Network Working Group
Request for Comments: 4870
Obsoleted By: 4871
Category: Historic

M. Delany
Yahoo! Inc
May 2007

Domain-Based Email Authentication Using Public Keys Advertised in the DNS (DomainKeys)

Status of This Memo

Copyright Notice

Abstract

"DomainKeys" creates a domain-level authentication framework for email by using public key technology and the DNS to prove the provenance and contents of an email.

This document defines a framework for digitally signing email on a per-domain basis. The ultimate goal of this framework is to unequivocally prove and protect identity while retaining the semantics of Internet email as it is known today.

Proof and protection of email identity may assist in the global control of "spam" and "phishing".

Table of Contents

1. Introduction

This document proposes an authentication framework for email that stores public keys in the DNS and digitally signs email on a domain basis. Separate documents discuss how this framework can be extended to validate the delivery path of email as well as facilitate per-user authentication.

The DomainKeys specification was a primary source from which the DomainKeys Identified Mail [DKIM] specification has been derived. The purpose in submitting this document is as an historical reference for deployed implementations written prior to the DKIM specification.

1.1. Lack of Authentication Is Damaging Internet Email

Authentication of email is not currently widespread. Not only is it difficult to prove your own identity, it is impossible to prevent others from abusing your identity.

While most email exchanges do not intrinsically need authentication beyond context, it is the rampant abuse of identity by "spammers", "phishers", and their criminal ilk that makes proof necessary. In other words, authentication is as much about protection as proof.

Importantly, the inability to authenticate email effectively delegates much of the control of the disposition of inbound email to the sender, since senders can trivially assume any email address. Creating email authentication is the first step to returning dispositional control of email to the recipient.

For the purposes of this document, authentication is seen from a user perspective, and is intended to answer the question "who sent this email?" where "who" is the email address the recipient sees and "this email" is the content that the recipient sees.

1.2. Digitally Signing Email Creates Credible Domain Authentication

DomainKeys combines public key cryptography and the DNS to provide credible domain-level authentication for email.

When an email claims to originate from a certain domain, DomainKeys provides a mechanism by which the recipient system can credibly determine that the email did in fact originate from a person or system authorized to send email for that domain.

The authentication provided by DomainKeys works in a number of scenarios in which other authentication systems fail or create complex operational requirements. These include the following:

- o forwarded email
- o distributed sending systems
- o authorized third-party sending

This base definition of DomainKeys is intended to primarily enable domain-level authenticity. Whether a given message is really sent by the purported user within the domain is outside the scope of the base definition. Having said that, this specification includes the possibility that some domains may wish to delegate fine-grained authentication to individual users.

1.3. Public Keys in the DNS

DomainKeys differs from traditional hierarchical public key systems in that it leverages the DNS for public key management, placing complete and direct control of key generation and management with the owner of the domain. That is, if you have control over the DNS for a given domain, you have control over your DomainKeys for that domain.

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

1.4. Initial Deployment Is Likely at the Border MTA

For practical reasons, it is expected that initial implementations of DomainKeys will be deployed on Mail Transfer Agents (MTAs) that accept or relay email across administrative or organizational boundaries. There are numerous advantages to deployment at the border MTA, including:

- a reduction in the number of MTAs that have to be changed to support an implementation of DomainKeys

- a reduction in the number of MTAs involved in transmitting the email between a signing system and a verifying system, thus reducing the number of places that can make accidental changes to the contents

- removing the need to implement DomainKeys within an internal email network.

However, there is no necessity to deploy DomainKeys at the border as signing and verifying can effectively occur anywhere from the border MTA right back to the Mail User Agent (MUA). In particular, the best place to sign an email for many domains is likely to be at the point of SUBMISSION where the sender is often authenticated through SMTP AUTH or other identifying mechanisms.

1.5. Conveying Verification Results to MUAs

It follows that testing the authenticity of an email results in some action based on the results of the test. Oftentimes, the action is to notify the MUA in some way -- typically via a header line.

The "Domainkey-Status:" header is defined in this specification for recording authentication results in the email.

1.6. Technical Minutiae Are Not Completely Covered

The intent of this specification is to communicate the fundamental characteristics of DomainKeys for an implementor. However, some aspects are derived from the functionality of the openssl command [OPENSSL] and, rather than duplicate that documentation, implementors are expected to understand the mechanics of the openssl command, sufficient to complete the implementation.

1.7. Motivation

The motivation for DomainKeys is to define a simple, cheap, and "sufficiently effective" mechanism by which domain owners can control who has authority to send email using their domain. To this end, the designers of DomainKeys set out to build a framework that:

- o is transparent and compatible with the existing email infrastructure
- o requires no new infrastructure
- o can be implemented independently of clients in order to reduce deployment time
- o does not require the use of a central certificate authority that might impose fees for certificates or introduce delays to deployment
- o can be deployed incrementally

While we believe that DomainKeys meets these criteria, it is by no means a perfect solution. The current Internet imposes considerable compromises on any similar scheme, and readers should be careful not to misinterpret the information provided in this document to imply that DomainKeys makes stronger credibility statements than it is able to do.

1.8. Benefits of DomainKeys

As the reader will discover, DomainKeys is solely an authentication system. It is not a magic bullet for spam, nor is it an authorization system, a reputation system, a certification system, or a trust system.

However, a strong authentication system such as DomainKeys creates an unimpeachable framework within which comprehensive authorization systems, reputations systems, and their ilk can be developed.

1.9. Definitions

With reference to the following sample email:

```
Line   Data
Number Bytes               Content
----   --- --------------------------------------------
  01    46 From: "Joe SixPack" <joe@football.example.com>
  02    40 To: "Suzie Q" <suzie@shopping.example.net>
  03    25 Subject: Is dinner ready?
  04    43 Date: Fri, 11 Jul 2003 21:00:37 -0700 (PDT)
  05    40 Comment: This comment has a continuation
  06    51   because this line begins with folding white space
  07    60 Message-ID: <20030712040037.46341@football.example.com>
  08    00
  09    03 Hi.
  10    00
  11    37 We lost the game. Are you hungry yet?
  12    00
  13    04 Joe.
  14    00
  15    00
```

Line 01 is the first line of the email and the first line of the headers.

