Network Working Group
Internet-Draft

Charles H. Lindsey
University of Manchester
September 2001

Signed Headers in Mail and Netnews

draft-lindsey-usefor-signed-01.txt

Status of this Memo

Abstract

The huge growth of Netnews/Usenet in recent years has been accompanied by many attempts to abuse the system by various forms of malpractice, particularly the forging of various headers, causing it to appear that articles came from parties other than those that actually injected them or conveyed some Approval that the real poster was not entitled to give. Insofar as Netnews is regularly gatwayed to and from Email systems, these problems also extend to the Email domain.

This document provides a cryptographically secure means whereby it can be established beyond doubt that relevant headers of a Netnews article or an Email message have not been tampered with in transit, and that they were indeed originated by the person purporting to have done so. It seeks to supplement, rather than to supplant, the existing protocols for signing the bodies of articles and messages.

[This proposal arises from the activities of the Usenet Format Working Group, which is charged with updating the Netnews standards. Comments are invited, preferably sent to the mailing list of the Group at

usenet-format@landfield.com.]

Table of Contents

1. Introduction

[Remarks enclosed in square brackets and aligned with the left margin, such as this one, are not part of this draft, but are editorial notes to explain matters amongst ourselves, or to point out alternatives, or to indicate work yet to be done.]

1.1. Scope and Objectives

[This is a Draft of a Draft, for discussion within the USEFOR mailing list until the best format for putting it forward has been decided on. It remains to be decided whether it should be aimed towards an Experimental Protocol or the Standards track.]

"Netnews" is a set of protocols [USEFOR] that enables news "articles" to be broadcast to potentially-large audiences, using a flooding algorithm which propagates copies throughout a network of participating hosts. The huge growth in the use of this protocol in recent years has been accompanied by many attempts to abuse the system by causing it to appear that articles came from parties other than those that actually injected them, or that they had been posted with some Approval that the real poster was not entitled to give, or that they otherwise appeared to be different from what they actually were. The effects of such abuse are particularly accute in the case of "Control" articles which can cause newsgroups to be created or removed on hosts worldwide, or which can cause unauthorized deletion of articles already received and stored on such hosts. It is therefore considered essential to provide a cryptographically secure means whereby it can be established beyond doubt that the source and structure of articles are exactly as they purport to be.

"Electronic Mail" is a system for routing "messages" [RFC 2822] between individual computer users, usually on a one-to-one basis. The formats of Email messages and News articles have deliberately been made to be similar, so that messages may be gatewayed to news systems and vice-versa. In order that the same protection may be provided end-to-end for articles passing through such gateways, the protocal described here has been designed so that it will also work in the Email environment. If it should be found to have further applications in the Email environment, then that would be an added bonus.

An existing experimental protocol "pgpverify" [PGPVERIFY] is already in widespread use for authenticating Control messages for creating and removing newsgroups within Usenet, and has proven itself very successful in mitigating the effects of malicious attacks against the integrity of Usenet. This present proposal is largely based upon pgpverify; however, pgpverify is unsuitable for more widespread use as it stands because it is unable to cope with folded headers and with the changes that mail messages in particular are likely to

undergo during transport. A second similar experimental protocol "pgpmoose" [PGPMOOSE] is also currently in use for protecting moderated newsgroups against unauthorized postings.

There also exist protocols for the cryptographic signature of bodies of articles, notably S/Mime and PGP/Mime [RFC 2015] and [RFC 2015bis] , and it is moreover common to sign such bodies using PGP alone without the use of Mime [RFC 2045] et seq at all. However, these protocols cannot, by their nature, be used to sign headers. Moreover, since the signature is applied after any Content-Transfer-Encoding [RFC 2045], it may be impossible to verify the signature if the Content-Transfer-Encoding should be changed as the message passes through a succession of sites during transport. Nevertheless, this present proposal does not attempt to usurp those protocols, but merely provides the means to sign headers, both of complete messages and of headers embedded in Mime messages and multiparts.

