

DKIM
Internet-Draft
Intended status: Standards Track
Expires: July 18, 2011

D. Crocker, Ed.
Brandenburg InternetWorking
M. Kucherawy, Ed.
Cloudmark
January 14, 2011

DomainKeys Security Tagging (DOSETA)
draft-crocker-dkim-doseta-00

Abstract

DomainKeys Security Tagging (DOSETA) is a component mechanism that enables development of a security-related service, such as authentication or encryption, with keys based on domain names; the name owner can be any actor involved in the handling of the data, such as the author's organization, a server operator or one of their agents. The DOSETA Library provides a collection of common capabilities, including canonicalization, parameter tagging, and retrieval of self-certified keys. The DOSETA Signing Template affixes a signature to data that is in a "header/content" form. Defining the meaning of the signature is the responsibility of the service that incorporates DOSETA. The signature is validated through a cryptographic signature and querying the signer's domain directly, to retrieve the appropriate public key.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 18, 2011.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

Internet-Draft

DOSETA

January 2011

This document is subject to [BCP 78](http://trustee.ietf.org/license-info) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	4
1.1.	DOSETA Features	5
1.2.	DOSETA Architecture	6
2.	Terminology and Definitions {rfc4871bis-02 2, subset}	6
2.1.	Identity	7
2.2.	Actors	7
2.3.	Syntax	8
3.	DOSETA Library	10
3.1.	Canonicalization {rfc4871bis-02 3.4}	11
3.2.	Tag=Value Parameters {rfc4871bis-02 3.2}	15
3.3.	Key Management {rfc4871bis-02 3.6}	16
3.4.	Selectors for Keys {rfc4871bis-02 3.1}	17
3.5.	DNS Binding for Key Retrieval {rfc4871bis-02 3.6.2} . . .	19
3.6.	Stored Key Data {rfc4871bis-02 3.6.1, subset, with preface}	20
4.	DOSETA H/C Signing Template	22
4.1.	Cryptographic Algorithms {rfc4871bis-02 3.3, subset} . . .	22
4.2.	Signature Data Structure {rfc4871bis-02 3.5, subset} . . .	24
4.3.	Signature Calculation {rfc4871bis-02 3.7, reorganized and clarified}	30
4.4.	Signer Actions {rfc4871bis-02 5}	33
4.5.	Verifier Actions {rfc4871bis-02 6}	37
4.6.	Requirements for Tailoring the Signing Service {added} . .	44
4.7.	Signing by Parent Domains {rfc4871bis-02 3.8}	44
5.	Semantics of Multiple Signatures {rfc4871bis-02 4}	45
5.1.	Example Scenarios {rfc4871bis-02 4.1}	45
5.2.	Interpretation {rfc4871bis-02 4.2}	46
6.	Considerations	47
6.1.	IANA Considerations {rfc4871bis-02 7, subset}	47
6.2.	Security Considerations {rfc4871bis-02 8, subset}	50

7.	References	56
7.1.	Normative References	56
7.2.	Informative References	57
Appendix A.	Creating a Public Key {rfc4871bis-02 C}	58
Appendix B.	Acknowledgements	59

Authors' Addresses	59
------------------------------	--------------------

Internet-Draft

DOSETA

January 2011

1. Introduction

DomainKeys Security Tagging (DOSETA) is a component mechanism that enables development of a security-related service, such as authentication or encryption, with keys based on domain names [[RFC1034](#)]; the name owner can be any actor involved in the handling of the data, such as the author's organization, a server operator or one of their agents. The DOSETA Library provides a collection of common capabilities, including canonicalization, parameter tagging, and retrieval of self-certified keys. The DOSETA Signing Template affixes a signature to data that is in a "header/content" form. Defining the meaning of the signature is the responsibility of the service that incorporates DOSETA. The signature is validated through a cryptographic signature and querying the signer's domain directly, to retrieve the appropriate public key.

The approach taken by DOSETA differs from previous approaches to data signing (such as, Secure/Multipurpose Internet Mail Extensions (S/MIME) [[RFC1847](#)], OpenPGP [[RFC4880](#)]) in that:

- o the message signature can be packaged independently of the data it is signing, so that neither human viewers of the data nor existing data handling software is confused by signature-related content appearing in the Content;
- o there is no dependency on public and private key pairs being issued by well-known, trusted certificate authorities;
- o there is no dependency on the deployment of any new Internet

protocols or services for public key distribution or revocation;

- o no attempt is made to include encryption as part of the mechanism;

DOSETA:

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

DOSETA is taken directly from the original DKIM Signing specification

[[RFC4871](#)], [[RFC5672](#)]. It has merely extracted the core portions of the specification, so that they can be applied to other security-related services. For example, the core could support a DKIM-like signing service for web pages, and it could support a data encryption mechanism using the same DNS-based, self-certified key service as DKIM.

NOTE: Much of the text for this draft is taken from the DKIM working group [draft-ietf-DKIM-rfc4871bis-02](#) revision. Sections in this document cross reference their source with the notation:

{rfc4871bis-02 xx}

where "xx" indicates the section number. It might also indicate that a subset is taken, such as when a portion is applied to the DKIM-over-DOSETA draft and a portion to the DOSETA draft. In some cases the base text also has been enhanced.

[1.1](#). DOSETA Features

DOSETA features include:

Identity: DOSETA separates the question of the identity of the DOSETA Creator from any other associated identifier, such as the data's purported author. In particular, a DOSETA header includes the identity of the signer. DOSETA Consumers can use the DOSETA identity information to decide how they want to process the data. That identity can be included in the attributes of the data or recorded elsewhere.

NOTE: DOSETA does not, itself, require that the identity it uses be required to match any other associated identifier. Those other identifiers already carry their own semantics which might conflict with the use of the identifier needed by DOSETA. However a particular service that incorporates DNS might choose to add constraints on the choice of identifier, such as having it match another identifier associated with the data.

Scalability: DOSETA is designed to support the extreme scalability requirements that characterize Internet data identification.

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

self-certifying keys.

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

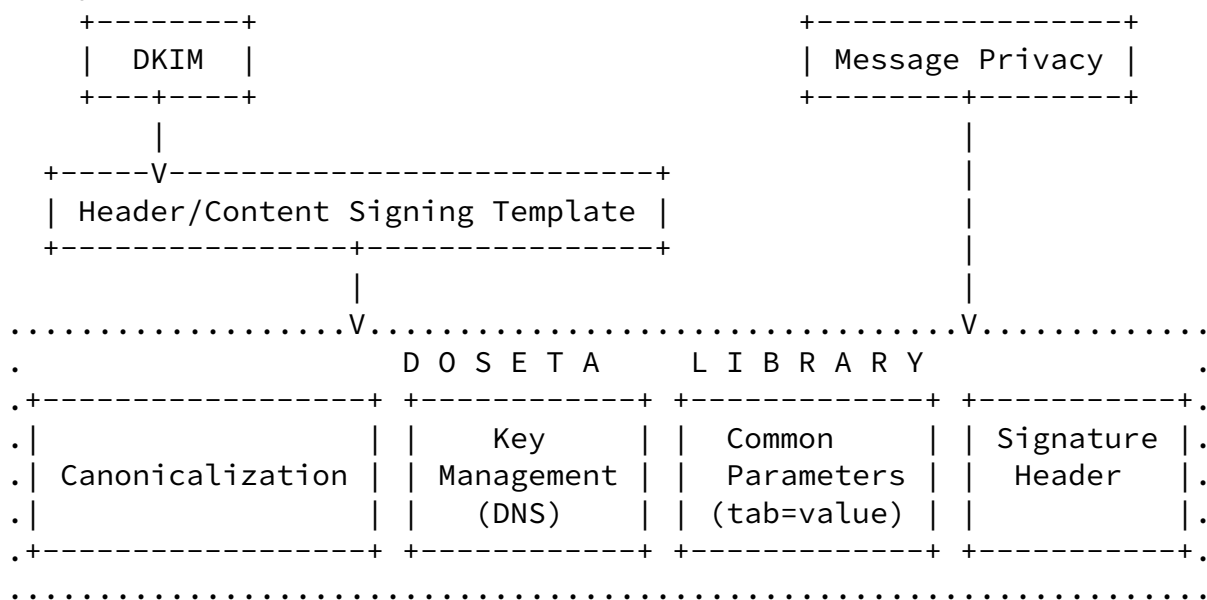
Data Integrity: A DOSETA signature associates its identifier with the computed hash of some or all of the data in order to prevent the re-use of the signature with different data. Verifying the signature asserts that the hashed data has not changed since it was signed, and asserts nothing else about "protecting" the end-to-end integrity of the data.

Assessment: For identity and authentication related processes,

this phase integrates the validation output for determining further action.

