DKIM E. Allman
Internet-Draft Sendmail, Inc.
Expires: October 15, 2006 J. Callas
PGP Corporation
M. Delany
M. Libbey
Yahoo! Inc
J. Fenton
M. Thomas
Cisco Systems, Inc.
April 13, 2006

DomainKeys Identified Mail Signatures (DKIM)
draft-ietf-dkim-base-01

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 proving and protecting 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].

Table of Contents

1. Introduction

[[Note: text in double square brackets (such as this text) will be deleted before publication.]]

1.1 Overview

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. S/MIME [RFC1847], OpenPGP [RFC2440]) in that:

- the message signature is written to the message header fields so that neither human recipients nor existing MUA (Mail User Agent) 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,
- it makes no attempt to include encryption as part of the mechanism.

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
- 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

o allows delegation of signing to third parties.

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 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.

> 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 over 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 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].

2.1 Signers

Elements in the mail system that sign messages 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 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[RFC2822].

o SWSP is streaming white space; it adds the CR and LF characters.

o FWS, also from [RFC2822], is folding white space. It allows multiple lines separated by CRLF followed by at least one white space, to be joined.

The formal ABNF for SWSP is:

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

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 / "+" / "/" / "=" / SWSP)
```

2.5 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, as they have been obsoleted and are expected to go away in future

editions of those RFCs.

The following tokens are imported from [RFC2821]:

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 [RFC2822]:

o WSP (space or tab)

o FWS (folding white space)

o field-name (name of a header field)

o dot-atom (in the local-part of an email address)

The following tokens are imported from [RFC2045]:

o qp-section (a single line of quoted-printable-encoded text)

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

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.

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

- 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 a fingerprint of the public key.

3.2 Tag=Value Lists

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; however, no encoding may include the semicolon (";") character, since that separates tag-specs.

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 =  0*VALCHAR    ; SWSP prohibited at beginning and end
     VALCHAR   =  %9 / %d32 - %d58 / %d60 - %d126
                          ; HTAB and SP to TILDE except SEMICOLON
     ALNUMPUNC =  ALPHA / DIGIT / "_"
```

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.

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 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. 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.

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 as the hash-alg. That hash is then encrypted 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.

3.3.2 The rsa-sha256 Signing Algorithm

The rsa-sha256 Signing Algorithm computes a message hash as described in Section 3.7 below using SHA-256 as the hash-alg. That hash is then encrypted by the signer using the RSA algorithm (actually 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 Other algorithms

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

3.3.4 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. Security 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:

o The practical constraint that large keys may not fit within a 512 byte DNS UDP response packet

- o The security constraint that keys smaller than 1024 bits are subject to off-line 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

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

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 an authentication 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 white-space replacement and header field line re-wrapping. 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. Further canonicalization algorithms MAY be defined in the future; verifiers MUST ignore any signatures that use unrecognized canonicalization algorithms.

In all cases, the header fields of the message are presented to the signing algorithm first in the order indicated by the signature header field and canonicalized using the indicated algorithm. 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, followed by the canonicalized body. Note that the header and body may use different 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 (e.g., 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 Field 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 white space MUST NOT be changed.

3.4.2 The "relaxed" Header Field Canonicalization Algorithm

The "relaxed" header canonicalization algorithm MUST apply the following steps in order:

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

- o 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.

- o 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.

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

- o 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.

[NON-NORMATIVE DOCUMENTATION NOTE: The only difference between "relaxed" header field canonicalization and "nowsp" listed in the previous version of this draft is that nowsp reduces all strings of streaming white space to zero characters while "relaxed" reduces strings of white space to one space.]

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. It makes no other changes to the message body. In more formal terms, the "simple" body canonicalization algorithm reduces "CRLF 0*CRLF" at the end of the body to a single "CRLF".

3.4.4 The "relaxed" Body Canonicalization Algorithm

[[This section may be deleted; see discussion below.]] The "relaxed" body canonicalization algorithm:

- o Ignores all white space at the end of lines. Implementations MUST NOT remove the CRLF at the end of the line.