[This document has been designed to fit on top of the draft currently in preparation for News [USEFOR]. If it is thought wise to issue this document before [USEFOR] is complete, then that reference will have to be to [RFC 1036] instead.]

1.2. Notations and Conventions

1.2.1. Requirements notation

Certain words, when capitalized, are used to define the significance of individual requirements. The key words "MUST", "SHOULD", "MAY" and the same words followed by "NOT" are to be interpreted as described in [RFC 2119].

1.2.2. Syntactic notation

This document uses the Augmented Backus Naur Form described in [RFC 2234]. A discussion of this is outside the bounds of this document, but it is expected that implementors will be able to quickly understand it with reference to the defining document.

1.3. Overview

This proposal makes provision for Signed headers to be included in news articles and in Mime messages and multiparts. A Signed header provides a cryptographic signature over a named set of other headers, including lower level headers contained in Mime messages and multiparts below the current level. Such signatures can give assurance to a recipient who verifies them that those headers have not been changed or added to in transit, and/or that the article was indeed sent by its purported originator.

The bodies of articles, Mime messages and multiparts are not directly included in the Signature. Rather, the intention is that each such body part should have a Content-MD5 (or similar) header computed for it, and that header should then be included in the Signature instead.

There is also provision for Verified headers which may be added by agents that have checked a Signed header. Verified headers may themselves be included in further Signed headers; this may be especially useful in the case of gateways which find it necessary to

change an article in ways that invalidate an original signature.

Every effort has been made to ensure that signatures remain verifiable in spite of all reasonable (and even unreasonable) changes to which they may be subjected in transit. These include changes to the Content-Transfer-Encoding of body parts (a principle reason for including them only via the Content-MD5 header), changes in the order of headers and of their layout, and encodings and re-encodings of unusual character sets. This is to be achieved by converting headers into a canonical form before they are signed. New headers, yet to be invented, need provide no problem, and there is no commitment to any particular character set (provided field-names remain in US-ASCII, as at present).

Provision is made for different protocols which may be required in the future. However, this proposal defines just one, recommended protocol, and it is not desirable that other protocols should be defined unless and until serious deficiencies in the existing ones have been revealed.

2. Basic Structure of Authenticating Headers

A Signed or a Verified header may appear in the headers of a news article or a mail message, or in the headers of a Mime multipart sub-part or of a Mime message/rfc822 object (or indeed of any similar Mime object yet to be invented). In all cases, the term "current level" encompasses the entire set of headers in that same object. Where the headers at the current level include a "Content-Type: multipart/*" or "Content-Type: message/*" header, lower-level headers can arise within its sub-parts.

Examples of Netnews articles and Email messages containing these headers may be found in section 5 below.

2.1. Syntax of the Signed header

```
Signed          = "Signed" ["-" DIGIT9] ":" 1*SP header-ref-list
                      1*( ";" header-parameter ) CRLF
DIGIT9          = %x31-39                          ; 1..9
header-ref-list= header-ref *( [CFWS] "," [CFWS] header-ref )
header-ref      = [ "+" / "-" ] [ subpart-indicator ] field-name /
                  [ subpart-indicator ] macro-name
subpart-indicator
                = 1*( DIGIT9 *DIGIT ":" )
field-name      ; see section [RFC 2822]
CFWS            ; see [RFC 2822]
FWS             ; see [RFC 2822]
macro-name      = '$' <a name, in the format of a field-name,
                   defined for the protocol indicated in the
```

```
                    signed-header-parameter>
      header-parameter
                    = signed-header-parameter / other-header-parameter
      signed-header-parameter
                    = signed-token "=" value
```

```
signed-token   = [CFWS] ( "protocol" / "key" / "sig" ) [CFWS] /
                 <Any other token defined for a particular protocol>
other-header-parameter
               = attribute "=" value
attribute      = iana-token / x-token
iana-token     = <A token defined in an experimental
                  or standards-track RFC and registered with
                  IANA>
value          = token / quoted-string
x-token        = [CFWS] "x-" token-core [CFWS]
token          = [CFWS] token-core [CFWS]
token-core     = 1*<any (US-ASCII) CHAR except SP, CTLs,
                           or tspecials>
tspecials      = "(" / ")" / "<" / ">" / "@" /
                 "," / ";" / ":" / "
                 "/" / "[" / "]" / "?" / "="
quoted-string  ; see [RFC 2822]
protocol-value = ietf-token / x-token
ietf-token     = <An extension token defined by a standards-track
                  or experimental protocol RFC and registered
                  with IANA>
key-id-value   = token
signature-value= [CFWS] DQUOTE [FWS] 1*( btext [FWS] ) DQUOTE [CFWS]
btext          = %x41-5A / %x61-7A / %x30-39 / "+" / "/" / "="
                 ; base 64 chars
```