1.2. DOSETA Architecture

As component technology, DOSETA is meant to be incorporated into a service. This specification provides an underlying set of common features and a template for using them to provide a signing service, such as for authenticating an identifier. Hence, the pieces can be depicted as follows, with DKIM being shown as a specific service that incorporates DOSETA:



2. Terminology and Definitions {rfc4871bis-02 2, subset}

This section defines terms used in the rest of the document.

Within the specification, the label "TEMPLATE" is used to indicate actions that are required to incorporate DOSETA into a service.

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

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

[2.1.](#) Identity

Identity: A person, role, or organization. In the context of DOSETA, examples include author, author's organization, an ISP along the handling path, an independent trust assessment service, and a data processing intermediary operator.

Identifier: A label that refers to an identity.

Signing Domain Identifier (SDID): A single domain name that refers to the identity owning the key to be used for DOSETA processing. The name has only basic domain name semantics; any possible owner-specific semantics are outside the scope of DOSETA. It is specified in [Section 4.2](#).

[2.2.](#) Actors

Creator: An element in the data handling system that produces a cryptographic encoding, on behalf of a domain, is referred to as a Creator.

Consumer: An element in the data handling system that processes an existing cryptographic encoding, on behalf of a domain, is referred to as a Consumer.

Signer: An element in the data handling system that signs messages on behalf of a domain is referred to as a signer. This element specifies the organization acting as a type of DOSETA Creator. It might be a client, server or other agent such as a reputation service. The key issue is that the data **MUST** be signed before it leaves the administrative domain of the signer.

Verifier: An element in the data handling system that verifies signatures is referred to as a verifier. This element is a Consumer of a signing service. It might be a client, server, or other agent, such as a reputation service. In most cases it is expected that a verifier will be close to an end user (Creator) of the data or some consuming agent such as a data processing intermediary.

[2.3.](#) Syntax

[2.3.1.](#) Whitespace

There are three forms of whitespace:

WSP: represents simple whitespace, that is, a space or a tab character (formal definition in [[RFC5234](#)]).

LWSP: is linear whitespace, defined as WSP plus CRLF (formal definition in [[RFC5234](#)]).

FWS: is folding whitespace. It allows multiple lines separated by CRLF followed by at least one whitespace, to be joined.

The formal syntax for these are (WSP and LWSP are given for information only):

ABNF:

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

The definition of FWS is identical to that in [[RFC5322](#)] except for the exclusion of obs-FWS.

[2.3.2.](#) Common ABNF Tokens

The following tokens are used in this document:

ABNF:

```
hyphenated-word = ALPHA
                  [ *(ALPHA / DIGIT / "-")
                    (ALPHA / DIGIT) ]
ALPHADIGITPS    = (ALPHA / DIGIT / "+" / "/" )
base64string    = ALPHADIGITPS *([FWS] ALPHADIGITPS)
                  [ [FWS] "=" [ [FWS] "=" ] ]
hdr-name        = field-name
qp-hdr-value    = D-quoted-printable
                  ; with "|" encoded
```

[2.3.3.](#) Imported ABNF Tokens

The following tokens are imported from other RFCs as noted. Those RFCs SHOULD be considered definitive.

From [[RFC5321](#)]:

<local-part> (implementation warning: this permits quoted strings)

<sub-domain>

From [[RFC5322](#)]:

<field-name> (name of a header field)

From [[RFC2045](#)]:

<qp-section> a single line of quoted-printable-encoded text

<hex-octet> a quoted-printable encoded octet)

NOTE: Be aware that the ABNF in [[RFC2045](#)] does not obey the rules of [[RFC5234](#)] and MUST be interpreted accordingly, particularly as regards case folding.

Other tokens not defined herein are imported from [[RFC5234](#)]. These are intuitive primitives such as SP, HTAB, WSP, ALPHA, DIGIT, CRLF, etc.

[2.3.4.](#) D-Quoted-Printable

The D-Quoted-Printable encoding syntax resembles that described in Quoted-Printable [[RFC2045](#)], [Section 6.7](#):

- o Any character MAY be encoded as an "=" followed by two hexadecimal digits from the alphabet "0123456789ABCDEF" (no lowercase characters permitted) representing the hexadecimal-encoded integer value of that character.

- o All control characters (those with values < %x20), 8-bit characters (values > %x7F), and the characters DEL (%x7F), SPACE

(%x20), and semicolon (";", %x3B) MUST be encoded.

- o All whitespace, including SPACE, CR, and LF characters, MUST be encoded.
- o After encoding, FWS MAY be added at arbitrary locations in order to avoid excessively long lines; such whitespace is NOT part of the value, and MUST be removed before decoding.

The formal syntax for D-Quoted-Printable is:

ABNF:

```
D-quoted-printable = *(FWS / hex-octet / D-safe-char)
                    ; hex-octet is from RFC2045
D-safe-char        = %x21-3A / %x3C / %x3E-7E
                    ; '!' - ':', '<', '>' - '~'
                    ; Characters not listed as "mail-safe"
                    ; in RFC2049 are also not
                    ; recommended.
```

D-Quoted-Printable differs from Quoted-Printable as defined in [RFC2045](#) in several important ways:

1. Whitespace in the input text, including CR and LF, MUST be encoded. [RFC2045](#) does not require such encoding, and does not permit encoding of CR or LF characters that are part of a CRLF line break.
2. Whitespace in the encoded text is ignored. This is to allow tags encoded using D-Quoted-Printable to be wrapped as needed. In particular, [RFC2045](#) requires that line breaks in the input be represented as physical line breaks; that is not the case here.
3. The "soft line break" syntax ("=" as the last non-whitespace character on the line) does not apply.
4. D-Quoted-Printable does not require that encoded lines be no more

than 76 characters long (although there might be other requirements depending on the context in which the encoded text is being used).

[3.](#) DOSETA Library

DOSETA is distinguished by a DNS-based, self-certifying public key mechanism, common canonicalization algorithms, and a common parameter

Crocker & Kucherawy

Expires July 18, 2011

[Page 10]

Internet-Draft

DOSETA

January 2011

encoding mechanism.

[3.1.](#) Canonicalization {rfc4871bis-02 3.4}

Some data handling systems modify the original data, potentially invalidating a cryptographic function, such as with a signature. For most signers, mild modification of data is immaterial to validation of the DOSETA domain name's use. For such signers, a canonicalization algorithm that survives modest handling modification is preferred.

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

To satisfy basic requirements, two canonicalization algorithms are defined: a "simple" algorithm that tolerates almost no modification and a "relaxed" algorithm that tolerates common modifications such as whitespace replacement and data line rewrapping.

Data handling systems sometimes treat different portions of text differentially and might be subject to more or less likelihood of breaking a signature. DOSETA currently covers two types of data:

Header: Attribute:value sets, in the style of an Internet Mail header fields

Content: Lines of ASCII text

Some DOSETA Creators might be willing to accept modifications to some portions of the data, but not other portions. For DOSETA, a Creator MAY specify either algorithm for one portion of the data and another for a different portion when signing the data.

If no canonicalization algorithm is specified, the "simple" algorithm defaults for all of the data. DOSETA Creators MUST implement both canonicalization algorithms. Because further canonicalization algorithms might be defined in the future, Creators MUST ignore any signatures that use unrecognized canonicalization algorithms.

Canonicalization simply prepares the data for presentation to the signing or verification algorithm. It MUST NOT change the transmitted data in any way. Canonicalization of distinct data portions is described below.

NOTE: This section assumes that data is already in "network normal" format (text is ASCII encoded, lines are separated with CRLF characters, etc.). See also [Section 4.4.3](#) for information about normalizing the message.

[3.1.1](#). Header Canonicalization Algorithms

This section specifies initial entry for the IANA registry defined in Table 2.

simple: The "simple" header canonicalization algorithm is for a set of "attribute:value" textual data structures, such as email header fields [[RFC5322](#)]. It does not change the original Header fields in any way. Header fields MUST be presented to the signing or verification algorithm exactly as they are in the message being signed or verified. In particular, header field names MUST NOT be case folded and whitespace MUST NOT be changed.

relaxed: The "relaxed" header canonicalization algorithm is for a set of "attribute:value" textual data structures, such as email header fields [[RFC5322](#)]. It does not change the original Header fields in any way. It MUST apply the following steps in order:

- * Convert all header field names (not the header field values) to lowercase. For example, convert "SUBJECT: AbC" to "subject:

AbC".

- * Unfold all header field continuation lines as described in [\[RFC5322\]](#); in particular, lines with terminators embedded in continued header field values (that is, CRLF sequences followed by WSP) MUST be interpreted without the CRLF. Implementations MUST NOT remove the CRLF at the end of the header field value.
- * Convert all sequences of one or more WSP characters to a single SP character. WSP characters here include those before and after a line folding boundary.
- * Delete all WSP characters at the end of each unfolded header field value.
- * Delete any WSP characters remaining before and after the colon separating the header field name from the header field value. The colon separator MUST be retained.

