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

Status of this Memo

This document may not be modified, and derivative works of it may not be created. This document may only be posted in an Internet-Draft.

By submitting this Internet-Draft, I certify that any applicable patent or other IPR claims of which I am aware have been disclosed, and any of which I become aware will be disclosed, in accordance with RFC 3668.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

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 the base framework of digitally signing email on a per-domain basis. Subsequent documents leverage this base framework to prove and validate email delivery paths as well as extend signing to facilitate per-user authentication.

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".

While this document presents technical details, it is not yet intended as a definitive or final specification, rather, the intent is to define a framework and sufficient technical detail to promote experimental deployment with a view to evolving into a comprehensive authentication standard for email.

Table of Contents

1. Introduction
   1.1 Lack of authentication is damaging Internet email
   1.2 Digitally signed email creates credible domain authentication
   1.3 Public-keys in the DNS
   1.4 Initial deployment is likely at the border MTA
   1.5 Conveying verification results to UAs
   1.6 Technical minutiae is not completely covered
   1.7 Motivation
   1.8 Benefits of DomainKeys
2. DomainKeys overview

3. DomainKeys detailed view
   3.1 Determining the sending domain of an email
   3.2 Retrieving the public-key given the "sending domain"
      3.2.1 Introducing "selectors"
      3.2.2 Public-key signing and verification algorithm
      3.2.3 Public-key representation in the DNS
   3.3 Canonical preparation of email for signing
   3.4 Storing the signature in the email header
   3.5 The signing process
      3.5.1 Identifying the sending domain
      3.5.2 Determining if an email should be signed
      3.5.3 Selecting a private-key and corresponding selector information
      3.5.4 Calculating the signature value
      3.5.5 Pre-pending the "DomainKey-Signature:" header
   3.6 Policy statement of sending domain
      3.6.1 Domain policy is nascent
      3.6.2 Interim sending domain policy
   3.7 The verification process
      3.7.1 Verification should render a binary result
      3.7.2 Extracting signature information from the headers
      3.7.3 Retrieve the public-key based on the signature information
      3.7.4 Verify the signature
      3.7.5 Retrieving sending domain policy
      3.7.6 Applying local policy
   3.8 Conveying verification results to UAs

4. Example of use
   4.1 The user composes an email
   4.2 The email is signed
   4.3 The email signature is verified

5. Association with a Certificate Authority
   5.1 The "DomainKey-X509:" header

6. Topics for discussion
   6.1 The benefits of selectors
   6.2 Canonicalization of email
   6.3 Mailing lists
   6.4 Roving users
   6.5 Envelope audit

7. Security Considerations
   7.1 DNS
      7.1.1 The DNS is not currently secure
      7.1.2 DomainKeys creates additional DNS load
   7.2 Key Management
   7.3 Implementation risks
   7.4 Privacy assumptions with forwarding addresses

8. The trial
   8.1 Goals
   8.2 Constraints on participation

9. Notes to Implementors
   9.1 TXT records

10. References
    10.1 Normative References
    10.2 Informative References

11. Acknowledgments

Appendix A - Syntax rules for the tag=value format
Appendix B - Sample "simple" canonicalization program: dk_simple_canonical
Appendix C - Sample signing shell script: dk_simple_sign
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.

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:

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

This base definition of DomainKeys is intended to only 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 initially 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 an interim 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 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 UA.

1.5 Conveying verification results to UAs

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 UA in some way—typically via a header line.

As yet, there is no standard for communicating authentication results to UAs. To complicate matters, there are currently a number of draft proposals that attempt to define UA notification mechanisms.

It makes a great deal of sense to consolidate the various notification mechanisms, and this specification will adopt a unified standard as soon as one becomes available.

However, until a unified standard exists, this specification proposes an interim header to communicate the results of the authentication.

1.6 Technical minutiae is not completely covered

The intent of this draft is to communicate the fundamental characteristics of DomainKeys and come close to an implementable specification. However, exacting detail is avoided in preference to conveying the main intent. Furthermore, where convenient, this document implies specifications based on the implementation of readily available software, in particular the openssl command [OPENSSL].