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

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

  [[NON-NORMATIVE DISCUSSION: The authors are undecided whether to leave the "relaxed" body canonicalization algorithm in to the specification or delete it entirely. We believe that for the vast majority of cases, the "simple" body canonicalization algorithm should be sufficient. We simply do not have enough data to know whether to retain the "relaxed" body canonicalization algorithm or not.]]

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. If the body length count is not specified then the entire message body is signed and verified.

INFORMATIVE IMPLEMENTATION NOTE: Body length limits could be useful in increasing signature robustness when sending to a

mailing list that both appends to content sent to it and does not sign its messages. However, using such limits enables an 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, 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 is made following the canonicalization algorithm; for example, any white space ignored by a canonicalization algorithm is not included as part of the body length count.

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.

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

3.4.6 Example

(In the following examples, actual white space is used only for clarity. The actual input and output text is designated using

bracketed descriptors: "<SP>" for a space character, "<TAB>" 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.)

A message reading:

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

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

```
a:X <CRLF>
b:Y <SP> Z <CRLF>
<CRLF>
<SP> C <CRLF>
D <SP> E <CRLF>
```

The same message canonicalized using simple canonicalization for both header and body results in:

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

When processed using relaxed header canonicalization and simple body canonicalization, the canonicalized version reads:

```
a:X <CRLF>
b:Y <SP> Z <CRLF>
<CRLF>
<SP> C <SP><CRLF>
D <SP><TAB><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 signaturee 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. 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 always included in the signature calculation, after the body of the message; however, when calculating or verifying the signature, the value of the b= tag (signature value) MUST be treated as though it were the null string. Unknown tags MUST be signed and verified 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 MIME Part One [RFC2045], with the additional conversion of semicolon characters to "=3B".

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

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.

ABNF:

```
sig-v-tag =
```

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   = "rsa-sha1" / "rsa-sha256" / x-sig-a-tag-alg
x-sig-a-tag-alg = hyphenated-word   ; for later extension
```

b= The signature data (base64; REQUIRED). Whitespace is ignored in this value and MUST be ignored when re-assembling 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-tagg-data  = base64string
```

bh= The hash of the body part of the message (base64; REQUIRED). Whitespace is ignored in this value and MUST be ignored when re-assembling 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.

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

ABNF:

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

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. Folding white space (FWS) MAY be included on either side of the colon separator. Header ffield 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 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 MAY be omitted. 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 =   %x69 [FWS] "=" [FWS] [ Local-part ] "@" Domain
```

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 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 aauthentication 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 after canonicalization 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 (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.

ABNF:

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

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 iis "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.

ABNF:

```
sig-q-tag        = %x71 [FWS] "=" [FWS] sig-q-tag-method
                     *([FWS] ":" [FWS] sig-q-tag-method)
sig-q-tag-method = sig-q-tag-type ["/" sig-q-tag-args]
sig-q-tag-type   = "dns" / x-sig-q-tag-type
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] Domain
```

t= Signature Timestamp (plain-text; 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.

ABNF:

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

x= Signature Expiration (plain-text; 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. 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, with the same contraints on the value in the "t=" tag. 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 (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 present when the message was signed. This field need not contain the same header fields listed in the "h=" tag. 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 [RFC2045] with vertical bars, colons, semicolons, and white space encoded in addition to the usual requirements.

Verifiers MUST NOT use the header field names or copied values for checking the signature in any way. Copied header field values are for diagnostic use only.

Header fields with characters requiring conversion (perhaps from legacy MTAs which 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 ":" [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 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 responsible signer (the "d=" tag of the DKIM-Signature header field), the signing identity (the "i=" tag), and the selector (the "s=" tag). The "i=" tag value could be ignored by some key services.

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

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

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

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 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.