[3.1.2.](#) Content Canonicalization Algorithms

This section specifies initial entry for the IANA registry defined in Table 3.

simple: The "simple" Content canonicalization algorithm is for lines of ASCII text, such as occur in the body of email [\[RFC5322\]](#). It ignores all empty lines at the end of the Content. An empty line is a line of zero length after removal of the line terminator. If there is no Content or no trailing CRLF on the Content, a CRLF is added. It makes no other changes to the Content. In more formal terms, the "simple" Content canonicalization algorithm converts "0*CRLF" at the end of the Content to a single "CRLF".

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

The sha1 value (in base64) for an empty Content (canonicalized to a "CRLF") is:

uoq1oCgLLTqpdDX/iUbLy7J1Wic=

The sha256 value is:

frcCV1k9oG9oKj3dpUqdJg1PxRT2RSN/XKdLCPjaYaY=

relaxed: The "relaxed" Content canonicalization algorithm is for lines of ASCII text, such as occur in the body of email [[RFC5322](#)]. It MUST apply the following steps (a) and (b) in order:

A. Reduce whitespace:

- + Ignore all whitespace at the end of lines. Implementations MUST NOT remove the CRLF at the end of the line.
- + Reduce all sequences of WSP within a line to a single SP character.

- B. Ignore all empty lines at the end of the Content. "Empty line" is defined in [Section 3.4.3](#). If the Content is non-empty, but does not end with a CRLF, a CRLF is added. (For email, this is only possible when using extensions to SMTP or non-SMTP transport mechanisms.)

The sha1 value (in base64) for an empty Content (canonicalized to a null input) is:

2jmj7l5rSw0yVb/vlWAYkK/YBwk=

The sha256 value is:

47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU=

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

[3.1.3](#). Canonicalization Examples (3.4.6)

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

bracketed descriptors: "<SP>" for a space character, "<HTAB>" for a tab character, and "<CRLF>" for a carriage-return/line-feed sequence. For example, "X <SP> Y" and "X<SP>Y" represent the same three characters.

Example 1: An email message reading:

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

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

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

and a Content reading:

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

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

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

and a Content reading:

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

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

```
a:X <CRLF>
```


b:Y <SP> Z <CRLF>

and a Content reading:

<SP> C <SP><CRLF>

D <SP><HTAB><SP> E <CRLF>

[3.2.](#) Tag=Value Parameters {rfc4871bis-02 3.2}

DOSETA uses a simple "tag=value" syntax in several contexts, including to represent associated cryptographic data and domain cryptographic key records.

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

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

Formally the syntax rules are as follows:

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

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

Tags MUST be interpreted in a case-sensitive manner. Values MUST be

processed as case sensitive unless the specific tag description of semantics specifies case insensitivity.

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

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

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

Unrecognized tags MUST be ignored.

Tags that have an empty value are not the same as omitted tags. An omitted tag is treated as having the default value; a tag with an empty value explicitly designates the empty string as the value.

[3.3.](#) Key Management {rfc4871bis-02 3.6}

Applications require some level of assurance that a cited public key is associated with the Creator using it. Many applications achieve this by using public key certificates issued by a trusted third party. For applications with modest certification requirements, DOSETA achieves a sufficient level of security, with significantly enhanced scalability, by simply having the Consumer query the purported Creator's DNS entry (or a supported equivalent) in order to retrieve the public key.

DOSETA keys might 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 DOSETA algorithm.

The key lookup algorithm is:
public-key = D-find-key(q-val, d-val, s-val)
where:

q-val: The type of the lookup, as specified in the "q=" tag
([Section 4.2](#))

d-val: The domain of the signature, as specified in the "d=" tag
([Section 4.2](#))

Internet-Draft

DOSETA

January 2011

s-val: The selector of the lookup as specified in the "s=" tag ([Section 4.2](#))

D-find-key: A function that uses q-val to determine the specific details for accessing the desired stored Key record.

This document defines a single binding, using DNS TXT records to distribute the keys, per [Section 3.5](#). Other bindings might be defined in the future.

[3.4](#). Selectors for Keys {rfc4871bis-02 3.1}

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 (for example, "sanfrancisco", "coolumbeach", and "reykjavik"), the signing date (for example, "january2005", "february2005", etc.), or even an individual user.

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

- o Domains that want to delegate signing capability for a specific address for a given duration to a partner, such as an advertising provider or other outsourced function.
- o Domains that want to allow frequent travelers to generate signed data locally without the need to connect to a particular server.
- o "Affinity" domains (such as, college alumni associations) that provide data forwarding, but that do not operate a data origination agent for outgoing data.

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

ABNF:
selector = sub-domain *("." sub-domain)

The number of public keys and corresponding selectors for each domain is determined by the domain owner. Many domain owners will be

satisfied with just one selector, whereas administratively distributed organizations might choose to manage disparate selectors and key pairs in different regions or on different 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 data might be in transit prior to verification. At the start of the transition period, the outbound 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.

NOTE: A key can also be revoked as described below. The distinction between revoking and removing a key selector record is subtle. When phasing out keys as described above, a signing domain would probably simply remove the key record after the transition period. However, a signing domain could elect to revoke the key (but maintain the key record) for a further period. There is no defined semantic difference between a revoked key and a removed key.

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

value, such as a fingerprint of the public key.

NOTE: The ability to reuse 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 data that didn't verify because the key is no longer valid versus a data that is actually forged. For this reason, signers are ill-advised to reuse selectors for new keys. A better strategy is to assign new keys to new selectors.

[3.5.](#) DNS Binding for Key Retrieval {rfc4871bis-02 3.6.2}

This section defines a binding using DNS TXT records as a key service. All implementations MUST support this binding.

[3.5.1.](#) Namespace

A DOSETA key is stored in a subdomain named:

ABNF:
dns-record = s "_domainkey." d
where:

s: is the selector of the lookup as specified in the "s=" tag
([Section 4.2](#))

d: is the domain of the signature, as specified in the "d=" tag
([Section 4.2](#))

NOTE: The string constant "_domainkey" is used to mark a sub-tree that contains unified DOSETA key information. This string is a constant, rather than being a different string for different key-based services, in the view that keys are agnostic about the service they are used for. That is, there is no semantic or

security benefit in having a different constant string for different key services.

Given a DOSETA-Signature field with a "d=" tag of "example.com" and an "s=" tag of "foo.bar", the DNS query will be for: foo.bar._domainkey.example.com

Wildcard DNS records (for example, *.bar._domainkey.example.com) do not make sense in this context and SHOULD not be used. Note also that wildcards within domains (for example, s._domainkey.*.example.com) are not supported by the DNS.

[3.5.2.](#) Resource Record Types for Key Storage

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

Strings in a TXT RR MUST be concatenated together before use with no intervening whitespace. TXT RRs MUST be unique for a particular

selector name; that is, if there are multiple records in an RRset, the results are undefined.

TXT RRs are encoded as described in [Section 3.6](#).

[3.6.](#) Stored Key Data {rfc4871bis-02 3.6.1, subset, with preface}

This section defines a syntax for encoding stored key data within an unstructured environment such as simple text.

TEMPLATE: (Key Retrieval) A service that incorporates DOSETA MAY define the specific mechanism by which Consumers can obtain associated public keys. This might be as easy as referencing an existing key management system or it might require a new set of conventions.

Absent an explicit specification for key retrieval, the default

mechanism is specified in [Section 3.5](#).

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

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

ABNF:

key-v-tag = %x76 [FWS] "=" [FWS] %x44 %x4B %x49 %x4D %x31

NOTE: The version string "DKIM1" is retained from the original DKIM specification, in order to preserve the installed base of records. In addition, there is no functional benefit in defining a new string, since the key record is not application-specific.

k= Key type (plain-text; OPTIONAL, default is "rsa"). Signers and verifiers MUST support the "rsa" key type. The "rsa" key type indicates that an ASN.1 DER-encoded [\[ITU-X660-1997\]](#) RSAPublicKey [\[RFC3447\]](#) (see Sections [Section 3.4](#) and A.1.1) is being used in the "p=" tag. (Note: the "p=" tag further encodes the value using the base64 algorithm.) Unrecognized key types MUST be ignored.

ABNF:

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

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

ABNF:

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

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

ABNF:

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

NOTE: If a private key has been compromised or otherwise disabled (for example, an outsourcing contract has been terminated), a signer might want to explicitly state that it knows about the selector, but also have all queries using that selector result in a failed verification. Verifiers SHOULD ignore any DOSETA-Signature header fields with a selector referencing a revoked key.

