

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 13, 2012

D. Crocker
Brandenburg InternetWorking
M. Kucherawy
Cloudmark
July 12, 2011

DomainKeys Security Tagging (DOSETA)
draft-crocker-doseta-base-03

Abstract

DomainKeys Security Tagging (DOSETA) is a component mechanism that enables easy development of security-related services, such as for authentication or encryption. It uses self-certifying keys based on domain names. The domain 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 key retrieval. The DOSETA Signing Template creates common framework for a signature of data that are in a "header/content" form. Defining the meaning of a signature is the responsibility of the service that incorporates DOSETA. Data security is enforced through the use of cryptographic algorithms.

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 January 13, 2012.

Copyright Notice

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

This document is subject to [BCP 78](#) and the IETF Trust's Legal

Internet-Draft

DOSETA

July 2011

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.

Table of Contents

1.	Introduction	3
1.1.	Comments and Issues	5
2.	Framework	5
2.1.	DOSETA Architecture	5
2.2.	Terminology	7
2.3.	Syntax	8
3.	DOSETA Library	11
3.1.	Normalization for Transport Robustness	11
3.2.	Canonicalization	12
3.3.	Tag=Value Parameters	16
3.4.	Key Management	17
3.5.	Selectors for Keys	18
3.6.	DNS Binding for Key Retrieval	20
3.7.	Stored Key Data	21
4.	DOSETA H/C Signing Template	23
4.1.	Cryptographic Algorithms	23
4.2.	Signature Data Structure	25
4.3.	Additional Tags	31
4.4.	Signature Calculations	31
4.5.	Signer Actions	35
4.6.	Verifier Actions	38
4.7.	Requirements for Tailoring the Signing Service	45
5.	Semantics of Multiple Signatures	46
5.1.	Example Scenarios	46
5.2.	Interpretation	47
6.	DOSETA Claims Registry Definition	48
7.	Considerations	49
7.1.	IANA Considerations	49
7.2.	Security Considerations	49
8.	References	50
8.1.	Normative References	50
8.2.	Informative References	51
Appendix A.	Creating a Public Key	52
Appendix B.	Acknowledgements	53

Appendix C. Example -- DKIM Using DOSETA	53
C.1. Signing and Verification Protocol	53
C.2. Extensions to DOSETA Template	54
Authors' Addresses	62

[1.](#) Introduction

DomainKeys Security Tagging (DOSETA) is a component mechanism enabling development of security-related services, such as for authentication or encryption; it uses self-certifying keys based on domain names [[RFC1034](#)]. The domain 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 key retrieval. The DOSETA Signing Template creates common framework for signing data that are in a "header/content" form. Defining the intended meaning of a signature is the responsibility of the service that incorporates DOSETA. Data security is enforced through the use of cryptographic algorithms.

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 security-related content appearing in the Content.
- o There is no dependency on having 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 Specific security services can be limited to those needed by the service using them.

DOSETA:

- o enables compatibility with the existing data handling infrastructure and is transparent to the fullest extent possible
- o requires minimal new infrastructure
- o can support a variety of implementation configurations, in order to reduce deployment time
- o can be deployed incrementally
- o allows delegation of signing to third parties

DOSETA derives from Domain Keys Identified Mail (DKIM) [[RFC5672](#)] and has extracted the core portions of the its signing specification [[DKIMSign](#)], 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 ion mechanism using the same DNS-based, self-certified key service as DKIM.

DOSETA features include:

Identity: DOSETA distinguishes the identity of the DOSETA signature's producer from that of any other identifier associated with the data, such as the data's purported author. In particular, the DOSETA header field includes the DOSETA Domain Identifier (DDI), per [Section 2.2.1](#). DOSETA consumers can use the DDI to decide how they want to process the data. The DDI can be directly included in the attributes of the data or can be recorded elsewhere.

NOTE: DOSETA does not, itself, specify that the identity it uses is 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 DOSETA-based security service might choose to add constraints on the choice of identifier, such as having it match another identifier that is associated with the data.

Scalability: DOSETA is designed to easily support the extreme scaling 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 under the domain name associated with the use of DOSETA directly, rather than requiring consultation of a certificate authority. That is, 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: When DOSETA is used to sign data -- independent of the semantics of that signature -- there is a computed hash of some or all of the data that ensures detection of changes to that data, between the times of signing and verifying.

[1.1.](#) Comments and Issues

[RFC EDITOR] Remove this sub-section prior to publication.

Possible applications:

- o JSON structure
- o XMPP message
- o XML object
- o vCard
- o vCal
- o Web page signing?

	Canonicalization	Management	Format	Header	
		(DNS)	(tag=value)	Field	
	+-----+	+-----+	+-----+	+-----+	
++-----++					

DKIM is as specified in [[DKIMSign](#)]. MIMESHA1 is an exemplar use of DOSETA, specified in [[mimeauth](#)]. Message Privacy is a generic term, indicating any service that provides encryption; it is expected that such a service can use the DOSETA core library, but not take advantage of the DOSETA signing template.

The library comprises:

Canonicalization: This ensures common data representation and robustness against some forms of data modification during transit. It is discussed in [Section 3.2](#) and [Section 3.1](#).