In part, this strategy is due to laziness, in part to avoid clutter, and in part by tacit acknowledgment that many of these details are best refined by further discussion and after trials of early implementations within the email community.

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 which:

- is transparent and compatible with the existing email infrastructure
- can be implemented independently of clients in order to potentially reduce deployment time
- does not require the use of a central certificate authority which might impose fees for certificates or introduce delays to deployment
- does not require a "flag" day.

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 can make stronger credibility statements than it can.

1.8 Benefits of DomainKeys

As the reader will discover, DomainKeys is nothing more than 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.

2. DomainKeys overview

Under DomainKeys, a domain owner generates one or more private/public key-pairs that will be used to secure 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 UAs?

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

- use deployed technology whenever possible
- prefer ease of implementation
- avoid trading risk for excessive flexibility or interoperability
- 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 domain 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. Clearly that is the RFC2822 From: address [RFC2822]. Therefore, a conforming DomainKey email must have a single valid From: header containing a domain name in the email address. This "sending domain" is used to formulate the query needed to retrieve the public-key.

### 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, single-level names below the ".domainkey." namespace. They can be any string value that is legal in both the DNS and in email headers (obviously ";" and perhaps "." should be excluded characters).
Examples of namespace using selectors are:

"sanfrancisco._domainkey.example.net"
"coolumbeach._domainkey.example.net"
"reykjavik._domainkey.example.net"
"default._domainkey.example.net"

and

"january2004._domainkey.example.net"
"february2004._domainkey.example.net"
"march2004._domainkey.example.net"

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

Beyond administrative convenience, selectors make it possible to seamlessly replace public-keys on a routine basis. If a domain wishes to change from using a public-key associated with selector "january" to a public-key associated with selector "february", 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 "february" private-key. At the end of the transition period, the "january" 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. E.g., if per-user keys are issued, the domain owner will need to make the decision as to whether to make this selector associated directly with the user name, or make it some unassociated random value, such as the fingerprint of the public-key.

3.2.2 Public-key signing and verification algorithm

The default signature is an RSA signed SHA1 digest of the email headers and content. Other candidate algorithms could include GnuPG [GPG] variants. (Suggestions for the mechanisms needed to include GnuPG and other freely available algorithms should be directed to the feedback address at the end of this document.)

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 384 bit private-key suitable for DomainKeys, is to use openssl like this:

$ openssl genrsa -out rsa.private 384

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

-----BEGIN RSA PRIVATE KEY-----
MIHyAgEAAjEAmS5epyLJyvJFXIIvxeAcz6j3B9KwtkXWp1NbeRgFmuE5Y6IDNwYKdwxWBi08gfaxBAgMBAAECMC7ONDewy1YRylrSNT3EiJFT/DYmEnW89Yg6xZdvwsgPfdxK9OGA5K22fK7eAQI2AMh+ggAq9GOGV/0yQkufdyyxi6LIdyhQI2AMoFq8rh/OJMdvuBvdKIMZPqHrCorxfWFjOJYvphCh1RVE/X16o9bxCpYMsDawI/FOOV
Ahqyh2ClGeyQiJBvj1YBDZ6suYm/A5VUCQRCOC8UTE0kFKFfE0k/0FCn8KKMTsQqIRC6Ig=
-----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 [MIME] format:

```
MrNZCpqRVgtLYdnvFYtP3jQAqIy3brRNGiwtZUaIdDzh68nx5qtW9p24kc5074m
```

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-----
MEwwDQYJKoZIhvcNAQEBBQADOwAwOAIxAJkhKcmScryTlyCL8XgHM+o9wfsSsLS18
KZTW3kYBTlHOwOiAzcGncF37dJQYHzsQQIDAQAB
-----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 signature. The openssl invocation to
verify a signature looks like this:

```
openssl dgst -verify rsa.public -sha1 -signature signature.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 PEM format [PEM], that is, as a Base64
representation of a DER encoded key. Here is an example of a 384 bit
RSA key in PEM form:

```
-----BEGIN PUBLIC KEY-----
MEwwDQYJKoZIhvcNAQEBBQADOwAwOAIxAJkhKcmScryTlyCL8XgHM+o9wfsSsLS18
KZTW3kYBTlHOwOiAzcGncF37dJQYHzsQQIDAQAB
-----END PUBLIC KEY-----
```

To save scarce DNS packet space and aid extensibility, the PEM
wrapping generated by openssl is removed and the remaining public-key
data along with other attributes relevant to DomainKeys functionality
are stored as tag=value pairs separated by semi-colons, e.g.:

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

The initial valid tags are:

- **g** - granularity of the key (the default of ’’ = all domain, which
  means that any left-hand-side of the @ is valid with this
  DomainKey)

  (Though not yet defined, one possible interpretation for
  non-empty values is that they could represent a Base64 SHA1
  fingerprint of the email address used to identify the sending
  domain. This, though, does not handle the notion of tagged
  addresses as well as one would like.)

- **k** - key type (rsa = the default)
n – Notes that may be of interest to a human. No interpretation is made by any program.

p – public-key data, encoded as a Base64 string. An empty value means that this public-key has been revoked.

t – testing mode (‘y’ means that this domain is testing DomainKeys and unverified email should not be treated differently from verified email. Recipient systems may wish to track testing mode results to assist the sender.)

(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 this example, the encoding creates 119 bytes of content for the TXT record. That this encoding is less than 128 bytes is of particular significance to some DNS implementations that do not handle large TXT records correctly.

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.

3.3 Canonical preparation of email for signing

DomainKeys is initially expected to be deployed at, or close to, the email borders of an organization rather than in UAs 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.

In this light, and in the interest of vastly simplifying trial implementations, it is proposed that the default preparation of signing material derived from the email include all headers and contents and that a lightweight canonical process be used to prepare these headers and content for submission to the signing and verifying algorithms.

The initial canonicalization method is designated "simple" and processes email as follows:

- Each line of the email is presented to the signing algorithm in the order it occurs in the email. Line one first, line two, second, etc.

- For each line in the email, any local line terminator is removed.

- Regardless of the presence or absence of a local line terminator, the signing algorithm is to calculate the signature over the line of data as if the line is terminated with a CRLF.

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

This canonicalization simply prepares the email for the signing algorithm. It does not change the transmitted data in any way.

Appendix B contains a sample program that demonstrates this "simple" canonical processing.

3.4 Storing the signature in the email header
The signature of the email is stored as a header line in the transmitted email. It is proposed that a single new header be introduced, called "DomainKey-Signature:" that contains all of the signature-related data.

When transmitting the signed email, the "DomainKey-Signature:" header line immediately precedes the original headers and content presented to the signature algorithm.

For extensibility, the "DomainKey-Signature:" header contains tag=value pairs separated by semi-colons, e.g.:

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

The initial valid tags are:

- **a** - The algorithm used to generate the signature. The default is "rsa-sha1", an RSA signed SHA1 digest.

- **b** - The signature data, encoded as a Base64 string.
  
  Whitespace is ignored in this value and must be removed when re-assembling the original signature. This is another way of saying that the signing process can safely insert 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. Currently the only valid value is "simple" which defines the algorithm described in this document. More sophisticated algorithms may evolve as part of the trials. There is no default value for this tag.

- **d** - The domain name of the signing domain. At this stage, the domain name must match the domain name found in the From: header. This tag is largely here to allow extensions that allow intermediate domain involvement. There is no default value for this tag.

- **q** - The query method used to retrieve the public-key. Currently the only valid value is "dns" which defines the DNS lookup algorithm described in this document. There is no default value for this tag.

- **s** - The selector used to form the query for the public-key. In the DNS query type, this value is prepended to the "_domainkey." namespace of the sending domain. There is no default value for this tag.

(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=dzdVyOfAKCdlkX0c9G2q8L0oS1EniSbav+yU4zGeeruDO01lszZVoG4ZHR1YzR;
```

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.
3.5 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.

>From the perspective of a signing server, signing an email with DomainKeys 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. That policy decision could vary depending on whether the email already appears to be signed, whether it claims to be from a domain which the local system can sign for, or a combination of those factors. In many cases, it is expected that the local policy will be unchanged by the introduction of DomainKeys.