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

[4.](#) DOSETA H/C Signing Template

This section specifies the basic components of a signing mechanism, which is similar to the one defined for DKIM. This template for a signing service can be mapped to a two-part -- header/content -- data model. As with DKIM it separates specification of the signer's identity from any other identifiers that might be associated with that data.

NOTE: The use of hashing and signing algorithms by DOSETA inherently provides a degree of data integrity protection, between the signing and verifying steps. However it does not necessarily "authenticate" the data that is signed. For example, it does not inherently validate the accuracy of the data or declare that the signer is the author or owner of the data.

TEMPLATE: (Header/Content Mapping) The service incorporating this mechanism MUST define the precise mappings onto the template provided in this section. (Note that data lacking a header component might still be possible to cast in a header/content form, where the header comprises on the DOSETA Signature information.)

The service also MUST define the precise meaning of a signature.

[4.1.](#) Cryptographic Algorithms {rfc4871bis-02 3.3, subset}

DOSETA supports multiple digital signature algorithms:

rsa-sha1: The rsa-sha1 Signing Algorithm computes a message hash as described in [Section 4.3](#) below using SHA-1 [[FIPS-180-2-2002](#)] as a hashing algorithm. That hash is then signed by the signer using the RSA algorithm (defined in PKCS#1 version 1.5 [[RFC3447](#)]) as the crypt-alg and the signer's private key. The hash MUST NOT be truncated or converted into any form other

than the native binary form before being signed. The signing algorithm SHOULD use a public exponent of 65537.

rsa-sha256: The rsa-sha256 Signing Algorithm computes a message hash as described in [[RFC5451](#)] below using SHA-256 [[FIPS-180-2-2002](#)] as the hash-alg. That hash is then signed by the signer using the RSA algorithm (defined in PKCS#1 version 1.5 [[RFC3447](#)]) as the crypt-alg and the signer's private key. The hash MUST NOT be truncated or converted into any form other than the native binary form before being signed.

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

Signers MUST implement and SHOULD sign using rsa-sha256. Verifiers MUST implement rsa-sha256.

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

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

Factors that ought to influence the key size choice include the following:

- o The practical constraint that large (for example, 4096 bit) keys might 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 data
- 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 other systems that employ digital signatures

Internet-Draft

DOSETA

January 2011

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

[4.2.](#) Signature Data Structure {rfc4871bis-02 3.5, subset}

A signature of data is stored into an data structure associated with the signed data. This structure contains all of the signature and key-fetching data. This DOSETA-Signature structure is a tag-list as defined in [Section 3.2](#).

TEMPLATE: (Signature Association) A service that incorporates DOSETA MUST define the exact means by which the Signature structure is associated with the data.

When the DOSETA-Signature is part of a sequence of structures -- such as being added to an email header -- it SHOULD NOT be reordered and SHOULD be prepended to the message. (This is the same handling as is given to email trace Header fields, defined in [Section 3.6 of \[RFC5322\]](#).)

The tags are specified below. Tags described as <qp-section> are encoded as described in [Section 6.7](#) of MIME Part One [[RFC2045](#)], with the additional conversion of semicolon characters to "=3B"; intuitively, this is one line of quoted-printable encoded text. The D-quoted-printable syntax is defined in [Section 2.3.4](#).

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

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

ABNF:

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

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

a= The algorithm used to generate the signature (plain-text; REQUIRED). Verifiers MUST support "rsa-sha1" and "rsa-sha256"; signers SHOULD sign using "rsa-sha256". See [Section 4.1](#) for a description of algorithms.

ABNF:

```
sig-a-tag       = %x61 [FWS] "=" [FWS] sig-a-tag-alg
sig-a-tag-alg   = sig-a-tag-k "-" sig-a-tag-h
sig-a-tag-k     = "rsa" / x-sig-a-tag-k
sig-a-tag-h     = "sha1" / "sha256" / x-sig-a-tag-h
x-sig-a-tag-k   = ALPHA *(ALPHA / DIGIT)
                 ; for later extension
x-sig-a-tag-h   = ALPHA *(ALPHA / DIGIT)
                 ; for later extension
```

b= The signature data (base64; REQUIRED). Whitespace is ignored in this value and MUST be ignored when reassembling the original signature. In particular, the signing process can safely insert FWS in this value in arbitrary places to conform to line-length limits. See Signer Actions ([Section 4.4](#)) for how the signature is computed.

ABNF:

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

bh= The hash of the canonicalized Content (body), as limited by the "l=" tag (base64; REQUIRED). Whitespace is ignored in this value and MUST be ignored when reassembling the original signature. In particular, the signing process can safely insert FWS in this value in arbitrary places to conform to line-length limits. See [Section 4.3](#) for how the Content hash is computed.

ABNF:

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

c= Message canonicalization (plain-text; OPTIONAL, default is "simple/simple"). This tag informs the verifier of the type of canonicalization used to prepare the message for signing. It consists of two names separated by a "slash" (%d47) character, corresponding to the header and Content canonicalization algorithms respectively:

ABNF:

```
sig-bh-tag      = %x63 [FWS] "=" [FWS] sig-c-header "/" sig-c-content
where:
```

sig-c-header: A value from IANA registry defined in
Table 2

sig-c-content: A value from IANA registry defined in
Table 3

These algorithms are described in [Section 3.1](#). If only one algorithm is named, that algorithm is used for the header and "simple" is used for the Content. 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 SDID doing the signing (plain-text; REQUIRED). Hence, the SDID value is used to form the query for the public key. The SDID MUST correspond to a valid DNS name under which the DOSETA key record is published. The conventions and semantics used by a signer to create and use a specific SDID are outside the scope of the DOSETA Signing specification, as is any use of those conventions and semantics. When presented with a signature that does not meet these requirements, verifiers MUST

consider the signature invalid.

Internationalized domain names MUST be encoded as described in [[RFC5890](#)].

TEMPLATE: (Semantics) The service incorporating DOSETA MUST define the semantics of a signature.

ABNF:

```
sig-d-tag      = %x64 [FWS] "=" [FWS] domain-name
domain-name    = sub-domain 1*("." sub-domain)
                  ; from RFC 5321 Domain,
```

; but excluding address-literal

h= Signed Header fields (plain-text, but see description; REQUIRED). A colon-separated list of header field names that identify the Header fields presented to the signing algorithm. The field MUST contain the complete list of Header fields in the order presented to the signing algorithm. The field MAY contain names of Header fields that do not exist when signed; nonexistent Header fields do not contribute to the signature computation (that is, they are treated as the null input, including the header field name, the separating colon, the header field value, and any CRLF terminator). The field MUST NOT include the DOSETA Signature header field that is being created or verified, but might include others. Folding whitespace (FWS) MAY be included on either side of the colon separator. Header field names MUST be compared against actual header field names in a case-insensitive manner. This list MUST NOT be empty. See [Section 4.4.4](#) for a discussion of choosing Header fields to sign.

ABNF:

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

By "signing" Header fields that do not actually exist, a signer can prevent insertion of those Header fields before verification. However, since a signer cannot possibly know all possible Header fields that might be created in the future, the security of this solution is not total.

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.

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

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

ABNF:

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

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

ABNF:

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


t= Signature Timestamp (plain-text unsigned decimal integer; RECOMMENDED, default is an unknown creation time). The time that this signature was created. The format is the number of seconds since 00:00:00 on January 1, 1970 in the UTC time zone. The value is expressed as an unsigned integer in decimal ASCII. This value is not constrained to fit into a 31- or 32-bit integer. Implementations SHOULD be prepared to handle values up to at least 10^{12} (until approximately AD 200,000; this fits into 40 bits). To avoid denial-of-service attacks, implementations MAY consider any value longer than 12 digits to be infinite. Leap seconds are not counted. Implementations MAY ignore signatures that have a timestamp in the future.

ABNF:

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

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

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

NOTE: Due to clock drift, the receiver's notion of when to consider the signature expired might not match exactly when the sender is expecting. Receivers MAY add a 'fudge factor' to allow for such possible drift.

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

[4.3.](#) Signature Calculation {rfc4871bis-02 3.7, reorganized and clarified}

Hashing and Cryptographic algorithms are combined into a procedure for computing a digital signature.

Creators will choose appropriate parameters for the signing process; Creators will use the tags that are then passed as an associated DOSETA Header field. [Section 4.2](#) defines the contents of the Header field.

In the following discussion, the names of the tags in the Header field that either exist (when consuming) or will be created (when creating) are used.

NOTE: Canonicalization (see [Section 3.1](#)) prepares a separate representation of the data for additional processing; it does not affect the transmitted data in any way.