Key Management: This covers the mechanisms for discovering and obtaining signature key information by a verifier. It is discussed in [Section 3.4](#), [Section 3.5](#), and [Section 3.6](#).

Format: This describes a simple syntax for encoding parametric information and is discussed in [Section 3.3](#).

Tags: These are common parameters for the stored public key record, defined in [Section 3.7](#) and the common parameters for the signature record that is associated with the signed data, defined in [Section 4.2](#).

[2.2](#). Terminology

Within the specification, the label "[[TEMPLATE](#)]" is used to indicate actions that are required for tailoring the use of DOSETA into a specific service.

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](#)].

Additional terms for this document are divided among Identity and Actors.

[2.2.1](#). 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. The primary example is a domain name.

DOSETA Domain Identifier (DDI): A single domain name that serves as an identifier, referring to the DOSETA key owner's identity. The DDI is specified in [Section 4.2](#). Within this specification, the name has only basic domain name semantics; any possible owner-specific semantics **MUST** be provided in the specification that incorporates DOSETA.

Identity Assessor: A module that consumes DOSETA's payload output. The module is dedicated to the assessment of the delivered identifier. Optionally, other DOSETA (and non-DOSETA) values can also be delivered to this module as well as to a more general message evaluation filtering engine. However, this additional activity is outside the scope of the DOSETA specification.

[2.2.2](#). Actors

Producer: An element in the data handling system that produces a cryptographic encoding, on behalf of a domain, is referred to as a Producer. For example, a signer is a type of producer.

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. For example, a verifier is a type of consumer.

Signer: An element in the data handling system that creates a

digital signature, on behalf of a domain, is referred to as a signer. This element specifies an actor that is a type of DOSETA producer. The actor might operate through a client, server or other agent such as a reputation service. The core requirement is that the data **MUST** be signed before it leaves the control of the signer's administrative domain.

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 of the data or some consuming agent such as a data processing intermediary.

[2.3.](#) Syntax

This section specifies foundational syntactic constructs used in the remainder of the document.

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

[2.3.1.](#) Whitespace

There are three forms of whitespace:

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

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

FWS: is folding whitespace. It allows multiple lines to be joined with each a separated by a sequence having CRLF followed by at least one whitespace.

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

July 2011

```
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's library of functional components is distinguished by a DNS-based, self-certifying public key mechanism, common data normalization and canonicalization algorithms, and a common parameter encoding mechanism.

[3.1.](#) Normalization for Transport Robustness

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. 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.

In order to minimize the chances of such breakage, signers convert the data to a suitable encoding, such as quoted-printable or base64, as described in [\[RFC2045\]](#) before signing. Specification and use of such conversions is outside the scope of DOSETA.

If the data is submitted to a DOSETA process with any local encoding that will be modified before transmission, that modification to a canonical form MUST be done before DOSETA processing. 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, a DOSETA producer MUST use the data as it is expected to be received by the DOSETA consumer rather than in some local or internal form.

[3.2.](#) Canonicalization

Some data handling systems modify the original data during transit, potentially invalidating a cryptographic function. In some cases, mild modification of data can be immaterial to the validity of a DOSETA-based service. In these cases, a canonicalization algorithm that survives modest handling modification is preferred.

In other cases, preservation of the exact, original bits is required; even minor modifications need to result in a failure. Hence a canonicalization algorithm is needed that does not tolerate any in-transit modification of the data.

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 presented for canonicalization MUST already be in "network normal" format -- text is ASCII encoded, lines are separated with CRLF characters, etc.) See [Section 3.1](#) for information about normalizing data.

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 Internet Mail header fields or MIME header fields

Content: Lines of ASCII text

Some DOSETA producers might be willing to accept modifications to some portions of the data, but not other portions. For DOSETA, a producer MAY specify one algorithm for the header and another for the content.

If no canonicalization algorithm is specified, the "simple" algorithm defaults for each part. DOSETA producers MUST implement both of the

base canonicalization algorithms. Because additional canonicalization algorithms might be defined in the future, producers MUST ignore any unrecognized canonicalization algorithms.

Canonicalization simply prepares the data for presentation to the DOSETA processing algorithm.

NOTE: Canonicalization operates on a copy of the data; it MUST NOT change the transmitted data in any way. Canonicalization of distinct data portions is described below.

[3.2.1](#). Header Canonicalization Algorithms

This section describes basic entries for the Header Canonicalization IANA registry defined in [\[DKIMSign\]](#), which also applies to DOSETA header canonicalization.

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 processing algorithm exactly as they are in the data being processed. 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. The following steps MUST be applied 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.

[3.2.2.](#) Content Canonicalization Algorithms

This section describes basic entries for the Message Canonicalization IANA registry defined in [[DKIMSign](#)], which also applies to DOSETA Content.

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 3.2.2](#). 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:

2jmmj7l5rSw0yVb/vlWAYkK/YBwk=

The sha256 value is:

NOTE: The relaxed Content canonicalization algorithm can enable certain types of extremely crude "ASCII Art" attacks in which a message can be conveyed, by adjusting the spacing between words. If this is a concern, the "simple" Content canonicalization algorithm is more appropriate for use.

3.2.3. Canonicalization Examples

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>
```

3.3. Tag=Value Parameters

DOSETA uses a simple "tag=value" parameter syntax in several contexts, such as when representing associated cryptographic data and domain 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 definition of a tag will determine the specific encoding for its associated value. Unencoded semicolon (";") characters MUST NOT occur in the tag value, since that separates tag-specs.