Lines 05 and 06 constitute the "Comment:" header.

Line 06 is a continuation header line.

Line 07 is the last line of the headers.

Line 08 is the empty line that separates the header from the body.

Line 09 is the first line of the body.

Lines 10, 12, 14, and 15 are empty lines.

Line 13 is the last non-empty line of the email.

Line 15 is the last line of the body and the last line of the email.

Lines 01 to 15 constitute the complete email.

Line 01 is earlier than line 02, and line 02 is later than line 01.

1.10. Requirements Notation

This document occasionally uses terms that appear in capital letters. When the terms "MUST", "SHOULD", "RECOMMENDED", "MUST NOT", "SHOULD NOT", and "MAY" appear capitalized, they are being used to indicate particular requirements of this specification. A discussion of the meanings of these terms appears in [RFC2119].

2. DomainKeys Overview

Under DomainKeys, a domain owner generates one or more private/public key pairs that will be used to sign messages originating from that domain. The domain owner places the public key in his domain namespace (i.e., in a DNS record associated with that domain), and makes the private key available to the outbound email system. When an email is submitted by an authorized user of that domain, the email system uses the private key to digitally sign the email associated with the sending domain. The signature is added as a header to the email, and the message is transferred to its recipients in the usual way.

When a message is received with a DomainKey signature header, the receiving system can verify the signature as follows:

1. Extract the signature and claimed sending domain from the email.

2. Fetch the public key from the claimed sending domain namespace.

3. Use public key to determine whether the signature of the email has been generated with the corresponding private key, and thus whether the email was sent with the authority of the claimed sending domain.

In the event that an email arrives without a signature or when the signature verification fails, the receiving system retrieves the policy of the claimed sending domain to ascertain the preferred disposition of such email.

Armed with this information, the recipient system can apply local policy based on the results of the signature test.

3. DomainKeys Detailed View

This section discusses the specifics of DomainKeys that are needed to create interoperable implementations. This section answers the following questions:

Given an email, how is the sending domain determined?

How is the public key retrieved for a sending domain?

As email transits the email system, it can potentially go through a number of changes. Which parts of the email are included in the signature and how are they protected from such transformations?

How is the signature represented in the email?

If a signature is not present, or a verification fails, how does the recipient determine the policy intent of the sending domain?

Finally, on verifying the authenticity of an email, how is that result conveyed to participating MUAs?

While there are many alternative design choices, most lead to comparable functionality. The overriding selection criteria used to choose among the alternatives are as follows:

- o use deployed technology whenever possible
- o prefer ease of implementation
- o avoid trading risk for excessive flexibility or interoperability
- o include basic flexibility

Adherence to these criteria implies that some existing email implementations will require changes to participate in DomainKeys. Ultimately, some hard choices need to be made regarding which requirements are more important.

3.1. Determining the Sending Address of an Email

The goal of DomainKeys is to give the recipient confidence that the email originated from the claimed sender. As with much of Internet email, agreement over what constitutes the "sender" is no easy matter. Forwarding systems and mailing lists add serious complications to an overtly simple question. From the point of view of the recipient, the authenticity claim should be directed at the domain most visible to the recipient.

In the first instance, the most visible address is clearly the RFC 2822 "From:" address [RFC2822]. Therefore, a conforming email MUST contain a single "From:" header from which an email address with a domain name can be extracted.

A conforming email MAY contain a single RFC 2822 "Sender:" header from which an email address with a domain name can be extracted.

If the email has a valid "From:" and a valid "Sender:" header, then the signer MUST use the sending address in the "Sender:" header.

If the email has a valid "From:" and no "Sender:" header, then the signer MUST use the first sending address in the "From:" header.

In all other cases, a signer MUST NOT sign the email. Implementors should note that an email with a "Sender:" header and no "From:" header MUST NOT be signed.

The domain name in the sending address constitutes the "sending domain".

3.2. Retrieving the Public Key Given the Sending Domain

To avoid namespace conflicts, it is proposed that the DNS namespace "_domainkey." be reserved within the sending domain for storing public keys, e.g., if the sending domain is example.net, then the public keys for that domain are stored in the _domainkey.example.net namespace.

3.2.1. Introducing "selectors"

To support multiple concurrent public keys per sending domain, the DNS namespace is further subdivided with "selectors". Selectors are arbitrary names below the "_domainkey." namespace. A selector value and length MUST be legal in the DNS namespace and in email headers with the additional provision that they cannot contain a semicolon.

Examples of namespaces using selectors are as follows:

```
"coolumbeach._domainkey.example.net"
"sebastopol._domainkey.example.net"
"reykjavik._domainkey.example.net"
"default._domainkey.example.net"
```

and

```
"2005.pao._domainkey.example.net"
"2005.sql._domainkey.example.net"
"2005.rhv._domainkey.example.net"
```

Periods are allowed in selectors and are to be treated as component separators. In the case of DNS queries, that means the period defines subdomain boundaries.

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

Beyond administrative convenience, selectors make it possible to seamlessly replace public keys on a routine basis. If a domain wishes to change from using a public key associated with selector "2005" to a public key associated with selector "2006", it merely makes sure that both public keys are advertised in the DNS concurrently for the transition period during which email may be in transit prior to verification. At the start of the transition period, the outbound email servers are configured to sign with the "2006" private key. At the end of the transition period, the "2005" public key is removed from the DNS.

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

3.2.2. Public Key Signing and Verification Algorithm

The default signature is an RSA signed SHA1 digest of the complete email.

For ease of explanation, the openssl command is used throughout this document to describe the mechanism by which keys and signatures are managed.

One way to generate a 768-bit private key suitable for DomainKeys is to use openssl like this:

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

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

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

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

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

which results in signature data similar to this when represented in Base64 [BASE64] format:

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

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

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

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

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

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

This public key data is placed in the DNS.