ABNF:

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

[[NON-NORMATIVE DISCUSSION POINT: "*" is legal in a dot-atom. This should probably use a different character for wildcarding. Unfortunately, the options are non-mnemonic

(e.g., "@", "(", ":"). Alternatively we could insist on escaping a "*" intended as a literal "*" in the address.]]

h= Accceptable 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.

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 RSA public key, as defined in [RFC3447], 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
```

[[NON-NORMATIVE DISCUSSION NOTE: In some cases it can be hard to separate h= and k=; for example DSA implies that SHA-1 will be used. This might be an actual change to the spec depending on how we decide to fix this.]]

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).

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 is defined by the k= tag.

ABNF:

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

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 courrse presumes that these keys are used in other services in the future.)

ABNF:

```
key-s-tag        = %x73 [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:

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.

ABNF:

```
key-t-tag        = %x74 [FWS] "=" [FWS] key-t-tag-flag
                   0*( [FWS] ":" [FWS] key-t-tag-flag )
key-t-tag-flag   = "y" / 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 as a key service is hereby defined. All implementations MUST support this binding.

3.6.2.1 Name Space

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 ""sample"", the DNS query will be for ""sample._domainkey.example.com"".

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

3.6.2.2 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-DKIM-RR].

TXT records are encoded as described in Section 3.6.1.

Verifiers SHOULD search for a DKK 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.

3.7 Computing the Message Hashes

Both signing and verifying message signatures starts with a step of computing two cryptographic hash 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 which 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 or verifier must compute two hashes, one over the body of the message and one over the header 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 or verifier MUST hash the message body, canonicalized using the header canonicalization algorithm specified in the "c=" tag and truncated to the length specified in the "l=" tag. That hash value is then converted to base64 form and inserted into the "XXX=" tag of the DKIM-Signature: header field.

In hash step 2, the signer or 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.

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 value portion of the "b=" tag (that is, the portion after the "=" sign) must be treated as though it were empty, 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.

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 heading 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 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."

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 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.

More formally, the algorithm for the signature is:

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

where crypt-alg is the encryption 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.

4. Semantics of Multiple Signatures

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 headers using the method described in section NN to sign trace headers. Signers should be cognizant that signing DKIM-Signature headers may result in signature failures with intermediaries that do not recognize that DKIM-Signature's are trace headers and unwittingly reorder them.

When evaluating a message with multiple signatures, a receiver should evaluate signatures independently and on their own merits. For example, a receiver that by policy chooses not to accept signatures with deprecated crypto algorithms should consider such signatures invalid. As with messages with a single signature, receievers are at liberty to use the presence of valid signatures as an input to local policy; likewise, the interpretation of multiple valid signatures in combination is a local policy decision of the receiver.

Signers SHOULD NOT remove any DKIM-Signature headers from messages

they are signing, even if they know that the headers cannot be verified.

5. Signer Actions

The following steps are performed in order by signers.

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 it is unwilling to be held responsible for the message; in particular, the signer SHOULD ensure that the submitter has a bona fide relationship with the signer and that the submitter has the right to use the address being claimed.

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.

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.

5.3 Normalize the Message to Prevent Transport Conversions

Some messages, particularly those using 8-bit characters, are subject to modification during transitt, 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.

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 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); 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, [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. 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 assert were present at the time of signing. 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=D 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.

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.

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 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 shoulld 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.

5.5 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 which will include the body hash and a signature of the header hash, where that header includes the DKIM-Signature header field itself.

To avoid possible ambiguity, a signer SHOULD either sign or remove any preexisting header fields which convey the results of previous verifications of the message signature prior to preparation for signing and transmission. Such header fields MUST NOT be signed if the signer is uncertain of the authenticity of the preexisting header field, for example, if it is not locally generated or signed by a previous DKIM-Signature line that the current signer has verified.

Entities such as mailing list managers that implement DKIM and which 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; such signing SHOULD include any prior verification status, if any, that was inserted upon message receipt.

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.

INFORMATIVE NOTE: A possible value to include in the "l=" tag would include the entire length of the message being signed, thereby allowing intermediate agents to append further information to the message without breaking the signature (e.g., a mailing list manager might add unsubscribe innformation to the body). A signer wishing to permit such intermediate agents to add another MIME body part to a "multipart/mixed" message should use a length that covers the entire presented message except for the trailing "--CRLF" characters; this is known as the "N-4" approach. Note that more than four characters may need to be stripped, since there could be postlude information that needs to be ignored.