Creators MUST compute hashes in the order defined. Consumers MAY compute them in any order convenient to the Creator, provided that the result is semantically identical to the semantics that would be the case had they been computed in this order.

The combined hashing algorithm is:

Step 1: The Creator/Creator hashes the Content portion, canonicalized using the Content canonicalization algorithm specified in the "c=" tag and then truncated to the length specified in the "l=" tag. That hash value is then converted to base64 form. Signers then insert it into the "bh=" tag of the DOSETA-Signature field; verifiers compare their hash with the value in the "bh=" tag.

Step 2: Pass the following to a second hash algorithm in the indicated order:

Internet-Draft

DOSETA

January 2011

- + Complete attribute:value 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 field MUST be terminated with a single CRLF.
- + The DOSETA-Signature field that exists (verifying), or will be inserted (signing), in the Header, with the value of the "b=" signature data tag (including all surrounding whitespace) deleted (that is, treated as the empty string), canonicalized using the Header canonicalization algorithm specified in the "c=" tag, and without a trailing CRLF.
- + The hash produced in Step 1.

When calculating or verifying a signature, the value of the "b=" tag (signature value) of that DOSETA-Signature structure MUST be treated as though it were an empty string. Unknown tags in the DOSETA-Signature header field MUST be included in the signature calculation but MUST be otherwise ignored by verifiers. Other DOSETA-Signature tags that are included in the signature SHOULD be treated as normal tags; in particular, the "b=" tag is not treated specially.

All tags and their values in the DOSETA-Signature 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 DOSETA-Signature field MUST be presented to the hash algorithm after the Content, rather than with the rest of the Header fields and MUST be canonicalized as specified in the "c=" (canonicalization) tag. The DOSETA-Signature header structure MUST NOT be included in its own h= tag, although other DOSETA-Signature Header fields MAY be signed (see [Section 5](#)).

When calculating the hash on data that will be transmitted using additional encoding, such as base64 or quoted-printable, signers MUST compute the hash after the encoding. Likewise, the verifier MUST incorporate the values into the hash before decoding the base64 or quoted-printable text. However, the hash MUST be computed before transport level encodings such as SMTP "dot-stuffing" (the modification of lines beginning with a "." to avoid confusion with the SMTP end-of-message marker, as specified in [\[RFC5321\]](#)).

With the exception of the canonicalization procedure described in [Section 3.1](#), the DOSETA signing process treats the Content as simply a string of octets. DOSETA Content MAY be either in plain-text or in MIME format; no special treatment is afforded to MIME content.

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

ABNF:

```
content-hash = bh-hash-alg (canon-content, l-param)
data-hash    = a-hash-alg (h-headers, D-SIG, content-hash)
signature    = sig-alg (d-domain, selector, data-hash)
```

where:

content-hash: is the output of a function to hash the data Content.

bh-hash-alg: is the hashing algorithm specified in the "bh" parameter.

canon-body: is a canonicalized representation of the body.

l-param: is the value of the l= length parameter.

data-hash: is the output from hashing the header, the DOSETA-Signature header, and the Content hash.

a-hash-alg: is the hash algorithm specified by the "a=" tag,

h-headers: is the list of headers to be signed, as specified in the h= parameter.

D-SIG: is the canonicalized DOSETA-Signature field sans the signature value itself.

canon-content: is the canonicalized data Content(respectively) as defined in [Section 3.1](#) (excluding the DOSETA-Signature

field).

signature: is the signature value produced by the signing algorithm.

sig-alg: is the signature algorithm specified by the "a=" tag,

d-domain: is the domain name specified in the d= parameter.

selector: is the value of the s= selector parameter

NOTE: Many digital signature APIs provide both hashing and application of the RSA private key using last two steps in the algorithm would probably be combined into a single call that would perform both the "a-hash-alg" and the "sig-alg".

[4.4.](#) Signer Actions {rfc4871bis-02 5}

The following steps are performed in order by signers.

[4.4.1.](#) Determine Whether the Data Should Be Signed and by Whom

A signer can obviously only sign data using 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 the data.

NOTE: Signing modules can be incorporated into any portion of a service, as deemed appropriate, including end-systems, servers and intermediaries. Wherever implemented, signers need to beware of the semantics of signing data. An example of how this can be problematic is that within a trusted enclave the signing address might be derived from the data according to local policy; the derivation is based on local trust rather than explicit validation.

If the data cannot be signed for some reason, the disposition of that

data is a local policy decision.

[4.4.2.](#) Select a Private Key and Corresponding Selector Information

This specification does not define the basis by which a signer ought to choose which private key and selector information to use. Currently, all selectors are equal, with respect to this specification. So the choices ought to largely be a matter of administrative convenience. Distribution and management of private keys is also outside the scope of this document.

NOTE: A signer SHOULD use a private key with an associated selector record that is expected to still be valid by time the verifier is likely to have an opportunity to validate the signature. The signer SHOULD anticipate that verifiers can choose to defer validation, perhaps until the message is actually read by the final recipient. In particular, when rotating to a new key pair, signing SHOULD immediately commence with the new private key, but the old public key SHOULD be retained for a reasonable validation

interval before being removed from the key server.

[4.4.3.](#) Normalize the Message to Prevent Transport Conversions

Some messages, particularly those using 8-bit characters, are subject to modification during transit, notably from conversion to 7-bit form. Such conversions will break DOSETA signatures. In order to minimize the chances of such breakage, signers SHOULD convert the data to a suitable encoding, such as quoted-printable or base64, as described in [\[RFC2045\]](#) before signing. Such conversion is outside the scope of DOSETA; the data SHOULD be converted to 7-bit prior to presentation to the DOSETA signing procedure.

Similarly, data that is not compliant with its associated standard, might be subject to corrective efforts intermediaries. See [Section 8 of \[RFC4409\]](#) for examples of changes that are commonly made to email. Such "corrections" might break DOSETA signatures or have other undesirable effects.

If the data is submitted to the signer with any local encoding that will be modified before transmission, that modification to a canonical form MUST be done before signing. For Text data in

particular, bare CR or LF characters (used by some systems as a local line separator convention) MUST be converted to the CRLF sequences before the data is signed. Any conversion of this sort SHOULD be applied to the data actually sent to the recipient(s), not just to the version presented to the signing algorithm.

More generally, the signer MUST sign the data as it is expected to be received by the verifier rather than in some local or internal form.

4.4.4. Determine the Header Fields to Sign

Signers SHOULD NOT sign an existing header field that is likely to be legitimately modified or removed in transit. Signers MAY include any other Header fields present at the time of signing at the discretion of the signer.

NOTE: The choice of which Header fields to sign is non-obvious. One strategy is to sign all existing, non-repeatable Header fields. An alternative strategy is to sign only Header fields that are likely to be displayed to or otherwise be likely to affect the processing of the Content at the receiver. A third strategy is to sign only "well known" headers. Note that verifiers might treat unsigned Header fields with extreme skepticism, including refusing to display them to the end user or even ignoring the signature if it does not cover certain Header fields.

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

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

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

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

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

EXAMPLE:

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

Received: <A>

Received:

Received: <C>

then the resulting DOSETA-Signature header field ought to read:

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

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

Signers need to be careful of signing Header fields that might have additional instances added later in the delivery process, since such Header fields might be inserted after the signed instance or otherwise reordered. Trace Header fields (such as Received) and Resent-* blocks are the only fields prohibited by [\[RFC5322\]](#) from being reordered. In particular, since DOSETA-Signature Header fields might be reordered by some intermediate MTAs, signing existing DOSETA-Signature Header fields is error-prone.

NOTE: Despite the fact that [\[RFC5322\]](#) permits Header fields to be reordered (with the exception of Received Header fields), reordering of signed Header fields with multiple instances by intermediate MTAs will cause DOSETA signatures to be broken; such anti-social behavior ought to be avoided.

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.

[4.4.5.](#) Compute the Message Signature

The signer MUST compute the message hash as described in [Section 4.3](#) and then sign it using the selected public-key algorithm. This will result in a DOSETA-Signature header field that will include the Content hash and a signature of the header hash, where that header includes the DOSETA-Signature header field itself.

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

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

[4.4.6.](#) Insert the DOSETA-Signature Header Field

Finally, the signer MUST insert the DOSETA-Signature header field created in the previous step prior to transmitting the data. The DOSETA-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

to the selector given in the "s=" tag of the DOSETA-Signature header field, as chosen above in [Section 4.4.2](#)

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

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

[4.5.](#) Verifier Actions {rfc4871bis-02 6}

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

A border or intermediate server MAY verify the data signature(s). An server that has performed verification MAY communicate the result of that verification by adding a verification header field to incoming data.