With the signature, canonical email contents and public key, a verifying system can test the validity of the signature. The openssl invocation to verify a signature looks like this:

```
 $ openssl dgst -verify rsa.public -sha1 -signature sig.file <input.file
```

3.2.3. Public key Representation in the DNS

There is currently no standard method defined for storing public keys in the DNS. As an interim measure, the public key is stored as a TXT record derived from a Privacy-Enhanced Mail (PEM) format [PEM], that is, as a Base64 representation of a DER encoded key. Here is an example of a 768-bit RSA key in PEM form:

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

To save scarce DNS packet space and aid extensibility, the PEM wrapping MUST be removed and the remaining public key data along with other attributes relevant to DomainKeys functionality are stored as tag=value pairs separated by semicolons, for example, as in the following:

```
brisbane._domainkey IN TXT "g=; k=rsa; p=MHww ... IDAQAB"
```

Verifiers MUST support key sizes of 512, 768, 1024, 1536 and 2048 bits. Signers MUST support at least one of the verifier supported key sizes.

The current valid tags are as follows:

g = granularity of the key. If present with a non-zero length value, this value MUST exactly match the local part of the sending address. This tag is optional.

The intent of this tag is to constrain which sending address can legitimately use this selector. An email with a sending address that does not match the value of this tag constitutes a failed verification.

k = key type (rsa is the default). Signers and verifiers MUST support the 'rsa' key type. This tag is optional.

n = Notes that may be of interest to a human. No interpretation is made by any program. This tag is optional.

p = public key data, encoded as a Base64 string. An empty value means that this public key has been revoked. This tag MUST be present.

t = a set of flags that define boolean attributes. Valid attributes are as follows:

y = testing mode. This domain is testing DomainKeys and unverified email MUST NOT be treated differently from verified email. Recipient systems MAY wish to track testing mode results to assist the sender.

This tag is optional.

(Syntax rules for the tag=value format are discussed in Appendix A.)

Keeping the size of the TXT record to a minimum is important as some implementations of content and caching DNS servers are reported to have problems supporting large TXT records. In the example above, the encoding generates a 182-byte TXT record. That this encoding is less than 512 bytes is of particular significance as it fits within a single UDP response packet. With careful selection of query values, a TXT record can accommodate a 2048 bit key.

For the same size restriction reason, the "n" tag SHOULD be used sparingly. The most likely use of this tag is to convey a reason why a public key might have been revoked. In this case, set the "n" tag to the explanation and remove the public key value from the "p" tag.

3.2.4. Key Sizes

Selecting appropriate key sizes is a trade-off between cost, performance, and risk. This specification does not define either minimum or maximum key sizes -- that decision is a matter for each domain owner.

Factors that should influence this decision include the following:

- the practical constraint that a 2048-bit key is the largest key that fits within a 512-byte DNS UDP response packet
- larger keys impose higher CPU costs to verify and sign email
- keys can be replaced on a regular basis; thus, their lifetime can be relatively short

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

In general, it is expected that most domain owners will use keys that are no larger than 1024 bits.

3.3. Storing the Signature in the Email Header

The signature of the email is stored in the "DomainKey-Signature:" header. This header contains all of the signature and key-fetching data.

When generating the signed email, the "DomainKey-Signature:" header MUST precede the original email headers presented to the signature algorithm.

The "DomainKey-Signature:" header is not included in the signature calculation.

For extensibility, the "DomainKey-Signature:" header contains tag=value pairs separated by semicolons, for example, as in the following:

```
DomainKey-Signature: a=rsa-sha1; s=brisbane; d=example.net;
  q=dns; c=simple
```

The current valid tags are as follows:

a = The algorithm used to generate the signature. The default is "rsa-sha1", an RSA signed SHA1 digest. Signers and verifiers MUST support "rsa-sha1".

b = The signature data, encoded as a Base64 string. This tag MUST be present.

Whitespace is ignored in this value and MUST be removed when reassembling the original signature. This is another way of saying that the signing process can safely insert folding whitespace in this value to conform to line-length limits.

c = Canonicalization algorithm. The method by which the headers and content are prepared for presentation to the signing algorithm. This tag MUST be present. Verifiers MUST support "simple" and "nofws". Signers MUST support at least one of the verifier-supported algorithms.

d = The domain name of the signing domain. This tag MUST be present. In conjunction with the selector tag, this domain forms the basis of the public key query. The value in this tag MUST match the domain of the sending email address or MUST be one of the parent domains of the sending email address. Domain name comparison is case insensitive.

> The matching process for this tag is called subdomain matching, as the sending email address must be the domain or subdomain of the value.

h = A colon-separated list of header field names that identify the headers presented to the signing algorithm. If present, the value MUST contain the complete list of headers in the order presented to the signing algorithm.

If present, this tag MUST include the header that was used to identify the sending domain, i.e., the "From:" or "Sender:" header; thus, this tag can never contain an empty value.

If this tag is not present, all headers subsequent to the signature header are included in the order found in the email.

A verifier MUST support this tag. A signer MAY support this tag. If a signer generates this tag, it MUST include all email headers in the original email, as a verifier MAY remove or render suspicious, lines that are not included in the signature.

In the presence of duplicate headers, a signer may include duplicate entries in the list of headers in this tag. If a header is included in this list, a verifier must include all occurrences of that header, subsequent to the "DomainKey-Signature:" header in the verification.

If a header identified in this list is not found after the "DomainKey-Signature:" header in the verification process, a verifier may "look" for a matching header prior to the "DomainKey-Signature:" header; however, signers should not rely on this as early experience suggests that most verifiers do not try to "look" back before the "DomainKey-Signature:" header.

Whitespace is ignored in this value and header comparisons are case insensitive.

q = The query method used to retrieve the public key. This tag MUST be present. Currently, the only valid value is "dns", which defines the DNS lookup algorithm described in this document. Verifiers and signers MUST support "dns".

s = The selector used to form the query for the public key. This tag MUST be present. In the DNS query type, this value is prepended to the "_domainkey." namespace of the sending domain.

(Syntax rules for the tag=value format are discussed in Appendix A.)

Here is an example of a signature header spread across multiple continuation lines:

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

Extreme care must be taken to ensure that any new tags added to this header are defined and used solely for the purpose of fetching and verifying the signature. Any semantics beyond verification cannot be trusted, as this header is not protected by the signature.

If additional semantics not pertaining directly to signature verification are required, they must only be added as subsequent headers protected by the signature. Semantic additions might include audit information describing the initial submission.

3.4. Preparation of Email for Transit and Signing

The fundamental purpose of a cryptographic signature is to ensure that the signed content matches the contents presented for verification. However, unlike just about every other Internet protocol, the email content is routinely modified as it enters and transits the email system.