NOTE: The "plain text" defined below, as "tag-value", only supports use of 7-bit characters. However, it is likely that support of UTF-8 Unicode [\[UTF8\]](#) data will eventually be deemed important.

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 / "_"
```

Internet-Draft

DOSETA

July 2011

NOTE: 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 interpret a VALCHAR as case-sensitive, unless the specific tag description of semantics specifies case insensitivity.

Tags MUST be unique; 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.

[3.4.](#) Key Management

Applications require some level of assurance that a producer is authorized to use a cited public. Many applications achieve this by using public key certificates issued by a trusted authority. For applications with modest certification requirements, DOSETA achieves a sufficient level of security, with excellent scaling properties, by simply having the consumer query the purported producer's DNS entry (or a supported equivalent) in order to retrieve the public key. The safety of this model is increased by the use of DNSSEC [[RFC4033](#)] for the key records in the DNS.

DOSETA keys might be stored in multiple types of key servers and in multiple formats. As long as the key-related information is the same and as long as the security properties of key storage and retrieval are the same, DOSETA's operation is unaffected by the actual source of a key.

The abstract 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" parameter ([Section 4.2](#))

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

s-val: The selector of the lookup as specified in the "s" parameter ([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 between the abstract lookup algorithm and a physical instance, using DNS TXT records, per [Section 3.6](#). Other bindings can be defined.

[3.5](#). Selectors for Keys

It can be extremely helpful to support multiple DOSETA keys for the same domain name. For example:

- * Rolling over from one key to another is a common security administration requirement; for an operational service this is made far easier when the old and new keys are supported simultaneously.
- * Domains that want to delegate signing capability for a specific address for a given duration to a partner, such as an advertising provider or other outsourced function.
- * Domains that want to allow frequent travelers to generate

signed data locally without the need to connect to a particular server.

- * "Affinity" domains (such as, college alumni associations) that provide data forwarding, but that do not operate a data origination agent for outgoing data.

To these ends, DOSETA includes a mechanism that supports 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", "columbeach", and "reykjavik"), the signing date (for example, "january2005", "february2005", etc.), or even an individual user.

For further administrative convenience, sub-division of selectors is

allowed, distinguished as dotted sub-components of the selector name. 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 the corresponding selectors for each domain are determined by the domain owner. Many domain owners will be satisfied with just one selector, whereas administratively distributed organizations might choose to manage disparate selectors and key pairs in different regions or on different servers.

As noted, 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, reuse of selectors with different keys is ill-advised. A better strategy is to assign new keys to new selectors.

[3.6.](#) DNS Binding for Key Retrieval

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

[3.6.1.](#) 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" parameter ([Section 4.2](#)); use of selectors is discussed in [Section 3.5](#)

d: is the domain of the signature, as specified in the "d" parameter ([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, with 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. That said, a new service is certainly free to define a new constant and maintain an entirely independent set of keys.

Given a DOSETA-Signature field with a "d" parameter of "example.com" and an "s" parameter 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 the context of DOSETA and their presence can be

problematic. Hence DNS wildcards with DOSETA SHOULD NOT be used. Note also that wildcards within domains (for example, s._domainkey.*.example.com) are not supported by the DNS.

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

The DNS Resource Record type used is specified by an option to the query-type ("q") parameter. The only option defined in this base specification is "txt", indicating the use of a DNS TXT Resource Record (RR), as defined in [Section 3.7](#). 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.

[3.7.](#) Stored Key Data

This section defines a syntax for encoding stored key data within an unstructured environment such as the simple text environment of a DNS TXT record.

[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 3.6](#). Use of this means sharing the set of public keys with DKIM and other DOSETA-based services.

The overall syntax is a tag-list as described in [Section 3.3](#). The base set of valid tags is described below. Other tags MAY be present and MUST be ignored by any implementation that does not understand them.

k= Key type (MAY be include; 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 3.5](#) and A.1.1) is being used in the "p" parameter. (Note: the "p" parameter

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 (MAY be included; 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 (MUST be included). 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" parameter.

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; it 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 for DKIM this 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. To the extent that authentication is meant by the presence of a signature, that needs to be specified as part of the semantics for the service based upon this template.

[TEMPLATE] (Header/Content Mapping) The service incorporating this mechanism MUST define the precise mappings onto the template provided in this section. (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

DOSETA supports multiple digital signature algorithms:

rsa-sha1: The rsa-sha1 Signing Algorithm computes a message hash as described in [Section 4.4](#) 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.

Internet-Draft

DOSETA

July 2011

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:

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

- * Keys can be replaced on a regular basis, thus their lifetime can be relatively short
- * The security goals of this specification are modest compared to typical goals of other systems that employ digital signatures

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

[4.2.](#) Signature Data Structure

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 3.3](#).

[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 structure is part of a sequence of structures -- such as being added to an email header -- it SHOULD NOT be reordered and SHOULD be pre-pended 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 an exact 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 (MUST be included). 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 (MUST be included). 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 4.5](#)) for how the signature is computed.

ABNF:

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

bh= The hash of the canonicalized Content (body), as limited by the "l" parameter (MUST be included). 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 4.4](#) 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= Data canonicalization (MAY be included; 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 Header Canonicalization IANA registry defined in [\[DKIMSign\]](#)

sig-c-content: A value from Message Canonicalization IANA registry defined in [[DKIMSign](#)]

These algorithms are described in [Section 3.2](#). 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
```