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

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

[4.5.1.](#) Extract Signatures from the Message

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

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

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

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

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

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

Verifiers SHOULD ignore any DOSETA-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, if it exists. However, verifiers MAY make note of the fact that an invalid signature was present for consideration at a

later step.

Internet-Draft

DOSETA

January 2011

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 ought to succeed even in the presence of the known-broken signature.

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

[4.5.2.](#) Validate the Signature Header Field

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

NOTE: An implementation might, of course, choose to also verify signatures generated by older versions of this specification.

If any tag listed as "required" in [Section 4.2](#) is omitted from the DOSETA-Signature header field, the verifier MUST ignore the DOSETA-Signature header field and return PERMFAIL (signature missing required tag).

NOTE: The tags listed as required in [Section 4.2](#) are "v=", "a=", "b=", "bh=", "d=", "h=", and "s=". Should there be a conflict between this note and [Section 4.2](#), is normative.

If the DOSETA-Signature header field does not contain the "i=" tag,

the verifier MUST behave as though the value of that tag were "@d", where "d" is the value from the "d=" tag.

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

If the "h=" tag does not include the From header field, the verifier

MUST ignore the DOSETA-Signature header field and return PERMFAIL (From field not signed).

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

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

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

[4.5.3.](#) Get the Public Key

The public key for a signature is needed to complete the verification process. The process of retrieving the public key depends on the query type as defined by the "q=" tag in the DOSETA-Signature header field. Obviously, a public key need only be retrieved if the process of extracting the signature information is completely successful. Details of key management and representation are described in [Section 3.3](#). The verifier MUST validate the key record and MUST ignore any public key records that are malformed.

NOTE: The use of wildcard TXT records in the DNS will produce a

response to a DOSETA query that is unlikely to be valid DOSETA key record. This problem applies to many other types of queries, and client software that processes DNS responses needs to take this problem into account.

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 might parallelize or reorder these steps, as long as the semantics remain unchanged:

1. Retrieve the public key as described in [Section 3.3](#) using the algorithm in the "q=" tag, the domain from the "d=" tag, and the selector from the "s=" tag.

2. If the query for the public key fails to respond, the verifier MAY defer acceptance of this data and return TEMPFAIL - key unavailable. (If verification is occurring during the incoming SMTP session, this MAY be achieved with a 451/4.7.5 SMTP reply code.) Alternatively, the verifier MAY store the message in the local queue for later trial or ignore the signature. Note that storing a message in the local queue is subject to denial-of-service attacks.
3. If the query for the public key fails because the corresponding key record does not exist, the verifier MUST immediately return PERMFAIL (no key for signature).
4. If the query for the public key returns multiple key records, the verifier might choose one of the key records or might cycle through the key records performing the remainder of these steps on each record at the discretion of the implementer. The order of the key records is unspecified. If the verifier chooses to cycle through the key records, then the "return ..." wording in the remainder of this section means "try the next key record, if any; if none, return to try another signature in the usual way".
5. If the result returned from the query does not adhere to the format defined in this specification, the verifier MUST ignore the key record and return PERMFAIL (key syntax error). Verifiers

are urged to validate the syntax of key records carefully to avoid attempted attacks. In particular, the verifier MUST ignore keys with a version code ("v=" tag) that they do not implement.

6. If the "h=" tag exists in the public key record and the hash algorithm implied by the a= tag in the DOSETA-Signature header field is not included in the contents of the "h=" tag, the verifier MUST ignore the key record and return PERMFAIL (inappropriate hash algorithm).
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 and return PERMFAIL (key revoked). There is no defined semantic difference between a key that has been revoked and a key record that has been removed.
8. If the public key data is not suitable for use with the algorithm and key types defined by the "a=" and "k=" tags in the DOSETA-Signature header field, the verifier MUST immediately return PERMFAIL (inappropriate key algorithm).

[4.5.4.](#) Compute the Verification

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

1. Based on the algorithm defined in the "c=" tag, the Content length specified in the "l=" tag, and the header field names in the "h=" tag, prepare a canonicalized version of the Content as is described in [Section 4.3](#) (note that this version does not actually need to be instantiated). When matching header field names in the "h=" tag against the actual message header field, comparisons MUST be case-insensitive.
2. Based on the algorithm indicated in the "a=" tag, compute the message hashes from the canonical copy as described in [Section 4.3](#)
3. Verify that the hash of the canonicalized Content computed in the

previous step matches the hash value conveyed in the "bh=" tag. If the hash does not match, the verifier SHOULD ignore the signature and return PERMFAIL (Content hash did not verify).

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

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 might also verify the signature on the message header before validating that the message hash listed in the "bh=" tag in the DOSETA-Signature header field matches that of the actual Content; however, if the Content hash does not match, the entire signature MUST be considered to have failed.

[4.5.5.](#) Communicate Verification Results

Verifiers wishing to communicate the results of verification to other parts of the data handling system can do so in whatever manner they see fit. For example, implementations might choose to add a Header field to the data before passing it on. Any such header field SHOULD be inserted before any existing DOSETA-Signature or preexisting verification status Header fields in the header field block. The Authentication-Results: header field ([\[RFC5451\]](#)) MAY be used for this

purpose.

Patterns intended to search for results Header fields to visibly mark authenticated data for end users SHOULD verify that the header field was added by the appropriate verifying domain. In particular, filters SHOULD NOT be influenced by bogus results header fields added by attackers. To circumvent this attack, verifiers SHOULD delete existing results Header fields after verification and before adding a new header field.

[4.5.6.](#) Interpret Results/Apply Local Policy

It is beyond the scope of this specification to describe what actions an Assessment phase will take, but data with a verified DOSETA signature presents an opportunity to an Assessor that unsigned data does not. Specifically, signed data creates a predictable identifier by which other decisions can reliably be managed, such as trust and reputation. Conversely, unsigned data typically lacks a reliable identifier that can be used to assign trust and reputation. It is usually reasonable to treat unsigned data as lacking any trust and having no positive reputation.

In general, verifiers SHOULD NOT reject data solely on the basis of a lack of signature or an unverifiable signature; such rejection would cause severe interoperability problems. However, if the verifier does opt to reject such data.

Temporary failures such as inability to access the key server or other external service are the only conditions that SHOULD use a temporary failure code. In particular, cryptographic signature verification failures MUST NOT return temporary failure replies.

Once the signature has been verified, that information MUST be conveyed to the Assessor (such as an explicit allow/whitelist and reputation system) and/or to the end user. If the SDID is not the same as the address in the From: header field, the data system SHOULD take pains to ensure that the actual SDID is clear to the reader.

The verifier MAY treat unsigned Header fields with extreme skepticism, including marking them as untrusted or even deleting them.

While the symptoms of a failed verification are obvious -- the signature doesn't verify -- establishing the exact cause can be more difficult. If a selector cannot be found, is that because the selector has been removed, or was the value changed somehow in transit? If the signature line is missing, is that because it was never there, or was it removed by an overzealous filter? For

diagnostic purposes, the exact nature of a verification failure SHOULD be made available to the policy module and possibly recorded in the system logs. If the data cannot be verified, then it SHOULD be rendered the same as all unverified data regardless of whether or

not it looks like it was signed.

[4.6.](#) Requirements for Tailoring the Signing Service {added}

This generic template requires additional details, to define a specific service:

D-Signature Association: Specify the means by which a DOSETA-Signature header field is associated with the data it signs. For example, DKIM uses the DKIM-Signature email header field. If the header field is encoded differently than defined for the DOSETA generic service, such as in XML, then that also needs to be specified including the algorithm for mapping between the common encoding provided here and the new encoding.

Semantics Signaling: The means by which a Consumer is to know what semantics apply must be indicated. This is likely to be the same means by which the Consumer knows what specific service is being used. For example, different service-specific DOSETA-Signature header fields might be used. "DKIM-Signature" signals a name-affixing service, while "DKVM-Signature" might signal a message validation service.

Semantics: Define the meaning of a signature. Note that exactly the same algorithms might be used for very different semantics. One might merely affix an identifier to some data, in a verifiable fashion, while the same set of mechanisms might separately be defined as authenticating the validity of that data.

Header/Content Mapping: The mappings between the generic service and data of a particular service needs to be defined. For example, with DKIM Header maps to the email header and Content maps to the email body.

[4.7.](#) Signing by Parent Domains {rfc4871bis-02 3.8}

In some circumstances, it is desirable for a domain to apply a signature on behalf of any of its subdomains without the need to maintain separate selectors (key records) in each subdomain. By default, private keys corresponding to key records can be used to sign messages for any subdomain of the domain in which they reside;

for example, a key record for the domain example.com can be used to verify messages where the AUID ("i=" tag of the signature) is sub.example.com, or even sub1.sub2.example.com. In order to limit the capability of such keys when this is not intended, the "s" flag MAY be set in the "t=" tag of the key record, to constrain the validity of the domain of the AUID. If the referenced key record contains the "s" flag as part of the "t=" tag, the domain of the AUID ("i=" flag) MUST be the same as that of the SDID (d=) domain. If this flag is absent, the domain of the AUID MUST be the same as, or a subdomain of, the SDID.