Fortunately most of the modifications typically made to email can be predicted and consequently accounted for when signing and verifying.

To maximize the chance of a successful verification, submitted email should be prepared for transport prior to signing, and the data presented to the signing algorithm is canonicalized to exclude the most common and minor changes made to email.

3.4.1. Preparation for Transit

The SMTP protocol defines a number of potential limitations to email transport, particularly pertaining to line lengths and 8-bit content.

While the editor has observed that most modern SMTP implementations accept 8-bit email and long lines, some implementations still do not. Consequently, a DomainKeys implementation SHOULD prepare an email to be suitable for the lowest common denominator of SMTP prior to presenting the email for signing.

3.4.2. Canonicalization for Signing

DomainKeys is initially expected to be deployed at, or close to, the email borders of an organization rather than in MUAs or SUBMISSION servers. In other words, the signing and verifying algorithms normally apply after an email has been packaged, transmogrified, and generally prepared for transmission across the Internet via SMTP and, thus the likelihood of the email being subsequently modified is reduced.

Nonetheless, empirical evidence suggests that some mail servers and relay systems modify email in transit, potentially invalidating a signature.

There are two competing perspectives on such modifications. For most senders, mild modification of email is immaterial to the authentication status of the email. For such senders, a canonicalization algorithm that survives modest in-transit modification is preferred.

For other senders however, any modification of the email - however minor -- results in a desire for the authentication to fail. In other words, such senders do not want a modified email to be seen as being authorized by them. These senders prefer a canonicalization algorithm that does not tolerate in-transit modification of the signed email.

To satisfy both requirements, two canonicalization algorithms are defined. A "simple" algorithm that tolerates almost no modification and a "nofws" algorithm that tolerates common modifications as whitespace replacement and header line rewrapping.

A sender may choose either algorithm when signing an email. A verifier MUST be able to process email using either algorithm.

Either algorithm can be used in conjunction with the "h" tag in the "DomainKey-Signature:" header.

Canonicalization simply prepares the email for the signing or verification algorithm. It does not change the transmitted data in any way.

3.4.2.1. The "simple" Canonicalization Algorithm

- Each line of the email is presented to the signing algorithm in the order it occurs in the complete email, from the first line of the headers to the last line of the body.

- If the "h" tag is used, only those header lines (and their continuation lines if any) added to the "h" tag list are included.

- The "h" tag only constrains header lines. It has no bearing on body lines, which are always included.

- Remove any local line terminator.

- Append CRLF to the resulting line.

- All trailing empty lines are ignored. An empty line is a line of zero length after removal of the local line terminator.

  If the body consists entirely of empty lines, then the header/body line is similarly ignored.

3.4.2.2. The "nofws" Canonicalization Algorithm

The "No Folding Whitespace" algorithm (nofws) is more complicated than the "simple" algorithm for two reasons; folding whitespace is removed from all lines and header continuation lines are unwrapped.

- Each line of the email is presented to the signing algorithm in the order it occurs in the complete email, from the first line of the headers to the last line of the body.

- Header continuation lines are unwrapped so that header lines are processed as a single line.

- If the "h" tag is used, only those header lines (and their continuation lines if any) added to the "h" tag list are included.

- The "h" tag only constrains header lines. It has no bearing on body lines, which are always included.

- For each line in the email, remove all folding whitespace characters. Folding whitespace is defined in RFC 2822 as being the decimal ASCII values 9 (HTAB), 10 (NL), 13 (CR), and 32 (SP).

- Append CRLF to the resulting line.

- Trailing empty lines are ignored. An empty line is a line of zero length after removal of the local line terminator. Note that the test for an empty line occurs after removing all folding whitespace characters.

  If the body consists entirely of empty lines, then the header/body line is similarly ignored.

3.5. The Signing Process

The previous sections defined the various components and mechanisms needed to sign an email. This section brings those together to define the complete process of signing an email.

A signer MUST only sign email from submissions that are authorized to use the sending address.

Once authorization of the submission has been determined, the signing process consists of the following steps:

- identifying the sending domain

- determining if an email should be signed

- selecting a private key and corresponding selector information

- calculating the signature value

- prepending the "DomainKey-Signature:" header

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

3.5.1. Identifying the Sending Domain

The sending domain is determined by finding the email address in the "Sender:" header, or, if the "Sender:" header is not present, the first email address of the "From:" header is used to determine the sending domain.

If the email has an invalid "From:" or an invalid "Sender:" header, it MUST NOT be signed.

If the signer adds the "h" tag to the "DomainKey-Signature:" header, that tag MUST include the header that was used to determine the sending domain.

3.5.2. Determining Whether an Email Should Be Signed

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

A signer MUST NOT sign an email if the email submission is not authorized to use the sending domain.

A signer MUST NOT sign an email that already contains a "DomainKey-Signature:" header unless a "Sender:" header has been added that was not included in the original signature. The most obvious case where this occurs is with mailing lists.

A signer SHOULD NOT remove an existing "DomainKey-Signature:" header.

3.5.3. Selecting a Private Key and Corresponding Selector Information

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

3.5.4. Calculating the Signature Value

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

To avoid possible ambiguity, a signing server may choose to remove any pre-existing "DomainKey-Status:" headers from the email prior to preparation for signing and transmission.

3.5.5. Prepending the "DomainKey-Signature:" Header

The final step in the signing process is that the signer MUST prepend the "DomainKey-Signature:" header prior to continuing with the process of transmitting the email.

3.6. Policy Statement of Sending Domain

While the disposition of inbound email is ultimately a matter for the receiving system, the introduction of authentication in email creates a need for the sender domain to indicate their signing policy and preferred disposition of unsigned email, in particular, whether a domain is participating in DomainKeys, whether it is testing, and whether it signs all outbound email.

The sending domain policy is very simple and is expressed in the _domainkey TXT record in the DNS of the sending domain. For example, in the example.com domain, the record is called _domainkey.example.com.

