Microsoft

The Coordinated Spam Reduction Initiative

A Technology and Policy Proposal

Microsoft Corporation

Published: February 13, 2004

Over recent years, the rate and frequency at which computer users receive unwanted and unsolicited e-mail (colloquially known as "spam") has increased significantly. In many regions the problem has become so severe so as to perhaps threaten the viability of e-mail as a useful communication medium. It is time that something be done. This paper is a draft-for-comment of a comprehensive proposal containing a technology and policy approach to actually reducing spam. Its technical contents are still in flux and not finalized, though some parts have received significant scrutiny to date and so are less likely to be subject to change. Comments, questions, suggestions and other feedback regarding the proposal are actively solicited. We believe that through coordinated action by the participants in the Internet e-mail community, significant progress in deterring spam can be made, and computer users everywhere will experience significant reductions in undesired and unwanted e-mail correspondence.

Copyright Notice

Feedback

The Author of the Specification welcomes public feedback and review and believes that such feedback can strengthen and improve the Specification. If you want to provide comments, questions, suggestions and other such feedback on the Specification ("Feedback"), you agree to make such Feedback available in accordance with the terms below. You should direct such Feedback to LessSpam@Microsoft.com.

You have no obligation to give the Author any Feedback. However, any Feedback you voluntarily provide may be used by the Author in any product, service, or technology (collectively, "Author Offerings") which in turn may be relied upon by other third parties to develop their own products. Accordingly, if you do give Author any Feedback, you agree: (a) Author may freely use, reproduce, license, distribute, and otherwise use in any Author Offerings; (b) you also grant third parties, without charge, only those patent rights necessary to enable other products that use or interface with any specific parts of an Author Offering that incorporates your Feedback; and (c) agree to not give Author any Feedback (i) that you have reason to believe is subject to any patent, copyright, or other intellectual property claim or right of any third party; or (ii) subject to license terms which seek to require any Author Offering incorporating or derived from such Feedback, to be licensed to or otherwise shared with any third party.

Table of Contents

Part I: Principles, Concepts, and Ideas

As of mid-2003, about **83%** of the e-mail messages received by Microsoft® Hotmail® on a typical day are spam, unwanted and unsolicited e-mail sent indiscriminately to users. That's around **2.5 billion** out of nearly **3 billion** messages, and the numbers keep climbing. While spam has been around virtually as long as there has been e-mail, in the early years the scale involved was small enough so as not to be a significant practical annoyance or problem. That's all changed now. In recent times, the rate at which spam has been in users' electronic mailboxes has skyrocketed. The problem has become one of significant proportions, costing users, industry, and the economy at large billions of dollars annually in terms of lost human productivity and wasted computer resources. Left unchecked, spam threatens the viability of e-mail as a useful communication medium.

It is time to put a stop to this problem. Since the spammers aren't going to change their behavior, it falls on the rest of us to change what we ourselves do so that the majority of the spammers no longer have a viable and profitable business. It is the intent of this Coordinated Spam Reduction Initiative to be a concrete plan of action. The initiative is a set of simple enhancements to e-mail distribution that the Internet community can collectively take on an incremental basis. As these measures are implemented and deployed more and more broadly, it will become more and more difficult to make a lucrative living as a spammer. We will never make the problem go away entirely: any system powerful enough to do that will necessarily unduly impinge on the free exchange of legitimate communication, making the solution worse than the problem it was designed to correct. But what we can do is reduce the problem back to the manageable and tolerable level it once was at. It's time we started.

1. Overview

The approach advocated in the Coordinated Spam Reduction Initiative consists of several parts. At its root, we see it as necessary that e-mail reader and e-mail server software be enhanced to provide **mail filtering features** by which incoming e-mail can be classified, sorted, and presented so that spam is hidden from the user's attention. The next steps are to provide means by which legitimate senders of e-mail take action so as to provide **evidence that they are not spammers**; mail received with such evidence should be honored by receivers' mail filters and not considered to be spam. Finally, a caller-id-like mechanism is defined by which the rampant practice of **sending e-mail with forged return addresses** can be significantly curtailed. This mechanism not only helps in the goal of providing evidence of not being a spammer, but is a significant benefit in its own right. The next several sections provide an overview of these ideas.

2. Mail Filters

Historically, the design of e-mail reading software and e-mail server software primarily focused on making the user experience of dealing with e-mail as efficient, useful, and pleasant as possible. The software had little or no understanding of the actual interest a user might have in a given piece of mail, so all received messages tended to be treated as equals and presented to the user on an egalitarian basis.

With today's ever increasing proliferation of spam, however, this approach just is no longer good enough. Software systems for receiving and reading e-mail need to be enhanced to incorporate features that favor the presentation of desired e-mail and penalize the presentation of unwanted spam. They need to include tools and mechanisms for classifying mail as more or less worthy of attention by the user. In an ideal system, mail considered to be spam would automatically be classified as such and subject to relatively severe penalties while mail believed not to be spam continues to be presented to the user in the efficient manner that has been used historically (that said, particular attention must be paid to the avoidance of 'false positives': non-spam mail that is inadvertently classified as spam). The actual penalties applied to mail believed to be spam will vary from system to system: perhaps spam is deposited in a designated 'Junk E-mail' folder, or maybe not even delivered to the user at all. The mechanisms by which spam is distinguished from non-spam will also vary greatly: some systems may examine each message and classify it as spam based upon words or phrases found therein; others might take advantage of the fact that the same spam is often sent to a large number of users and use collective voting approaches to identify these messages.

Broad deployment of software with such "penalty boxes" for mail considered to be spam is in fact an important part of the overall framework by which the world-wide volume of spam can ultimately be reduced. As e-mail software gets better and better at avoiding bothering the user with spam while leaving non-spam mail unaffected, the economic incentive for actually sending spam in the first place will become less and less. The particular classification and spam-penalization features and technologies used in such software are not as important as the

mere fact that penalization of spam becomes a ubiquitous aspect of deployed e-mail receiving and reading software, thus broadly reducing the value of sending spam at all.

2.1. New Inputs to Mail Filters

It is useful to draw a distinction between the details of *how* spam is classified and penalized within a given e-mail system and the *inputs* available to that system upon which such classification and penalization technologies can be based. No industry coordination is necessary or even desirable on the details of how the classification and penalization features of mail filters should work, and none is proposed here. To the contrary, vigorous competitive innovation in these areas is beneficial to all.

However, the nature of e-mail is that systems from disparate vendors and sources need to interoperate in order for e-mail to be successfully transmitted, and standards therefore exist (and must necessarily so) as to how e-mail is handed off from system to system. In order to increase the ability of mail filtering software to reliably distinguish desirable e-mail from unwanted spam, such standards need to be enhanced so as to provide additional information that can be then input to the mail filtering infrastructure on the receiving side. Enhancing the e-mail transmission standards to provide this additional information requires coordination within the industry.

3. Providing Evidence That You're Not a Spammer

Once a broad part of the Internet community has a mail filter in place to catch and penalize incoming spam, we will have made significant steps forward. However, we can do even better if we arm those filters with additional information that they can use to do their work. Today's filters typically function by coming up with an estimate based on various factors of how much a given message looks like spam, and classifying it as such if these indications exceed a certain threshold. What we aim to do here is to define new indicators that provide evidence that a message is *non*-spam. These measures are designed so that legitimate senders can take advantage of them while spammers cannot. By including such non-spam indicators in the mail they send, legitimate senders can have greater assurance that their mail will not be inadvertently misclassified, and mail filters can be more aggressive at finding spam without a fear of increasing their false-positive rate. We envision two basic kinds of non-spam indicators: a *policy* approach based on a sender's reputation, and a *resource expenditure* approach based on economic behavior.

3.1. Trade on Your Reputation

One way to show that mail you send isn't spam is to get someone whose opinion your recipients value to say that you are not a spammer. This is an approach based on *policy* and *reputation*: if you are an organization large and upstanding enough that an outside party can pragmatically certify that indeed you are who you say you are and you do what you say you do, then this certification can be used by the mail systems and filters of those who receive your mail as an indication that your messages are worthy of their users' attention.

This approach has a close analogy to what has occurred with respect to the issue of online privacy. In that sphere, organizations such as TRUSTe (see reference [8]) and others have articulated policies of reasonable behavior and conduct regarding the handling of sensitive personal information. The policy-setting organizations then certify various information-handling organizations as conforming to a given policy. Certain policies have come to be seen by users as valuable and useful ones, and thus web sites run by organizations certified as conforming to these policies are perceived as more worthy of a user's attention and time.

We expect that things will develop similarly in the spam deterrence sphere. There will likely be a range of policies of e-mail transmission that will come to be generally accepted as reasonable, each addressing a particular operational need. That said, this Coordinated Spam Reduction Initiative does *not* itself seek to define such policies; rather, what we attempt to define here are the technical *mechanisms* by which an indication of conformance to a given policy can be conveyed from sender to receiver. Having standardization on the mechanisms of policy dissemination helps to encourage the growth of and experimentation with new and interesting policies of reasonable e-mail transmission.

3.2. Do Something That It's Uneconomical for a Spammer To Do

While the policy-based approach works very well when it's feasible, unfortunately there is a substantial faction of the Internet community which is unable to participate in such a framework. Many senders of e-mail are associated with organizations which are simply too small to have a reputation on which an e-mail-transmission-policy-setting

body could base a policy-conformance-attestation. Suppose that you are a small sole-proprietor business with (say) less than ten employees: it is not likely that you will have sufficient traceable and auditable electronic information associated with your business that a policy-certifying organization could actually certify your conformance with their policies. We see this situation in analogy today, for example, with respect to the issuance of certificates that verify the authenticity of SSL servers: if you're a "small potatoes" domain, it's very difficult if not impossible today to purchase an SSL server certificate which will attest to your legitimate ownership of your domain, simply because you lack information by which the certificate issues can investigate your authenticity and ultimately hold you accountable.

Therefore, in this initiative we need some alternate mechanism in addition to attestation of policy conformance. It is simply not acceptable that we would create a world in which there were 1st class and 2nd class e-mail senders: those who could take measures to positively indicate that their mail was not spam, and those who could not. The alternate measure we propose is that legitimate senders *do something that it is uneconomic for a spammer to do, and show that they did that.* Specifically, we suggest that the sender expends some limited resource on behalf of each of the messages that he sends.

In that context, an obvious resource to consider having senders expending is real money. If a sender were to, say, actually spend 1 US cent on behalf of a message, and his recipients could be convinced that the money indeed was no longer in the sender's possession, then reasonable recipients ought to be convinced that the message in question is not spam: this amount of expense per message is beyond the level that spammers can economically afford. However, while in theory this approach is attractive, actually moving real money around on behalf of sending messages has a number of pragmatic questions that need to be addressed, among them the following:

1. Who should actually get the money? The receiver of the message? A third party? If the latter, then what value is he adding to the system that would justify his receiving a windfall stream of income?
2. How should the money transfer actually be effected? Will a selected mechanism work on a global basis? In each of the worlds two hundred or so countries and currencies?
3. What are the privacy issues involved? Can the money be transferred anonymously, or does this introduce some new mechanism by which e-mail can be forensically classified and analyzed?
4. In each of the world's thousands of taxing jurisdictions, what are the income tax and other tax implications of having this money change hands?
5. What are the politics involved? To raise but one issue, if the party receiving the money is a government-related agency (perhaps a postal service, for example), then many will perceive such a measure as introducing a formal new tax.

Actually trying to cope with these and related questions would be a daunting challenge.

Fortunately, actual money is not the only resource we need consider having senders expend. As has been observed by others (see references [1] and [4]), it is a fortunate coincidence that the computers of a majority of smaller organizations are typically lightly loaded. These computers have lots of "wasted" CPU cycles: on average, the microprocessors in these computers have almost nothing to do. On the other hand, the same is not true of the computers used by those whose large volumes of spam: their computers are busy almost 100% of the time doing the work of sending out their millions of messages. If we can define a means by which senders could demonstrate that a certain amount of CPU time was expended on behalf of the sending of a given message, that would be a cost that could be borne easily by small organizations yet not by spammers, and, if the CPU expenditure is large enough, would be a good indication that the message is not spam.

The utilization of a CPU for a particular amount of time is actually a two-fold cost. First, as a certain fraction of the overall cost of running the computer in question, the expenditure of CPU cycles represents an actual expenditure of money. Second, the expenditure of significant CPU time per message also inherently slows down the rate at which messages can be generated, which is in itself a deterrent to spam.

4. Caller ID for E-mail: Preventing the Use of Forged E-mail Domains

When e-mail is transmitted over the Internet from one organization to another today, typically no authentication of the sender of the e-mail or the computers delivering it on the sender's behalf occurs. That is, no verification is done to ensure that mail which purports to be sent from, say, `someone@example.com` does in fact originate from computers under the control of the "Example" organization. It is a trivial matter for virtually anyone with a computer connected to the Internet to send mail and appear to be someone else in doing so. This is a form of

forgery, and is often politely called “spoofing”. Automated infrastructure for preventing such forgeries is lacking: short of a human forensic investigation, there is no means to distinguish such mail from the real thing. Needless to say, spammers routinely exploit this capability.

One way to begin to address this state of affairs is to mirror what the telephone system provides with caller id. In the telephone system, caller id technology reliably informs the receiver of a phone call as to which telephone number is on the other end of the line. This is often related to the human being you end up talking to, but is not one-and-the-same concept. A more direct analogy to e-mail is most evident if one considers the transmission of faxes over the telephone: caller id provides the phone-company-provided phone number of the party sending the faxes, but this is a separate notion from the identity of the party who is responsible for the contents of the fax itself. That said, in most cases one expects the two notions to be related, and would be suspicious if they were not.

In the world of e-mail on the Internet today, we’re missing this sort of caller id mechanism. However, as has been observed by others (see references [2], [3] and [10]), a caller id mechanism can be achieved in e-mail relatively simply by having administrators of domains publish the Internet addresses of their *outgoing* e-mail servers in the Domain Name System (DNS) in addition to the *incoming* e-mail servers that they list there today (the term “domain” refers to the part of an e-mail address that follows the at sign “@”). DNS is a planet-wide, distributed database that provides numerical Internet Protocol (IP) addresses and other information about Internet domains, much as a paper telephone book lists addresses and telephone numbers under the names of telephone subscribers. When e-mail is transmitted from one organization to another, the computers of the sending organization need to determine which computer the receiving organization has designated as the one that handles its incoming mail. For example, the DNS entry for the domain `example.com` might include the fact that the computer at Internet address 111.222.123.121 is one to which mail destined for e-mail addresses `@example.com` may be delivered. To actually deliver mail to this organization, the sending computer opens up a connection to 111.222.123.121 and hands over messages one-by-one using the conversational rules of a standard known as the Simple Mail Transfer Protocol, or SMTP.

With the information listing outgoing mail servers in place, software that receives an incoming message can now verify that the computer used to send the message was in fact under the control of the domain from which the message purports to have originated, just as (in analogy) the receiver of a fax sent over the phone can use caller id information to check that the phone number from which the fax was transmitted is consistent with what the contents of the fax actually say. With this infrastructure in place, e-mail software will be able to distinguish, for example, whether a message that claims to be from `someone@example.com` was actually sent from the `example.com` organization or not. If not, the mail is a forgery.

Note that the caller id mechanism defined here is not the equivalent of fully authenticated e-mail: there still remains for example the question as to whether the user named `someone` did in fact send the e-mail or whether, say, some rogue administrator at `example.com` sent the e-mail instead. Even so, having reliable domain information is a huge step forward, and will be of great utility to e-mail filtering and classification systems, allowing them to distinguish between spam and non-spam e-mail with greater certainty. This in turn allows organizations to more harshly penalize e-mail they believe to be spam with less fear of misclassification or “false positives”.

5. Conclusion

Taken together, the proposals of this Coordinated Spam Reduction Initiative lead to a *virtuous circle*. Each one helps to distinguish legitimate organizations – large and small – from spammers, and legitimate e-mail from spam. As these approaches are deployed more and more broadly, the severity of the penalties applied by mail filters to mail suspected of being spam can be made larger and larger without fear of false positives. Users get the mail they want, and they get dramatically less spam. In the long term, a point will be reached where the effective visibility of spam mail to users is so small that there is little point in sending it in the first place. We, as do many others, eagerly look forward to that day.

Part II: Technical Specifications

Part I of the Coordinated Spam Reduction Initiative gave an overview of the problem space and motivation for an approach to a solution. This Part II of the initiative gives the technical details and specification of that solution. It is intended that all the information needed by software implementers to intimately understand and to implement the proposal be provided here.

It should be noted that the content presented here in §6 and §7 is also available in a separate, independent specification entitled "Caller ID for E-mail". The technical content of that specification is a proper subset of what is presented here, and there are no differences in technical design.

6. Technical Preliminaries

From a conceptual point of view, the intent of the technical specifications here is broadly two-fold.

The first intent is to provide means by which senders of e-mail can provide ***evidence that they are not spammers***. By providing such evidence when they send their mail, individuals and companies can communicate with those with whom they've never communicated before and have a reasonable assurance that their message not be misinterpreted as spam. This ability for people to communicate with new correspondents that they have not previously been introduced to is a very important property of the e-mail communication medium that ought to be preserved. We define two fundamental technical strategies for providing such evidence:

1. *Have the sender provide evidence that he has done something that it is uneconomical for a spammer to do*. In this specification, we suggest this be accomplished by exhibiting solutions to computational puzzles, demonstrating that the sender has consumed a certain amount of CPU cycles on behalf the sending of the message (§11).
2. *Have the sender provide evidence that someone whose opinion the receiver values says that the sender is a non-spammer.* This is a matter of *policy*. While we do not here define specific policies of reasonable e-mail transmission (but see §8), we define two mechanisms by which conformance with pretty much any imaginable such policy can be conveyed:
    a. The signing of transmitted e-mail under the auspices of certificates which attest that the sender conforms to a particular transmission policy or policies (§9).
    b. As a matter of performance and convenience, a means by which a domain can indicate that *all* of the mail that it transmits conforms to a certain transmission policy or policies without the need to sign each individual e-mail (§10).

    The separation of the details of a particular policy from the mechanisms by which such conformance to such policies is conveyed is important. Keeping the issues of policy and mechanism orthogonal enables one to write and deploy software that allows senders to indicate conformance with a widest possible range of policies, even those that might not have been articulated at the time such software was shipped, as well as allowing receiving systems to similarly choose to value or not to value such policies. By decoupling from the need to deploy new software, the orthogonality of policy and mechanism also encourages the marketplace of ideas and principles underlying such policies to arise, flourish, and grow. In contrast, a bundling together of policy and mechanism both hinders the ability to develop and deploy new software that aids and benefits users, and impeded the articulation and adoption of new and interesting policies.

The second main intent of these technical specifications is to provide at least ***a minimal level of authentication*** within the e-mail infrastructure so the wild abandon with which today any sender can forge the identity of any other is diminished. Concretely, we define a caller id-like means by which the forging of the DNS domain of the sending address of e-mail can be eliminated (§7). While by no means a panacea, these measures are a great step forward.

These components are interrelated. For example, the non-spoofed domain provided by §7 is a key aspect of the mechanism of §10. Knowing that domains are not spoofed makes the maintenance of known-sender lists considerably more valuable, and this in turn makes the computational puzzle approach more usable overall. Other interconnections exist as well.

The remaining sections of this document detail each of these technological components in turn. However, we begin in the present section with some preliminaries.

6.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in RFC2119.

6.2. Storing E-mail Policy Information in DNS

At various points below and elsewhere, it is necessary that new kinds of policy and other information regarding the e-mail policies of a DNS domain be published. This section defines an approach to publishing such information.