[5.](#) Semantics of Multiple Signatures {rfc4871bis-02 4}

[5.1.](#) Example Scenarios {rfc4871bis-02 4.1}

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

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

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

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

Of course, a message might also have multiple signatures because it passed through multiple signers. A common case is expected to be

that of a signed message that passes through a mailing list that also

Internet-Draft

DOSETA

January 2011

signs all messages. Assuming both of those signatures verify, a recipient might choose to accept the message if either of those signatures were known to come from trusted sources.

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

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

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

[5.2.](#) Interpretation {rfc4871bis-02 4.2}

A signer that is adding a signature to a message merely creates a new DOSETA-Signature header, using the usual semantics of the h= option. A signer MAY sign previously existing DOSETA-Signature Header fields using the method described in [Section 4.4.4](#) to sign trace Header fields.

NOTE: Signers need to be cognizant that signing DOSETA-Signature Header fields might result in verification failures due to modifications by intermediaries, such as their reordering DOSETA-Signature header fields. For this reason, signing existing DOSETA-Signature Header fields is unadvised, albeit legal.

NOTE: If a header field with multiple instances is signed, those Header fields are always signed from the "bottom" up (from last to first). Thus, it is not possible to sign only specific

DOSETA-Signature Header fields. For example, if the message being signed already contains three DOSETA-Signature header fields A, B, and C, it is possible to sign all of them, B and C only, or C only, but not A only, B only, A and B only, or A and C only.

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

algorithms.

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

When evaluating a message with multiple signatures, a verifier SHOULD evaluate signatures independently and on their own merits. For example, a verifier that by policy chooses not to accept signatures with deprecated cryptographic algorithms would consider such signatures invalid. Verifiers MAY process signatures in any order of their choice; for example, some verifiers might choose to process signatures corresponding to the From field in the message header before other signatures. See [Section 4.5.1](#) for more information about signature choices.

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

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

[6.](#) Considerations

[6.1.](#) IANA Considerations {rfc4871bis-02 7, subset}

This section carries forward the IANA registrations made by DKIM [RFC4871]. The original DKIM language has been retained, including the use of "DKIM" in the name of these registries, in order to provide continuity from the original specification.

In all cases, new values are assigned only for values that have been documented in a published RFC that has IETF Consensus [RFC5226].

6.1.1. DKIM-Signature Tag Specifications

NOTE: The content of a DOSETA-Signature structure match those in a DKIM-Signature header field.

A DKIM-Signature provides for a list of tag specifications. IANA has established the DKIM-Signature Tag Specification Registry for tag

specifications that can be used in DKIM-Signature fields.

The initial entries in the registry comprise:

TYPE	REFERENCE
v	(this document, Section 4.2)
a	(this document, Section 4.2)
b	(this document, Section 4.2)
bh	(this document, Section 4.2)
c	(this document, Section 4.2)
d	(this document, Section 4.2)
h	(this document, Section 4.2)
q	(this document, Section 4.2)
s	(this document, Section 4.2)
t	(this document, Section 4.2)
x	(this document, Section 4.2)

Table 1: DKIM-Signature Tag Specification Registry Initial Values

6.1.2. DKIM-Signature Query Method Registry

The "q=" tag-spec (specified in [Section 4.2](#)) provides for a list of query methods.

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

The initial entry in the registry comprises:

TYPE	OPTION	REFERENCE
dns	txt	(this document, Section 4.2 , "q=")

DKIM-Signature Query Method Registry Initial Values

6.1.3. DKIM-Signature Canonicalization Registry

The "c=" tag-spec (specified in [Section 4.2](#)) provides for a specifier for canonicalization algorithms for the header and Content.

IANA has established the DKIM-Signature Canonicalization Algorithm Registry for algorithms for converting a message into a canonical

form before signing or verifying using DKIM.

The initial entries in the Header registry comprise:

TYPE	REFERENCE
simple	(this document, Section 3.1.1)
relaxed	(this document, Section 3.1.1)

Table 2: DKIM-Signature Header Canonicalization Algorithm Registry Initial Values

The initial entries in the Content (Body) registry comprise:

TYPE	REFERENCE
------	-----------

	simple		(this document, Section 3.1.2)	
	relaxed		(this document, Section 3.1.2)	
+-----+-----+-----+-----+-----+				

Table 3: DKIM-Signature Body Canonicalization Algorithm Registry Initial Values

[6.1.4.](#) _domainkey DNS TXT Record Tag Specifications

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

The initial entries in the registry comprise:

+-----+-----+-----+-----+-----+				
	TYPE		REFERENCE	
+-----+-----+-----+-----+-----+				
	v		(this document, Section 3.6)	
	k		(this document, Section 3.6)	
	n		(this document, Section 3.6)	
	p		(this document, Section 3.6)	
+-----+-----+-----+-----+-----+				

DOSETA _domainkey DNS TXT Record Tag Specification Registry Initial Values

[6.1.5.](#) DKIM Key Type Registry

The "k=" <key-k-tag> (specified in [Section 3.6](#)) and the "a=" <sig-a-tag-k> (specified in [Section 4.2](#)) tags provide for a list of mechanisms that can be used to decode a DKIM signature.

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

The initial entry in the registry comprises:

+-----+-----+-----+		
	TYPE	
	REFERENCE	

rsa	[RFC3447]
-----	-----------------------------

DKIM Key Type Initial Values

[6.1.6.](#) DKIM Hash Algorithms Registry

The "h=" <key-h-tag> (specified in [Section 3.6](#)) and the "a=" <sig-a-tag-h> (specified in [Section 4.2](#)) tags provide for a list of mechanisms that can be used to produce a digest of message data.

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

The initial entries in the registry comprise:

TYPE	REFERENCE
sha1	[FIPS-180-2-2002]
sha256	[FIPS-180-2-2002]

DKIM Hash Algorithms Initial Values

[6.2.](#) Security Considerations {rfc4871bis-02 8, subset}

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

[6.2.1.](#) Misappropriated Private Key

If the private key for a user is resident on their computer and is not protected by an appropriately secure mechanism, it is possible for malware to send mail as that user and any other user sharing the

same private key. The malware would not, however, be able to generate signed spoofs of other signers' addresses, which would aid in identification of the infected user and would limit the possibilities for certain types of attacks involving socially engineered messages. This threat applies mainly to MUA-based implementations; protection of private keys on servers can be easily achieved through the use of specialized cryptographic hardware.

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

One countermeasure is to use a user-entered passphrase to encrypt the private key, although users tend to choose weak passphrases and often reuse them for different purposes, possibly allowing an attack against DOSETA 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 (for example, SMTP Authentication), possibly validate the message itself (for example, 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.

[6.2.2.](#) 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.

6.2.3. Attacks Against the DNS

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

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

DOSETA 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 [[RFC4880](#)] and S/MIME [[RFC5751](#)] address those requirements.

A second security issue related to the DNS revolves around the increased DNS traffic as a consequence of fetching selector-based data as well as fetching signing domain policy. Widespread deployment of DOSETA will result in a significant increase in DNS queries to the claimed signing domain. In the case of forgeries on a large scale, DNS servers could see a substantial increase in queries.

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

6.2.4. 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

Internet-Draft

DOSETA

January 2011

signatures.

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

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

[6.2.5.](#) Limits on Revoking Keys

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

[6.2.6.](#) Intentionally Malformed Key Records

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

[6.2.7.](#) Intentionally Malformed DOSETA-Signature Header Fields

Verifiers **MUST** be prepared to receive messages with malformed

DOSETA-Signature Header fields, and thoroughly verify the header field before depending on any of its contents.

[6.2.8.](#) 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.

[6.2.9.](#) Remote Timing Attacks

In some cases it might be possible to extract private keys using a remote timing attack [[BONEH03](#)]. Implementations SHOULD obfuscate the timing to prevent such attacks.

[6.2.10.](#) Reordered Header Fields

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

[6.2.11.](#) RSA Attacks

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

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

[6.2.12.](#) Inappropriate Signing by Parent Domains

The trust relationship described in [Section 4.7](#) could conceivably be used by a parent domain to sign messages with identities in a subdomain not administratively related to the parent. For example, the ".com" registry could create messages with signatures using an "i=" value in the example.com domain. There is no general solution to this problem, since the administrative cut could occur anywhere in the domain name. For example, in the domain "example.podunk.ca.us" there are three administrative cuts (podunk.ca.us, ca.us, and us), any of which could create messages with an identity in the full domain.