Since the verification process is fail-negative - that is, a failed verification implies negative disposition of that email - a particularly simplistic signing server could sign all email regardless of whether it should or not, and let the verifying servers determine whether the signature validly applies to the From: domain. Appendix C contains a sample shell script that demonstrates a simple signing server.

3.5.1 Identifying the sending domain

The sending domain is determined by finding the first email address in the first From: header of the email. If that first email address does not contain a domain for which the signing system is responsible, then the email cannot be signed. In this context, "first" means the header closest to the first line of the email.

3.5.2 Determining if an email should be signed

The signing server 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.

The signing server should not normally attempt to sign an email that already contains a "DomainKey-Signature:" header. The presumption is that the signature was generated by another signing server for that domain. This decision is ultimately a local policy matter. An alternative local policy could be to remove such headers and re-sign if the email is from a domain that the signing server can sign for.

3.5.3 Selecting a private-key and corresponding selector information

This specification does not define the basis by which a signing server 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 signing server must use one of the defined canonicalization processed to present the email to the signing algorithm. At this stage that means preparation using the "simple" canonicalization process. This canonicalization process 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 Pre-pending the "DomainKey-Signature:" header

The final step in the signing process is to pre-pend the "DomainKey-Signature:" header and continue 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 they are testing and whether it signs all outbound email.

3.6.1 Domain policy is nascent

As yet there is no standard for a domain to indicate sending policy to recipient MTAs. Indeed the whole notion of sender policy is relatively new to email. To complicate matters, there are currently a number of incompatible draft proposals that attempt to define comprehensive sender policy mechanisms.

Fortunately, the IETF MARID Working Group [MARID] has recently decided to develop a standard policy language and retrieval mechanism that is likely to be flexible enough to cover all the policy needs of DomainKeys.

However, until a unified standard exists, this specification proposes a minimalist and interim sender policy mechanism. The sole purpose of this policy is to define the level of participation in DomainKeys and consequently how unverified email should be treated relative to verified email from that domain.

To re-iterate this last point, we strongly take the view that sending domain policy will ultimately encompass much more than just the dispositional needs of DomainKeys. To that end, implementors are encouraged to completely separate their DomainKeys verification process from the determination and application of sender domain policy.

3.6.2 Interim sending domain policy

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

The contents of this TXT record are stored as tag=value pairs separated by semi-colons, e.g.:

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

The initial valid tags are:
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)

t = testing mode ('y' means that this domain is testing DomainKeys so unsigned and unverifiable email should 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.

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

Recipient systems should only retrieve this policy TXT record to determine when an email fails to verify. Recipient systems should not retrieve this 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.

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 should 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:

- Extract the signature information from the headers
- Retrieve the public-key based on the signature information
- 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 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 over-zealous 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 it looks like it was signed or not.

3.7.2 Extracting signature information from the headers

The signature information consists of data from the "DomainKey-Signature:" line and identification of the claimed sending domain from the From: header.

The process of extracting the signature information consists of:

1. Scanning the email for the last occurrence of the "DomainKey-Signature:" header. If found, verify that the contents conform to this specification and extract the values for subsequent use. In this context, "last" means the header line farthest from the first line of the email.

   If a From: header is found prior to the last "DomainKey-Signature:" header, treat the email as unverified.

   Only lines subsequent to the "DomainKey-Signature:" are included in the signature calculation, thus any pre-pended lines are ignored as far as verification data is concerned.

   If multiple "DomainKey-Signature:" headers are found, all but the last one should be ignored.

2. Scanning the remaining headers to find the first occurrence of the From: line and extract the claimed sending domain from the first address in the From: line. In this context, "first" means the line closest to the first line of the email.

   If a claimed sending domain cannot be extracted from the From: line, then treat the email as unverified.

   If the claimed sending domain does not match the domain identified with the ‘d’ tag of the "DomainKey-Signature:" header, then treat the email as unverified.

Implementors should program defensively when scanning for the signature information as the contents are completely under the control of the sending system - some of which may have malicious intent.

3.7.3 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:

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.