The natural mechanism with which to publish information regarding a DNS domain is of course to use DNS itself in a straightforward, traditional manner. That said, unfortunate logistical complications are found in the selection of DNS resource record types that must be used to represent such information. A seemingly-obvious choice is to select appropriately-defined new record types for such new information. However, the use of new record types generally requires that DNS client software, server software, and administration tools all be updated to be made aware of the new types. Were we to require such updates here, the adoption of this proposal would be significantly impeded.

Accordingly, an alternate means of representing the new information has been chosen, one that only makes use of existing, commonly understood DNS resource record types. Specifically, XML-encoded information stored in the TXT resource records in the '`_ep`' subdomain of a DNS domain in question is used. These records form a record set, which can thus be retrieved with a single DNS query, an important performance concern.

The use and interpretation of this record set is as follows. Each TXT record in this record set MUST be of at most 2K characters in length (this accommodates restrictions of deployed DNS clients and servers, specifically broadly deployed versions of BIND). If there is just a single record in the record set, then the E-mail Policy Document for the domain is simply the sequence of strings in that one record concatenated together. Otherwise, the concatenated sequence of strings in each TXT record in the record set MUST begin with two distinguished characters (the two-digit decimal representation (with leading zeros as needed) of a non-negative integer is recommended) which is unique within the record set. Upon retrieval, the TXT records in the record set are sorted in ascending lexicographical order by these two characters, the first two characters removed from each record, and the results concatenated together to form one overall sequence of characters which is the E-mail Policy Document for the domain.

The E-mail Policy Document for the domain MUST be a valid UTF8-encoded XML document; in order to promote interoperability, other XML character encodings MUST NOT be used.

The XML E-mail Policy Document of a domain SHOULD contain information which the administrators of the domain attest is pertinent to the policies under which the domain receives or sends mail. The XML schema definition which SHOULD be used for such XML documents is found in §13; an E-mail Policy Document containing elements conforming to a different schema SHOULD be ignored by those applications which do not understand it.

XML E-mail Policy Documents SHOULD be cached in the normal manner for information returned from DNS. In addition, applications may find performance benefits from also caching at a higher semantic level the information derived from parsing these documents, but whether they in fact do so or not is their own local concern.

More will be said later regarding the details of E-mail Policy Documents, but the simple cases are quite simple: one writes the XML, it fits in a single TXT record, and this is copied to the DNS configuration file. For example (see also §7.1.1), the following illustrative fragment of a DNS configuration file shows an XML E-mail Policy Document indicating that the outbound mail servers of the domain `example.com` are exactly whatever the domain's inbound mail servers are. Indeed, for those domains where this is the applicable policy, this literal boilerplate TXT record can be used verbatim.

```
@ IN SOA ns.example.com. postmaster.example.com. (1 36000 600 86400 3600)
         _ep TXT ("<ep xmlns='http://ms.net/1'><out><m><mx/></m></out></ep>")
```

This next slightly more contrived example illustrates how an E-mail Policy Document can be split across possibly several TXT records:

```
@ IN SOA ns.example.com. postmaster.example.com. (1 36000 600 86400 3600)
          _ep TXT ("02.3.4</a>"
                   "      </m>"
                   "      <m><mx/></m>"
                   "   </out>"
                   "</ep>" )
              TXT ("01<ep xmlns='http://ms.net/1'>"
                   "   <out>"
                   "      <m>"
                   "         <a>1.2" )
```

Here, the e-mail policy information of the `example.com` domain is the following XML document (note that, per the usual XML rules, with the exeption of the space character between `ep` and `xmlns`, whitespace is not significant):

```
<ep xmlns='http://ms.net/1'>
   <out>
      <m>
         <a>1.2.3.4</a>
      </m>
      <m><mx/></m>
   </out>
</ep>
```

6.3. Maintenance of Known-Sender Lists

Within a mail-receiving organization's computer systems, a particularly important aspect of the mail filtering infrastructure is the maintenance of known-sender lists, that is, lists of known bona-fide correspondents of the users on whose behalf a mail-receiving system operates. Such lists are used to modify what would otherwise be the normal mail filtering behavior of the receiving system, with the aim of reliably preventing mail sent from addresses on such lists as being inadvertently classified as spam.

As was mentioned above, there is a significant interaction of the use of known-sender lists with the technology initiatives of this proposal. For example, the domain spoofing elimination provided by the mechanisms of §7 increases the value of known-sender lists, as it becomes considerably more difficult for an attacker to send a message and have it appear that the message was sent from an address legitimately on the receiver's list. Conversely, if a message sender has knowledge that he is on his recipient's known-sender list, then he can reasonably avoid expending effort to otherwise proof to that recipient that he is not a spammer. This is particularly important in the face of the computational puzzle proposal of §11.

In support of these interactions, it seems very worthwhile to define a technical means by which the additions or deletions of a mail sender to given mail recipient's known-sender list can be communicated from the recipient to the sender. In this subsection, we define such a means. We also discuss important security and privacy implications of its use.

6.3.1. The KnownSender: Header

That a given sending address has been placed on or removed from some mail recipient's known-sender list can be communicated to the sending address through the use of a `KnownSender:` header on some message traveling in the reverse direction (that is, from said recipient back to the sender).This header is a structured header in the sense of §2.2.2 of RFC2822. Its syntax is defined as follows (grammar non-terminals not defined here are from RFC2822):

```
knownSender = "KnownSender:" knownSenderStatus mailbox-list
knownSenderStatus = "on" \ "off"
```

Multiple `KnownSender:` headers are permitted on one message; if multiple headers exist, such are to be processed in reverse order from last header to first.

The information conveyed by a `KnownSender:` header is that each of the addresses in the mailbox-list is either now on or now absent from (according respectively to whether the `knownSenderStatus` is `on` or `off`) the known-sender list of the mailbox(es) listed in the `From:` header (see §3.6.2 of RFC2822) of the message. If the message lacks a `From:` header, then no information is conveyed.

6.3.2. KnownSender: Header Usage: Security and Privacy Considerations

There are significant security and privacy considerations surrounding both the transmission and the processing of `KnownSender:` headers.

6.3.2.1. Issues in processing KnownSender: headers

From the point of view of a message receiver *r* receiving `KnownSender:` headers apparently from a sender *s*, the biggest security concern is one of forgery: whether one can reasonably trust and so take action on the `KnownSender:` header(s) that may be present in a received message. In the absence of any verification steps, such headers are completely spoofable, and an attacker could thus fraudulently convince receiver *r* that he either is or is not on sender *s*'s known-sender list, with consequences ranging from the r's subsequent mail to *s* being classified as spam to his needless expenditure of resources (using the techniques of this proposal) in trying to indicate that his messages indeed are not spam.

To that end, it is RECOMMENDED that the recipient *r* of messages apparently from *s* that contain `KnownSender:` headers *should ignore* such headers unless:

1. the message is seen through the use of the mechanisms of §7 not be domain-spoofed, and
2. the `From:` header contains just one mailbox (namely that of *s*), and that matches the purported responsible address of the message as defined in §7.2.

While not foolproof, these measures provide a level of assurance that the `KnownSender:` headers can be reasonably acted on. Other approaches may be possible (perhaps involving the use of digital signatures) by which *r* may be able to reasonably ascertain the authenticity of the message and so be able to act on the `KnownSender:` headers therein, but these are beyond the scope of this document.

6.3.2.2. Issues in sending KnownSender: headers

It should be mentioned that there are considerable design issues surrounding exactly what actions (automatic and / or manual) should cause an entry to be added or removed from a user's known-sender list; however, such issues are beyond our scope here. Rather, what is important here to examine are the issues surrounding the timing of when the presence or absence of a sender on a given user's list should be transmitted to that sender.

As was seen in the previous subsection, recipients of `KnownSender:` headers will only process them if the messages have certain characteristics. Thus, messages SHOULD NOT be sent with `KnownSender:` headers unless they conform to said characteristics. Of the conformant messages, appropriate `KnownSender:` headers SHOULD be attached under any of the following conditions:

1. If a message is sent by sender *s* to recipient *r*, *r* is on *s*'s known-sender list, it is not against *s*'s policy (as determined by some means beyond the scope of this document) to disclose that fact, and since the previous time *s* communicated that fact to *r* (which may of course have never yet occurred) *s* has received from *r* a message that contains an anti-spam measure which *r* could have avoided if he had only known he was on *s*'s list, or then the message SHOULD contain a `KnownSender:` informing *r* of his presence on *s*'s list.
2. If a message is sent by sender *s* to recipient *r*, where *r* was previously on *s*'s known-sender list but has subsequently been removed, and it is not against *s*'s policy to disclose this fact, and *r* has not yet been informed at least three times of his removal, then the message SHOULD contain a `KnownSender:` informing *r* of his absence from on *s*'s list.

In the interests of privacy the transmission of `KnownSender:` headers except when demonstrably useful or necessary SHOULD be avoided.

It is RECOMMENDED that `KnownSender:` headers be attached to messages which would still have otherwise been sent: the automatic generation of messages primarily or solely with the intent of propagating this header is discouraged: such messages convey little information to users, and, especially in the context of distribution lists, may cause considerable inconvenience and annoyance.

It should be noted that the inclusion of a `KnownSender:` header for a recipient who is BCC'd on the message leaks information to the other message recipients, allowing them to infer with reasonable cause that the BCC'd recipient was present. Accordingly, a message containing a `KnownSender:` header which mentions a BCC'd recipient SHOULD be a bifurcated copy of the original message that is delivered to that recipient alone; in the original message, the BCC'd recipient MUST be omitted from all `KnownSender:` headers. If for technical or other reasons message bifurcation is not possible, `KnownSender:` headers regarding the BCC'd recipient SHOULD simply not be transmitted.

7. Caller ID for E-mail: Preventing Forged Domains in E-mail

Today, when e-mail is handed off from one organization to another, it is necessary for the computers of the organization sending the mail to learn which computer the receiving organization has designated as the one that handles its incoming mail. This is accomplished with the help of DNS. For example, the DNS entry for the domain '`example.com`' might (using a MX resource record) include the fact that the computer at Internet address 111.222.123.121 is one to which mail destined for e-mail addresses '`@example.com`' may be delivered. To actually deliver mail to this organization, the sending computer opens up a connection to 111.222.123.121 and hands over messages using SMTP.

It is important to understand that today, as a rule, absolutely no authentication of the sender of the e-mail or the computers delivering it on his behalf happens as part of this SMTP transmission process. That is, there is no means to verify that mail being handed to `example.com` for delivery which purports to be sent from, say, `someone@otherexample.com` is in fact being provided by computers under the control of the 'Other Example' organization. Indeed it is a trivial and simple technical matter for virtually anyone with a computer connected to the Internet to send mail and appear to be someone else in doing so. This is not uncommonly called 'spoofing'. Automated infrastructure for preventing this is lacking: short of a human forensic investigation, there is no means to distinguish such mail from the real thing. Needless to say, spammers routinely exploit this capability.

Fortunately, there are some simple steps which can be taken to significantly alleviate this problem, steps which mimic within the e-mail infrastructure the caller id mechanism found in today's telephone system. Specifically, based on the ideas of Hadmut Danisch and others (see references [2] and [3]), the present proposal specifies that in addition to today's practice of publishing in DNS the addresses of their *incoming* mail servers, administrators of domains SHOULD also publish the addresses of their *outgoing* mail servers, the addresses of the computers from which mail legitimately originating from that domain will be sent from. This information will be then used in enhancements to software that receives incoming mail to verify that computers handing off a message to them in fact are authorized to do so by the legitimate administrator of the domain from which the message is purported to have been sent.

This simple provision of a caller id mechanism for e-mail will virtually eliminate the problem of domain spoofing, in that while it does not prevent its occurrence, it provides a means to detect when it is taking place. For example, when someone receives mail appearing to be from `johndoe@example.com`, their software will be able to with virtual certainty[1] to ascertain whether it legitimately originated from within the `example.com` organization instead of an imposter thereof. Note that this is not the equivalent of fully authenticated e-mail: there still remains with this proposal the question as to whether `johndoe` did in fact send the mail or whether, say, some rogue administrator at `example.com` sent the mail instead. Even so, having reliable domain information is a huge step forward, and will be of great utility to mail filtering and classification systems, allowing them to distinguish between spam and non-spam mail with greater certainty, and thus be able with less fear of 'false positives' to be able to penalize more harshly mail they believe to be spam. In particular, with domain spoofing no longer a problem, the consequent reduction in impersonated e-mail will significantly increase the value of the maintenance of known-sender lists.

From a technical perspective, the caller id mechanism is roughly separable into three parts:

1. the publication in DNS of the **outgoing mail servers** for a given domain in addition to the *incoming* mail servers which are already published today,
2. the identification of a particular **domain as being responsible** for a given message, and
3. the **correlation and checking of these** two pieces of information to determine whether the IP address from which the message was received was reasonable or not.

The following subsections examine each of these parts in turn. Significant subtleties and surprises lurk in each: readers are cautioned to read and consider this information carefully.

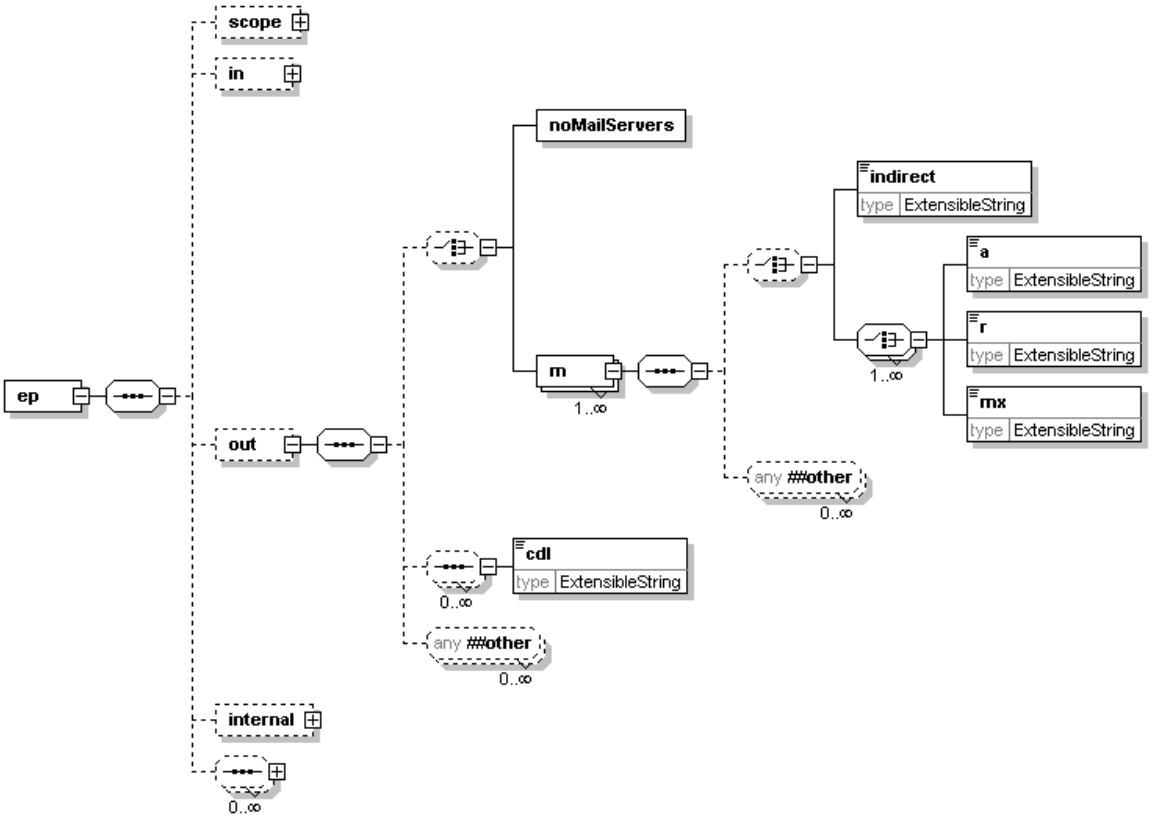
7.1. Publication of Outgoing Mail Servers

In this proposal, the identity of the outgoing mail servers for a domain SHOULD be published in the XML E-mail Policy Document for the domain, specifically in the `ep/out` element. If the XML E-mail Policy Document does

[1] The remaining uncertainty lies with the fact that the entire DNS system itself is not, in theory, secure. The details of this issue are elaborated below.

not exist for a domain or the document exists but does not contain either an `ep/out/m` element or an `ep/out/noMailServers` element, then the domain makes no statement about its outbound mail servers. If the document exists and contains an `ep/out/noMailServers` element, then the domain is making the statement that there are *no* outbound servers for that domain.

The portion of the E-mail Policy Document relevant to outbound mail servers can be depicted as follows (a more complete exposition of this schema is found in §13 below). Note that XML attributes (notably those on the element `m`) are not illustrated in this diagram; only XML elements are shown.



The outbound mail servers for the domain are listed as one or more occurrences of an `m` element within the element `ep/out`. The interpretation of each `m` element ultimately provides two essential pieces of information about the server in question:

1. whether or not it makes use of the "enhanced SMTP noop" functionality (described in §7.5.2 below) when making outbound SMTP connections, and
2. the IP address or addresses of the server.

Within the design, several important points were considered, including:

a. Some domains (many of them well known) have literally thousands of outgoing mail servers. Thus, a means must be provided by which the need to list the address each server individually can be avoided. This is addressed by permitting legal *address range* (using address prefix list information in the style of in RFC3123) to be listed in place of individual addresses.

b. Although not as common as it once was, SMTP outbound relaying must be accommodated. In such situations, the outbound servers that actually perform the inter-domain delivery of mail (as seen by the receiving domain) may not be administratively under the direct control of the sending domain. A means of indirection is thus provided, wherein a sending domain can avoid having to repeatedly and continually update its XML E-mail Policy Document to reflect the now-current addresses of the servers used by the relaying organization.

From the point of view of specification, the set of IP addresses from which mail from a given domain *d* may be transmitted to a receiving domain is denoted as *OutGoing(d).* The function *OutGoing(d)* is defined as follows.

1. If there exits an element `ep/out/noMailServers` in the XML E-mail Policy Document of *d*, then *OutGoing(d)* is defined to be the empty set.
2. If there exists at least one element `ep/out/m` in the XML E-mail Policy Document of *d*, then *OutGoing(d)* is defined to be the union over all the `ep/out/m` elements *m* in the XML E-mail Policy Document of *d* of the set of IP addresses *OutGoing(m,d)* mapped to by each.
3. Otherwise, the result of the function *OutGoing(d)* is undefined.

In turn the function *OutGoing(m,d)* is defined as follows. If *m* is an element of the form `ep/out/m` in domain *d*, then *OutGoing(m,d)* is the set of IP addresses mapped to by *m* as calculated in the following manner.

