H-B's suggested compliance section:

5) there should be a compliance section. I think that the spec should say what it takes for an email sender, recipient and MTA to comply with the spec in one place. In particular I think that somewhere there needs to be the statement that an SMTP forwarder is compliant with this spec IFF all messages that bear a signature are forwarded in a manner that preserves each of the canonicalization mechanisms specified.

8. IANA Considerations

Use of the _domainkey prefix in DNS records will require registration by IANA.

The DKK and DKP RR types must be registered by IANA.

9. Security Considerations

It has been observed that any mechanism that is introduced which 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.

9.1 Misuse of Body Length Limits ("l=" Tag)

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

9.1.1 Addition of new MIME parts to multipart/*

If the body length limit does not cover a closing MIME multipart header field (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.

*** Example appropriate here ***

9.1.2 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>
```

9.2 Misappropriated Private Key

If the private key for a user is resident on their computer and is not protected by an appropriately secure passphrase, it is possible for malware to send mail as that user. The malware would, however, not 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.

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 supposed 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 passphrase, 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 sender and will sign the message using its key which is normally authorized for all addresses

in the domain. 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.

9.3 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.

9.4 Attacks Against DNS

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

While the DNS is currently insecure [RFC3833], it is expected that the security problems should and will be solved by DNSSEC [RFC4033], and all users of the DNS will reap the benefit of that work.

Secondly, the types of DNS attacks relevant to DKIM are very costly and are far less rewarding than DNS attacks on other Internet applications.

To systematically thwart the intent of DKIM, an attacker must conduct a very costly and very extensive attack on many parts of the DNS over an extended period. No one knows for sure how attackers will respond, however the cost/benefit of conducting prolonged DNS attacks of this nature is expected to be uneconomical.

Finally, 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 [RFC2633] 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.

9.5 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 information via existing collaborative systems.

9.6 Limits on Revoking Key Authorization

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 which have not yet been verified or which 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 key authorization on a per-address basis have been proposed but require further study as to their utility and the DNS load they represent.

9.7 Intentionally malformed Key Records

It is possible for an attacker to publish key records in DNS which 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 DNS and be robust

against intentionally as well as unintentionally malformed key records.

9.8 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.

10. References

10.1 References -- Normative

[ID-AUTH-RES]
Kucherawy, M., "Message header field for Indicating Sender Authentication Status", draft-kucherawy-sender-auth-header field-02 (work in progress), 2005.

[ID-DK-RR]
"[*] dk rr", draft-dkk-rr-xx (work in progress), 2005.

[ID-DKPOLICY]
"[*] dk policy", draft-allman-dkim-policy (work in progress), 2005.

[ID-MAIL-ARCH]
Crocker, D., "Internet Mail Architecture", draft-crocker-email-arch-02 (work in progress), April 2005.

[OPENSSL] Team, C&D., "", WEB http://www.openssl.org/docs/, June 2005.

[RFC1421] Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures", RFC 1421, February 1993.

[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.

[RFC2234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[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.

[RFC3491] Hoffman, P. and M. Blanchet, "[*] Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)", RFC 3491, March 2003.

10.2 References -- Informative

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

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

[RFC2633] Ramsdell, B., "S/MIME Version 3 Message Specification", RFC 2633, June 1999.

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

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

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-24/2
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

Appendix A. Appendix A -- Usage Examples (INFORMATIVE)

This section taken directly from IIM without serious editing; it should be updated or deleted before publication. In no case should these examples be used as guidance when creating an implementation.

A.1 Simple message transfer

The above sections largely describe the process of signing and verifying a message which goes directly from one user to another. One special case is where the recipient has requested forwarding of the email message from the original address to another, through the use of a Unix .forward file or equivalent. In this case the message is typically forwarded without modification, except for the addition of a Received header field to the message and a change in the Envelope-to address. In this case, the eventual recipient should be able to verify the original signature since the signed content has not changed, and attribute the message correctly.

A.2 Outsourced business functions

Outsourced business functions represent a use case that motivates the need for user-level keying. Examples of outsourced business functions are legitimate email marketing providers and corporate benefits providers. In either case, the outsourced function would like to be able to send messages using the email domain of the client company. At the same time, the client may be reluctant to register a key for the provider that grants the ability to send messages for any address in the domain.

With user-level keying, the outsourcing company can generate a keypair and the client company can register the public key for a specific address such as 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 authority to revoke the key registration at any time.

A.3 PDAs and Similar Devices

PDAs are one example of the use of multiple keys per user. Suppose that John Doe wanted to be able to send messages using his corporate email address, jdoe@example.com, and the device did not have the ability to make a VPN connection to the corporate network. If the device was equipped with a private key registered for jdoe@example.com by the administrator of that domain, and appropriate software to sign messages, John could send IIM messages through the outgoing network of the PDA service provider.

A.4 Mailing Lists

There is a wide range of behavior in forwarders and mailing lists (collectively called "forwarders" below), ranging from those which make no modification to the message itself (other than to add a Received header field and change the envelope information) to those which may add header fields, change the Subject header field, add content to the body (typically at the end), or reformat the body in some manner.

Forwarders which do not modify the body or signed header fields of a message with a valid signature MAY re-sign the message as described below.

Forwarders which make any modification to a message that could result in its signature becoming invalid SHOULD sign or re-sign using an appropriate identification (e.g., mailing-list-name@example.net). Since in so doing the (re-)signer is taking responsibility for the content of the message, modifying forwarders MAY elect to forward or re-sign only for messages which were received with valid signatures or other indications that the messages being signed are not spoofed.

Forwarders which wish to re-sign a message MUST apply a Sender header field to the message to identify the address being used to sign the message and MUST remove any preexisting Sender header field as required by RFC 2822 [5]. The forwarder applies a new IIM-Sig header field with the signature, public key, and related information of the forwarder. Previously existing IIM-Sig header fields SHOULD NOT be removed.

A.5 Affinity Addresses

"Affinity addresses" are email addresses that users employ to have an email address that is independent of any changes in email service provider they may choose to make. 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, but (currently) usually depend on the user to send outgoing messages through their own service provider's MTA. They usually have an associated Web application which authenticates the user and allows the forwarding address to be changed.

With DKIM, affinity domains could use the Web application to allow users to register their own public keys to be used to sign messages on behalf of their affinity address. 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 decide to start handling outgoing mail, and could operate 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.

A.6 Third-party Message Transmission

Third-party message transmission refers to the authorized sending of mail by an Internet application on behalf of a user. For example, a website providing news may allow the reader to forward a copy of the message to a friend; this is typically done using the reader's email address. 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.

One way this can be handled is to continue to put the reader's email address in the From field of the message, but put an address owned by the site into the Sender field, and sign the message on behalf of the Sender. A verifying MTA SHOULD accept this and rewrite the From field to indicate the address that was verified, i.e., From: John Doe via news@news-site.com <jdoe@example.com>.

Appendix B. Appendix B -- Example of Use (INFORMATIVE)

This section taken directly from DK without serious editing; it should be updated or deleted before publication. In no case should these examples be used as guidance when creating an implementation.

This section shows the complete flow of an email from submission to final delivery, demonstrating how the various components fit together.

B.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.
```