   E.g., 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, 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, treat the email as unverified.

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

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

3.7.4 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.

Even though there is currently only one canonicalization algorithm and one verification algorithm, implementors should consider the possibility that this list may grow.

3.7.5 Retrieving sending domain policy

In the event that an email fails to verify, the policy of the sending domain should be consulted. For now that means consulting the _domainkey TXT record in the DNS of the claimed sending domain.

While a receiver can always do whatever they want, it behooves a receiver to consider the sending domain policy statement and act accordingly. The range of possibilities is up to the receiver, but it could include rejecting the email.

3.7.6 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 make, but an authenticated email presents an opportunity to a receiving system that unauthenticated email cannot. Specifically, an authenticated email creates a predictable identifier by which other decisions can reliably be managed, such as trust and reputation.

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

3.8 Conveying verification results to UAs

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 in 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 pre-pended to the email.

The "DomainKey-Status:" header contains a single value that indicates the result of the verification. Valid values are:

- "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.

A client can normally just look for "good" and can safely assume that all other values imply that the verification failed in some way.

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 demean 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=dzdVyOfAKCdLXdJOc9G2q8LoXS1Enlsbav+yuU4zGeerud001szZ
```
Hi.

We lost the game. Are you hungry yet?

Joe.

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

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 extracts the domain "football.example.com" 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

Signature verification starts from the line following the "DomainKey-Signature:" line. The email is obviously canonically prepared for verifying with the "simple" method as discussed previously.

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

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 – 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 "DomainKey-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 S/MIME [SMIME] and 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=dzdVyOfAKCdLxdJ0c9G2q8LoXS1En1Sbav+yuU4zGeeruD001szZVoG4ZHRRNyYzR;
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 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, they 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 TTL expiration, a domain owner has course-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 email software routinely (and often unnecessarily) transforms email as it transits through the network. Such transformations conflict with a fundamental purpose of cryptographic signatures - to detect modifications.

The "best" method for canonicalizing an email for presentation to the signing algorithm will no doubt be a matter of much debate. Should all headers be signed? Should whitespace be trimmed? Should header lines be unwrapped? Should known format headers such as Date: be converted to a common form? Should the whole email be encapsulated as a MIME object? Alternatively, is the lightweight canonicalization (called "simple") proposed in this document in fact, too heavy?

At this stage, in the interest of early interoperability tests, a very simplistic, default canonicalization is used. In part this is based on the assumption that early DomainKey implementations will likely be deployed at the border and that most email transformations occur within such borders.

Gaining rough consensus on the best canonicalization algorithm for the long-term will no doubt be a challenging process.

6.3 Mailing lists

Mailing list managers (MLMs) must resolve a number of issues prior to participating in DomainKeys.

First, MLMs need to decide whether they simply pass through a submission and signature unchanged (and quite possibly unverified), or whether they verify the inbound signature and apply their own signature prior to redistribution to the list recipients.

One argument for the latter is that MLMs often make significant changes to the submission. They may add tag lines, rewrite Subject: lines and re-package contents in a digest form.

Secondly, MLMs usually re-distribute email with the original From: header unchanged. However, if they want to apply their own signature to the distributed email, they have to replace the From: line. Such a change presents a surprise to mailing list participants that don’t render the Sender: header.

One possibility is to adapt DomainKeys to specifically recognize MLM
traffic, e.g. recognizing that MLMs often insert a Sender: line and searching for that as the sending domain.

Another possibility is for MLMs to adopt strict and consistent conventions regarding header generation at the point of redistribution. E.g., From: should always be converted to Resent-From:.

A final possibility is that MLMs may not need to participate in DomainKeys as recipients have other means of sufficiently recognizing legitimate MLM traffic, such as List-ID: headers.

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 their 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 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.

Possible semantics and mechanisms for per-user DomainKeys are canvassed in a separate document. Suffice to say here that the base definition specifically supports the possibility of per-user DomainKeys.

6.5 Envelope audit

[ To be discussed: Identify the preconditions in the base document that allow for envelope auditing to protect against replay and joe-jobs ]

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 [DNS-THREATS].

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 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 which 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.
8. The trial

The first step in validating DomainKeys as a viable email authentication system is to conduct a trial of the technology. A number of organizations have committed to developing and deploying DomainKeys and Open Source implementations are expected to become available. Interested parties are encouraged to participate in this trial and help evolve this specification based on those experiences.

8.1 Goals

The primary goals of the trial are to:

- understand the operational implications of running a DNS-based public-key system for email
- evolve the likely canonical needs of future specifications
- determine the best method for MLM participation
- experiment with possible per-user key deployment models
- fully define the semantics of the "DomainKey-X509:" header

8.2 Constraints on participation

Due to the "simple" canonicalization algorithm, participants in the trial should ensure that they:

- do not re-arrange or modify headers or contents
- are 8BITMIME compliant
- sign email after any transfer-encoding transformations have been made prior to transmission

Acknowledging the possibility that early DomainKey implementations may be less than perfect and that senders may be merely testing their implementations, recipient systems should be reticent about applying strict policy to unverified email. Particularly if the sending domain policy or the selector information has the testing mode set.

9. Notes to Implementors

9.1 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
10.2 Informative References


11. Acknowledgments

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

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. In a later draft, a formal grammar will be used to precisely define the rules, for now, this section summarized the most salient syntactic rules for this encoding:

- A tag-value pair consists of three tokens, a "tag", the "=" character and the "value"
- A tag is one character long and must be a lower-case alphabetic character
- Duplicate tags are not allowed
- A value can only consist of characters that are valid in RFC2822 headers, DNS TXT records and are within the ASCII range of
characters from SPACE (0x20) to TILDE (0x7E) inclusive. Values cannot contain a colon or semi-colon but they may contain "=" characters.

- A tag=value pair is terminated by a semi-colon or the end of the data
- Values are case sensitive, (though some values may have the same semantics regardless of case)
- Values are a minimum of zero bytes long
- Whitespace can surround any of the tokens, however whitespace within a value forms part of that value unless explicitly excluded. Currently the only tag that specifically ignores embedded whitespace is the ‘b’ tag in the signature header.
- Tag=value pairs that represent the default value need not be present
- Unrecognized tags and values should be ignored

Appendix B - Sample "simple" canonicalization program: dk_simple_canonical

This sample perl program called dk_simple_canonical, demonstrates the "simple" canonical processing by converting stdin to canonical form on stdout:

```perl
my $pendingTerminators = 0;
my $canonCRLF = "\r\n";
while(<>) {
    chomp;  # Remove local line terminator
    if (length == 0) {
        $pendingTerminators++;  # Count potential trailing empties
        next;  # and avoid printing them
    }

    # A non-empty line means printing all the potentials prior to
    # printing the data line.
    print $canonCRLF while $pendingTerminators-- > 0;
    print;  # print the data line
    $pendingTerminators = 1;  # data is not yet terminated
}
print $canonCRLF if $pendingTerminators > 0;
exit(0);
```

Appendix C - Sample signing shell script: dk_simple_sign

This sample shell script demonstrates how to sign an email that matches the DomainKeys specification. It assumes that the private-key has been previously generated and that the dk_simple_canonical program is in the PATH. This script also assumes that the From: domain matches the domain of the selector.

```bash
#!/bin/sh

file=$1
domain=$2
selector=$3

if [ ! -r "$file" -o "$domain" -o "$selector" ]; then
echo >&2 Usage: dk_simple_sign filename domain selector
exit 1
fi

echo "DomainKey-Signature: a=rsa-sha1; q=dns; c=simple;"
```
echo " s=$selector; d=$domain;"

dk_simple_canonical <$file |
   openssl dgst -sign rsa.private -sha1 |
   perl -MMIME::Base64 -0777 -ne 'print encode_base64($_)' |
   sed -e '1s/^/b=/' -e 's/ /' |

cat <$file

exit 0

Author’s Address

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

domainkeys-feedbackbase00@yahoo.com

Please send comments to the author at the above address.

RCS: $Id: draft-delany-domainkeys-base-00.txt,v 1.4 2004/05/18 00:42:11 markd Exp $