The contents of this TXT record are stored as tag=value pairs separated by semicolons, for example, as in the following:

```
_domainkey   IN TXT "t=y; o=-; n=notes; r=emailAddress"
```

All tags are optional. The current valid tags are as follows:

n = Notes that may be of interest to a human. No interpretation is made by any program.

o = Outbound Signing policy ("-" means that this domain signs all email; "~" is the default and means that this domain may sign some email with DomainKeys).

r = A reporting email address. If present, this defines the email address where invalid verification results are reported. This tag is primarily intended for early implementers -- the content and frequency of the reports will be defined in a separate document.

t = a set of flags that define boolean attributes. Valid attributes are as follows:

y = testing mode. This domain is testing DomainKeys, and unverified email MUST NOT be treated differently from verified email. Recipient systems MAY wish to track testing mode results to assist the sender).

Note that testing mode cannot be turned off by this tag; thus, policy cannot revert the testing mode setting of a Selector.

This tag is optional.

(Syntax rules for the tag=value format are discussed in Appendix A.)

Recipient systems SHOULD only retrieve this policy TXT record to determine policy when an email fails to verify or does not include a signature. Recipient systems SHOULD not retrieve this policy TXT record for email that successfully verifies. Note that "testing mode" SHOULD also be in the Selector TXT record if the domain owner is running a DomainKeys test.

If the policy TXT record does not exist, recipient systems MUST assume the default values.

There is an important implication when a domain states that it signs all email with the "o=-" setting, namely that the sending domain prefers that the recipient system treat unsigned mail with a great deal of suspicion. Such suspicion could reasonably extend to rejecting such email. A verifying system MAY reject unverified email if a domain policy indicates that it signs all email.

Of course, nothing compels a recipient MTA to abide by the policy of the sender. In fact, during the trial, a sending domain would want to be very certain about setting this policy, as processing by recipient MTAs may be unpredictable. Nonetheless, a domain that states that it signs all email MUST expect that unverified email may be rejected by some receiving MTAs.

3.7. The Verification Process

There is no defined or recommended limit on the lifetime of a selector and corresponding public key; however, it is recommended that verification occur in a timely manner with the most timely place being during acceptance or local delivery by the MTA.

Verifying a signature consists of the following three steps:

- o extract signature information from the headers

- o retrieve the public key based on the signature information

- o check that the signature verifies against the contents

In the event that any of these steps fails, the sending domain policy is ascertained to assist in applying local policy.

3.7.1. Presumption that Headers Are Not Reordered

Indications from deployment of previous versions of this specification suggest that the canonicalization algorithms in conjunction with the "h" tag in the "DomainKey-Signature:" header allows most email to cryptographically survive intact between signing and verifying.

The one assumption that most of the early deployments make is that the headers included in the signature are not reordered prior to verification.

While nothing in this specification precludes a verifier from "looking" for a header that may have been reordered, including being moved to a position prior to the "DomainKey-Signature:" header, such reordered email is unlikely to be successfully verified by most implementations.

A second consequence of this assumption -- particularly in the presence of multiple "DomainKey-Signature:" headers -- is that the first "DomainKey-Signature:" header in the email was the last signature added to the email and thus is the one to be verified.

3.7.2. Verification Should Render a Binary Result

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

For diagnostic purposes, the exact reason why the verification fails SHOULD be recorded; however, in terms of presentation to the end user, the result SHOULD be presented as a simple binary result: either the email is verified or it is not. If the email cannot be verified, then it SHOULD be rendered the same as all unverified email regardless of whether or not it looks like it was signed.

3.7.3. Selecting the Most Appropriate "DomainKey-Signature:" Header

In most cases, a signed email is expected to have just one signature -- that is, one "DomainKey-Signature:" header. However, it is entirely possible that an email can contain multiple signatures. In such cases, a verifier MUST find the most appropriate signature to use and SHOULD ignore all other signatures.

The process of finding the most appropriate signature consists of the following "best match" rules. The rules are to be evaluated in order.

1. Selecting the sending domain

   If the email contains a "Sender:" header, the sending domain is extracted from the "Sender:" address. If this extraction fails, the email SHALL fail verification.

   If no "Sender:" header is present, the sending domain is extracted from the first address of the "From:" header. If this extraction fails, the email SHALL fail verification.

2. Domain matching

   A signature can only match if the sending domain matches the "d" tag domain -- according to the "d" tag subdomain matching rules.

3. "h" tag matching

   If the signature contains the "h" tag list of headers, that list must include the header used to extract the sending domain in rule 1, above.

4. Most secure signing algorithm

   While it is not yet the case, in the event that additional algorithms are added to this specification, a verifier MUST use the signature that contains the most secure algorithm as defined by the future specification. For current implementations, that means verifiers MUST ignore signatures that are coded with an unrecognized signing algorithm.

5. Earlier signatures are preferred

   If multiple signatures are equal as far as these rules apply, the signature from the earlier header MUST be used in preference to later signature headers.

Implementors MUST meticulously validate the format and values in the "DomainKey-Signature:" header; any inconsistency or unexpected values MUST result in ignoring that header. Being "liberal in what you accept" is definitely a bad strategy in this security context.

In all cases, if a verification fails, the "DomainKey-Status:" header SHOULD be generated and include a message to help explain the reason for failure.

3.7.4. Retrieve the Public Key Based on the Signature Information

The public key is needed to complete the verification process. The process of retrieving the public key depends on the query type as defined by the "q" tag in the "DomainKey-Signature:" header line. Obviously, a public key should only be retrieved if the process of extracting the signature information is completely successful.

Currently, the only valid query type is "dns". The public key retrieval process for this type is as follows:

1. Using the selector name defined by the "s" tag, the "_domainkey" namespace and the domain name defined by the "d" tag, construct and issue the DNS TXT record query string.

   For example, if s=brisbane and d=example.net, the query string is "brisbane._domainkey.example.net".

2. If the query for the public key fails to respond, the verifier SHOULD defer acceptance of this email (normally this will be achieved with a 4XX SMTP response code).

3. If the query for the public key fails because the corresponding data does not exist, the verifier MUST treat the email as unverified.

4. If the result returned from the query does not adhere to the format defined in this specification, the verifier MUST treat the email as unverified.

5. If the public key data is not suitable for use with the algorithm type defined by the "a" tag in the "DomainKey-Signature:" header, the verifier MUST treat the email as unverified.

Implementors MUST meticulously validate the format and values returned by the public key query. Any inconsistency or unexpected values MUST result in an unverified email. Being "liberal in what you accept" is definitely a bad strategy in this security context.