cl= A list of semantic claims, asserting the set of "meanings" intended by the signer, such as author validity or content validity (SHOULD be included; default is "handled"). The list of supported claims comprises values from the DOSETA Claims IANA registry, defined in [Section 6](#) using values listed in [Section 7.1.2](#).

ABNF:

```
sig-cl-tag      = %x63 %x6C [FWS] "=" [FWS]
                  sig-cl-tag-claim
                  ["/" sig-c-tag-claim]
sig-c-tag-claim  = hyphenated-word
                  ; per DOSETA Claims Registry
```

d= The DDI doing the signing (MUST be included). Hence, the DDI value is used to form the query for the public key. The DDI 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 DDI 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 (MUST be included). 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 4.5.3](#) 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 (MAY be included; 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 (MUST be included).

ABNF:

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

t= Signature Timestamp (SHOULD be included; 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 (SHOULD be included; default is no expiration). The format is the same as in the "t" parameter, 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

Internet-Draft

DOSETA

July 2011

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" parameter MUST be greater than the value of the "t" parameter if both are present.

NOTE: The "x" parameter 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.](#) Additional Tags

Some applications can benefit from additional, common functional enhancements. These are defined here, as options to the core mechanism.

id= The local identifier, restricting the scope of the DDI, such as to a specific user. This value is combined with the DDI, to specify the identifier to be used for assessment. Administrative choices for selectors can provide different keys for different local identifiers. See [Section 3.5](#)

ABNF:

```
key-id-tag   = %x69 %64 [FWS] "=" [FWS] hyphenated-word
```

[4.4.](#) Signature Calculations

Hashing and cryptographic signature algorithms are combined into a

procedure for computing a digital signature. Producers will choose appropriate parameters for the signing process. Consumers will use the tags that are then passed as an associated DOSETA-Signature header field. [Section 4.2](#). In the following discussion, the names of the tags are parameters in that field.

The basic operations for producing a signature are canonicalization, hashing and signing. Canonicalization removes irrelevant variations. Hashing produces a very short representation for the data and signing produces a unique, protected string to be exchanged.

NOTE: Canonicalization (see [Section 3.2](#)) prepares a separate representation of the data for additional processing; it does not affect the original, transmitted data in any way.

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

The combined hashing and signing algorithms are:

Content Hash:

1. Truncate the content to the length specified in the "l" parameter.
2. Canonicalize the truncated content, using the algorithm specified in the "c" parameter.
3. Hash the canonicalized content, using the hashing algorithm specified in the "a" parameter.
4. Convert the resulting hash to base64 form.
5. Signers then insert the base64 result into the "bh" parameter of the DOSETA-Signature field; verifiers compare their hash with the value in the "bh" parameter.

Header Hash:

1. Select the header fields specified by the "h" parameter.
2. Ensure that each field is terminated by a single CRLF.
3. Canonicalize each of these fields, using the header canonicalization algorithm specified in the "c" parameter.
4. Select the D0SETA-Signature field that exists (verifying), or will be inserted (signing), in the header.

5. From that field, delete the value portion of the "b" parameter, including all surrounding whitespace; that is, treat the "b" parameter as containing an empty string.
6. Canonicalize the resulting field, using the Header canonicalization algorithm specified in the "c" parameter, but remove the trailing CRLF, if present.
7. Using the hashing algorithm specified in the "a" parameter, hash the sequence:
 1. the canonicalized header fields, in the order specified by the "h" parameter,
 2. the output from the content hash, and
 3. the canonicalized D0SETA-Signature field.

Signature:

1. Obtain the relevant key associated with the relevant domain and selector; for signing this is a private key; for verifying this is a public key.
2. Obtain the header hash produced by the previous calculation.
3. Using the signing algorithm specified in the "a" parameter, and the relevant key, process the hash.

All tags cited in the "h" parameter MUST be included even if they are not understood by the verifier. Note that the DOSETA-Signature field is presented to the hash algorithm after the content hash is processed, rather than with the rest of the header fields that are processed before the content hash. The DOSETA-Signature header structure MUST NOT be cited in its own h= tag. If present, other DOSETA-Signature header fields MAY be cited and included in the signature process (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 3.2](#), the DOSETA signing process treats the content as a simple string of octets. DOSETA content MAY be either simple lines of plain-text or as a MIME object; no special treatment is afforded to MIME content.

Formally, the algorithm for the signature is as follows:

ABNF:

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

where:

content-hash: is the output from hashing the content, using hash-alg.