1. If the child element *m*`/indirect` exists, then it MUST contain a fully qualified domain name *d'*. *OutGoing(m,d)* is then defined as follows:
    a. If *d'* contains an E-mail Policy Document in its DNS entries, then *OutGoing(m,d)* is defined to be equal to *OutGoing(d')*.
    b. Otherwise, *OutGoing(m,d)* is defined to be the same set of addresses that are associated with the inbound mail servers of the domain *d'* as determined by the usual MX and A record processing (see §5 of RFC2821).
2. Otherwise, if any child elements *m*`/a`, *m*`/r`, or *m*`/mx` exist, then *OutGoing(m,d)* is defined to be the set difference $A_{+}(m,d) - A_{-}(m,d)$ of the two sets $A_{+}(m,d)$ and $A_{-}(m,d)$ which respectively represent explicitly listed addresses and explicitly listed address exclusions for *m*. These two sets are specified as follows:

    The set $A_{+}(m,d)$ is defined to be the union over all child elements *m*`/a`, *m*`/r`, or *m*`/mx` of the contributions of each as defined in the following manner:

    a. An *m*`/a` element MUST contain one of the following
        i. the (usual) dotted-decimal textual representation of an IPv4 address as defined by §3.4.1 of RFC1035 (for example, "`11.22.33.44`"); in this case, the contribution to the set $A_{+}(m,d)$ is the IPv4 so noted.
        ii. any of the three forms of the textual representation of an IPv6 address as defined by §2.2 of RFC2373 (for example, "`1080:0:0:0:8:800:200C:417A`" or "`::FFFF:129.144.52.38`"); in this case, the contribution to the set $A_{+}(m,d)$ is the IPv6 address so noted.
        iii. a fully qualified domain name *d'*. In this case, the contribution to the set $A_{+}(m,d)$ is the set of IPv4 and / or IPv6 addresses listed in DNS for the domain *d'* using (respectively) A and / or AAAA records.
        iv. an empty string. Here, the contribution to the set $A_{+}(m,d)$ is the set of IPv4 and / or IPv6 addresses listed in DNS for the current domain *d* using (respectively) A and / or AAAA records.
    b. An *m*`/r` element MUST contain the textual representation of a single prefix-denoted range of IP addresses as defined by §5 of RFC3123. Examples include "1:192.168.32.0/21" and "!1:192.168.38.0/28". If the representation begins with a '!' character, then its contribution to the set $A_{+}(m,d)$ is the empty set; otherwise, its contribution is the set of IP addresses denoted by the textual representation.
    c. An *m*`/mx` element MUST contain one of the following
        i. a fully qualified domain name *d'*. In this case, the contribution to the set $A_{+}(m,d)$ is the same set of addresses that are associated with the inbound mail servers of the domain *d'* as determined by the usual MX and A record processing (see §5 of RFC2821).
        ii. an empty string. Here, the contribution to the set $A_{+}(m,d)$ is the same set of addresses that are associated with the inbound mail servers of the current domain *d* as determined by the usual MX and A record processing (see §5 of RFC2821).

    The set $A_{-}(m,d)$ is defined to be the union over all child elements *m*`/r` of the addresses indicated by each in the following manner:

a. As was just noted, an `m/r` element MUST contain the textual representation of a single prefix-denoted range of IP addresses as defined by §5 of RFC3123. If the representation begins with a '!' character, then its contribution to the set *A_(m,d)* is the set of IP address denoted by the textual representation but with the '!' removed; otherwise, its contribution is the empty set.

3. Otherwise, *OutGoing(m,d)* is defined to be the same set of addresses that are associated with the inbound mail servers of domain *d* as determined as determined by the usual MX and A record processing (see §5 of RFC2821).

Note that, as usual, internal DNS processing will automatically follow CNAME resource record aliasing should such aliases be used. Be aware that the use of the indirection mechanism or the use of the MX and / or A record processing per §5 of RFC2821 and the like will result in more DNS queries during the resolution than would otherwise be the case.

Implementations of this functionality should take internal measures to detect cycles of recursion as the evaluation progresses. While detecting actual recursion loops is RECOMMENDED, a simple approach is merely to count recursion depth; if that approach is used, then to promote interoperability a depth of at least eight recursions SHOULD be accommodated. If such a recursion cycle is detected, the results of the overall evaluation SHOULD be considered undefined, resulting in an inability to discern the outbound servers of the domain in question: the querying system SHOULD act in the same manner as if the outbound information were not published at all. If a DNS query involved in the evaluation times out or otherwise fails, then a reasonable approach is similarly to treat the query as unable to discern the outbound servers of the domain. Moreover, in order to deter a denial of service attack against querying mail servers through the use of deep and broad trees of indirection, a querying system MAY itself impose a time limit on the duration of the execution of the overall query including all its indirections. If present, such a limit is implementation-dependent, but SHOULD NOT be less than twenty seconds.

7.1.1. Examples of Outbound Mail Servers Published in E-mail Policy Documents

Here are some simple illustrative examples:

(1) My outbound mail servers are whatever my inbound mail servers are:

```
<ep xmlns='http://ms.net/1'><out><m><mx/></m></out></ep>
```

(2) My outbound mail server is at the single address `192.168.210.101`:

```
<ep xmlns='http://ms.net/1'><out><m><a> 192.168.210.101</a></m></out></ep>
```

(3) My domain has three particular outbound mail servers:

```
<ep xmlns='http://ms.net/1'><out><m>
    <a>192.168.210.101</a>
    <a>192.168.210.102</a>
    <a>192.168.210.107</a>
</m></out></ep>
```

(4) My domain has no outbound mail servers:

```
<ep xmlns='http://ms.net/1'><out><noMailServers/></out></ep>
```

(5) The outbound mail servers of my domain all live within one particular group of 16 IP addresses:

```
<ep xmlns='http://ms.net/1'><out><m>
    <r>192.168.210.101/28</r>
</m></out></ep>
```

(6) My domain employs `contoso.com` to send mail on its behalf. In addition, my domain also sends mail outbound from its inbound servers and from the specific address `192.168.210.101`:

```
<ep xmlns='http://ms.net/1'><out>
    <m><indirect>contoso.com</indirect></m>
    <m><mx/></m>
    <m><a>192.168.210.101</a></m>
</out></ep>
```

(7) My domain has multiple subdomains, all of which use the same outbound mail servers.

- Create CNAME records for each subdomain:

```
_ep.sub1.example.com  IN CNAME outbound.example.com
_ep.sub2.example.com  IN CNAME outbound.example.com
_ep.sub3.example.com  IN CNAME outbound.example.com
```

- In `_ep.outbound.example.com` (a dummy domain), register the required E-mail Policy Document.

(8) My domain has hundreds of scattered outbound e-mail servers and will not open port 53 to TCP traffic. This can be accommodated by using `<indirect>` to dummy subdomains in order to build a tree of internal nodes, all of which can fit in a DNS UDP response, then populating the leaves of that tree with pieces of the necessary information. For example:

- In `_ep.example.com` publish this E-mail Policy Document:

```
<ep xmlns='http://ms.net/1'><out><m>
   <indirect>list1.outbound.example.com</indirect>
   <indirect>list2.outbound.example.com</indirect>
</m></out></ep>
```

- In `_ep.list1.outbound.example.com` (a dummy domain) publish a subset of my outbound server addresses information. For example:

```
<ep xmlns='http://ms.net/1'><out><m>
    <a>192.168.210.101</a>
    <a>192.168.210.102</a>
    <a>192.168.210.107</a>
    <a>     ...       </a>
</m></out></ep>
```

- In `_ep.list2.outbound.example.com` (a dummy domain) publish the remaining outbound server address information. For example:

```
<ep xmlns='http://ms.net/1'><out><m>
    <a>192.168.210.253</a>
    <a>192.168.93.17</a>
    <a>192.168.93.21</a>
    <a>     ...     </a>
</m></out></ep>
```

(9) The outbound servers of my domain reside at the address(es) of the domain `dynamic.example.com` as listed in DNS for that domain:

```
<ep xmlns='http://ms.net/1'><out><m><a>dynamic.example.com</a></m></out></ep>
```

7.1.2. Bringing Outbound Mail Servers Online and Offline

Domain administrators managing the publication of the identity of outbound mail servers for their domain should be cognizant of the fact that the time-to-live (TTL) value they publish on their records will adversely affect the speed with which a new outgoing mail server for the domain can be pragmatically brought online. Such effects can often be mitigated by the pre-publication of the IP addresses that such new servers will use, if such addresses can be predicted ahead of time. In particular, if the domain owns a particular range of IP addresses, and can administratively arrange that its new outgoing mail servers will use addresses in that range, the adverse effects can be eliminated by the one-time pre-publication of the address range.

An analogous issue arises involving the *retirement* of the addresses of *old* servers that are taken offline. Generally speaking, even after the last message has been transmitted from a retiring server address, there may be a period of time before all the messages sent from that address have been evaluated against a caller id check. This period of time may be considerable: if the caller id check is carried out in an e-mail client (for example, see §12, Conformance Level R/I/1a), the message may perhaps sit in a POP3 server for weeks or more while a user is on vacation. In order to bound this otherwise indeterminate length of time, it is required that a caller id check on a given message MUST be carried out within 28 days (672 hours) of the message having been received by the checking organization (that is, within 28 days of the message having crossed the inter-organizational SMTP hop);

after such a period has elapsed, a caller id check MUST NOT be carried out. Organizations wishing to retire addresses of old mail servers SHOULD keep those addresses listed in their domain's E-mail Policy Document until this amount of time has elapsed since the last message sent from that address has crossed the inter-organizational SMTP hop and so been received by the destination domain. Once this time period has elapsed, the address can safely be removed from the E-mail Policy Document.

7.2. Mapping a Message to a Purported Responsible Domain

For each message transferred through SMTP from one organization to another (information regarding published outbound mail servers need not and likely ought not to be interrogated as mail is relayed within an organization, where presumably the trust relationships amongst the computers involved are such that they need not guard themselves against spam originating from within the organization itself) a ***purported responsible address*** is determined as the first from the following list of items which is both present and non-empty:

1. the first `Resent-Sender` header in the message, unless (per the rules of RFC2822) it is preceded by a `Resent-From` header and one or more `Received` or `Return-Path` headers occur after said `Resent-From` header and before the `Resent-Sender` header (see §3.6.6. of RFC2822 for further information on `Resent` headers),
2. the first mailbox in the first `Resent-From` header in the message,
3. the `Sender` header in the message, and
4. the first mailbox in the `From` header in the message.

Every well-formed e-mail message will contain at least one of the above headers. Any message that is sufficiently ill-formed as to not contain any of these SHOULD be considered very heavily suspect as spam (otherwise, spammers will just resort to omitting all of these headers). The particular listed order in which these headers are searched determines in a manner consistent with the defined semantics and usage of each a purported responsible address which is most immediately responsible for the transmission of a message. Roughly speaking (but see RFC2822 for details), these semantics can be characterized as follows:

`Resent-Sender: addr1@example.com`
: The message was previously delivered to the mailbox of its final destination, but it was subsequently reintroduced into the e-mail transport system, and `addr1@example.com` was the agent that actually carried out the reintroduction on behalf of the party listed in the `Resent-From:` header (which per RFC2822 MUST be present) whose desire it was that this occur.

`Resent-From: addr2@example.com`
: The message was previously delivered to the mailbox of its final destination, but it was subsequently reintroduced into the e-mail transport system, and `addr2@example.com` was the party whose desire it was to have the message reintroduced; that is, `addr2@example.com` is responsible for the message's reintroduction.

`Sender: addr3@example.com`
: `addr3@example.com` is the agent responsible for the actual transmission of the message. That is, it was `addr3@example.com` that actually caused the message to be injected into the e-mail transmission system. For example, if a secretary were to send a message for another person, the mailbox of the secretary would appear in the `Sender:` field and the mailbox of the actual author would appear in the `From:` field.

`From: addr4@example.com`
: `addr4@example.com` is the author of the message. That is, `addr4@example.com` is responsible for the writing of the message.

The ***purported responsible domain*** of a message is defined to be the domain part of the message's purported responsible address.

7.2.1. Implications for Mobile Users

Among the today's broad community of e-mail users are some who legitimately send e-mail from IP addresses which are not administratively connected with their home domain.

A common case is that of a mobile user, who may be using a laptop PC, an e-mail-enabled mobile phone, or other device. For example, suppose that Adam from `example.com` uses a portable e-mail messaging device device to

send much of his mail. Adam has an e-mail account `adam@example.com` that he wishes his recipients to think of as his 'actual' address. However, he also has an account `adam@consolidatedmessenger.com` with the system run by his mobile device provider, and, indeed, when he sends mail from his device the message travels directly to its destination using servers under the control of `consolidatedmessenger.com`; the servers of `example.com` are not involved. To make this situation function without appearing to have a spoofed domain, Adam should ideally (although it may not be technically possible today) configure his mobile e-mail messaging device so that the following headers appear in the message:

```
From: adam@example.com
Sender: adam@consolidatedmessenger.com
```

The purported responsible domain of Adam's mail is then `consolidatedmessenger.com`.

Many other mobile scenarios can be accommodated with an analogous approach, including that of Web-originated e-mail from greeting card sites and the like. That said, an alternate, different methodology is also applicable in many mobile scenarios. Since there is little legitimate point in having an e-mail address that can send mail but cannot receive it, there necessarily already exist servers which are administratively associated with the domains of each legitimate user, namely the servers which manage the receipt of the user's inbound mail. In many cases it is reasonable for mobile users to connect to these servers (or other administratively-related servers) to also relay their outbound messages. Adam might, for example, use his mobile e-mail messaging device in its alternate transmission mode wherein it communicates with his `adam@example.com` mailbox, which in turn sends the mail to its ultimate recipient in a manner indistinguishable from Adam having been sitting in front of his `example.com` mail reader in his home office. Where feasible, this approach is straightforward and is often to be preferred.

It is worth nothing in passing that it remains solely a decision for each individual domain such as `example.com` to choose whether or when to protect their reputation by taking the steps of §7.1. As more and more domains take such steps, spammers will necessarily migrate their spoofing to the remaining domains, which in turn may appear more and more suspect, but in the end each domain owner is empowered to act on their own behalf.

7.2.2. Implications for Mailing Lists

The core functionality provided by mailing list software is the provision of e-mail addresses (the address of each list) that take the mail received thereat and retransmit it on to the registered members of the list. In order to avoid the appearance of domain spoofing, such software SHOULD add a `Resent-From:` header to each retransmitted message to indicate that this action has been carried out. Alternately, a `Sender:` header MAY be used instead if one is not already present: indeed a number of existing mailing list agents already add such a `Sender:` header, and so do not need adjustment to conform to this proposal.

Continuing our example, if Adam sends mail from his mobile e-mail messaging device to the mailing list `asrg@ietf.org`, when the message is transmitted to the list members it might look in part like the following

```
Resent-From: asrg@ietf.org
Resent-Date: Tue, 16 Dec 2003 14:33:13 -0800
Received: from ... by ietf-mx.ietf.org ...
...
To: asrg@ietf.org
From: adam@example.com
Sender: adam@consolidatedmessenger.com
```

Because of the `Resent-From:` header, the purported responsible domain of this message is `ietf.org`.

Note that the `Resent-Date:` header is included here per the requirements of RFC2822. Also note that the `Resent-*` headers are prepended *earlier* in the message than the corresponding `Received:` header; this is necessary in order to reliably demarcate the possibly multiple groups of `Resent-*` headers that may accrete as the message makes its way along its journey.

7.2.3. Implications for Mail Forwarders

Today many mail forwarding services exist, wherein mail sent to one e-mail address will be automatically forwarded to a second address. The details of exactly how such services operate differ from service to service, but a widespread practice is to carry out the forwarding by retransmitting messages *verbatim*, preserving exactly both original the SMTP-level envelope information as well as the entire original message body. That is, in the existing

verbatim retransmission practice legitimate mail sent from any domain can be legitimately delivered to its recipient's systems from virtually any IP address, all in a manner entirely indistinguishable from the actions of a forger. If this behavior were to be preserved, any attempt to distinguish forgeries from legitimate mail would be problematic (see also the discussions of this issue in §10 of reference [2] and §3.4 of reference [3]).

Therefore, the practice of forwarding mail using verbatim retransmission is to be discouraged. We propose a very small modification to forwarding practices in order to allow forwarded mail to be distinguished from spam, namely that forwarding retransmission services SHOULD annotate the message as they carry out the retransmission action.

As before, this can be accomplished concretely by the inclusion by the forwarding agent of a new `Resent-From:` header in the message. Continuing our previous example yet again, suppose I subscribe to the ASRG mailing list using the address `bob@forwarderexample.com`, and further suppose that I have configured `bob@forwarderexample.com` to forward mail that it receives to `bob@example.com`. By the time Adam's message reaches me at this latter address where I actually read the mail, it's headers in part should look like the following.

```
Received: from ... by example.com ...
...
Resent-From: bob@forwarderexample.com
Received: from ... by forwarderexample.com ...
...
Resent-From: asrg@ietf.org
Resent-Date: Tue, 16 Dec 2003 14:33:13 -0800
Received: from ... by ietf-mx.ietf.org ...
...
To: asrg@ietf.org
From: adam@example.com
Sender: adam@consolidatedmessenger.com
```

The purported responsible domain of the message is now `forwarderexample.com`.

That all said, the legacy practice of verbatim retransmission can be accommodated to a limited degree at the expense of introducing narrow but entirely open holes in the forgery detection infrastructure. Fortunately, it is the case that most e-mail users are aware of the forwarding addresses that indirect to them, as they usually were involved in their establishment. In the above example, for instance, `bob@example.com` is likely aware of the existence of `bob@forwarderexample.com`'s forwarding behavior, since Bob probably set up the forwarding relationship in the first place. Given this state of affairs, it is not unreasonable that the user of the forwarded-to system be provided with mechanisms in his e-mail software by which the filtering actions which would otherwise be carried out for the forwarded mail may be overridden. Specifically, the original destination address (here `bob@forwarderexample.com`) might be listed as a "known recipient" for `bob@example.com`: if this address appears in the `To:` or `Cc:` lines of mail delivered to `bob@example.com`, then the message would not be subjected to normal forgery detection processing. The forwarded mail thus will get through; however, this can be exploited by spammers should they become aware of this relationship between these two addresses. Such controlled and focused disabling of the forgery analysis strikes a balance between accommodating legitimate legacy forwarding systems on the one hand and defeating domain-spoofing spammers on the other.

7.2.4. Implications Regarding Multiple Apparent Responsible Identities

Each of the four categories of identification of §7.2 used in determining the purported responsible address for a message has slightly different semantics, the details of which are articulated in RFC2822 (see "§3.6.2 Originator Fields" and "§3.6.6 Resent Fields"). It is quite possible that a given piece of received mail contains a different address or addresses within the headers of each of these different four categories. As has been seen, sometimes such a combination of identities is reasonable, while other times it is not.

The mechanisms and policies of §7.1 through §7.2.1 determine the purported party who was most immediately responsible for the transmission of a message and verify that the message has not been spoofed with respect to the domain of that party. Moving beyond this to consider the general relationship among the various possible categories of purported responsible address is generally beyond the scope of this proposal per se, though §7.4 does address one important case.. More properly such is a role of mail filtering and e-mail client software within a receiving system. That said, we do here offer some suggestions regarding how that might work.

Common historical practice in mail reading software regarding the mail originator and resent headers has been to present only the contents of the `From:` header to the users; the other related headers (`Sender:`, `Resent-From:`,

Resent-Sender:) have not been shown. This behavior SHOULD change. Messages with combinations of identities in the originator headers SHOULD be rendered differently than messages in which the identities are the same. Specifically, it is RECOMMENDED that if the purported responsible addresses of a message is not the same as the address that would be rendered as the From: address that both these addresses be exhibited to the user. For example, the message in the example from §7.2.3 might be presented by e-mail client software as being

> From bob@forwarderexample.com on behalf of adam@example.com

or

> From adam@example.com via bob@forwarderexample.com

instead of the historical

> From adam@example.com

Indeed, this suggestion is essentially a straightforward generalization of the behavior that exists today in some newer mass market e-mail clients with respect to the processing of From: and Sender: headers.

7.2.5. Summary of Implications

Examples of the implications of the various scenarios described in the immediately previous sections, as well as some additional ones, can be summarized in the following table.