The header-parameters MUST include a "protocol" parameter and a "sig" parameter, of which the "sig" paramameter MUST be the last parameter and MUST NOT be followed by CFWS (though it MAY be followed by WS). Any other-header-parameter that is present SHOULD be ignored.

> NOTE: The requirement for an explicit SP after the ":" is to ensure compatibility with the syntax of Netnews [USEFOR]; it is not strictly necessary for Email.

The use of a DIGIT9 in the Signed header allows for 10 distinct such headers at any one level. This is more than sufficient for the intended usage (it would be most unusual to get beyond Signed-2) whilst still permitting implementations to check field-names against a fixed list of valid names. There MUST NOT be more than one Signed header with no DIGIT9, or the same DIGIT9, within one set of headers.

The header-ref-list indicates those header-refs, at or below the current level, which are covered by the signature. The ordering of this list is significant. A header-ref prefixed by a "+", or not prefixed at all, indicates a header-ref to be added to the list defined by those preceding it (unless already present therein), and a header-ref prefixed by "-" indicates a header-ref to be removed from the header-refs defined by the list preceding it. Any macro-names in

the header-ref-list MUST be defined, in the definition of the protocol, to expand into a header-ref-list which does not itself contain any subpart-indicators or further macro-names.

NOTE: If some header-ref in the list matches no header in the actual article, then it comprises an assertion that no such header was present when the article was signed. Headers which are routinely added to or altered as the article progresses through transports (such as Path, Received and Xref, and even Sender and Content-Transfer-Encoding) SHOULD NOT be included in a header-ref-list, and neither should any header which appears twice in the set of headers. A header-ref prefixed by "-" may be used to exclude any header-ref arising from the expansion of a macro-name.

Tokens are case-insensitive. "PGP-Head-1" (section 3.2) is the preferred protocol defined by this proposal. It is desirable to keep the number of recognized protocols to an absolute minimum, and it is anticipated that further protocols would only be needed in the event that serious cryptographic deficiencies were to be found in the existing ones.

The "key" parameter identifies the key used to generate the signature in a notation dependent upon the protocol (but commonly "0x" followed by hexadecimal digits). The CFWS following it MAY include a comment containing an identification of the person or entity which owns that key.

2.2. Semantics of the Signed header

Where the headers at the current level include a "Content-Type: multipart/*" or "Content-Type: message/*" header, lower-level headers within its sub-parts may be referenced as follows:

(i) A header-ref not containing any subpart-indicator references the header of that name, if any, at the current level. Header-refs are, for this purpose, considered as case-insensitive.

(ii) A header-ref of the form "<m>:XXXX/" (or "<m>:<n>:...XXXX"), where <m> and <n> are numbers and the current level contains a "Content-Type: multipart/*" header, references the header that would be referenced by "XXXX" alone (or by "<n>:...XXXX") in the <m>th sub-part of that multipart, that sub-part now being regarded as the current level.