Latency critical implementations may wish to initiate the public key query in parallel with calculating the SHA-1 hash, as the public key is not needed until the final RSA is calculated.

3.7.5. Verify the Signature

Armed with the signature information from the "DomainKey-Signature:" header and the public key information returned by the query, the signature of the email can now be verified.

The canonicalization algorithm defined by the "c" tag in the "DomainKey-Signature:" header defines how the data is prepared for the verification algorithm, and the "a" tag in the same header defines which verification algorithm to use.

3.7.6. Retrieving Sending Domain Policy

In the event that an email fails to verify, the policy of the sending domain MUST be consulted. For now, that means consulting the _domainkey TXT record in the DNS of the domain in the sending domain as defined in Section 3.5.1. For example, if example.net is the sending domain the TXT query is:

```
_domainkey.example.net
```

A verifier SHOULD consider the sending domain policy statement and act accordingly. The range of possibilities is up to the receiver, but it MAY include rejecting the email.

3.7.7. Applying Local Policy

After all verification processes are complete, the recipient system has authentication information that can help it decide what to do with the email.

It is beyond the scope of this specification to describe what actions a recipient system should take, but an authenticated email presents an opportunity to a receiving system that unauthenticated email cannot. Specifically, an authenticated email creates a predictable identifier by which other decisions can reliably be managed, such as trust and reputation.

Conversely, unauthenticated email lacks a reliable identifier that can be used to assign trust and reputation. It is not unreasonable to treat unauthenticated email as lacking any trust and having no positive reputation.

3.8. Conveying Verification Results to MUAs

Apart from the application of automated policy, the result of a signature verification should be conveyed to the user reading the email.

Most email clients can be configured to recognize specific headers and apply simple rules, e.g., filing into a particular folder. Since DomainKey signatures are expected to be initially verified at the border MTA, the results of the verification need to be conveyed to the email client. This is done with the "DomainKey-Status:" header line prepended to the email.

The "DomainKey-Status:" header starts with a string that indicate the result of the verification. Valid values are as follows:

"good"	- the signature was verified at the time of testing
"bad"	- the signature failed the verification
"no key"	- the public key query failed as the key does not exist
"revoked"	- the public key query failed as the key has been revoked
"no signature"	- this email has no "DomainKey-Signature:" header
"bad format"	- the signature or the public key contains unexpected data
"non-participant"	- this sending domain has indicated that it does not participate in DomainKeys

Verifiers may append additional data that expands on the reason for the particular status value.

A client SHOULD just look for "good" and assume that all other values imply that the verification could not be performed for some reason. Policy action as a consequence of this header is entirely a local matter.

Here are some examples:

```
DomainKey-Status: good
DomainKey-Status: bad format
```

Although it is expected that MTAs will be DomainKey aware before MUAs, it is nonetheless possible that a DomainKey-aware MUA can be fooled by a spoofed "DomainKey-Status:" header that passes through a non-DomainKey-aware MTA.

If this is perceived to be a serious problem, then it may make sense to preclude the "good" value and only have values that effectively demote the email as far as the UA is concerned. That way successful spoofing attempts can only serve to demote themselves.

4. Example of Use

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

4.1. The User Composes an Email

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

Hi.

We lost the game. Are you hungry yet?

Joe.
```

4.2. The Email Is Signed

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

```
DomainKey-Signature: a=rsa-sha1; s=brisbane; d=football.example.com;
  c=simple; q=dns;
  b=dzdVyOfAKCdLXdJOc9G2q8LoXSlEniSbav+yuU4zGeeruD00lszZ
    VoG4ZHRNiYzR;
Received: from dsl-10.2.3.4.football.example.com  [10.2.3.4]
     by submitserver.football.example.com with SUBMISSION;
     Fri, 11 Jul 2003 21:01:54 -0700 (PDT)
From: "Joe SixPack" <joe@football.example.com>
To: "Suzie Q" <suzie@shopping.example.net>
Subject: Is dinner ready?
Date: Fri, 11 Jul 2003 21:00:37 -0700 (PDT)
Message-ID: <20030712040037.46341.5F8J@football.example.com>

Hi.

We lost the game. Are you hungry yet?

Joe.
```

The signing email server requires access to the private key associated with the "brisbane" selector to generate this signature. Distribution and management of private keys are outside the scope of this document.

4.3. The Email Signature Is Verified

The signature is normally verified by an inbound SMTP server or possibly the final delivery agent. However, intervening MTAs can also perform this verification if they choose to do so.

The verification process uses the domain "football.example.com" extracted from the "From:" header and the selector "brisbane" from the "DomainKey-Signature:" header to form the DNS TXT query for:

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

Since there is no "h" tag in the "DomainKey-Signature:" header, signature verification starts with the line following the "DomainKey-Signature:" line. The email is canonically prepared for verifying with the "simple" method.

The result of the query and subsequent verification of the signature is stored in the "DomainKey-Status:" header line. After successful verification, the email looks like this:

```
DomainKey-Status: good
 from=joe@football.example.com; domainkeys=pass
Received: from mout23.brisbane.football.example.com (192.168.1.1)
          by shopping.example.net with SMTP;
          Fri, 11 Jul 2003 21:01:59 -0700 (PDT)
DomainKey-Signature: a=rsa-sha1; s=brisbane; d=football.example.com;
 c=simple; q=dns;
 b=dzdVyOfAKCdLXdJOc9G2q8LoXSlEniSbav+yuU4zGeeruD00lszZ
   VoG4ZHRNiYzR;
Received: from dsl-10.2.3.4.network.example.com  [10.2.3.4]
     by submitserver.example.com with SUBMISSION;
     Fri, 11 Jul 2003 21:01:54 -0700 (PDT)
From: "Joe SixPack" <joe@football.example.com>
To: "Suzie Q" <suzie@shopping.example.net>
Subject: Is dinner ready?
Date: Fri, 11 Jul 2003 21:00:37 -0700 (PDT)
Message-ID: <20030712040037.46341.5F8J@football.example.com>

Hi.

We lost the game. Are you hungry yet?