Scenario	Action Required	Action Required By	Example Headers
Ordinary e-mail	Publish outbound IP addresses Perform caller id check	Sender Receiver	To: bob@cohowinery.com From: adam@example.com
Mobile user (User: sends mail over a 3rd party carrier network (e.g. consolidatedmessenger .com) but wishes mail to appear to be from his/her regular corporate account)	Add Sender: header	Sender's MUA or carrier network's MTA	From: adam@example.com Sender: adam@consolidatedmessenger.com To: bob@cohowinery.com
Mailing lists	Add Sender: or Resent-From: header	MTA associated with the mailing list	From: adam@example.com Sender: asrg@ietf.org To: asrg@ietf.org *- or -* Resent-From: asrg@ietf.org Received: from ... by ... ... Sender: adam@consolidatedmessenger.com From: adam@example.com To: asrg@ietf.org
Mail forwarding	Add Resent-From: header	Forwarding MTA	Received: ... by cohowinery.com ... Resent-From: bob@forwardexample.com Received: ... by forwardexample.com ... From: adam@example.com To: bob@forwarderexample.com
Web-generated e-mail (E-vites, E-greets, mail this news article to a friend, etc.)	Add Sender: header	Web application server	From: adam@example.com Sender: articles@dailynews.com To: bob@cohowinery.com
Anonymous e-mail	Add Resent-From: header	"Anonymizing" MTA	Received: ... by anonexample.com ... Resent-From: admin@anonexample.com <intermediate Received: headers removed> ...

			`From: randomname@anonexample.com` `To: bob@cohowinery.com`

Outsourced e-mail service provider	Use `<indirect>` element of E-mail Policy Document to point to outsourced outbound IP addressees	Sender	`From: sales@example.com`
			`To: bob@cohowinery.com`
			`Return-Path: admin@outsourcexample.com`

"Bring your own IP." (User sends mail using their ISP's SMTP network but has their primary e-mail account elsewhere.)	Add Sender header	Sender's MUA, or ISP's MTA	`From: adam@example.com`
			`Sender: adam@ispexample.com`
			`To: bob@cohowinery.com`
			`Return-Path: adam@example.com`

7.3. Correlation and Checking of Purported Responsible Domains

Once a purported responsible domain *p* for a message *m* has been determined, the IP address from which *m* was received should be compared with the IP addresses from which mail transmitted by the domain *p* are expected.

To that end, the XML E-mail Policy Document of *p* is consulted with the evaluation of *OutGoing(p)* to learn the set of possible IP addresses from which mail sent by this domain can be validly transmitted (note that an actual DNS lookup need not occur for each message: significant and even pro-active caching of this information can be carried out). This set of IP addresses is compared to the IP address of the SMTP client transmitting the message (as indicated by the actual TCP connection) in order to determine whether domain spoofing is occurring or not. The outcome of this determination then forms input to the receiving system's mail filter.

Moreover, if it is determined that spoofing is occurring then the receiving system SHOULD NOT (§6.1 of RFC2821 notwithstanding) send any delivery receipts or delivery status notifications with regard to this message. This policy avoids a form of denial-of-service attack wherein such error notification messages are maliciously sent to a third party.

7.3.1. Checking Purported Responsible Domains in E-mail Clients, Etc.

Complications arise when the verification of the legitimacy of a purported responsible domain is carried out in e-mail client software or other software not actually on the organizational edge (see also §12.2.2). Specifically, it is in general quite difficult for such software to know the IP address from which mail was actually delivered into the organization (and against which, of course, the domain check is to be made). Software running on the organizational edge owns the TCP connection in question, and thus can easily find this information; however, this is not true for other software.

In many non-edge cases, though, this missing address information can be gleaned through careful and judicious processing of `Received:` headers (see §3.6.7 of RFC2822 and §4.4 of RFC2821). As mail is relayed from SMTP server to SMTP server, `Received:` headers noting each hop are prepended to the beginning of the message. Therefore, for messages received into a given organization, there is in general some initial prefix of the `Received:` headers which were added by the organization itself, with the remainder of these headers following this prefix having been already present in the message before it relayed into the organization's edge SMTP server. Because they were added by the organization itself, it is reasonable that software within the organization can trust the headers in this prefix set. In particular, software can reasonably trust the last header in this prefix (the one added by the organization's edge SMTP server), which quite often has contained within it exactly the TCP-reported IP address needed to perform the caller id check.

The thorny difficulty, of course, lies in partitioning the `Received:` headers into the trustable prefix and the untrustable suffix. In general, this is not reliably solvable: there will always continue to remain deployment situations where this information simply is not available, and thus the forgery check cannot be performed by software within the organizational interior. However, various approaches can be used to reduce these situations to a minimum.

7.3.1.1. Look for your inbound mail servers

A first such approach seeks to locate `Received:` headers which indicate that they were added by the *inbound mail servers* of the domain in question. While the formal syntax of a `Received:` header is quite liberal, generally speaking a Received: header does actually contain both the SMTP server sending and the SMTP server receiving the message. The former is listed as the 'From' address, while the latter is listed as the 'By' address. For example (but see §4.4 of RFC2821 for details):

```
Received: from sendingDomain.com ([1.2.3.4]) by mx.receivingDomain.com; Tue, 4
Nov 2003 14:03:14 -0800
```

An interesting observation is the following. Suppose that within your organization you find a `Received:` header in a message where the 'By' address is one of your inbound mail servers (or, more accurately, where the set IP addresses resolved to by the indicated 'By' address has non-empty overlap with the set published inbound mail addresses for your domain per §5 of RFC2821). Then it is reasonable to assume *for the purposes of domain spoof checking* that all headers up to and including the first such header (call it header r_{e0}) were in fact added by your organization. That this is reasonable *for this purpose* can be seen as follows:

1. If your domain does indeed add such a header r_{e0}, then the header r_{e0} and all previous headers were added by an organization you trust, and so the content of the headers can also be trusted.
2. If your domain does *not* add such a header r_{e0}, then if present in the message it must have been provided by an attacker from outside the organization in an attempt, presumably, to defeat the caller id check logic. Note, however, that even if the attacker instead did nothing at all (that is, didn't add r_{e0}) that the caller id check would not in this scenario have been performed, due to lack of knowledge of the IP address from which the mail was delivered into the organization. Thus, the attacker cannot achieve any benefit by adding r_{e0} that he would not have received anyway.

Once r_{e0} is located, then immediately subsequent headers may also be attributed as being added by your organization so long as at least one of the following is true:

a. The header also has a 'By' address which is also one of your inbound mail servers
b. The header has a 'By' address which resolves to an IP address set which contains only non-Internet routable addresses (see RFC1918).

Let r_e be the last header added by your organization as determined by this algorithm. Then, if the 'From' address of r_e contains a TCP-reported IP address (per the `TCP-info` production of §4.4 of RFC2821), it is reasonable to believe that that address is the address from which your organization received the message, and to perform the caller id check on the message using that address as input.

It is worth repeating that the algorithm of this first approach is not reliable, in that there are many ways in which an organization's systems may be configured so as to prevent software inside the organization which parses `Received:` headers from finding the desired IP address. In particular, there historically has been little technical incentive to ensure that one's inbound mail servers are evident from one's added 'By' addresses, and so it should not be surprising that some domains are not configured in this manner. Though it is usually easy to rectify this situation, the next section presents an alternative approach which may also be pursued.

7.3.1.2. Look for something your domain administrator tells you to

Locating the `Received:` header added by your organizational edge would be trivially easy if two conditions were met.

1. There existed some distinguished character string *s* that all edge SMTP servers in your organization placed in the `Received:` headers that they add to incoming messages.
2. The string *s* was *not* present in the `Received:` headers added by any of the non-edge SMTP servers in your organization

If these two conditions hold, then the `Received:` header added by your organizational edge is simply the first `Received:` header which contains the distinguished string *s*. Moreover, in practice, it is often quite straightforward to arrange that these conditions hold, often amounting to little more than using existing administrative infrastructure on the edge servers in question to add some newly concocted distinguished string. Having located the `Received:` header added by the organizational edge, the IP address from which the message in question was delivered into the organization can be located as described in the previous section.

A less straightforward issue is that of how to communicate to software in the interior of an organization that wishes to perform caller id checks (for example, e-mail client software, or internal e-mail servers) whether these conditions hold and, if so, what the distinguished string *s* is. From an operational perspective, this could be carried out in an ad-hoc internal non-standardized manner, since, after all, all the computers in question are under the control of a single administrative entity. That said, having a standardized mechanism for disseminating this information is valuable, as it enables and supports interoperation of software from different vendors.

To that end, the following mechanism is defined. The E-mail Policy Document may contain zero or more elements of the form `ep/internal/edgeHeader`, each of which contains an arbitrary string. If a non-zero number of such elements is present, then the set of strings contained in the collective set of such headers denotes the distinguished strings that are found in the `Received:` headers added by the edge SMTP servers of this domain. That is, within the domain in question, the first `Received:` header which contains any of these strings should be understood to be the header which was added by the edge.

For example the following E-mail Policy Document indicates that the edge SMTP servers of the domain (which is presumably example.com) always add a `Received:` header which contains either the string "`>***example.com edge***`" or the string "`mx.example.com`" and that the `Received:` headers by SMTP servers inside the organization never contain either of these strings.

```
<ep xmlns="http://ms.net/1">
   <internal>
      <edgeHeader>***example.com edge***</edgeHeader>
      <edgeHeader>mx.example.com</edgeHeader>
   </internal>
</ep>
```

7.3.1.3. Parsing Received: headers

Formally, the specification of the syntax of `Received:` headers of Internet mail messages is found in §4.4 of RFC2821. Unfortunately, actual practice exhibits considerable deviation from this specification, almost (but not quite) to the point of being free-form text. Naturally, that presents difficulty in parsing them to locate the 'From' and 'By' addresses therein. In practice, the following heuristics to locating this information have been found to be workable.

1. If the first case-insensitive whitespace-separated word of the header isn't "From", then the header cannot be successfully parsed.
2. Find the first occurrence of the case-insensitive whitespace-separated word "By" before the first semicolon, outside of any parentheses, brackets and quotation marks. If none exists, then the header cannot be successfully parsed.
3. In between, if anything looks like an IP address, that's the 'From' address. An IP address, for these purposes, is four sets of digits, separated by periods, and optionally followed by a colon and another set of digits. Otherwise, if anything in between looks like a domain name, then *that's* the 'From' address. A domain name, for these purposes, is sets of domain-legal characters separated by periods. It must have a least one period, and its last character must be alphabetic. Otherwise, the header cannot be successfully parsed.
4. If the first word after the "By" looks like a domain name, then that's the 'By' address. Otherwise, the header cannot be successfully parsed.

7.4. Domains Which Only Send Mail Directly to Recipients

One important sub-case of the general issue discussed in §7.2.4 regarding multiple apparent responsible identities warrants special attention and consideration. As we explored previously, the life of an e-mail message as it flows from its original sender to its ultimate recipient can in general be a complex one, possibly involving many occurrences of the message being delivered to a mailing list or forwarding service where it is updated and subsequently reintroduced and sent on to a different destination.

However, not all mail is subject to the generality of such an arbitrary pattern of flow. In particular, there is an important volume of mail, largely that of business-to-consumer correspondence involving billing statements, account maintenance, and the like, which will never legitimately be sent to mailing-lists. From the sender's perspective, such mail is being delivered directly to what they believe to be the ultimate recipient. If the

organizations which have this policy can communicate that fact to mail receivers, then receiving systems can with greater confidence apply significantly pejorative penalties to mail they receive which is in violation of the policy.

Domains which have this kind of policy can indicate so in their E-mail Policy Document. Specifically, a domain owner can indicate that mail from their domain only sends mail directly to their ultimate recipients by including on the `ep/out` element of their E-mail Policy Document a `directOnly` attribute with a true value.

A receiving system SHOULD interrogate this information once they have determined for a given message that domain spoofing is not taking place. Specifically, if the Caller ID check passes, and the purported responsible domain of the message is different than the domain of the first mailbox in the `From` header in the message (the "From Domain"), then the receiving system SHOULD retrieve the E-mail Policy Document of the From Domain. If the document exists and contains a `directOnly` attribute on its `ep/out` element which has a true value, then the receiving system can conclude that the message is in violation of the delivery policy of its sender. As a consequence, the receiving system SHOULD apply significantly pejorative penalties to mail, unless it has additional local knowledge (such as known recipient lists) that might reasonably override that behavior.

Note that the policy of sending mail directly to recipients can only be expressed on the granularity of a domain. As a result, domain owners who wish to avail themselves of this facility will need to separate their direct-only mail from the rest and use a different domain for each. For example, if Humongous Insurance wished to use this mechanism, they might use `humongousinsurance.com` for the direct-only correspondence they have with their customers and something like `corp.humongousinsurance.com` as the domain by which their employees conducted their business with outside partners and collaborators. Alternately, `humongousinsurance.com` might be set up as the employee domain, and designated subdomains such as `sales.humongousinsurance.com` and `support.humongousinsurance.com` could be used for the direct-only customer correspondence.

7.5. Security Considerations of Caller ID for E-mail

There are some subtle consequences of the approach presented here for providing caller id information for e-mail. These we now explore (in no particular order).

7.5.1. DNS Security Considerations

The use of DNS described in §7.1 warrants careful examination from a security perspective.

The DNS protocol has a simple request-response design. DNS requests and responses are traditionally transported using UDP, though TCP is also broadly supported (see RFC1035). All DNS communications are carried out in a single message format; the same format is used in both requests and responses. The top level format of the message is divided into five sections, some of which are empty in certain cases:

1. a header,
2. a question for the DNS server,
3. some resources records answering the question,
4. some resource records pointing towards other servers that might be of better assistance in answering the question, and
5. resource records holding additional information.

An important aspect of the DNS architecture is that answers may be cached by clients so that subsequent requests to the server may be avoided; the duration over which this caching may be permitted is indicated by the server as part of each answer, and is known as a "time to live" (TTL).

One particular piece of information in the DNS header is used to match up responses to outstanding requests: a 16-bit identification field is provided by the requesting client, and this value is copied into the corresponding response by the DNS server (the questions in the request are similarly copied to the response). If a client is careful in the manner in which it reuses values for the identification field, this allows several DNS requests to be issued in parallel from a given *source IP address, source UDP port* pair.

The same mechanism, however, provides opportunity for a straightforward hijacking attack, though one difficult to achieve in practice: if an attacker manages to deliver a matching response for an outstanding request before the arrival of the legitimate response from the DNS server, such a fraudulent response will be indistinguishable from the legitimate one. The primary impediments to carrying out this attack include providing the correct value of the

identification field, ensuring that the questions in the response match those in the request, and achieving the necessary packet timing.

If the DNS client software uses a good random number for the generation of the identification field, an attacker on average must attempt roughly 2^{16} responses differing significantly only in this field in order to be successful in achieving the correct value; all but one of these will be seen by the DNS client as ill-formed. Moreover, all these responses must be sent quickly, in order that they arrive before the legitimate response. These characteristics taken together provide a means of defense: DNS client software ought to monitor for such situations and consider themselves under attack if detected. Unfortunately, such sophistication is rare today.

Normally, it is virtually impossible for an attacker to successfully carry out a DNS hijack attack. In addition to the difficulty in matching the identification field value, attackers usually have no intrinsic means to know when a particular DNS client might happen to make a certain particular query request, so providing a corresponding fraudulent response before the DNS server does is very, very difficult.

However, if we are not careful, then depending on the particular usage scenario (see §12) the same may not always hold true of the use of DNS as articulated above for publishing the identity of outbound mail servers. Specifically, if an attacker attempting to send domain-spoofed mail to an SMTP server also carries out a DNS hijacking attack on the server, he can to some degree use the timing of his delivery of the mail in order to help estimate when the receiving infrastructure will query DNS for the outbound mail server information of the domain in question. Moreover, given the specification of XML E-mail Policy Documents, the attacker knows to a very high degree the nature of the questions in these query requests. By using this timing and question-knowledge information, the attacker has an easier time succeeding than would otherwise be the case.

That said, important mitigating factors should be noted. First, while the architecture of DNS does indeed admit attacks, it is unclear whether it is economical for a spammer to exploit them, especially on a scale of any significant volume. Second, unlike many spamming activities, this particular form of attack is a criminal act in many jurisdictions. Moreover, due to the nature of the DNS caching architecture, the execution of the attack tends to leave persistent evidence of its existence. Finally, the degree of vulnerability varies considerably depending on the particular usage scenario: for example, in the terminology of §12, scenarios R/I/1a, R/I/1b, and R/I/2 are of less concern than R/E/1, R/E/2, and R/E/3 because there is a lesser timing correlation with the actions of the attacker. Fortunately, in these latter scenarios (the edge ones), there is already an expectation of SMTP queuing and delay, which allows us to take certain countermeasures.

With these considerations in mind, the following additional mitigating approaches may be employed:

1. Applications SHOULD where possible use DNS client software that monitors for and guards against attacks.
2. Especially for DNS queries for XML E-mail Policy Documents issued against DNS servers that reside outside of one's organization, applications SHOULD where possible specify to their DNS client software that TCP-based DNS queries be used and UDP-based queries be avoided (note that this approach is also driven by size considerations). If this technique is used, then attackers attempting to spoof the result of the query face the additional burden of guessing TCP/IP sequence numbers used within the connection. This significantly decreases the feasibility of a successful attack.
3. SMTP servers receiving mail SHOULD introduce a random delay between the determination that the outbound servers for a certain domain are required and the issuance of the actual DNS query that requests their addresses. Though clearly a greater variability in this delay provides greater protection, even variability on the order of several seconds is very worthwhile.
4. As XML E-mail Policy Documents cached by a DNS client on behalf of an SMTP server expire under the TTL caching policy indicated by the server, they MAY be proactively refreshed by the SMTP server on a schedule independent of the further reception of mail from the domains in question. For these domains, this decouples the timing of the XML E-mail Policy Document DNS queries from the timing of transmission of mail from those domains, thus making attacks more difficult.

The benefit to be achieved in adopting any one or all of these mitigating approaches needs to be balanced against the costs and impediments each has to successful and broad deployment (indeed, for these reasons, use of DNSSEC is not among the list of recommended mitigations).

We must also keep in perspective that the consequence of a successful attack is simply that some mail that ought to be considered to be spam may fail to be detected as such and thus appear with more prominence to a user; that is, such consequences are considerably less severe than attacks in other situations, such as computer viruses.

7.5.2. SMTP Security Considerations

In using the IP address as reported by TCP as part of the process of eliminating domain spoofing described above, there is a small concern that TCP hijacking may successfully be able to forge this address. While such hijacking is not easy and is somewhat expensive to achieve, particularly with reasonable TCP/IP implementations which well-randomize their initial TCP sequence numbers and monitor for sequence number guessing attacks, it is unfortunately the case that the SMTP protocol is of the nature that one can accomplish significant mayhem (specifically, one can send mail) by hijacking just the *sending* side of the two-way TCP connection; this is in contrast to many other protocols where hijacking of both directions of the connection is effectively necessary in order to wreak havoc, a much more difficult attack to carry out.

In order to alleviate the concern of such an attack, we propose here a small enhancement to the SMTP protocol wherein the SMTP client demonstrates to the SMTP server that it has actually received some unique data returned previously by the server. With such enhancements in use, one-sided TCP hijackings are no longer effective against SMTP.

Because this enhancement operates at the SMTP level, it can be used without the need to *rely* on particular safety-conscious behavior of the perhaps varying underlying TCP implementations on which the SMTP software may be deployed, thus providing an increase in overall resilience of the system to attack.

Specifically, prior to sending any SMTP MAIL commands, an SMTP client SHOULD send to its server an SMTP NOOP command (see §4.1.1.9 of RFC2821) with a parameter containing the following:

- the 40 case-insensitive characters of the hexadecimal encoding of the SHA1 hash of the entire data provided previously by the server in its EHLO response (that is, the entire sequence of characters matching the `ehlo-ok-rsp` production found in §4.1.1.1 of RFC2821).