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

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

l-param: is the length-of-content value of the "l" parameter.

data-hash: is the output from using the hash-alg algorithm, to hash the header including the DKIM-Signature header, and the content hash.

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

D-SIG: is the canonicalized DOSETA-Signature field without the signature value portion of the parameter, itself; that is, an empty parameter value.

canon-content: is the canonicalized data content (respectively) as defined in [Section 3.2](#) (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" parameter.

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

selector: is the selector value specified in the "s" parameter.

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

[4.5.](#) Signer Actions

The following steps are performed in order by signers.

[4.5.1.](#) 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.

[4.5.2.](#) 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.

4.5.3. 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" parameter 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" parameter 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" parameter 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" parameter.

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.

[4.5.4.](#) Compute the Message Signature

The signer **MUST** compute the message hash as described in [Section 4.4](#) 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.

[4.5.5.](#) 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" parameter **MUST** be the appropriately signed hash computed in the previous step, signed using the algorithm specified in the "a" parameter of the DOSETA-Signature header field and using the private key corresponding to the selector given in the "s=" tag of the DOSETA-Signature header field, as chosen above in [Section 4.5.2](#)

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.

[4.6.](#) Verifier Actions

Since a signer **MAY** remove or revoke a public key at any time, it is recommended that verification occur in a timely manner. In many

configurations, the most timely place is during acceptance by the border MTA or shortly thereafter. In particular, deferring

verification until the message is accessed by the end user is discouraged.

A border or intermediate 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.

[4.6.1.](#) 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.

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.

[4.6.2](#). 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.

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" parameter, 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"

parameter. If not, the DOSETA-Signature header field MUST be ignored and the verifier SHOULD return PERMFAIL (domain mismatch).

If the "h" parameter 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" parameter and the signature has expired.

Verifiers MAY ignore the DOSETA-Signature header field if the domain used by the signer in the "d" parameter 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.

[4.6.3.](#) 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" parameter 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 3.4](#). 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 following order -- 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 3.4](#) using the algorithm in the "q=" tag, the domain from the "d" parameter, and the selector from the "s" parameter.
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.
6. If the "h" parameter 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" parameter, the verifier MUST ignore the key record and return PERMFAIL (inappropriate hash algorithm).

7. If the public key data (the "p" parameter) 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" parameters in the DOSETA-Signature header field, the verifier MUST immediately return PERMFAIL (inappropriate key algorithm).

4.6.4. 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" parameter, the Content length specified in the "l" parameter, and the header field names in the "h" parameter, prepare a canonicalized version of the Content as is described in [Section 4.4](#) (note that this version does not actually need to be instantiated). When matching header field names in the "h" parameter against the actual message header field, comparisons MUST be case-insensitive.
2. Based on the algorithm indicated in the "a" parameter, compute the message hashes from the canonical copy as described in [Section 4.4](#)
3. Verify that the hash of the canonicalized Content computed in the previous step matches the hash value conveyed in the "bh" parameter. 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" parameter, verify the signature against the header hash using the mechanism appropriate

for the public key algorithm described in the "a" parameter. 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" parameter 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.

4.6.5. 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.

4.6.6. 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 DDI is not the same as the address in the From: header field, the data system SHOULD take pains to ensure that the actual DDI 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.

[4.7.](#) Requirements for Tailoring the Signing Service

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

Association: Specify the means by which the signature field is associated with the data it signs. This identifies the specific signing service and the mechanics of attaching the signature. 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 or JSON, then that also needs to be specified, including the algorithm for mapping between the common encoding provided here and the new encoding.

Internet-Draft

DOSETA

July 2011

Semantics: Define the meaning(s) of a signature that is intended by the signer. 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. A single service can support multiple semantics for a signature. These SHOULD be specified using the "cl=" claims mechanism defined in [Section 4.2](#) and [Section 6](#) with values listed in [Section 7.1.2](#)

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

Required Fields: Specify the header fields that MUST, SHOULD or MAY be included in the signature calculation.

Required Algorithms: Specify the hashing and signing algorithms that MUST, SHOULD or MAY be supported by participants in the service.

[5.](#) Semantics of Multiple Signatures

[5.1.](#) Example Scenarios

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

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" parameter (to satisfy strict verifiers) and a second time with a limited set of headers and an "l" parameter (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 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

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 4.5.3](#) 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 instances of header fields. For example, if the message being signed already contains three DOSETA-Signature header fields (from the bottom, up) A, B, and C, it is possible to sign all of them, A and B only, or A only, but not C only, B only, B and C 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 4.6.1](#) 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.](#) DOSETA Claims Registry Definition

A registry entry MUST contain:

Crocker & Kucherawy	Expires January 13, 2012	[Page 48]
---------------------	--------------------------	-----------

Internet-Draft	DOSETA	July 2011
----------------	--------	-----------

Label: Specifies a textual name for claim, to be used in the "cl=" tag.

Description: Explains the semantics of the claim being asserted.

The registry entries are contained in the IANA DOSETA Claims Registry, defined in [Section 7.1.2](#)

[7.](#) Considerations

[7.1.](#) IANA Considerations

[7.1.1.](#) DKIM Registries

DOSETA relies on IANA registration data bases specified by DKIM [[DKIMSign](#)]. Services that incorporate DOSETA might need to define new registries or add to existing ones.