5.6 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" SHOULD be inserted before any header fields that it signs 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.

6. Verifier Actions

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.

Verifiers MUST apply the following steps in the order listed. In many cases these steps say that a "DKIM-Signature" header field must be ignored, e.g., because it is malformed or because the signature verification failed. In such cases verifiers SSHOULD proceed to the next signature, and treat the message as verified if any signature succeeded, ignoring the bad signatures. The order in which signatures are tried is a matter of local policy for the verifier and is not defined here. A verifier MAY treat a message that has one or more bad signatures and no good signatures differently from a message with no signature at all; again, this is local policy and is beyond the scope of this document.

6.1 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 cause the header field to be completely ignored. 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 and no valid signatures with more suspicion than a message with no signature at all.

6.2 Get the Public Key

The public key is needed to complete the verificatiion 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 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 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 reply 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 ignore the signature. 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 (the "p=" tag) is empty then this key has been revoked and the verifier MUST treat this as a failed signature check.

8. 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 signature is to be ignored, verifiers SHOULD search for another signature in the message.

6.3 Compute the Verification

Given a signer and a public key, verifying a signature consists of 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, create a canonicalized copy of the email as is described in Section 3.7. 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.

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

3. Compare the decrypted signature to the message hash. If they are identical, the hash computed by the signer must be the same as the hash computed by the verifier, and hence the signature verifies; otherwise, the signature fails.

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 match, the entire signature must be considered to have failed.

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 eveen 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 to treat a message containing bytes beyond the indicated body length with suspicion. Verifiers MAY truncate the message at the indicated body length or reject the signature outright.

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 described in the informative note inSection 5.5. 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.

6.4 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. An example proposal for a header field is the Authentication-Results header field [ID-AUTH-RES]. Any such 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 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 sender identity that will be displayed by the MUA. In particular, MUA patterns should not be influenced by bogus results header fields added by attackers.

6.5 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 fetch the public key, 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 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.

INFORMATIVE NOTE: If the authentication status is to be stored in the message header field, the Authentication-Results header field [ID-AUTH-RES] may be used to convey this information.

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.

6.6 MUA Considerations

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. MUAs MAY visually mark the unverified part of the body in a distinctive font or color to the end user.

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]. Such modifications MUST be performed after the final verification

step since they will break the signature. 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.

The MUA MAY hide or mark portions of the message body that are not signed wh en using the "l=" tag.

7. IANA Considerations

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

To avoid conflicts, tag names for the DKIM-Signature header and key records should be registered with IANA.

Tag values for the "a=", "c=", and "q=" tags in the DKIM-Signature header field, and the "h=", "k=", "s=", and "t" tags in key records should be registered with IANA for the same reason.

The DKK RR type must be registered by IANA.

8. 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. See also [ID-DKIM-THREATS].

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.

*** Example appropriate here ***

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, 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 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 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 [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 siggning 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.

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 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 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 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 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 unintentiionally 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.

9. References

9.1 Normative References

[ID-DKIM-RR]
"DKIM Key Resource Records (To be written)", draft-dkim-dkk-rr-xx (work in progress), 2005.

[ID-SHA] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and HMAC-SHA)", draft-eastlake-sha2-02 (work in progress), January 2006.

[OPENSSL] Team, C&D., "OpenSSL Documents", http://www.openssl.org/docs/,

<http://www.openssl.org/docs/>.

[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.

[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.

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

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

9.2 Informative References

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

[ID-DKIM-THREATS] Fenton, J., "Analysis of Threats Motivating DomainKeys Identified Mail (DKIM)", draft-fenton-dkim-threats-02 (work in progress), December 2005.

[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.

[RFC3766] "", 2005.

[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.

[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. 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.

A.1 The user composes an email

```
From: Joe SixPack <joe@foootball.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: 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.