By including a random string as part of its EHLO response, an SMTP server can have reasonable confidence that, upon the receipt of such a NOOP command, the SMTP client is in fact receiving its responses, and one-side TCP/IP hijacking is not occurring. Note that, as a security consideration, the SMTP server provides no indication in response to the NOOP as to whether a correct hash value was provided.

Use or support of this enhanced NOOP command on the part of SMTP clients (and servers) is wholly OPTIONAL: absent the need to send this NOOP, no e-mail or DNS server software need be updated on the client side in order to for an SMTP client to participate in the design proposed here for publication of outbound mail server information; this simplicity is a significant factor that ought to foster its adoption. In order to preserve this characteristic while also allowing for enhanced NOOP support where possible, we provide a means by which the administrator of a domain can indicate whether any or all of its outbound mail servers support the enhanced NOOP. Such administrative indication on the part of the legitimate domain owner defeats an attempt by a potential hijacker to merely pretend that the domain he wishes to impersonate merely does not have enhanced-NOOP-aware software.

The administrative mechanism makes use of the optional `eNoop` XML attribute which may be found on `ep/out/m` elements in XML E-mail Policy Documents (see §13 below for details of this XML schema). This attribute, if present, indicates published knowledge of the behavior of particular outbound mail server(s) with respect to their using the enhanced NOOP; if the attribute is absent, then no information is provided as to whether the outbound servers will use the NOOP or not.

The `eNoop` XML attribute is interpreted as follows. Let T be the set of four enumerated values {*useUnknown, uses, doesntUse*}. Let *Meld*: $T \times T \rightarrow T$ be the symmetric binary function defined by the following table:

	useUnknown	***uses***	***doesntUse***
useUnknown	*useUnknown*	*uses*	*doesntUse*
uses	*uses*	*uses*	*uses*
doesntUse	*doesntUse*	*uses*	*doesntUse*

Let D be the set of all fully qualified domain names, IP be the set of all IP addresses, and A be the set of all XML attributes of Boolean type. Define *Uses*: $D \times IP \times A \times T \rightarrow T$ to be the function where *Uses(d, ip, a, t)* is computed as follows.

Let M_d be the set of all the `ep/out/m` XML elements m in the XML E-mail Policy Document of d such that ip is in the set *OutGoing(m,d)*. Then,

1. Let t_d be a variable initialized with the value *t.*
2. For each element *m* in M_d,
   a. If the XML child element `m/indirect` exists, then let $t_{indirect}$ be *Uses(d',ip,t)* where *d'* is the fully qualified domain name contained therein; otherwise, let $t_{indirect}$ be *useUnknown*.
   b. If the XML attribute *a* exists as a child of *m*, then let t_m be *uses* if the attribute is true and *doesntUse* if the attribute is false; otherwise, let t_m be *useUnknown*.
   c. Update t_d to be the value $Meld(t_m, Meld(t_{indirect}, t_d))$.
3. *Uses(d, ip, t)* is then defined to be the resulting value of t_d.

Given these definitions, whether or not the SMTP client at IP address *ip* which according to §7.1 is a legitimate outgoing mail server for domain *d* will use the enhanced NOOP functionality when connecting to SMTP servers is given by *Uses(d, ip, eNoop, useUnknown):* if that value is *uses*, then an SMTP client purportedly from domain *d* which omits the use of enhanced NOOP should be considered suspect.

8. E-mail Transmission Policies

Looked at from one point of view, the goal of this initiative is not to prevent or hinder people or organizations from sending spam, but rather to reward and be able to distinguish those who instead choose to adhere to socially reasonable, well-mannered etiquette of e-mail transmission.

In order to facilitate this, it helps considerably if some organization or organizations undertake the task of defining policies or codes of conduct with the aim of achieving broad consensus and acceptance as representing reasonable, 'non-spam' behavior. Generally speaking, it will commonly be the case that due to the practices they engage in spammers will be unable to comply with such policies,. Received mail that can be demonstrated to have been sent under such policies is thus likely mail that users actually want, not spam.

This approach has in analogy already been successfully carried out in other areas of Internet technology. For example, with respect to privacy issues surrounding information collected online, organizations such as TRUSTe (see reference [8]) and others have defined polices which have come to be widely appreciated as representing reasonable, appropriate and respectful behavior for the collection, management, and dissemination of personal information. There is every reason to expect that similar efforts can produce equally valued policies with respect to practices surrounding the sending e-mail.

Indeed, a number of such efforts are already significantly underway, though it is unclear at this time which organizations will in the end produce the most widely valued policies. It does not matter which (if any) predominates, as there is no need for a "one size fits all" solution. Rather, what is desired is that polices are articulated and seen to be adhered to that are appropriate for each particular Internet community by organizations trusted and valued by each. Such communities in different regions of the globe are likely to value and appreciate policies different than those valued by communities in other locations.

Thus, we in this proposal have relatively little to say regarding the details of exactly which policies of reasonable e-mail behavior might be seen as broadly valuable by any particular group. Rather, we focus here on means by which adherence to a particular policy, whatever it may be, may be communicated from the sender to the receiver during transmission of mail, and so form part of the enhanced set of inputs to the receiver's mail filter system. The coupling of a useful policy together with a means by which it can be seen to be adhered to forms a powerful basis for weeding out those who wish to be unreasonable in their e-mail transmissions.

That said, there is one class of policies worthy of explicit mention here, namely those policies which at their heart involve objective evaluations of the externally observable behavior of an e-mail sender: that (for example) they send e-mail only at a relatively low rate (not the thousands per second that spammers do), particularly to recipients with whom they have not previously corresponded, that they do not frequently send mail to large numbers of addresses, and so on. A distinguishing feature of this class of policies is that their adherence can be straightforwardly observed, audited and monitored by organizations other than those which are actually sending the mail. An E-mail Mailbox Provider, for example, might as a condition of service impose and enforce such a policy on the mailboxes that it hosts.

8.1. E-mail Transmission Policy Certificates

A technical data format is needed by which adherence to a particular policy of e-mail transmission can be indicated. This is accomplished using X509 certificates of a particular form. An E-mail Transmission Policy Certificate (known hereafter simply as an "ETP Certificate") is an X509v3 certificate with several specific characteristics, as listed in the following subsections.

8.1.1. Small

ETP Certificates SHOULD be issued with minimizing the size of the resulting certificate as an important goal. Many ETP certificates will be used in considerably high volume, and often as attestations regarding messages which themselves quite small. Both of these factors argue that X509v3 options and features which lead to large certificates should be avoided where possible: ETP certificates SHOULD be no larger than 1K bytes.

8.1.2. Certificate is authorized for sending e-mail

An ETP certificate MUST contain an Extended Key Usage extension (§4.2.1.13 of RFC3280) that contains at least the `id-kp-e-mailProtection` attribute (whose ASN object id is 1.3.6.1.5.5.7.3.4) and other aspects of the certificate are consistent with the restrictions imposed on the use of this attribute by §4.2.1.13 of RFC3280.

An ETP certificate MAY contain a Key Usage extension (§4.2.1.3 of RFC3280). If such extension is present, then at a minimum the extension MUST have its `digitalSignature` bit set. As an absence of a Key Usage Extension implies that all uses of the certificate are valid, an ETP certificate MAY instead omit such an extension.

8.1.3. Certificate contains indication of transmission policy

An ETP Certificate MUST contain an indication of one or more policies of e-mail transmission that the issuer of the certificate warrants that the subject of the certificate has agreed to conform with. Such policy or policies SHOULD be indicated by the use of appropriate object identifiers (and corresponding optional qualifiers) within a Certificate Policies extension in the certificate (see §4.2.1.5 of RFC3280).

Each policy of e-mail transmission SHOULD as part of its specification define a new unique object identifier for use in such an extension, and such identifier SHOULD be made known to all those systems who wish to certify or test for adherence to the policy in question. The definition of any qualifiers that may optionally accompany this identifier SHOULD also be part of the specification of the policy.

8.1.4. Certificate should contain revocation information

Should it prove necessary to revoke an ETP Certificate, it is useful and valuable that the certificate be capable of being revoked quickly and efficiently, as considerable amounts of spam may be transmitted by a wayward sender in a relatively short amount of time. Accordingly, it is RECOMMENDED that the need to potentially revoke an ETP Certificate be managed by an alternate means in addition to the traditional certificate revocation lists (CRLs), as the standards and mechanisms by which updates and changes to CRLs are disseminated introduce significant latencies and transmission sizes, thus hindering the rapid revocation of ETP Certificates. That said, for legacy deployment reasons, the publication of revocation information using CRLs SHOULD still also be supported

Specifically, it is RECOMMENDED that the Online Certificate Status Protocol (OCSP; see RFC2560) be used for managing the revocation of ETP Certificates. The Validate service of XKMS (see §3.3 of reference [12]) MAY also be used, as MAY a service indicated and described using the UDDI Business Registry (see reference [9]). In addition, any other means mutually agreed upon by the party revoking the certificate (usually the Certificate Authority or an agent acting on its behalf) and the certificate-verification-agent wishing to check for revocation (usually the receiving e-mail system) MAY be used.

In the XKMS, OCSP, and UDDI cases, the location of the service in question SHOULD be indicated in an Authority Information Access extension (§4.2.2.1 of RFC3280) in the certificate. The details of this extension vary from case to case:

1. In the OCSP case the details of the `AccessDescription` SHOULD be as indicated in §4.2.2.1 of RFC3280.
2. In the XKMS case the `accessMethod` of the `AccessDescription` SHOULD contain the value `id-ad-xkms-soap12-http` defined below, and the `accessLocation` SHOULD contain a URI at which the XMKS service may be contacted using the XKMS v2 SOAP 1.2 binding over HTTP.
3. In the UDDI case, `accessMethod` SHOULD contain either the value `id-ad-uddiv2` or `id-ad-uddiv3` defined below, and the `accessLocation` correspondingly contain either a UDDI version 2 `serviceKey` or a UDDI version 3 `serviceKey` that is to be resolved in the global UDDI Business Registry.

The values of the identifiers mentioned are as illustrated in the following ASN.1 fragment:

```
id-ad-xkms-soap12-http        OBJECT IDENTIFIER ::= { id-csri 1 }
id-ad-uddiv2                  OBJECT IDENTIFIER ::= { id-csri 2 }
id-ad-uddiv3                  OBJECT IDENTIFIER ::= { id-csri 3 }
id-csri                       OBJECT IDENTIFIER ::= { id-microsoft 2 1 101 }
id-microsoft                  OBJECT IDENTIFIER ::= {
       iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1)
       microsoft(311) }
```

Interoperability considerations arise from the decisions made by certificate authorities and certificate-verification-agents as to the revocation infrastructure they are capable of using. In the normal course of events, as part of the process of issuing a certificate, a certificate authority will indicate the means, protocols, and locations by which a revocation of the certificate will later be disseminated, should the need arise: when such a revocation occurs, the revoking party clearly can only distribute this information through a certain set of services, and in order that

certificate-verification-agents can find these services, they need to be identified within the certificates themselves. However, a given set of services that may be indicated will only support a certain set of revocation-dissemination protocols: if none of the protocols supported by a given identified service are understood by a given certificate-verification-agent, then that agent will not be able to make use of the service.

Therefore, in order to reduce the degree to which this problem occurs, it is RECOMMENDED that all receiving e-mail systems that are conformant with the present proposal be capable of retrieving revocation information through either OCSP or certificate revocation lists (CRLs). Conversely, support for XKMS validation services and indirection through UDDI is OPTIONAL.

8.2. Applicability of ETP Certificates to an E-mail Address

In some situations, some of which are described below, it is necessary to indicate that the policies articulated in ETP Certificate as described in §8.1.3 in fact apply to a particular e-mail address. This is accomplished using *at least one* of the following techniques:

1. The certificate contains a `SubjectAltName` extension (see §4.2.1.7 of RFC3280) one of whose `GeneralNames` is an `rfc822Name` whose value is the e-mail address in question.
2. The distinguished name which is the subject of the certificate contains an `E-mailAddress` attribute (see §4.1.2.6 and §4.2.1.11 of RFC3280) whose value is the e-mail address in question.

Per both RFC3280 and §3 of RFC2632, the former approach is preferred over the latter.

This approach is the same as that used in S/MIME to indicate the applicability of a certificate to an e-mail address. Details may be found in §3 of RFC2632.

8.3. Applicability of ETP Certificates to a DNS Domain

In other situations, it is necessary to indicate that the policies articulated in ETP Certificate as described in §8.1.3 in fact apply to a particular DNS domain name. This is accomplished by using *both* of the following techniques (which mirror the techniques used in SSL/TLS Server certificates to indicate such domain name connection).

1. The first common name (CN) attribute contained in the distinguished name which is the subject of the certificate has a value which is the domain in question.
2. The certificate contains an Extended Key Usage extension (see §4.2.1.13 of RFC3280) is present in the certificate containing (at least) the `id-kp-serverAuth` attribute (whose ASN object id is 1.3.6.1.5.5.7.3.1), and other aspects of the certificate are consistent with the restrictions imposed on the use of this attribute by §4.2.1.13 of RFC3280.)

9. Per-Message Signing with E-mail Policy Indication

The receipt of a message digitally signed under the auspices of an ETP Certificate whose certified e-mail transmission policy is considered valuable by the recipient's system is likely a good indication that at least as originally transmitted, the message is not spam. This section specifies the technical details of how such messages are to be formed.

Before going further, however, it should be noted that it is RECOMMENDED that where possible the per-message signing mechanisms described here should be avoided in preference to the more space and time efficient connection-oriented policy indication detailed in §10. Signing of messages introduces what may be significant increases in message size and consequent message storage cost; therefore, its use should be avoided unless some significant offsetting value is actively being operationally achieved.

Moreover, signed messages on their own are, generally speaking, open to replay attacks by spammers wherein they forward the message on to possibly large lists of recipients of their choosing. Signing a message attests to properties associated with its point of origin, not properties associated with the system by which it was ultimately delivered. Accordingly, it remains valuable and interesting to used signed messages as described herein in conjunction with the Caller ID methodology of §7; the two are complementary technologies.

The signing technology used here is based on S/MIME Version 3 (see RFC2633). The S/MIME Version 3 Message Specification provides a rich framework for sending and receiving secure e-mail messages. The full generality of this richness is well beyond what is needed to achieve the aims of the present proposal. Accordingly, in order to increase both simplicity and interoperability we here specify a profile of S/MIMEv3 focused more specifically on our goals at hand.

The design of the per-message signing approach articulated here took into consideration several goals and constraints, among them the following.

1. The ability to leverage the installed base of e-mail clients to create conformant signed messages. In the present design, existing, deployed S/MIME-aware e-mail client software that allows its users to send clear-text-signed S/MIME messages can send conformant signed messages merely by using appropriate ETP Certificates in the signing process: no software changes are required. It is difficult to overstate the importance of this design goal to the success of this effort.
2. The ability of many deployed e-mail servers to avoid accidentally invalidating the signature of a signed message only if the message is formed in one of a limited set of signature formats. For such servers, S/MIME is usually among this set. In contrast, a newly-created signature format would not be; consequently, messages signed in such a format would not successfully survive transmission through these mail servers.
3. The preservation, to the greatest extent possible, of the ease with which e-mail can continue to be effectively used in receiving systems which are only intermittently connected (perhaps over dial-up, for example). Thus, the addition of information which must be transmitted out-of-band of the main message flow has been minimized. Moreover, the existing S/MIME practice of including certificates in-band with the transmission of each message has been retained (but see §9.3).
4. The avoidance here of the unnecessary re-specification (and attendant re-interpretation, re-implementation, and bug-for-bug compatibility testing) of numerous design decisions and enumerations already well-covered in existing signing specifications. Except where we are introducing new concepts, there is no need to till up well-trodden ground.
5. The recognition that signed messages will often not successfully survive transit through a great deal of the deployed mailing list agents. Many such mailing list agents, very often driven by operational business interests, routinely modify the body of messages that flow through them in order to append, for example, various sorts of advertising-related material. Some agents also support as value-add for their list subscribers various forms of content-format conversion, wherein they might offer to reformat HTML messages as plain text or strip suspect attachments before retransmitting the message to each recipient. Of course, such practices cannot be carried out while also maintaining the original message submitter's signature on the message.

The next several subsections go over the design in detail.

9.1. S/MIME Signing with E-mail Transmission Policy Indication

An S/MIME e-mail signed and sent with the intent of conveying an indication of the policy of reasonable e-mail behavior associated with the transmission is known an "ETP S/MIME Message". An ETP S/MIME Message MUST be an S/MIMEv3-compliant message with several specific characteristics, as listed in the following subsections

9.1.1. Use of multipart/signed

The ETP S/MIME Message MUST be of format `multipart/signed`, as is defined in §3.4.3 of RFC2633; to promote interoperability, the signature formats `application/pkcs7-signature` and `application/pkcs7-mime` MUST NOT be used.

Such a message consists of exactly two MIME parts: a first MIME entity that is to be signed (which is often, but not necessarily, a clear-text representation of the message), and a second entity containing the so-called "detached signature" thereon. The protocol parameter to the `multipart/signed` Content-Type MUST either be `application/pkcs7-signature` or `application/x-pkcs7-signature`, and SHOULD be the latter (we specifically require support for `application/x-pkcs7-signature` to accommodate existing deployed systems).

Similarly, the (detached) signature entity MUST be of either Content-Type `application/pkcs7-signature` or Content-Type `application/x-pkcs7-signature`, and SHOULD be the latter. It MUST consist of a Base64 encoding of an RFC2630-defined `ContentInfo` ASN.1 structure, which in turn MUST contain an RFC2630-defined `SignedData` structure. A `SignedData` is a container of zero or more `SignerInfo`s, one for each signature on the message (note that S/MIMEv3 only *requires* support for zero or one such signature, and we make no stronger requirements here).

9.1.2. Use of ETP certificate

The signature in at least one `SignerInfo` in the `SignedData` of the S/MIME message MUST be created under the auspices of an ETP Certificate (`SignerInfo`s created under the auspices of non-ETP certificates are ignored for the purposes of evaluation of e-mail transmission policy).

9.1.3. Proof of freshness indication in lieu of / in addition to revocation check

For usability and other reasons, certificates that are issued for use in e-mail tend to be relatively long lived, often having valid lifetimes on the order of months or years. Should a certificate be revoked by its issuer or an agent thereof prior to its expiration, that information needs to be communicated to systems wishing to rely on the certificate. As was discussed in §8.1.4, to assure themselves that they have up-to-date revocation information, such systems classically poll or otherwise subscribe to information published by the revocation services indicated in the various certificates they wish to rely on.

An alternate approach, often called 'proof-of-freshness', can provide the same assurances, but shift some of the cost of verifying the lack of revocation from the system wishing to rely upon the certificate to another party, such as the system (in our case here) that originates some signed mail. The basic idea of proof-of-freshness is that this other party contacts the revocation service in the normal way, obtains a signed result therefrom which attests to the lack of revocation as of the moment of query, and then disseminates this signed result to those systems that wish to know. In the relying systems, what would have necessitated a communication with the revocation service now instead requires only a verification of the signature of and trust in the presented signed freshness result. This is both operationally cheaper and has better usability in intermittently connected receiving systems.

Being able to shift the burden-of-proof as to lack of revocation from the receiver to the sender of mail thus seems very worthwhile. To that end, we note the following. There are two validation services, XKMS and OCSP, whose query results are most pragmatically useful in conveying proof-of-freshness. An XKMS validation result is an XML element `ValidateResult` in the XMKS namespace (see reference [12]). An OCSP validation result is an ASN.1 `OCSPResponse` structure defined in RFC2650; this can be encoded in XML using the XML element `ocspResponse` defined in §13. Each of the XML elements `ValidateResult` and `ocspResponse` can in turn be used in of `KeyInfo` XML elements defined by XML-DSIG, specifically inside the `KeyInfo\X509Data` container (where it architecturally may be a sibling of other similar information such as CRLs; see reference [13]).