[7.1.2.](#) Claims Registry

Per [\[RFC2434\]](#), IANA is requested to establish a DOSETA Claims Registry, for assertions (claims) that are meant by the presence of the DOSETA-based signature that contains the claims. See [Section 6](#) for the definition of the columns in the registry table.

LABEL	CLAIM DESCRIPTION
handled	The signer claims they have had a role in processing the object. (This claim is approximately equivalent to the semantics of DKIM.)
validauth	If there is a standardized field listing the purported author of the data, the signer claims that the value in that field is valid.
validdata	The signer claims that all of the data in the object valid.
validfields	The signer claims that the portions of the object that are covered by the signature hash are valid.

Table 1: DOSETA Claim Registry (with initial values)

[7.2.](#) Security Considerations

Any mechanism that attempts to prevent or detect abuse is subject to intensive attack. DOSETA needs to be carefully scrutinized to identify potential attack vectors and the vulnerability to each. See

also [\[RFC4686\]](#).

DOSETA core technology derives from DKIM [\[DKIMSign\]](#). The Security Considerations of that specification applies equally to DOSETA.

The DOSETA "cl=" claims list provides a list of claimed meanings for a DOSETA signature. An opportunity for security problems comes from failing to distinguish between a signer "claim" and claim validity. Whether to trust claims made by a signer requires a level of assessment beyond DOSETA.

[8.](#) References

[8.1.](#) 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.
- [RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", [RFC 2434](#), October 1998.
- [RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", [RFC 3447](#), February 2003.

Crocker & Kucherawy Expires January 13, 2012 [Page 50]

Internet-Draft DOSETA July 2011

- [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.

[8.2.](#) Informative References

- [DKIMSign] Allman, E., Callas, J., Delany, M., Libbey, M., Fenton, J., and M. Thomas, "DomainKeys Identified Mail (DKIM) Signatures", [RFC 4871](#), May 2007.
- [RFC1847] Galvin, J., Murphy, S., Crocker, S., and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", [RFC 1847](#), October 1995.
- [RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Content", [RFC 2047](#), November 1996.
- [RFC3766] Orman, H. and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", [BCP 86](#), [RFC 3766](#), April 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.
- [RFC4870] Delany, M., "Domain-Based Email Authentication Using Public Keys Advertised in the DNS (DomainKeys)", [RFC 4870](#), May 2007.
- [RFC4880] Callas, J., Donnerhake, L., Finney, H., and R. Thayer, "OpenPGP Message Format", [RFC 4880](#), November 2007.

- [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.
- [UTF8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", [RFC 3629](#), November 2003.
- [mimeauth] Crocker, D. and M. Kucherawy, "MIME Content Authentication using DOSETA (MIMEAUTH)", I-D [draft-crocker-doseta-mimeauth](#), 2011.

[Appendix A](#). Creating a Public Key

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
jx0CKG9v3b4jYfcTNh5ijSsq631uBItLa7od+v/RtdC2UzJ1lWT947qR+Rcac2gb
to/NMqJ0fzfVjH40uKhitdY9tf6mcwGjaNBcWToIMmPSPDdQPNUYckcQ2QIDAQAB
AoGBALmn+XwWk7akvkUlbq+dOxyLB9i5VBVfje89Teolwc9YJT36BGN/l4e0l6QX
/1//6DWUTB3KI6wFcm7TWJcxbS0tcKZX7FsJvUz1SbQnks54DJck1EZ0/BLa5ckJ
gAYIaqLA9C0ZwM6i58lLLPadX/rthb7pWzeNcZHjKrjM461ZAkEA+itss2nRlmyO
n1/5yDyCluST4dQf08kAB3toSEVc7DeFeDhnC1mZdjASZNvdHS4gbLIA1hUGEF9m
3hKsGUMMPwJBAPW5v/U+AWTADFCs22t72NUurgzeAbzb1HWMq04y4+9Hpjk5wvL/
eVYizyu3e3/fGke7aRYw/ADKygmJdW8H/0cCQQDz50Qb4j2QDpPZc0Nc4QlbvMsj
7p7otWR05xRa6SzXqqV3+F0VpqvDmshEBkoCydaYwc2o6WQ5EBmExeV8124XAkEA
qZzGsIxVP+sEVRWZmW6KNFSdVUpk3qzK0Tz/WjQMe5z0UunY9Ax9/4PVhp/j61bf
eAYXunajbBSOLlx4D+TunwJBANKPI5S9iyLSbLS6NkaMHV6k5ioHBBmgCak95JGX
GMot/L2x0IYyMLAz6oLWh2hm7zwtb0Cg0rPo1ke44hFYnfc=
```

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

Internet-Draft

DOSETA

July 2011

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

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

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

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

```
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDwIRP/UC3SBsEmGqZ9ZJW3/DkM  
oGeLnQg1fWn7/zYtIxN2SnFCjxOCKG9v3b4jYfcTNh5ijSsq631uBItLa7od+v/R  
tdC2UzJ1lWT947qR+Rcac2gbto/NMqJ0fzfVjH40uKhItY9tf6mcwGjaNBcWToI  
MmPSPDdQPNUYckcQ2QIDAQAB
```

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

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