NOTE: This is considered an acceptable risk for the same reason that it is acceptable for domain delegation. For example, in the example above any of the domains could potentially simply delegate "example.podunk.ca.us" to a server of their choice and completely replace all DNS-served information. Note that a verifier MAY ignore signatures that come from an unlikely domain such as ".com", as discussed in [Section 4.5.2](#).

[6.2.13.](#) Attacks Involving Addition of Header Fields

Many email implementations do not enforce [\[RFC5322\]](#) with strictness. As discussed in [Section 4.4.3](#) DOSETA processing is predicated on a valid mail message as its input. However, DOSETA implementers need to be aware of the potential effect of having loose enforcement by email components interacting with DOSETA modules.

For example, a message with multiple From: Header fields violates [Section 3.6 of \[RFC5322\]](#). With the intent of providing a better user experience, many agents tolerate these violations and deliver the message anyway. An MUA then might elect to render to the user the value of the last, or "top", From: field. This might also be done simply out of the expectation that there is only one, where a "find first" algorithm would have the same result. Such code in an MUA can be exploited to fool the user if it is also known that the other From: field is the one checked by arriving message filters. Such is the case with DOSETA; although the From: field MUST be signed, a malformed message bearing more than one From: field might only have the first ("bottom") one signed, in an attempt to show the message

with some "DOSETA passed" annotation while also rendering the From: field that was not authenticated. (This can also be taken as a demonstration that DOSETA is not designed to support author validation.)

To resist this specific attack the signed header field list can include an additional reference for each field that was present at signing. For example, a proper message with one From: field could be signed using "h=From:From:..." Due to the way header fields are canonicalized for input to the hash function, the extra field references will prevent instances of the cited fields from being added after signing, as doing so would render the signature invalid.

The From: field is used above to illustrate this issue, but it is only one of > several fields that [Section 3.6 of \[RFC5322\]](#) constrains in this way. In reality any agent that forgives malformations, or is careless about identifying which parts of a message were authenticated, is open to exploitation.

[7.](#) References

[7.1.](#) Normative References

[FIPS-180-2-2002]

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

[ITU-X660-1997]

"Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", 1997.

[RFC1034] Mockapetris, P., "DOMAIN NAMES - CONCEPTS AND FACILITIES", [RFC 1034](#), November 1987.

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

- [RFC2049] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples", [RFC 2049](#), November 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", [RFC 3447](#), February 2003.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", [RFC 4234](#), January 2008.
- [RFC5321] Klensin, J., "Simple Mail Transfer Protocol", [RFC 5321](#), October 2008.
- [RFC5322] Resnick, P., "Internet Message Format", [RFC 5322](#), October 2008.
- [RFC5890] Klensin, J., "Internationalizing Domain Names in Applications (IDNA): Definitions and Document Framework", [RFC 5890](#), August 2010.

[7.2.](#) Informative References

- [BONEH03] "Remote Timing Attacks are Practical", Proceedings 12th USENIX Security Symposium, 2003.
- [RFC1847] Galvin, J., Murphy, S., Crocker, S., and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", [RFC 1847](#), October 1995.
- [RFC3766] Orman, H. and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", [BCP 86](#), [RFC 3766](#), April 2004.
- [RFC3833] Atkins, D. and R. Austein, "Threat Analysis of the Domain

Name System (DNS)", [RFC 3833](#), August 2004.

- [RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", [BCP 90](#), [RFC 3864](#), September 2004.
- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", [RFC 4033](#), March 2005.
- [RFC4409] Gellens, R. and J. Klensin, "Message Submission for Mail", [RFC 4409](#), April 2006.
- [RFC4686] Fenton, J., "Analysis of Threats Motivating DomainKeys Identified Mail (DKIM)", [RFC 4686](#), September 2006.
- [RFC4871] Allman, E., Callas, J., Delany, M., Libbey, M., Fenton, J., and M. Thomas, "DomainKeys Identified Mail (DKIM) Signatures", [RFC 4871](#), May 2007.
- [RFC4880] Callas, J., Donnerhacke, L., Finney, H., and R. Thayer, "OpenPGP Message Format", [RFC 4880](#), November 2007.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 5226](#), May 2008.
- [RFC5451] Kucherawy, M., "Message Header Field for Indicating Message Authentication Status", [RFC 5451](#), April 2009.
- [RFC5672] Crocker, D., Ed., "[RFC 4871](#) DomainKeys Identified Mail (DKIM) Signatures: Update", [RFC 5672](#), August 2009.
- [RFC5751] Ramsdell, B., "Secure/Multipurpose Internet Mail

Extensions (S/MIME) Version 3.1 Message Specification",
[RFC 5751](#), January 2010.

[Appendix A](#). Creating a Public Key {rfc4871bis-02 C}

The default signature is an RSA signed SHA256 digest of the complete

email. For ease of explanation, the openssl command is used to describe the mechanism by which keys and signatures are managed. One way to generate a 1024-bit, unencrypted private key suitable for DOSETA is to use openssl like this:

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

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

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

```
-----BEGIN RSA PRIVATE KEY-----
```

```
MIICXwIBAAKBgQDwIRP/UC3SBsEmGqZ9ZJW3/DkMoGeLnQg1fWn7/zYtIxN2SnFC
jx0CKG9v3b4jYfctNh5ijSsq631uBIItLa7od+v/RtdC2UzJ1lWT947qR+Rcac2gb
to/NMqJ0fzfVjH40uKhItDY9tf6mcwGjaNBcWToIMmPSPDdQPNUYckcQ2QIDAQAB
AoGBALmn+XwWk7akvkUlbq+d0xyLB9i5VBVfje89Teolwc9YJT36BGN/l4e0l6QX
/1//6DWUTB3KI6wFcm7TWJcxbS0tcKZX7FsJvUz1SbQnkS54DJck1EZ0/BLa5ckJ
gAYIaqLA9C0ZWm6i58lLLPadX/rthb7pWzeNcZHjKrjM461ZAKEA+itss2nRlmyO
n1/5yDyCluST4dQf08kAB3toSEVc7DeFeDhnc1mZdjASZNvdHS4gbLIA1hUGEF9m
3hKsGUMMPwJBAPW5v/U+AWTADFCs22t72NUurgzeAbzb1HWMq04y4+9Hpjk5wvL/
eVYizyu3/fGke7aRYw/ADKygmJdW8H/0cCQQDz50Qb4j2QDpPZc0Nc4QlbvMsj
7p7otWR05xRa6SzXqqV3+F0VpqvDmshEBkoCydaYwc2o6WQ5EBmExeV8124XAKEA
qZzGsIxVP+sEVRWZmW6KNFSdVUpk3qzK0Tz/WjQMe5z0UunY9Ax9/4PVhp/j61bf
eAYXunajbBS0Llx4D+TunwJBANKPI5S9iylsbLs6NkaMHV6k5ioHBBmgCak95JGX
GMot/L2x0IYyMLAz6oLWh2hm7zwtb0CgOrPo1ke44hFYnfc=
```

```
-----END RSA PRIVATE KEY-----
```

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

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

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

```
-----BEGIN PUBLIC KEY-----
```

```
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDwIRP/UC3SBsEmGqZ9ZJW3/DkM
oGeLnQg1fWn7/zYtIxN2SnFCjx0CKG9v3b4jYfctNh5ijSsq631uBIItLa7od+v/R
tdC2UzJ1lWT947qR+Rcac2gbto/NMqJ0fzfVjH40uKhItDY9tf6mcwGjaNBcWToI
MmPSPDdQPNUYckcQ2QIDAQAB
```

```
-----END PUBLIC KEY-----
```

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

```
brisbane IN TXT
      ("v=DKIM1; p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQ"
      "KBgQDwIRP/UC3SBsEmGqZ9ZJW3/DkMoGeLnQg1fWn7/zYt"
      "IxN2SnFCjxOCKG9v3b4jYfcTNh5ijSsq631uBIItLa7od+v"
      "/RtdC2UzJ1lWT947qR+Rcac2gbto/NMqJ0fzfVjH40uKhi"
      "tdY9tf6mcwGjaNBcWToIMmPSPDdQPNUYckcQ2QIDAQAB")
```

[Appendix B](#). Acknowledgements

DOSETA is derived from DKIM [[RFC4871](#)].

Authors' Addresses

D. Crocker (editor)
Brandenburg InternetWorking
675 Spruce Dr.
Sunnyvale
USA

Phone: +1.408.246.8253
Email: dcrocker@bbiw.net
URI: <http://bbiw.net>

M. Kucherawy (editor)
Cloudmark
128 King St., 2nd Floor
San Francisco, CA 94107
USA

Email: msk@cloudmark.com