XML-DSIG-defined `KeyInfos` can be placed inside the `UnsignedAttributes` or `SignedAttributes` field of a given `SignerInfo` (§5.3 of RFC2630) of the `SignedData` (§5.1 of RFC2630) comprising the signature of the message using an `xmldsKeyInfo`, which is defined as follows:

```
xmldsKeyInfo ATTRIBUTE ::= {
       WITH SYNTAX XmldsKeyInfo
       ID id-xmlds-keyinfo
       }

XmldsKeyInfo ::= XmlElement   -- an XML DSIG KeyInfo element
XmlElement   ::= UTF8String   -- arbitrary XML element

id-xmlds-keyinfo OBJECT IDENTIFIER ::= { id-csri 5 }
```

When proof-of-freshness is desired to be used to vouch for the credentials used to sign an ETP S/MIME Message, a `KeyInfo` containing the freshness result SHOULD be placed in the `SignedAttributes`. If the logistics and timing of the generation of signature and the generation of the proof-of-freshness are such that this is not possible (for example, if the proof-of-freshness is not available before the signature is made), the `KeyInfo` MAY reside in the `UnsignedAttributes` instead.

9.2. Publication of Policy

A domain's policy as to whether or not one of its outbound mail servers either does or does not always send signed mail may be indicated by making use of the Boolean-valued `ep/out/m/@allETPSigned` XML attribute within the domain's XML E-mail Policy Document. Absent such information, it must be assumed that outbound mail may or may not be signed.

Receipt of mail which is not signed apparently from a mail server known to always send signed mail is a likely indication of foul play.

9.3. Implementation Considerations

Several points are worthy of note in the context of efficiently processing received ETP S/MIME messages.

A first observation is the fact that `SignedData` elements that comprise S/MIME signatures and the certificates that may be found in them present significant opportunity for compression. The formats in question were not designed with space efficiency being a primary goal; if the resource cost of storing large numbers of these messages is a significant concern, much can be done wholly within such storage systems to reduce costs using S/MIME-specific compression techniques.

Moreover, as, generally speaking, many messages may be transmitted over time to a given destination domain under the auspices of a given certificate, another opportunity for compression arises from avoiding the redundant storage of the repeated parts of the `SignedData` elements across these various messages, in particular the certificates contained therein. That is, if a given sender sends, say, a couple of hundred messages to (perhaps different) recipients within a given domain, the certificate chains used in each of these messages is likely to be the same. Moreover, two or more senders from a given sending domain will commonly differ in only the end-entity certificates they use to sign their messages; the certificate for the certificate authority issuing those certificates as well as certificates superior to that of the certificate authority are likely to be in common. Thus, significant space savings can result if the receiving domain internally avoids storing duplicate copies of such certificates.

It is therefore RECOMMENDED that e-mail systems significantly concerned about the resource cost of storing large numbers of S/MIME messages implement an internal storage system whereby the `certificates` member of the `SignedData` comprising the S/MIME signature is decomposed into its constituent certificates; each such certificate, identified uniquely by its cryptographic hash, its (issuer name, serial number) identity, or other appropriate mechanism is then stored only once within the storage system. When a given S/MIME message is later retrieved from the storage system, the reverse construction is applied, thus limiting the knowledge of the use of this optimization to just the storage system itself.

Another observation worth noting is that while validation of a certificate chain may perhaps in relative terms be considered computationally intensive, the results of such computations can be cached, thus amortizing the cost

over subsequent validations of the same certificate chain or suffix thereof. This is particularly true if revocation checking is factored out of the validation computation: the non-revocation-aspects of the validation can usually be cached for a significantly longer interval than the validation as a whole, with revocation included. Thus, a two tier validation system which combines relatively short-lived revocation-check-caches with longer lived caches of the results of validation exclusive of revocation checking is likely to be worthwhile. Moreover, since the validation of a certificate chain conceptually involves the validation of the first certificate in the chain, then the recursive validation of the suffix of the chain exclusive of the first certificate, and since (as was mentioned in the previous paragraph) many certificate chains will share a common suffix, benefits can be achieved by caching the results of validating each suffix of a chain instead of simply each chain taken as a whole.

A final potential implementation concern involves the computational cost of performing perhaps vast quantities of public key decryption operations in order to validate vast quantities of signed e-mail. Installations for which this concern is a significant one ought to investigate the use of hardware accelerators for these functions. As of the time of writing (mid-2003), PCI cards costing just a few hundred dollars US yet capable of more than 20,000 1024-bit RSA operations per second are available off-the-shelf from vendors such as Cavium Networks and their competitors.

10. Per-Domain Indication of E-mail Transmission Policy

Recall from above that the situation to be addressed here is that where all of the mail transmitted from a given domain purports to conform to a certain policy. We seek therefore a connection-oriented mechanism by which this information can be conveyed, rather than the message-by-message expense of signed S/MIME.

The specific mechanism selected here is reminiscent of various list-maintenance protocols historically deployed on the Internet in the ongoing quest to deter spam. Here, the mechanism is used to maintain what we term "conformant domain" lists. The idea is that the membership in the list represents adherence to a certain set of policies. The exact set of policies associated with a given list is specified out of band; at a technical level, the *name* of the list summarizes and captures the totality of these policies and thus the totality of what it means to be a list member. By placing a domain on a list, the administrators of the list assert that the domain in question adheres to the list's policies.

At a technical level, the name of a conformant domain list is a fully qualified domain name.

To see if a given domain *d* is on a given list *list*, one simply forms the DNS request which asks for the A resource records of the following domain:

`d`.query.`list`

For example, if one wishes to enquire whether the domain `example.com` is present on the conformant domain list `exampleList.org`, one should issue the query:

```
example.com.query.examplelist.org
```

A query to a conformant domain list can have one of three possible results:

1. If the address `127.0.0.10` is returned as (one of) the addresses in the A resource record set, then the domain *d* is present on the conformant domain list *list*.
2. Otherwise, if the address `127.0.0.255` is returned as (one of) the addresses in the A resource record set, then the domain *d* is absent from the conformant domain list *list*.
3. Otherwise (perhaps a communication error occurred, or the list was misconfigured) the status of the domain *d* with respect to its membership in the list *list* is unknown.

In both the first two cases, the results may be cached per the usual DNS conventions according to the TTL returned for the resource records in question.

To learn of e-mail transmission policies associated with a given e-mail message, one queries to see whether the purported responsible domain of the message (see §7.2) is a member of a conformant domain list whose opinion you value.

A means is provided in the E-mail Policy Document by which a domain administrator may list the conformant domain lists that it believes it is a member of. Specifically, the E-mail Policy Document may contain zero or more `ep/out/cdl` elements, each of which is a string containing the fully qualified domain name of a list. For example, the following E-mail Policy Document lists two such lists:

```
<ep xmlns='http://ms.net/1'>
   <out>
      <cdl>examplelist.org</cdl>
      <cdl>otherexamplelist.org</cdl>
   </out>
</ep>
```

By including in its E-mail Policy Document the conformant domain lists of which it believes itself to be a member, a domain can help increase the likelihood that its membership in (for example) a recently created list will be noticed by those who to whom it sends mail. However, this list of lists should be considered merely as a hint one can use to aid and optimize querying: the list of lists cannot be considered exhaustive, for there may indeed be conformant domain lists that add domains as members without the domain's knowledge; nor can the list of list can be considered authoritative: a list may of course remove a domain as member should it be seen to no longer conform to the list's policies.

11. Computational Puzzles for Spam Deterrence

One important fact must be mentioned regarding the certification approaches outlined in the immediately previous sections (§9 and §10), namely that achieving such certification is likely to be considerably expensive. There are several reasons for this, including the simple observation that certification authorities ought to be reasonably compensated for their investigatory and administrative services, as well as the fact that if such certifications are too inexpensive spammers will have little deterrence to attempting to fraudulently purchase large quantities of them. As a result, achieving certification will likely be beyond the financial wherewithal of a significant fraction of the smaller organizations which today (and in the future) wish to send e-mail.

11.1. Approaches for Smaller Organizations

Other approaches must therefore be found by which the mail sent by such smaller organizations can avoid being classified as spam by mail filtering technologies, one that does not involve such large up front cash expenditure.

We propose here two such techniques.

11.1.1. Relaying Through Certified Organizations

The first is that the smaller organization enter into a new business relationship with a larger, certified organization wherein the smaller organization relay its outgoing mail through computers controlled by the larger for the purposes of having the larger enforce some policy of reasonable e-mail behavior on it. Usually, such policies will be ones involving objective evaluations of the externally observable behavior of the mail sender, as was mentioned in §8.

From the point of view of the larger organization, its relationship to the smaller one is very much like that of a traditional E-mail Mailbox Provider who does anti-spam enforcement, except for the fact that it need not handle incoming mail, only outgoing. The larger organization ought to be able to provide these services to many smaller ones, thus being able to amortize the cost of certification across them and offer affordable prices.

A drawback of even this approach is that it requires the smaller organization to enter into new business and technical relationships that are presently not necessary, and thus shoulder the ongoing burden of the fees and expenses associated therewith. Our second alternative avoids these problems: it provides an approach which is essentially free to small organizations yet represents expenditures of real money to spammers. Because of the lack of additional cost to small organizations beyond today's practice, it is quite important that this alternative be present and broadly supported.

11.1.2. Solving Computational Puzzles

It is a fortunate coincidence that the computers of smaller organizations are typically lightly loaded. There are lots of 'wasted' CPU cycles: on average, the microprocessors in these computers have almost nothing to do. This is not true of the computers used by those whose large volumes of spam: their computers are busy almost 100% of the time doing the work of sending out their millions of messages. If we could by some means demonstrate that a substantial amount of CPU time was expended on behalf of the sending of one e-mail message, that would be a cost that could be borne easily by small organizations yet not by spammers. The expenditure of significant CPU time per message also inherently slows down the rate at which messages can be generated, which is in itself a deterrent to spam (remember, though, that not all messages need this treatment: if I know I am on your known-sender list, for example, there's no need for me to pay this price).

The techniques by which an e-mail sender can believably demonstrate that significant CPU cycles were expended on behalf of the transmission of a given e-mail message are based on the work of Cynthia Dwork and Moni Naor, as well as Adam Back. Generally speaking, they take the form of having the sender's computer solve certain computational puzzles that are expensive to solve yet efficient to verify. Dwork's & Naor's ideas can be found in reference [4]; Back's can be found in reference [1].

11.2. Definition of Computational Puzzles

The framework articulated by Dwork & Naor and independently by Back admits a wide variety of different computational puzzles that achieve the required characteristics. Within the general framework, however, there are two distinct classes of puzzles: those whose solution is CPU-bound, and those whose solution is memory-bound. Generally speaking, CPU-bound puzzles are as of the present writing both better understood as well as simpler

and considerably more straightforward to specify and implement. Accordingly, despite the fact that they suffer more than do memory-bound puzzles from variations in running time due to differences in processing power, we RECOMMEND that one specific CPU-bound spam-deterring puzzle be used. This we call the HashedPuzzle puzzle. It is defined as follows.

The HashedPuzzle puzzle *P* takes the following parameters as input:

1. an e-mail 'to address' *t*
2. a 'degree of difficulty' *n* for which a puzzle solution is sought
3. a message identifier *m*
4. an e-mail 'from address' *f*
5. a date *d*, and
6. a subject line *s.*

From these a document *D* is formed as follows. First the following six character strings are formed:

a. the representation of address *t*, whose syntax must be conformant with the `addr-spec` production from §3.4.1. of RFC2822 (note that this syntax contains embedded occurrences of `[CFWS]`, and so comments may be present: parse carefully).

b. a decimal integer representation of *n*

c. the identifier *m*, which must be conformant with the syntax

```
id-left "@" id-right
```

(which is that of a `Message-Id` but without the surrounding angle brackets or `[CFWS]`) where these non-terminals are taken from §3.6.4 of RFC2822

d. the representation of address *f*, whose syntax must be conformant to the syntax as was described in (a)

e. the representation of *d* in a manner conformant with §3.3 of RFC2822

f. the contents of the characters of *s.*

Each of these six strings is next escaped by replacing in each every occurrence of the '\' character (backslash) with the two character sequence '\\' and every occurrence of the ';' character (semicolon) with the two character sequence '\;'. The six escaped strings are assembled to form the document *D* by concatenating them together in order but separating each from the others with (five occurrences of) a string conformant with the syntax:

```
[FWS] ";" [FWS]
```

The following is a simple example of the result of applying these steps:

fred@example.com; 2;1c3a295$ae06$6a8c0@contoso.com; barney@contoso.com; Mon, 3 Nov 2003 21:37:03 -0800; Hi!

Given the sequence of bytes comprising a document *D*, the computational task involved in the puzzle is to find and exhibit a set of sixteen documents δ such that both of the following are true:

1. When each δ is prepended to the hash under the Son-of-SHA-1 hash algorithm *H* (see §11.7 below) of *D* with its whitespace removed and then hashed again to form $H(\delta \circ H(NWS(D)))$, the result is zero in at least the first *n* bits (taken most significant bit first within each byte taken in order). Here *NWS* is the function that takes a sequence of bytes as input, removes all those which are legal characters which could match the `FWS` production of RFC2822, and produces the remaining as output.
2. The last 12 bits of each of the documents δ are the same (the particular 12-bit suffix value shared by these documents does *not* matter).

That is, the answer to the puzzle *P(t, n, m, f, d, s)* is a set of 16 documents δ each with these characteristics. The hash *H(NWS(D))* is used as the suffix to which each δ is prepended rather than simply *D* in order to minimize the effect of variation in the length of *D* on the length of time required to solve the puzzle. Whitespace is stripped from *D* before being input to the hash in order to minimize sensitivity to the encoding of *D* in header fields where it may be subjected to folding.

No means other than brute force is known by which satisfactory δ can be located; however, that a given set of δ indeed answers the puzzle can be very quickly verified. Indeed, the particular brute force approach of first attempting all one-byte solutions, then attempting all two-byte solutions, then all three-byte solutions, and so on is as good of a solution algorithm as any other but has the additional benefit that solutions found will be as small as

possible. Furthermore, for puzzles with reasonable degrees of difficulty, solutions with four or fewer bytes will be the norm.

Among things of note with regard to this overall approach is the fact that it does not involve any additional back and forth exchanges between sender and receiver: it leaves the existing SMTP protocol completely untouched. The computational work of solving the puzzles can if desired be done speculatively on the part of the sender, without any communication with recipients; if the receiving software is not 'puzzle-aware', the extra headers will be safely ignored. One can conceptualize this characteristic as being a challenge / response system in which the "challenge" is known ahead of time and so need not be explicitly communicated from recipient back to sender.

The overall goal of this computational effort is to have the sender of the mail be seen (in analogy) to have purchased a 'ticket' to send mail from one specific sender address to a specific recipient address over a specific interval of time. The ticket is only good for a message with the corresponding actual headers; this effectively connects the ticket to a single message (attempting to connect the ticket to a single message in a more secure manner, such as by including a hash of the message body, has been found by experience to suffer from fragility, and is thus considered for the purposes of this proposal more trouble than benefit).

Note specifically that reuse cannot be carried out across multiple recipients, as each recipient will ignore puzzles that are solved against parameters that do not include its own address (which, of course, it knows). As for reuse within one recipient's mailbox (which is likely uninteresting to a spammer), the reuse of puzzle solutions can easily be automatically detected. Moreover, in the event that the automation of such duplicate detection is not done, the duplication of subject and date headers, which are typically shown to users, will likely lead to quick human disposal of the mail (again, discouraging the attempted duplication in the first place).

In actual practice, a sender of mail SHOULD for a given message solve the puzzles *P(t, n,m, f, d, s)* where

1. *t* ranges over the set of recipients of the message as communicated through the SMTP command line
2. *n* is appropriate for the domain of the recipient *t* as described below
3. *m* is the message-id of the message per §3.6.4 of RFC2822. If the message as submitted lacks a `Message-Id:` header, then the puzzle-solving system SHOULD add one before solving the puzzle.
4. *f* is the purported responsible address of the message as described above in §7.2.
5. *d* is the origination date of the message per §3.6.1 of RFC2822
6. *s* is the subject of the message per §3.6.5 of RFC2822

11.3. When to Solve Puzzles

It should be emphasized that it is not expected that computational puzzles will be solved for *all* outgoing mail.

First, mail originating from within organizations whose e-mail transmission policies are demonstrated by the mechanisms of §9 and §10 MAY omit the affixing of puzzle solutions thereto, as they will add little value, if any, beyond such attestations. Conversely, however, organizations that receive mail MUST NOT use the presence of puzzle solutions as the ONLY input to their mail filter that they draw from the present proposal: if computational puzzles are used, then the connection-oriented mechanisms of §10 MUST also be supported, and support for the mechanisms of §9 is STRONGLY RECOMMENDED. More discussion of conformance issues is found below in §12.

Second, as is described in the next section §11.4, senders of mail MAY avoid computing puzzle solutions when they have no reason to believe that they will in fact be valued by a particular recipient.

Moreover, once a sender comes to believe that it has been placed on a recipient's known-sender list, it need no longer solve puzzles when it sends mail to that particular receiver (thus, the solution of puzzles will likely be of use with first time or infrequent correspondents).

The use of known-sender lists will be especially important in the case of inter-organizational distribution lists. Unless a distribution list exploder can been seen by means of the mechanisms of §9 and §10 to be adhering to reasonable e-mail transmission policies (or, in the case of §9, the sender of the mail to the list can so be seen), the exploder is faced with the need to solve a computational puzzle for each member of the list. This is likely to generally be an unreasonably large task; as a consequence, mail sent to distribution lists will often arrive at the mailboxes of the members of the list with no related policy attestation or puzzle solution (though, with domain spoofing absent, it can still be known to have in fact been delivered from the distribution list, and not an imposter thereof). If the distribution list is on the member's known-sender list, however, the user experience will still be

normal. Therefore, it is RECOMMENDED that mail filtering systems include mechanisms by which distribution lists to which users subscribe are placed on their known-sender lists.

11.4. Specification of Desired Puzzle Parameters

11.4.1. Puzzle Parameters Desired by a Domain

The degree of difficulty *n* considered sufficient, along with the tolerable width of the interval *v* and whether *d* and *s* may reasonably be absent, and indeed whether the solving of computational puzzles is valued at all are each policy parameters that conceptually need to be decided upon by systems receiving e-mail and communicated to systems wishing to send it. In order to accomplish this, it is RECOMMENDED that receiving mail systems that evaluate whether a computational puzzle has been solved on an incoming mail message and take some action based thereon publish this fact in DNS entries for their domain; e-mail sending systems MAY assume that domains for which such DNS information is absent do not at all value the solution of computational puzzles, and so the expense of their computation can be omitted with no adverse effect.

To accomplish the publication of this information, the XML E-mail Policy Document of the DNS domain in question is used, specifically the `ep/in/hashedPuzzle` elements therein. Systems wishing to send mail to a given domain SHOULD consult such XML documents (if they exist) to learn the policy parameters the domain finds valuable with respect to spam-deterring puzzles.

11.4.2. Puzzle Parameters Desired by a Particular Receiver

An alternate means of disseminating this information is provided, one which is more point-to-point centric rather than domain-centric. This takes the form of the `E-mailPolicy:` mail header. Such a header is defined to be an unstructured header (see RFC2822) whose contents (the B & Q encodings etc. of RFC2047 notwithstanding) MUST be either a UTF-8 encoded XML document instance (if the characters therein are wholly US ASCII and so usable directly in a message header) or a Base64 encoding of such a document; the two cases can be distinguished according to whether the first non-whitespace character is '<' or not. As in the previous subsection, the XML schema definition which SHOULD be used for such XML documents is found in §13.