Joe.
```

5. Association with a Certificate Authority

A fundamental aspect of DomainKeys is that public keys are generated and advertised by each domain at no additional cost. This accessibility markedly differs from traditional Public Key Infrastructures where there is typically a Certificate Authority (CA) who validates an applicant and issues a signed certificate -- containing their public key -- often for a recurring fee.

While CAs do impose costs, they also have the potential to provide additional value as part of their certification process. Consider financial institutions, public utilities, law enforcement agencies, and the like. In many cases, such entities justifiably need to discriminate themselves above and beyond the authentication that DomainKeys offers.

Creating a link between DomainKeys and CA-issued certificates has the potential to access additional authentication mechanisms that are more authoritative than domain-owner-issued authentication. It is well beyond the scope of this specification to describe such authorities apart from defining how the linkage could be achieved with the "DomainKey-X509:" header.

5.1. The "DomainKey-X509:" Header

The "DomainKey-X509:" header provides a link between the public key used to sign the email and the certificate issued by a CA.

The exact content, syntax, and semantics of this header are yet to be resolved. One possibility is that this header contains an encoding of the certificate issued by a CA. Another possibility is that this header contains a URL that points to a certificate issued by a CA.

In either case, this header can only be consulted if the signature verifies and MUST be part of the content signed by the corresponding "DomainKey-Signature:" header. Furthermore, it is likely that MUAs rather than MTAs will confirm that the link to the CA-issued certificate is valid. In part, this is because many MUAs already have built-in capabilities as a consequence of Secure/Multipurpose Internet Mail Extensions (S/MIME) [SMIME] and Secure Socket Layer (SSL) [SSL] support.

The proof of linkage is made by testing that the public key in the certificate matches the public key used to sign the email.

An example of an email containing the "DomainKey-X509:" header is:

```
   DomainKey-Signature: a=rsa-sha1; s=statements;
     d=largebank.example.com; c=simple; q=dns;
     b=dzdVyOfAKCdLXdJOc9G2q8LoXSlEniSbav+yuU4zGeeruD00lszZ
       VoG4ZHRNiYzR;
   DomainKey-X509: https://ca.example.net/largebank.example.com
   From: "Large Bank" <statements@largebank.example.com>
   To: "Suzie Q" <suzie@shopping.example.net>
   Subject: Statement for Account: 1234-5678
   ...
```

The format of the retrieved value from the URL is not yet defined, nor is the determination of valid CAs.

The whole matter of linkage to CA-issued certificates is one aspect of DomainKeys that needs to be resolved with relevant CA's and certificate-issuing entities. The primary point is that a link is possible to a higher authority.

6. Topics for Discussion

6.1. The Benefits of Selectors

Selectors are at the heart of the flexibility of DomainKeys. A domain administrator is free to use a single DomainKey for all outbound mail. Alternatively, the domain administrator may use many DomainKeys differentiated by selector and assign each key to different servers.

For example, a large outbound email farm might have a unique DomainKey for each server, and thus their DNS will advertise potentially hundreds of keys via their unique selectors.

Another example is a corporate email administrator who might generate a separate DomainKey for each regional office email server.

In essence, selectors allow a domain owner to distribute authority to send on behalf of that domain. Combined with the ability to revoke by removal or Time to Live (TTL) expiration, a domain owner has coarse-grained control over the duration of the distributed authority.

Selectors are particularly useful for domain owners who want to contract a third-party mailing system to send a particular set of mail. The domain owner can generate a special key pair and selector just for this mail-out. The domain owner has to provide the private key and selector to the third party for the life of the mail-out.

However, as soon as the mail-out is completely delivered, the domain owner can revoke the public key by the simple expedient of removing the entry from the DNS.

6.2. Canonicalization of Email

It is an unfortunate fact that some email software routinely (and often unnecessarily) transforms email as it transits through the network. Such transformations conflict with the fundamental purpose of cryptographic signatures - to detect modifications.

While two canonicalization algorithms are defined in this specification, the primary goal of "nofws" is to provide a transition path to "simple". With a mixture of "simple" and "nofws" email, a receiver can determine which systems are modifying email in ways that cause the signature to fail and thus provide feedback to the modifying system.

6.3. Mailing Lists

Integrating existing Mailing List Managers (MLMs) into the DomainKeys authentication system is a complicated area, as the behavior of MLMs is highly variable. Essentially, there are two types of MLMs under consideration: those that modify email to such an extent that verification of the original content is not possible, and those that make minimal or no modifications to an email.

MLMs that modify email in a way that causes verification to fail MUST prepend a "Sender:" header and SHOULD prepend a "List-ID:" header prior to signing for distribution to list recipients.

A participating SUBMISSION server can deduce the need to re-sign such an email by the presence of a "Sender:" or "List-ID:" header from an authorized submission.

MLMs that do not modify email in a way that causes verification to fail MAY perform the same actions as a modifying MLM.

6.4. Roving Users

One scenario that presents a particular problem with any form of email authentication, including DomainKeys, is the roving user: a user who is obliged to use a third-party SUBMISSION service when unable to connect to the user’s own SUBMISSION service. The classic example cited is a traveling salesperson being redirected to a hotel email server to send email.

As far as DomainKeys is concerned, email of this nature clearly originates from an email server that does not have authority to send on behalf of the domain of the salesperson and is therefore indistinguishable from a forgery. While DomainKeys does not prescribe any specific action for such email, it is likely that over time, such email will be treated as second-class email.

The typical solution offered to roving users is to submit email via an authorized server for their domain -- perhaps via a Virtual Private Network (VPN) or a web interface or even SMTP AUTH back to a SUBMISSION server.

While these are perfectly acceptable solutions for many, they are not necessarily solutions that are available or possible for all such users.

One possible way to address the needs of this contingent of potentially disenfranchised users is for the domain to issue per-user DomainKeys. Per-user DomainKeys are identified by a non-empty "g" tag value in the corresponding DNS record.

7. Security Considerations

7.1. DNS

DomainKeys is primarily a security mechanism. Its core purpose is to make claims about email authentication in a credible way. However, DomainKeys, like virtually all Internet applications, relies on the DNS, which has well-known security flaws [RFC3833].

7.1.1. The DNS Is Not Currently Secure

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

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

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

Finally, DomainKeys is only intended as a "sufficient" method of proving authenticity. It is not intended to provide strong cryptographic proof about authorship or contents. Other technologies such as GnuPG and S/MIME address those requirements.

7.1.2. DomainKeys Creates Additional DNS Load

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

7.2. Key Management

All public key systems require management of key pairs. Private keys in particular need to be securely distributed to each signing mail server and protected on those servers. For those familiar with SSL, the key management issues are similar to those of managing SSL certificates. Poor key management may result in unauthorized access to private keys, which in essence gives unauthorized access to your identity.

7.3. Implementation Risks

It is well recognized in cryptographic circles that many security failures are caused by poor implementations rather than poor algorithms. For example, early SSL implementations were vulnerable because the implementors used predictable "random numbers".

While some MTA software already supports various cryptographic techniques, such as TLS, many do not. This proposal introduces cryptographic requirements into MTA software that implies a much higher duty of care to manage the increased risk.

There are numerous articles, books, courses, and consultants that help programming security applications. Potential implementors are strongly encouraged to avail themselves of all possible resources to ensure secure implementations.

7.4. Privacy Assumptions with Forwarding Addresses

Some people believe that they can achieve anonymity by using an email forwarding service. While this has never been particularly true, as bounces, over-quota messages, vacation messages, and web bugs all conspire to expose IP addresses and domain names associated with the delivery path, the DNS queries that are required to verify DomainKeys signature can provide additional information to the sender.

In particular, as mail is forwarded through the mail network, the DNS queries for the selector will typically identify the DNS cache used by the forwarding and delivery MTAs.

7.5. Cryptographic Processing Is Computationally Intensive

Verifying a signature is computationally significant. Early indications are that a typical mail server can expect to increase CPU demands by 8-15 percent. While this increased demand is modest compared to other common mail processing costs -- such as Bayesian filtering -- any increase in resource requirements can make a denial-of-service attack more effective against a mail system.

A constraining factor of such attacks is that the net computational cost of verifying is bounded by the maximum key size allowed by this specification and is essentially linear to the rate at which mail is accepted by the verifying system. Consequently, the additional computational cost may augment a denial-of-service attack, but it does not add a non-linear component to such attacks.

8. The Trial

The DomainKeys protocol was deployed as a trial to better understand the implications of deploying wide-scale cryptographic email authentication.

Open Source implementations were made available at various places, particularly Source Forge [SOURCEFORGE], which includes links to numerous implementations, both Open Source and commercial.

8.1. Goals

The primary goals of the trial were to:

- o understand the operational implications of running a DNS-based public key system for email

- o measure the effectiveness of the canonicalization algorithms

- o experiment with possible per-user key deployment models

- o fully define the semantics of the "DomainKey-X509:" header

8.2. Results of Trial

The DomainKeys trial ran for approximately 2 years, in which time numerous large ISPs and many thousands of smaller domains participated in signing or verifying with DomainKeys. The low order numbers are that at least one billion DomainKey signed emails transit the Internet each day between some 12,000 participating domains.

The operational and development experience of that trial was applied to DKIM.

9. Note to Implementors Regarding TXT Records

The DNS is very flexible in that it is possible to have multiple TXT records for a single name and for those TXT records to contain multiple strings.

In all cases, implementors of DomainKeys should expect a single TXT record for any particular name. If multiple TXT records are returned, the implementation is free to pick any single TXT record as the authoritative data. In other words, if a name server returns different TXT records for the same name, it can expect unpredictable results.

Within a single TXT record, implementors should concatenate multiple strings in the order presented and ignore string boundaries. Note that a number of popular DNS command-line tools render multiple strings as separately quoted strings, which can be misleading to a novice implementor.

10. References

10.1. Normative References

[BASE64] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

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

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

10.2. Informative References

[DKIM] Allman, E., Callas, J., Delany, M., Libbey, M., Fenton, J., and M. Thomas, "DomainKeys Identified Mail (DKIM) Signatures", RFC 4871, May 2007.

[DNSSEC] http://www.ietf.org/html.charters/dnsext-charter.html

[OPENSSL] http://www.openssl.org

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

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

[SMIME] Ramsdell, B., Ed., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", RFC 3851, July 2004.

[SOURCEFORGE] http://domainkeys.sourceforge.net

[SSL] http://wp.netscape.com/security/techbriefs/ssl.html

Appendix A - Syntax Rules for the Tag=Value Format

A simple tag=value syntax is used to encode data in the response values for DNS queries as well as headers embedded in emails. This section summarized the syntactic rules for this encoding:

- A tag=value pair consists of three tokens, a "tag", the "=" character, and the "value"
- A tag MUST be one character long and MUST be a lowercase alphabetic character
- Duplicate tags are not allowed
- A value MUST only consist of characters that are valid in RFC 2822 headers and DNS TXT records and are within the ASCII range of characters from SPACE (0x20) to TILDE (0x7E) inclusive. Values MUST NOT contain a semicolon but they may contain "=" characters.
- A tag=value pair MUST be terminated by a semicolon or the end of the data
- Values MUST be processed as case sensitive unless the specific tag description of semantics imply case insensitivity.
- Values MAY be zero bytes long
- Whitespace MAY surround any of the tokens; however, whitespace within a value MUST be retained unless explicitly excluded by the specific tag description. Currently, the only tags that specifically ignore embedded whitespace are the "b" and "h" tags in the "DomainKey-Signature:" header.
- Tag=value pairs that represent the default value MAY be included to aid legibility.
- Unrecognized tags MUST be ignored

Acknowledgments

The editor wishes to thank Russ Allbery, Eric Allman, Edwin Aoki, Claus Asmann, Steve Atkins, Jon Callas, Dave Crocker, Michael Cudahy, Jutta Degener, Timothy Der, Jim Fenton, Duncan Findlay, Phillip Hallam-Baker, Murray S. Kucherawy, John Levine, Miles Libbey, David Margrave, Justin Mason, David Mayne, Russell Nelson, Juan Altmayer Pizzorno, Blake Ramsdell, Scott Renfro, the Spamhaus.org team, Malte S. Stretz, Robert Sanders, Bradley Taylor, and Rand Wacker for their valuable suggestions and constructive criticism.

Author’s Address

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

EMail: markd+domainkeys@yahoo-inc.com

Acknowledgement

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