```
brisbane IN TXT
```

```
("v=DKIM1; p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQ"  
"KBgQDwIRP/UC3SBsEmGqZ9ZJW3/DkMoGeLnQg1fWn7/zYt"  
"IxN2SnFCjxOCKG9v3b4jYfcTNh5ijSsq631uBItLa7od+v"  
"/RtdC2UzJ1lWT947qR+Rcac2gbto/NMqJ0fzfVjH40uKhi"  
"tdY9tf6mcwGjaNBcWToIMmPSPDdQPNUYckcQ2QIDAQAB")
```

[Appendix B.](#) Acknowledgements

DOSETA is derived from DKIM [[DKIMSign](#)]. DKIM is an evolution of DomainKeys [[RFC4870](#)], which was developed by Mark Delany, then of Yahoo!. In particular, the key management service, based on the DNS, and the user-INvisible tagging scheme was developed by him.

[Appendix C.](#) Example -- DKIM Using DOSETA

This example re-specifies DKIM in terms of DOSETA, while retaining bit-level compatibility with the existing DKIM specification [[DKIMSign](#)].

NOTE: This section is merely an example. Any use of normative language in this section is strictly for completeness of the example and has no normative effect on the DOSETA specification.

[C.1.](#) Signing and Verification Protocol

The DOSETA template specifies TEMPLATE information that is required to tailor the signing service:

Association: The DOSETA-Signature data are stored in a DKIM-Signature header field that is part of the header of the message being signed. This contains all of the signature and key-fetching data, per [Section 4.2](#).

Semantics: A DKIM signature means that the owner of the signing domain is taking "some" responsibility for the message. Hence, the payload, or output, of DKIM is:

- + A validated domain name, specifically the d= parameter in the DKIM-Signature header field
- + An indication that its use has been validated

The nature and extent of a DKIM signer's responsibility can vary widely and is beyond the scope of this specification.

Structural Mapping: DKIM maps the DOSETA Header processing to an email header and the DOSETA Content to an email body, per [\[RFC5322\]](#)

Required Support

Field Hash: The basic rule for choosing fields to include is to select those fields that constitute the "core" of the message content. Hence, any replay attack will have to include these in order to have the signature succeed; but with these included, the core of the message is valid, even if sent on to new recipients.

Common examples of fields with addresses and fields with textual content related to the body are: From (MUST be included), Reply-To, Subject, Date, To, Cc, Resent-Date,

Resent-From, Resent-To, Resent-Cc, In-Reply-To, References,
List-Id, List-Help, List-Unsubscribe, List-Subscribe, List-
Post, List-Owner, List-Archive.

Required Algorithms: Use DOSETA defaults.

[C.2.](#) Extensions to DOSETA Template

This section contains specifications that are added to the basic
DOSETA H/C Signing Template.

Crocker & Kucherawy

Expires January 13, 2012

[Page 54]

Internet-Draft

DOSETA

July 2011

[C.2.1.](#) Signature Data Structure

These are DKIM-specific tags:

i= The Agent or User Identifier (AUID) on behalf of which the
SDID is taking responsibility (DOSETA-quoted-printable;
OPTIONAL, default is an empty <local-part> followed by an "@"
followed by the domain from the "d=" tag).

The syntax is a standard email address where the <local-part>
MAY be omitted. The domain part of the address MUST be the
same as, or a subdomain of, the value of the "d=" tag.

Internationalized domain names MUST be converted using the
steps listed in [Section 4 of \[RFC5890\]](#) using the "ToASCII"
function.

ABNF:
sig-i-tag = %x69 [FWS] "=" [FWS]
 [local-part] "@" domain-name

The AUID is specified as having the same syntax as an email

address, but is not required to have the same semantics. Notably, the domain name is not required to be registered in the DNS -- so it might not resolve in a query -- and the <local-part> MAY be drawn from a namespace unrelated to any mailbox. The details of the structure and semantics for the namespace are determined by the Signer. Any knowledge or use of those details by verifiers or assessors is outside the scope of the DOSETA Signing specification. The Signer MAY choose to use the same namespace for its AUIDs as its users' email addresses or MAY choose other means of representing its users. However, the signer SHOULD use the same AUID for each message intended to be evaluated as being within the same sphere of responsibility, if it wishes to offer receivers the option of using the AUID as a stable identifier that is finer grained than the SDID.

NOTE: The <local-part> of the "i=" tag is optional because in some cases a signer might not be able to establish a verified individual identity. In such cases, the signer might wish to assert that although it is willing to go as

far as signing for the domain, it is unable or unwilling to commit to an individual user name within their domain. It can do so by including the domain part but not the <local-part> of the identity.

NOTE: Absent public standards for the semantics of an AUID, an assessment based on AUID requires a non-standardized basis.

NOTE: This specification does not require the value of the "i=" tag to match the identity in any Header field. This is considered to be an assessment-time policy issue. Constraints between the value of the "i=" tag and other identities in other Header fields might seek to apply basic authentication into the semantics of trust associated with a role such as content author. Trust is a broad and complex topic and trust mechanisms are subject to highly creative attacks. The real-world efficacy of any but the most basic bindings between the "i=" value and other identities is not well established, nor is its vulnerability to subversion by an attacker. Hence reliance on the use of these options

needs to be strictly limited. In particular, it is not at all clear to what extent a typical end-user recipient can rely on any assurances that might be made by successful use of the "i=" options.

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

ABNF:

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

NOTE: Use of the "l=" tag might allow display of fraudulent content without appropriate warning to end users. The "l=" tag is intended for increasing signature robustness when sending to intermediaries that append data to Content, such as mailing lists that both modify their content and do not sign their messages. However, using the "l=" tag enables attacks in which an intermediary with malicious intent modifies a message to include content that solely benefits

the attacker. It is possible for the appended content to completely replace the original content in the end recipient's eyes and to defeat duplicate message detection algorithms. Examples are described in Security Considerations [Section 7.2](#). To avoid this attack, signers need be extremely wary of using this tag, and verifiers might wish to ignore the tag or remove text that appears after the specified content length.

NOTE: The value of the "l=" tag is constrained to 76 decimal digits. This constraint is not intended to predict the size of future data or to require implementations to use an integer representation large enough to represent the maximum possible value, but is intended to remind the implementer to

check the length of this and all other tags during verification and to test for integer overflow when decoding the value. Implementers might need to limit the actual value expressed to a value smaller than 10^{76} , for example, to allow a message to fit within the available storage space.

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

The Header fields referenced by the "h=" tag refer to the fields in the [\[RFC5322\]](#) Header, not to any copied fields in the "z=" tag. Copied header field values are for diagnostic use.

Header fields with characters requiring conversion (perhaps from legacy MTAs that are not [\[RFC5322\]](#) compliant) SHOULD be converted as described in MIME Part Three [\[RFC2047\]](#).

ABNF:

sig-z-tag = %x7A [FWS] "=" [FWS]