The semantics of such the `E-mailPolicy:` header are that of the sender of the message making a statement about e-mail policies associated with the particular e-mail address which is listed in the `From:` header of the message content. Such headers may be particularly useful for disseminating information about certain addresses within whose domain not all of the infrastructure for receiving e-mail has been upgraded to become aware of the details of this proposal. In particular, such headers are useful when only e-mail client software in a domain has been so upgraded.

The intended use of the `E-mailPolicy:` header is that supporting software will routinely and of a matter of course attach to its outbound messages a header which describes the policies adhered to by the system(s) used by the sender of such messages (indicated in the `From:` header). That is, information conveyed in `E-mailPolicy:` headers spreads 'virally' by catching a ride on mail that would be sent anyway.

By examining these headers, recipients of such messages may come to learn which particular policies are valued at that particular address, and thus be able to adapt accordingly their behavior when subsequently originating mail destined thereto.

It should be specifically and especially noted that `E-mailPolicy:` mail headers may be attached to non-delivery-receipts (NDRs), delivery-status-notifications (DSNs), and other error indications which might be sent in response to a received message that was determined to be spam by the receiver's mail filter. This proposal in no way *requires* such error indications to be sent, but if they are sent, the attachment of `E-mailPolicy:` mail headers thereto provides a convenient means by which the original sender can learn to correct its behavior.

11.4.3. Spammer Economics: Recommended Degree of Difficulty

Though as was just mentioned it is entirely the purview of each receiving e-mail system to indicate the particular puzzle parameters that the receiving system considers valuable, it is nevertheless constructive if that process of the selection of such parameters is carried out on a reasoned and rational basis. This section attempts to provide such analysis, and so offers guidance to administrators as to how to select reasonable puzzle parameterizations. While many approaches to the analysis are possible and valuable, we present here some that have proved particularly useful, concrete, and enlightening.

We proceed in two steps. First, we present a methodology for computing a certain amount of time per message that constitutes an effective spam deterrent. Second, that time-per-message is converted to a degree of difficulty parameter by means of inspection of the characteristics of our particular HashedPuzzle puzzle. It is interesting to note in passing that the methodological analysis is applicable to a broad range of spam-deterring-puzzle variations.

11.4.3.1. Computation Time Per Message

The key fact central to us here is the observation that the solving of a spam-deterring puzzle is *both* an expenditure of actual money (in the form of a fraction of the amortized yearly cost of a CPU) *as well as* the expenditure of a certain amount of wall-clock time. As there are only so many seconds in a day, these two ideas can be connected.

Consider the following four variables:

- *secondsPerMessage*, the amount of CPU time spent per message in solving the puzzle (actually of course this is per message per recipient, but for brevity in the sequel we will simply say "per message")
- *messagesPerResponse*, the number of messages that must be sent to get one actual successful response. For example, if only one in ten thousand spam messages generate a successful response, then the value of this variable is 10,000.
- *revenuePerResponse*, the amount of revenue generated by each successful response. In our analysis here, this should be thought of as the revenue net of both the actual cost of goods and any amortized overhead.
- *cpuCostPerYear*, the burdened cost of a CPU: the amount of money necessary to acquire, house, power, air-condition, maintain and administer a CPU for a year.

With those definitions in hand, we can straightforwardly calculate an upper bound on the maximum average profit that a spammer can make in a day. This calculation is illustrated in the following function (written in Maple 9; see reference [6]):

```
MaxProfitPerDay :=
proc(secondsPerMessage, messagesPerResponse, revenuePerResponse, cpuCostPerYear)
      local secondsPerDay, secondsPerYear, cpuCostPerSecond, costPerResponse,
            costPerMessage, grossProfitPerResponse, maxMessagesPerDay,
            maxResponsesPerDay, maxProfitPerDay;
      secondsPerDay  := 3600 * 24;
      secondsPerYear := secondsPerDay * (365 + 1/4);
      cpuCostPerSecond := cpuCostPerYear / secondsPerYear;
      costPerMessage := secondsPerMessage * cpuCostPerSecond;
      costPerResponse := costPerMessage * messagesPerResponse;
      grossProfitPerResponse := revenuePerResponse - costPerResponse;
      maxMessagesPerDay := secondsPerDay / secondsPerMessage;
      maxResponsesPerDay := maxMessagesPerDay / messagesPerResponse;
      maxProfitPerDay := maxResponsesPerDay * grossProfitPerResponse;
      return maxProfitPerDay;
end proc:
```

Very interestingly, the function collapses to a simple closed form:

$$\frac{86400\left(revenuePerResponse - \frac{1}{31557600} secondsPerMessage\ \ cpuCostPerYear\ \ messagesPerResponse\right)}{secondsPerMessage\ \ messagesPerResponse} \quad (1)$$

The curious may wish to note that 86,400 is the number of seconds in a day and 31,557,600 is (a close approximation to) the number of seconds in a year.

A crucial observation is the following: *if the average maximum profit per day is less than or equal to zero, then we can know that a spammer must be unprofitable*. We can compute the seconds-per-message threshold at which this condition arises by equating the above expression to zero and solving for the *secondsPerMessage* parameter in terms of the remaining three. This leads us to the following equation:

$$secondsPerMessage = \frac{31557600\, revenuePerResponse}{cpuCostPerYear \;\; messagesPerResponse} \qquad (2)$$

What this says is that given values in hand for the three variables *messagesPerResponse*, *revenuePerResponse*, and *cpuCostPerYear* that one considers reasonable, then the expenditure of an amount of CPU time per message beyond the threshold indicated in equation (2) is *guaranteed* to deter a profit-driven spammer. (Note that this equation is independent of any particular currency: it simply requires that *revenuePerResponse* and *cpuCost-PerYear* be denominated in the same units.)

This equation can be understood in one of two ways, depending on when one considers a response to have occurred. Specifically, *messagesPerResponse* is an indication of either

1. the *click-through* rate, the rate at which the content of spam messages are actively acted upon by their recipients, or
2. the *conversion* rate, the rate at which spam messages actually result in a sale or other transaction involving a transfer of money.

Depending on the viewpoint one chooses, the *revenuePerResponse* must of course be adjusted accordingly.

Rough real-world estimates for each of the two interpretations of the *messagesPerResponse* rate together with their corresponding profit levels can be gleaned from various and sundry sources, especially the many press interviews of spammers that have appeared. These sources indicate with high probability that the following are not unreasonable estimates for these parameters for the majority of spammers:

Parameter	***Approximate Value Range***
Click-through rate	1-in-2000 to 1-in-10,000
Revenue per click	USD$1 – USD$2
Conversion rate	1-in-50,000 to 1-in-100,000
Revenue per conversion	USD$20 – USD$80

In order to get a sense of the implications of what this means in concrete terms, if we assume for the moment that *cpuCostPerYear* is for the average spammer in the ballpark of USD$200, then using equation (2) we can conclude that:

- At a 1-in-2000 click-through rate, 78 seconds of computation deters $1 per click.
- At a 1-in-10,000 click-through rate, 31 seconds of computation deters $2 per click.
- At a 1-in-50,000 conversion rate, 78 seconds of computation deters $25 per transaction
- At a 1-in-100,000 conversion rate, 63 seconds of computation deters $40 per transaction.

All in all, given the uncertainty in our parameter estimates and the fact that our model here actually omits several factors of cost to the spammer, it seems entirely reasonable to believe that around **one minute** of computation is a very significant spam deterrent, and likely a level that will guarantee that a majority of spammers are simply not profitable. Moreover, for these parameter values the numbers illustrated are an absolute *upper bound*: it is very possible that significantly smaller thresholds are also quite effective deterrents.

It is worthwhile in passing to again recall that in deployed systems puzzles need not be solved if the sender of a message is on his recipient's known-sender list.

An entirely different "pure rate-limiting" analysis can also be used to justify a CPU-time-per-message threshold for spam deterrence. A number of published press reports indicate that successful high-volume spammers, using relatively high-end machines presumably running completely flat out, can send upwards of 650,000 messages per hour per CPU. Forcing each message to consume a minute of CPU would limit this to 60 messages per hour per CPU, which would be a reduction by a factor of more than 10,000 in the message transmission rate. Put another way: in order to maintain current levels of message transmission (and therefore current levels of income, profit, and lifestyle), at a minute per message such spammers would have to deploy nearly *two million* servers running at 100% CPU utilization instead of the approximately 200 servers they employ today.

Indeed, for these spammers, even just **10 seconds** per message (requiring 360,000 servers) or **1 second** per message (requiring 36,000 servers) would have a very significant impact on their current infrastructure costs and so also on the amount of spam they can profitably transmit. Moreover, it is worth remembering that with the

techniques of §7 in place that it is considerably more difficult than at present for spammers to surreptitiously enlist the aid of unsuspecting hijacked computers in their endeavor. Thus the increased infrastructure costs must be borne more or less directly by the spammer, though the exact impact is difficult to quantify or assess precisely.

In sum, it is RECOMMENDED that, using the above analyses together with other data (including their own observations), e-mail receiving systems should form their own informed opinion as to the amount of CPU expenditure they feel to be an effective spam deterrent. That said, it is further highly RECOMMENDED that, in the interests of minimizing the adverse impacts to legitimate senders of e-mail and in the face of the uncertainty involved in the analysis of spammer economics, receiving systems should initially request smaller CPU expenditures rather than larger ones (**one second is suggested**), and only increase their requested threshold should the smaller threshold prove by observed experience to be of insufficient deterrent.

11.4.3.2. Time to Degree-of-difficulty Conversion

It has been observed by way of experiment that on high-end mass-market personal computers as of the current writing (mid 2003) a degree of difficulty of approximately 9 corresponds roughly to a minute of CPU time. The running time is exponential in nature: for example, a degree of difficulty of 10 corresponds to two minutes of CPU time, a degree of difficulty 6 corresponds to about 10 seconds of CPU time, and a difficulty of 3 to about 1 second.

11.5. Presenting Puzzle Answers: The HashedPuzzle: Header

That the sender has solved the puzzle *P* with respect to a given recipient of a given message SHOULD be conveyed to the receiver by exhibiting both the document *D* and an encoding of the answer documents δ. This is accomplished using a combination of `HashedPuzzle:` and `HashedPuzzle-Suffix:` mail headers.

Both of these headers are structured headers in the sense of §2.2.2 of RFC2822. Their syntax is defined as follows (as before, grammar non-terminals not defined here are from RFC2822):

```
hashedPuzzle = “HashedPuzzle: ” [FWS] [puzzleSolutions] [FWS] “;” [FWS] docPrefix
puzzleSolutions = word; the base64 encoding of the solutions δ
hashedPuzzle-suffix = “HashedPuzzle-Suffix: ” docSuffix
docPrefix = unstructured; a prefix of the document D for which the puzzle is solved
docSuffix = unstructured; a suffix of the document D for which the puzzle is solved
```

Each of the sixteen answer parts δ is used to form the representation matching the `puzzleSolutions` production in the following manner. First, each byte sequence δ is prefixed with a one-byte datum whose value contains the number of bytes in δ itself. These resulting length-counted solutions are then concatenated in an arbitrary order and the overall byte sequence encoded in base64 to form the `puzzleSolutions` value (see §6.8 of RFC2045, especially the admonition at the top of page 25 noting that “Any characters outside of the base64 alphabet are to be ignored in base64-encoded data”, though in our situation here the semicolon will of course be interpreted as delimiter rather than being ignored). Though syntactically optional according to the grammar for reasons of future extensibility, the `puzzleSolutions` value will of course always be present in headers containing actual puzzle solutions.

If a message has multiple recipients, then generally speaking it will have multiple `HashedPuzzle:` headers, one for each recipient, as each recipient needs a different puzzle solution. However, the exhibition of the actual document *D* used in any given solution can be split between the `docPrefix` of the `HashedPuzzle:` header and the `docSuffix` of the `HashedPuzzle-Suffix:` which may also appear in the message. In this way, verbose common portions of the various documents *D* can avoid being repetitiously exhibited. In particular, it is useful to lift the recipient portion of *D* into the `docPrefix`.

At most one `HashedPuzzle-Suffix:` header may appear in a message. If present, then its `docSuffix` applies to all `HashedPuzzle:` headers in the message. Each document *D* is formed as the literal concatenation of the content (see §3.2.5 of RFC2822) found in a `docPrefix` and a `docSuffix`.

Dealing with BCC message delivery warrants special attention in order to avoid inadvertent information leakage. Specifically, a message containing a puzzle solution computed on behalf of a BCC’d recipient SHOULD be a bifurcated copy of the original message that is delivered to that recipient alone; in the original message, the `HashedPuzzle:` header targeted at the BCC’d recipient MUST be omitted. If for technical or other reasons

message bifurcation is not possible, then puzzle solutions for BCC'd recipients SHOULD simply not be transmitted.

11.6. Verifying Puzzle Answers

On the receiving side, for each message received and for each indicated recipient thereof, software should take

1. the purported responsible address from the message body (see §7.2), and
2. the recipient's address or addresses as understood by the receiving side's infrastructure and into whose mailbox the message is being delivered (this is the recipient address from the SMTP RCPT command, together with possibly other addresses known and trusted as aliases for this address)

then check to see if the headers of the message indicated that the puzzle *P* has been correctly solved for that sender-receiver pair. The date, subject, and message-id should also be checked for semantic correspondence with the actual e-mail headers received.

If the puzzle is correctly solved, the receiving system should then evaluate its opinion of the strength of that particular puzzle in deterring spam, and use that information as input to its mail filter. If instead an exhibited puzzle is found to be incorrectly solved, the receiving system MUST within its mail filtering infrastructure treat the associated message as if the errant exhibited puzzle solution were not present. That is, there **MUST NOT** be any penalty for sending what is ultimately perceived to be an erroneous or insufficiently difficult puzzle solution. This philosophy encourages the solving of puzzles without fear of inadvertently receiving a penalty instead of a benefit for doing so. It also makes the overall design more robust in the face of various message transformations that a message may be subject to but which the message sender cannot be aware of.

11.7. Son-of-SHA-1 Hash Algorithm

The Son-of-SHA1 algorithm is defined as a constrained perturbation of the SHA-1 algorithm (see reference [5] for the specification of SHA-1). The intent of defining a new hash algorithm unique to the proposed use of computational puzzles for spam reduction is to reduce the ease with which hardware accelerators can be applied to reduce the cost and duration of puzzle solving. Indeed, in conformant systems the Son-of-SHA1 algorithm MUST NOT be implemented in hardware.

In "§5 Functions Used" of the specification of SHA-1 (FIPS PUB 180-1), a set of eighty functions are defined that are subsequently used in the core of the algorithm specified in §7 and §8. Each f_t, 0 <= t <= 79, operates on three 32-bit words B, C, D and produces a 32-bit word as output

The Son-of-SHA-1 algorithm differs from SHA-1 only in the specification of these functions. Specifically, where SHA-1 specifies the eighty functions as follows:

f_t(B,C,D) = (B AND C) OR ((NOT B) AND D) (0 <= t <= 19)

f_t(B,C,D) = B XOR C XOR D (20 <= t <= 39)

f_t(B,C,D) = (B AND C) OR (B AND D) OR (C AND D) (40 <= t <= 59)

f_t(B,C,D) = B XOR C XOR D (60 <= t <= 79)

the Son-of-SHA-1 algorithm instead specifies the first of them as involving an additional XOR operation:

f_t(B,C,D) = g(B,C,D) XOR ((B AND C) OR ((NOT B) AND D)) (0 <= t <= 19)

f_t(B,C,D) = (B XOR C XOR D) (20 <= t <= 39)

f_t(B,C,D) = (B AND C) OR (B AND D) OR (C AND D) (40 <= t <= 59)

f_t(B,C,D) = (B XOR C XOR D) (60 <= t <= 79)

The supporting function g (B,C,D) is defined as follows:

g_t(B,C,D) = n(r(m(B,C), m(C,D)))

The binary function m() takes two 32-bit words as input and produces a non-negative 64-bit integer as output by concatenating the two 32-bits words together with the first word forming the high-order bits of the result:

m(B,C) = (B << 32) OR C

The unary function n() takes a single 64-bit integer as input and returns the word consisting of the lower 32 bits thereof.

n(x) = x AND FFFFFFFF

Finally, the binary function *r*() takes two 64-bit integers as input and computes the 64-bit integer which is the remainder of the first when divided by the second (unless the latter is zero). Specifically, *r*(*x*,*y*) is defined by the following relations:

If $y \neq 0$: $x = k\,y + r(x,y)$ for some non-negative integer *k*, where $0 <= r(x,y) < y$

If $y = 0$: $x = r(x,y)$

In all other ways, the Son-of-SHA-1 algorithm is identical to SHA-1. The following are examples of specific test data and together with their corresponding Son-of-SHA-1 hash output:

Test Data	***Son-of-SHA-1 Hash Output***
the string "abc"	`EBF90F28 917D0F67 A0994009 290FAC95 A0B32507`
the string "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq"	`76BD1E7E CB11C355 EB9AC016 FC5C2D29 9C9617DB`
a string consisting of 1,000,000 a's	`37362E7A 6C3339B4 3FC135CC 0E2E9707 9C2E53D2`
an empty string	`C60F381D 0342B6DE 22C66FE1 C37968BC D7D97C08`

12. Promoting Interoperability: Conformance Levels

This proposal details several technical features and mechanisms of which senders and receivers of mail can avail themselves in order to help reduce the overall worldwide volume of spam. As has been seen, there is a good deal of flexibility within which senders of mail may choose one or possibly several of these features and mechanisms to attest to the non-spam nature of the mail they send. Correspondingly, receivers of mail also have flexibility as to which of these features and mechanisms they value as inputs to their mail filtering software. Given this broad flexibility, unless we here provide concrete guidance and otherwise limit the number of ways in which the various features can be mixed and matched, overall system interoperability will significantly suffer, as will our collective effectiveness in deterring spam. To that end, this section articulates several specific conformance levels of feature combination.

The particular feature combination choices which may be reasonable in either the sending or the receiving case depend on the nature of the software in which (respectively) the attestations and filtering occur. Though there are many possible actual configurations, an important distinction is that between software which is on the edge of an organization, either sending or receiving mail, and other software which is in the organizational interior, such as e-mail client software. A primary difference between these two is that the edge software is in direct contact with the organization on the other side, whereas interior software is only in indirect contact, and so has to communicate through what are perhaps older, legacy components which may mask certain information.

In sum, there are thus four important groups of software which will have interest in this proposal:

1. edge software that sends e-mail,
2. interior software that sends mail,
3. edge software that receives mail, and
4. Interior software that receives mail.

This section defines conformance levels of feature combination for each of these groups. Note that the requirements in each of these conformance levels are additional ones over and above those mentioned previously throughout this specification.

12.1. Software That Sends E-mail

12.1.1. Sending Software on the Organizational Edge

For software on the organizational edge which sends mail (which are necessarily SMTP clients in the terminology of RFC2821), there exist the following conformance levels that are consistent with this proposal.

Conformance Level S/E/1: Publication of Outbound Mail Servers

At this minimum conformance level for sending software on the organizational edge, the identity of an organization's outbound mail servers MUST be published in DNS in the manner described in §7.1.

In addition, at this conformance level, the enhanced NOOP functionality of §7.5.2 SHOULD be employed on the part of the SMTP client, though this is *not* REQUIRED.

Conformance Level S/E/2: Mail Sent Seen not to be Spam

At this conformance level, sending software on the organizational edge MUST conform to all the requirements of level S/E/1.