B.2 The email is signed

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

```
DKIM-Signature: a=rsa-sha1; s=brisbane; d=example.com;
      c=simple; q=dns; i=joe@football.example.com;
      h=Received : From : To : Subject : Date : Message-ID;
      b=dzdVyOfAKCdLXdJOc9G2q8LoXSlEniSbav+yuU4zGeeruD00lszZ
        VoG4ZHRNiYzR;
Received: from dsl-10.2.3.4.football.example.com  [10.2.3.4]
      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. Distribution and management of private-keys is outside the scope of this document.

B.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=" header field and the selector "brisbane" from the "s=" tag in the "DKIM-Signature" header field to form the DNS DKIM query for:

```
brisbane._dkim.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 the "Authentication-Results" header field line. After successful verification, the email looks like this:

```
Authentication-Status: XXX
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: a=rsa-sha1; s=brisbane; d=example.net;
       c=simple; q=dns; i=joe@football.example.com;
       h=Received : From : To : Subject : Date : Message-ID;
       b=dzdVyOfAKCdLXdJOc9G2q8LoXSlEniSbav+yuU4zGeeruD00lszZ
          VoG4ZHRNiYzR
Received: from dsl-10.2.3.4.network.example.com  [10.2.3.4]
         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 C. Appendix C -- Creating a public key (INFORMATIVE)

Drop this section? It seems like this could clarify things for some people.

The default signature is an RSA signed SHA1 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 768 bit private-key suitable for DKIM, is to use openssl like this:

```
$ openssl genrsa -out rsa.private 768
```

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


```
-----BEGIN RSA PRIVATE KEY-----
MIIByQIBAAJhAKJ2lzDLZ8XlVambQfMXn3LRGKOD5o6lMIgulclWjZwP56LRqdg5
ZX15bhc/GsvW8xW/R5Sh1NnkJNyL/cqY1a+GzzL47t7EXzVc+nRLWT1kwTvFNGIo
AUsFUq+J6+OprwIDAQABAmBOX0UaLdWWusYzNol++nNZ0RLAtr1/LKMX3tk1MkLH
+Ug13EzB2RZjjDOWlUOY98yxW9/hX05Uc9V5MPo+q2Lzg8wBtyRLqlORd7pfxYCn
Kapi2RPMcR1CxEJdXOkLCFECMQDTO0fzuShRvL8q0m5sitIHlLA/L+0+r9KaSRM/
3WQrmUpV+fAC3C31XGjhHv2EuAkCMQDE5U2nP2ZWVlSbxOKBqX724amoL7rrkUew
ti9TEjfaBndGKF2yYF7/+g53ZowRkfcCME/xOJr58VN17pejSl1T8Icj88wGNHCs
FDWGAH4EKNwDSMnfLMG4WMBqd9rzYpkvGQIwLhAHDq2CX4hq2tZAt1zT2yYH7tTb
weiHAQxeHe0RK+x/UuZ2pRhuoSv63mwbMLEZAjAP2vy6Yn+f9SKw2mKuj1zLjEhG
6ppw+nKD50ncnPoP322UMxVNG4Eah0GYJ4DLP0U=
          -----END RSA PRIVATE KEY-----
```


Once a private-key has been generated, the openssl command can be used to sign an appropriately prepared email, like this:

```
$ openssl dgst -sign rsa.private -sha1 <input.file
```

This results in signature data similar to this when represented in Base64 [MIME] format:


```
   aoiDeX42BB/gP4ScqTdIQJcpAObYr+54yvctqc4rSEFYby9+omKD3pJ/TVxATeTz
msybuW3WZiamb+mvn7f3rhmnozHJ0yORQbnn4qJQhPbbPbWEQKW09AMJbyz/0lsl
```


How this signature is added to the email is discussed later in this

document. 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-----
MHwwDQYJKoZIhvcNAQEBBQADawAwaAJhAKJ2lzDLZ8XlVambQfMXn3LRGKOD5o6l
MIgulclWjZwP56LRqdg5ZX15bhc/GsvW8xW/R5Sh1NnkJNyL/cqY1a+GzzL47t7E
XzVc+nRLWT1kwTvFNGIoAUsFUq+J6+OprwIDAQAB
          -----END PUBLIC KEY-----
```

This public-key data (without the BEGIN and END tags) is placed in the DNS. With the signature, canonical email contents and public key, a verifying system can test the validity of the signature. The openssl invocation to verify a signature looks like this: openssl dgst -verify rsa.public -sha1 -signature signature.file <input.file

Appendix D. Glossary

Appendix E. Acknowledgements

The authors wish to thank Russ Allbery, Edwin Aoki, Claus Asmann, Steve Atkins, Fred Baker, Mark Baugher, Nathaniel Borenstein, Dave Crocker, Michael Cudahy, Dennis Dayman, Jutta Degener, Patrik Faltstrom, Duncan Findlay, Elliot Gillum, Phillip Hallam-Baker, Tony Hansen, Arvel Hathcock, Don Johnsen, Harry Katz, Murray S. Kucherawy, Barry Leiba, John Levine, Simon Longsdale, David Margrave, Justin Mason, David Mayne, Russell Nelson, Dave Oran, Shamim Pirzada, Juan Altmayer Pizzorno, Sanjay Pol, Blake Ramsdell, Christian Renaud, Scott Renfro, Dave Rossetti, the Spamhaus.org team, Malte S. Stretz, Robert Sanders, Rand Wacker, 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 <http://domainkeys.sourceforge.net/license/patentlicense1-1.html>.

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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