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._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-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: 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.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 B. Usage Examples (INFORMATIVE)

Studies in this appendix are for informational purposes only. In no case should these examples be used as guidance when creating an implementation.

B.1 Simple Message Forwarding

In some cases the recipient may request forwarding of email messages from the original address to another, through the use of a Unix .forward file or equivalent. In this case messages are 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.

B.2 Outsourced Business Functions

Outsourced business functions represent a use case that motivates the need for selectors (the "s=" signature tag) and granularity (the "g=" key tag). 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.

The outsourcing company can generate a keypair and the client company can register the public key using a unique selector for a specific address such as winter-promotions@example.com by specifying a granularity of "g=winter-promotions" or "g=*-promotions" (to allow a range of addresses). 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.

B.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 signed messages through the outgoing network of the PDA service provider.

B.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 [RFC2822]. The forwarder applies a new DKIM-Signature header field with the signature, public key, and related information of the forwarder.

B.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 aassociated 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 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.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 C. Creating a public key (INFORMATIVE)

XXX Update to 1024 bit key and SHA-256 and adjust examples accordingly. XXX

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+OpprwIDAQABAmBOX0UaLdWWusYzNol++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-----
```


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
```

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 elsewhere in this document.

Appendix D. Acknowledgements

The authors wish to thank Russ Allbery, Edwin Aoki, Claus Assmann, 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, Amir Herzberg, Craig Hughes, Don Johnsen, Harry Katz, Murray S. Kucherawy, Barry Leiba, John Levine, Simon Longsdale, David Margrave, Justin Mason, David Mayne, Steve Murphy, Russell Nelson, Dave Oran, Doug Otis, Shamim Pirzada, Juan Altmayer Pizzorno, Sanjay Pol, Blake Ramsdell, Christian Renaud, Scott Renfro, Dave Rossetti, Hector Santos, 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>.

Appendix E. Edit History

[[This section to be removed before publication.]]

E.1 Changes since -ietf-00 version

The following changes were made between draft-ietf-dkim-base-00 and draft-ietf-dkim-base-01:

- o Added section 8.9 (Information Leakage).
- o Replace section 4 (Multiple Signatures) with much less vague text.
- o Fixed ABNF for base64string.
- o Added rsa-sha256 signing algorithm.
- o Expanded several examples.
- o Changed signing algorithm to use separate hash of the body of the message; this is represented as the "bh=" tag in the DKIM-

Signature header field.

- o Changed "z=" tag so that it need not have the same header field names as the "h=" tag.

- o Significant wordsmithing.

E.2 Changes since -allman-01 version

The following changes were made between draft-allman-dkim-base-01 and draft-ietf-dkim-base-00:

- o Remove references to Sender Signing Policy document. Such consideration is implicitly included in Section 6.5.

- o Added ABNF for all tags.

- o Updated references (still includes some references to expired drafts, notably [ID-AUTH-RES].

- o Significant wordsmithing.

E.3 Changes since -allman-00 version

The following changes were made between draft-allman-dkim-base-00 and draft-allman-dkim-base-01:

- o Changed "c=" tag to separate out header from body canonicalization.

- o Eliminated "nowsp" canonicalization in favor of "relaxed", which is somewhat less relaxed (but more secure) than "nowsp".

- o Moved the (empty) Compliance section to the Sender Signing Policy document.

- o Added several IANA Considerations.

- o Fixed a number of grammar and formatting errors.

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might oor might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

The IETF has been notified of intellectual property rights claimed in regard to some or all of the specification contained in this document. For more information consult the online list of claimed rights.

Disclaimer of Validity

Copyright Statement

Acknowledgment

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