In addition, at this conformance level, such software MUST annotate the mail that it transmits with its choice of one or more of the mechanisms described in §9, §10, and §11 in order to attest as to its non-spam nature. The particular choice of mechanism or mechanisms is up to the software and the transmitting organization on whose behalf it operates. The mechanisms will, generally speaking, vary according to the nature and needs of the organization.

12.1.2. Sending Software within the Organizational Interior

Generally speaking, updates to e-mail client and other mail-sending software deployed within an organizational interior often have difficulty in effecting changes to the exterior-facing view of the organization. For example, it is logistically difficult by means of the installation of an update to e-mail client software to cause changes in the

organization's DNS structure. Accordingly, the conformance levels regarding mail-sending software not on the organizational edge are somewhat less stringent.

Conformance Level S/I/1: Mail Sent Seen not to be Spam

At this conformance level, sending software within an organizational interior MUST annotate the mail that it transmits with its choice of one or more of the mechanisms described in §9, §10, and §11 in order to attest as to its non-spam nature, though of these §9 and §11 are easier than §10 for this category of software. The particular choice of mechanism or mechanisms is up to the software and the users and / or organization on whose behalf it operates. The mechanisms will, generally speaking, vary according to the nature and needs of the users and / or organization.

12.2. Software That Receives E-mail

The conformance levels for software receiving e-mail are somewhat more involved than on the sending side.

12.2.1. Receiving Software on the Organizational Edge

For software on the organizational edge which sends mail (which are necessarily SMTP servers in the terminology of RFC2821), there exist the following conformance levels that are consistent with this proposal.

Conformance Level R/E/1: Execution of Caller ID Check

At this minimum conformance level for receiving software on the organizational edge, the legitimacy of domain identity of the sending organization MUST be verified using the caller id mechanism described in §7. The results of such evaluation MUST be capable of affecting the behavior of the mail filtering infrastructure within the receiving system.

In addition, at this conformance level, the enhanced NOOP functionality of §7.5.2 SHOULD be employed on the part of the SMTP server, though this is *not* REQUIRED.

Conformance Level R/E/2: Evaluation of Per-Domain Policies and Computational Puzzles

At this conformance level, receiving software on the organizational edge MUST conform to all the requirements of level R/E/1.

In addition, at this conformance level, such software MUST also check for and evaluate its opinion of any per-domain e-mail transmission policy indications published by the sending organization in the manner of §10. The results of such evaluation MUST be capable of affecting the behavior of the mail filtering infrastructure within the receiving system. The rationale for this clause lies in the fact that for the organizations whose policies allow them to avail themselves of it, the approach of §10 is by far the most efficient to deploy. In particular, it is expected that many relatively large corporations and E-mail Mailbox Providers will use this mechanism, perhaps as the *only* mechanism by which they attest to the non-spam nature of the mail they send. It is important that such an approach be a reasonable option for such organizations

In addition, at this conformance level, receiving software on the organizational edge MUST check for and evaluate (using minimum degree-of-difficulty and other puzzle parameters of its own choosing) its opinion of any computational puzzle solutions which may be present per §11 in e-mail that it receives. The results of such evaluation MUST be capable of affecting the behavior of the mail filtering infrastructure within the receiving system. It is STRONGLY RECOMMENDED that the particular degree-of-difficulty and other puzzle parameters valued by the system in such evaluation be published in DNS in the manner described in §11.

Computational puzzles are important as they are the *only* means by which the lowest end of e-mail senders can reasonably attest to the non-spam nature of the mail that they transmit. It is thus critically important that such attestations are broadly valued. And, as verifying that puzzles have been solved correctly is simple and cheap for receiving software to carry out, requiring such support at this conformance level is not a significant burden.

Note that the above three clauses are *not* separable within this conformance level. In particular, an implementation which supports just one of connection-oriented evaluation and computational-puzzle evaluation is *not* consistent with this proposal.

Conformance Level R/E/3: Evaluation of S/MIME-Conveyed Policies

At this conformance level, receiving software on the organizational edge MUST conform to all the requirements of level R/E/2.

In additional, at this conformance level, such software MUST also check for and evaluate its opinion of any S/MIME-conveyed e-mail transmission policy indications which may be present per §9 in messages that it receives. The results of such evaluation MUST be capable of affecting the behavior of the mail filtering infrastructure within the receiving system.

Of all the mechanisms defined here, the signed e-mail approach is arguably the most flexible and robust to the various scenarios in which e-mail is used throughout the world. Broad support for this most general mechanism is thus very valuable.

12.2.2. Receiving Software within the Organizational Interior

As with sending software within an organizational interior, the conformance levels for receiving software within the organizational interior are somewhat less stringent due to difficulties which would result from imposing necessary interactions with an organization's administrative infrastructure.

Conformance Level R/I/1a: Execution of Caller ID Check

At this conformance level for receiving software within the organizational interior, the legitimacy of domain identity of the sending organization MUST be verified using the caller id mechanism described in §7. The results of such evaluation MUST be capable of affecting the behavior of the mail filtering infrastructure within the receiving system.

Depending on the particulars of the infrastructure of the receiving system, this functionality may be difficult to achieve in practice, as determining the IP address used to send a message across the inter-organizational SMTP hop can be difficult.

Conformance Level R/I/1b: Evaluation of Computational Puzzles

At this conformance level, receiving software within the organizational interior MUST check for and evaluate (using minimum degree-of-difficulty and other puzzle parameters of its own choosing) its opinion of any computational puzzle solutions which may be present per §11 in e-mail that it receives. The results of such evaluation MUST be capable of affecting the behavior of the mail filtering infrastructure within the receiving system. It is RECOMMENDED that the degree-of-difficulty and other puzzle parameters valued by the system in such evaluation be routinely published in e-mail policy headers on outbound messages in the manner described in §11.

Conformance Level R/I/2: Evaluation of S/MIME-Conveyed Policies

At this conformance level, receiving software within the organizational interior edge MUST conform to all the requirements of level R/I/1b.

In additional, at this conformance level, such software MUST also check for and evaluate its opinion of any S/MIME-conveyed e-mail transmission policy indications which may be present per §9 in messages that it receives. The results of such evaluation MUST be capable of affecting the behavior of the mail filtering infrastructure within the receiving system.

13. Related XML Schema

As was described above, the element ep from the following XML schema should be used to indicate e-mail policies associated with a given DNS domain or e-mail address. Example instances of this element follows the schema specification itself.

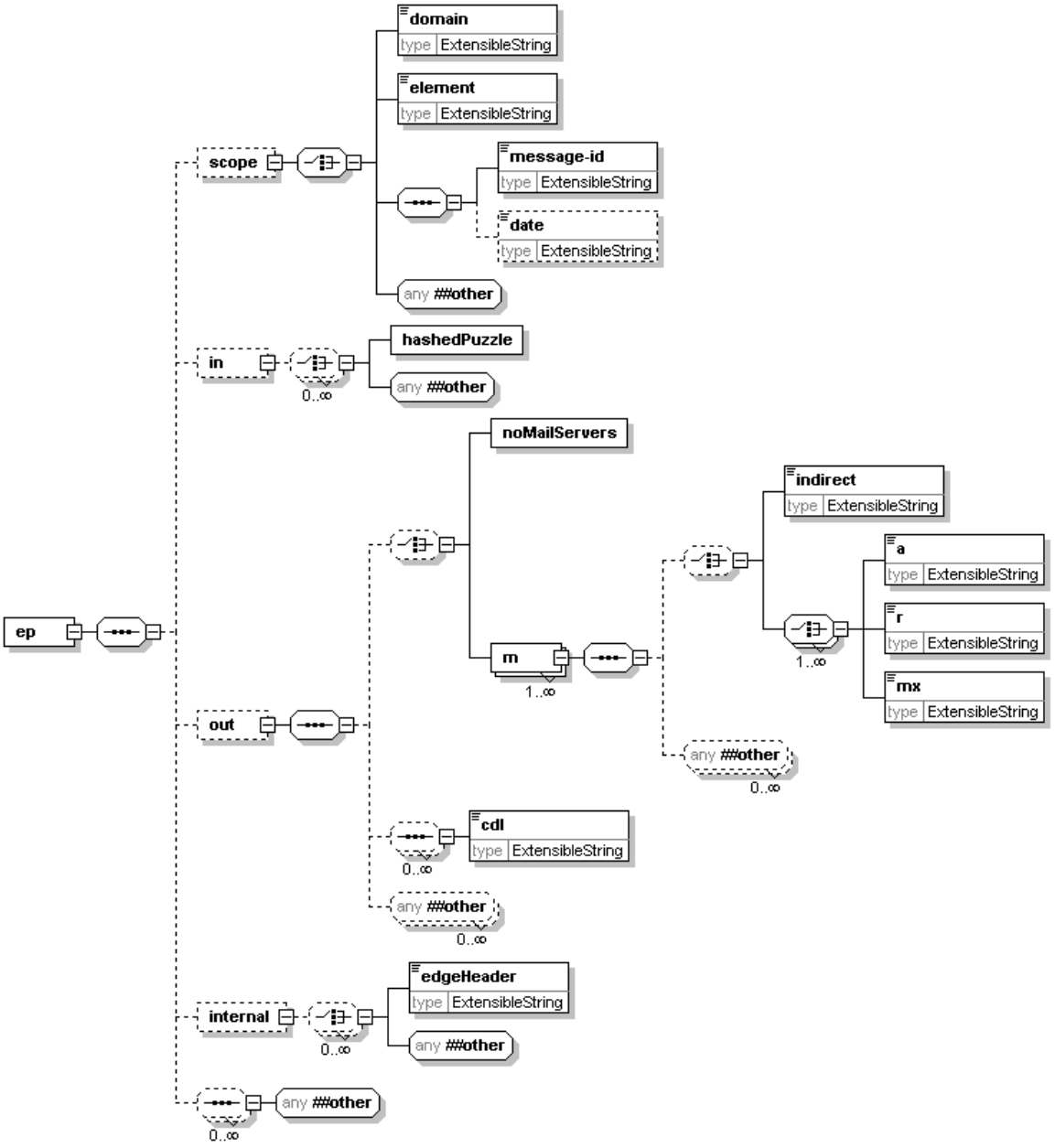
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://ms.net/1" xmlns="http://ms.net/1" xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" blockDefault="#all">
  <xs:complexType name="ExtensibleString">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:element name="ep">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="scope" minOccurs="0">
          <xs:complexType>
            <xs:choice>
              <xs:element name="domain" type="ExtensibleString"/>
              <xs:element name="element" type="ExtensibleString"/>
              <xs:sequence>
                <xs:element name="message-id" type="ExtensibleString"/>
                <xs:element name="date" type="ExtensibleString" minOccurs="0"/>
              </xs:sequence>
              <xs:any namespace="##other" processContents="lax"/>
            </xs:choice>
            <xs:anyAttribute namespace="##other" processContents="lax"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="in" minOccurs="0">
          <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
              <xs:element name="hashedPuzzle">
                <xs:complexType>
                  <xs:attribute name="nMin" type="xs:nonNegativeInteger"/>
                  <xs:anyAttribute namespace="##other" processContents="lax"/>
                </xs:complexType>
              </xs:element>
              <xs:any namespace="##other" processContents="lax"/>
            </xs:choice>
            <xs:anyAttribute namespace="##other"/>
          </xs:complexType>
```

```
</xs:element>
<xs:element name="out" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:choice minOccurs="0">
        <xs:element name="noMailServers">
          <xs:complexType>
            <xs:anyAttribute namespace="##other" processContents="lax"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="m" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:choice minOccurs="0">
                <xs:element name="indirect" type="ExtensibleString"/>
                <xs:choice maxOccurs="unbounded">
                  <xs:element name="a" type="ExtensibleString"/>
                  <xs:element name="r" type="ExtensibleString"/>
                  <xs:element name="mx" type="ExtensibleString"/>
                </xs:choice>
              </xs:choice>
              <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="eNoop" type="xs:boolean" use="optional"/>
            <xs:attribute name="allETPSigned" type="xs:boolean" use="optional"/>
            <xs:anyAttribute namespace="##other" processContents="lax"/>
          </xs:complexType>
        </xs:element>
      </xs:choice>
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element name="cdl" type="ExtensibleString"/>
      </xs:sequence>
      <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="directOnly" type="xs:boolean" use="optional"/>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
</xs:element>
<xs:element name="internal" minOccurs="0">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="edgeHeader" type="ExtensibleString"/>
      <xs:any namespace="##other" processContents="lax"/>
    </xs:choice>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
```

```
        </xs:element>
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
          <xs:any namespace="##other" processContents="lax"/>
        </xs:sequence>
      </xs:sequence>
      <xs:attribute name="testing" type="xs:boolean" use="optional" default="false"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  </xs:element>
</xs:schema>

<xs:schema targetNamespace="http://ms.net/2" xmlns="http://ms.net/2" xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" blockDefault="#all">
  <xs:element name="ocspResponse" type="xs:base64Binary">
    <xs:annotation>
      <xs:documentation>Base64 encoding of an RFC2560 OCSPResponse.</xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:schema>
```

Pictorially, the first schema above can be represented as follows (note that XML attributes are not illustrated in this diagram; only XML elements are shown):



The `testing` attribute which may optionally be present on an `ep` element provides a means by which an E-mail Policy Document can be tentatively deployed in an experimental mode. Documents in which such attribute is present with a true value SHOULD be entirely ignored (one should act as if the document were absent) unless one has cause by some other means (such as a private arrangement with the document publisher) to do otherwise.

Most of the XML elements defined here allow for an extensible set of attributes to be attached; this is signified by the presence of definitions of the form <xs:anyAttribute namespace="##other"/> in the schema. Software interpreting an E-mail Policy Document SHOULD ignore such extended attributes that it finds which it doesn't semantically understand; those wishing to avail themselves of this mechanism MUST in their design allow that this may occur.

Similarly, several places in the schema permit extensibility in the form of the introduction of arbitrary new XML elements; this is signified by the presence in the schema of wildcard definitions of the form <xs:any namespace="##other" processContents="lax"/> and is illustrated in the schema diagrams above with the construct:

With one exception, software interpreting an E-mail Policy Document SHOULD simply ignore any extended elements that it finds which it doesn't semantically understand. The exception pertains to the extensible elements permitted in the `scope` element: E-mail Policy Documents indicated as applying to scopes not understood by the interpreting application should be ignored in their entirety. Those wishing to avail themselves of the use of extended XML elements in E-mail Policy Documents MUST in their design accommodate these processing rules.

The optional `scope` element indicates the applicability of this particular policy. The possible options are a given DNS domain, a particular e-mail address, or a particular e-mail message as designated by the value of its `Message-Id:` header; alternate scoping mechanisms can be accommodated in the wildcard child of the `scope` element. If the `scope` element is omitted, the scope it is to be inferred from contextual use. If an `ep` element is stored in the E-mail Policy Document of a given DNS domain, then the `scope` MUST either be present with a value of `domain` equal to the domain in question, or `scope` MUST be omitted.

Use of e-mail policy documents with `message-id` scope begins to enable a form of a machine-interpretable challenge-response mechanism. When attached to a transmitted message (in an `E-mailPolicy:` header) such a policy document indicates policy that the address listed in the `From:` header of the message has with regard to a second message, namely the one whose `Message-Id:` header is as indicated (and where, if present, the `scope/date` element, which must contain a date formatted per §3.6.1 of RFC2822, also semantically matches the `Date:` header on the message). On a challenging side, the policy indicated usually has its useful semantic content beneath the `ep/in` element, where it can list information that it wishes were known for the second identified message; on the response side, useful information is usually contained beneath the `ep/out` element. Further details of such mechanisms are beyond the scope of this specification.

The optional `internal` element indicates e-mail related policies that might be of interest internally within the scope (usually a domain) with which this policy is associated. Publishing this internal information in DNS might be a convenient mechanism for disseminating the information through the organization associated with the domain. Be aware, however, that generally speaking, such information will be publicly accessible, and so private or sensitive information should not be placed here. The one architected element within `internal` is `edgeHeader` which denotes, as previously described, a distinguished string which will reliably be found in `Received:` headers added the incoming SMTP serves on the edge of a given domain (in e-mail policy documents of non-domain scope, the `edgeHeader` element has no meaning). Publishing this information in this publicly accessible way should cause little concern, since the contents of the `Received:` headers in one's mail undoubtedly already make their way outside the organization as mail is forwarded, bounced, and so on, and so ought not to be sensitive information. That said, those organizations for which this remains a concern should simply not avail themselves of the `edgeHeader` mechanism.

Readers are reminded that XML Schema specifies that values of type boolean may use either "true" or "1" to represent a true value and either "false" or "0" to represent a false value.

The following is an example XML instance of an `ep` element that applies to the `example.com` domain. It lists an inbound policy regarding the `HashedPuzzle` computational puzzle of §11 which says that a puzzle solution with at least 13 zero bits is acceptable. `example.com` list two outbound mail servers: the IP address 1.2.3.4, together with whatever IP addresses result from DNS address resolution on the domain `example.com` itself.

Note that, per reference [11], the encoding declaration `<?xml ... ?>` may generally be omitted on XML instances that are UTF-8 encoded.

```
<?xml version="1.0" encoding="UTF-8"?>
<ep xmlns='http://ms.net/1'>
   <scope><domain>example.com</domain></scope>
   <in><hashedPuzzle nMin='13'/></in>
   <out>
```

```
      <m>
         <a>1.2.3.4</a>
      </m>
      <m><mx/></m>
   </out>
</ep>
```

14. References

[1] Back, Adam, et al. *Hash Cash – A Denial of Service Counter-Measure*, http://cypherspace.org/hashcash/hashcash.pdf. See also http://cypherspace.org/hashcash and http://www.hashcash.org.

[2] Danisch, Hadmut, *The RMX DNS RR and method for lightweight SMTP sender authorization*, http://www.danisch.de/work/security/txt/draft-danisch-dns-rr-smtp-03.txt.

[3] DeKok, A. (Ed.), *Lightweight MTA Authentication Protocol (LMAP) Discussion and Applicability Statement*, November 3, 2003, http://asrg.kavi.com/apps/group_public/download.php/18/draft-irtf-asrg-lmap-discussion-00.txt

[4] Dwork, C., and M. Naor, *Pricing via Processing, Or, Combating Junk Mail, Advances in Cryptology*, CRYPTO '92, Lecture Notes in Computer Science No. 740, Springer, 1993, pp. 139-147. A more complete version is available at http://www.wisdom.weizmann.ac.il/~naor/PAPERS/pvp.ps

[5] FIPS PUB 180-1, *Secure Hash Standard*, available at http://www.itl.nist.gov/fipspubs/fip180-1.htm

[6] Maplesoft, *Maple 9*, http://www.maplesoft.com.

[7] Santesson, S., R. Housley, and T. Freeman, *Internet X.509 Public Key Infrastructure, Logotypes in X.509 certificates*, http://www.ietf.org/internet-drafts/draft-ietf-pkix-logotypes-11.txt, July 2003

[8] TRUSTe, http://www.truste.org

[9] UDDI, *Universal Description, Discovery, and Integration of Web Services*, http://www.uddi.org

[10] Wong, Meng Weng, Hadmut Danish, Gordon Fecyk Mark Lentczner, *Sender Permitted From*, http://spf.pobox.com.

[11] Worldwide Web Consortium, *Extensible Markup Language (XML) 1.0 (Second Edition)*, http://www.w3.org/TR/REC-xml

[12] Worldwide Web Consortium, *XML Key Management Specification*, http://www.w3.org/TR/xkms/, http://www.w3.org/2001/xkms, http://www.w3.org/2001/XKMS/Drafts/XKMS/xkms-part-1.html, and http://www.w3.org/2001/XKMS/Drafts/XKMS/xkms-part-2.html

[13] Worldwide Web Consortium, *XML-Signature Syntax and Processing*, http://www.w3.org/TR/xmldsig-core/

Top of document