```
sig-z-tag-copy
*( "|" [FWS] sig-z-tag-copy )
sig-z-tag-copy = hdr-name [FWS] ":"
qp-hdr-value
```

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

```
DKIM-Signature: v=1; a=rsa-sha256; d=example.net;
s=brisbane; c=simple; q=dns/txt; i=@eng.example.net;
t=1117574938; x=1118006938;
h=from:to:subject:date;
z=From:foo@eng.example.net|To:joe@example.com|
Subject:demo=20run|
Date:July=205,=202005=203:44:08=20PM=20-0700;
bh=MTIzNDU2Nzg5MDZyMzQ1Njc4OTAxMjM0NTY3ODkwMTI=;
b=dzdVy0fAKCdLXdJOc9G2q8LoXSlEniSbav+yuU4zGeeruD00lszZVoG4ZHRNiYzR
```

[C.2.1.1](#). Content Length Limits

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

This capability is provided because it is very common for intermediate data handling services to add trailers to text (for example, instructions how to get off a mailing list). Until such data is signed by the intermediate handler, the text length count can be a useful tool for the verifier since it can, as a matter of policy, accept messages having valid signatures that do not cover the additional data.

NOTE: Using text length limits enables an attack in which an attacker modifies a message to include content that solely benefits the attacker. It is possible for the appended content to completely replace the original content in the end recipient's eyes and to defeat duplicate message detection algorithms. To avoid this attack, signers need to be wary of using this tag, and verifiers might wish to ignore the tag or remove text that appears after the specified content length, perhaps based on other criteria.

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

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

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

Creators wishing to ensure that no modification of any sort can occur will specify the "simple" canonicalization algorithm for all data portions and will and omit the text length counts.

C.2.1.2. Signature Verification

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

NOTE: Verifiers that truncate the Content at the indicated Content length might pass on a malformed MIME message if the signer used the "N-4" trick (omitting the final "--CRLF") described in the informative note in [Appendix C.2.1.1](#). Such verifiers might wish to check for this case and include a trailing "--CRLF" to avoid breaking the MIME structure. A simple way to achieve this might be to append "--CRLF" to any "multipart" message with a Content length; if the MIME structure is already correctly formed, this will appear in the postlude and will not be displayed to the end user.

Internet-Draft

DOSETA

July 2011

[C.2.2.](#) Stored Key Data

This section defines additions to the DOSETA Library, concerning stored key data.

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

ABNF:

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

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

Internet-Draft

DOSETA

July 2011

ABNF:

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

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

* matches all service types

email electronic mail (not necessarily limited to SMTP)

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

ABNF:

```
key-s-tag      = %x73 [FWS] "=" [FWS]
                  key-s-tag-type
```

```

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

```

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

Crocker & Kucherawy Expires January 13, 2012 [Page 61]

Internet-Draft

DOSETA

July 2011

y This domain is testing DOSETA. Verifiers MUST NOT treat data from signers in testing mode differently from unsigned data, even if the signature fails to verify. Verifiers MAY wish to track testing mode results to assist the signer.

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

ABNF:

```

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

```

Authors' Addresses

D. Crocker

Brandenburg InternetWorking
675 Spruce Dr.
Sunnyvale
USA

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

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

Email: msk@cloudmark.com