(iii) A header-ref of the form "1:XXXX", where the current level contains a "Content-Type: message/rfc822" header (or any other message type which provides for its own set of headers), references the header that would be referenced by "XXXX" alone in that message object.

(iv) A header-ref that does not match up with multipart or message Content-Type headers as indicated above MUST NOT be used.

(v) For example "3:2:Content-MD5" references the Content-MD5 header of the second part of a multipart, which is itself the third part of a multipart established at the current level.

A protocol, as established by this proposal or by any extension to it, comprises three parts: a "canonicalization algorithm", a "cryptographic algorithm", and possibly a "macro definition".

The signature of a Signed header is constructed in accordance with a given header-ref-list as follows:

1. A partial Signed header is constructed from that header-ref-list and such header-parameters (excluding "sig") as are required by the protocol, including at least a "protocol" parameter and, most likely, a "key" parameter identifying the cryptographic key used (possibly followed by a comment indicating the person or entity responsible), all followed by a CRLF.

2. A reduced header-ref-list is obtained by first expanding any macro-names defined in the macro definition of the protocol, duplicating any subpart-indicator in front of each header-ref in the expansion; then any "+" prefixes are stripped, and finally, working from left to right, if a header-ref duplicates a preceding one the second copy is removed, and if a header-ref is prefixed by "-", that copy and any previous ones (without that prefix) are removed.

3. The partial Signed header (with its original header-ref-list) followed by all the headers referenced by the reduced header-ref-list (being headers at the current level or encapsulated within multiparts at any lower level and taken in their order within the reduced header-ref-list) are concatenated to produce a list of headers to be signed.

4. The list of headers to be signed is subjected to the canonicalization algorithm of the protocol to produce a canonicalized list.

5. The canonicalized list is subjected to the cryptographic algorithm of the protocol to produce a character stream representing the signature encoded in base64 [RFC 2045].

6. A "sig" parameter is appended to the partial Signed header, its value consisting of a quoted-string containing the base64-encoded stream, split into convenient lines by the insertion of FWS.

7. The Signed header thus constructed is then incorporated into the set of headers at the current level.

The signature of a Signed header is verified as follows:

1. The "sig" parameter is removed from the Signed header to give a partial Signed header.

2-4.The corresponding steps of the process that constructed the header are taken, producing a canonicalized list.

5. The public key identified according to the "protocol" parameter is now used by the cryptographic algorithm of that protocol to verify that canonicalized list. This may result in a simple pass-fail, or it may return some indication of the privileges (such as the authority to issue certain news control messages or to manage some mailing list) enjoyed by the owner of that key.

The purpose of a Signed header is solely to establish that the headers referenced in it were present in an article when that article passed through the hands of the person or entity that generated the signature (and hence that it did indeed pass through those hands). It SHOULD NOT be taken as an endorsement of whatever is contained in the body of the article. If the contents of the body require such endorsement, then the body SHOULD be signed separately, for example in accordance with PGP/Mime [RFC 2015] and [RFC 2015bis].

Signatures will typically be generated by the originators of articles (to prove the origin), by moderators of moderated newsgroups (to testify to their Approved header), by managers of mailing lists, and occasionally by gateways. They SHOULD NOT be generated by intermediate transports and relayers through which the article might pass. This is intended to be an end-to-end protocol, and signatures SHOULD ONLY be added when new, hitherto unsigned, information is added. Moreover, the set of headers included within the signature SHOULD be no more than is necessary to achieve the security desired.

> NOTE: It will be observed that no provision has been made to include the bodies of an article or of its sub-parts in the signature. If (as will indeed often be the case) it is required to attest that the body (or sub-part) dispatched along with the set of headers is the same as the body that was delivered at the far end, then the proper procedure is to construct a Content-MD5 header [RFC 1864] for that body (or sub-part) and to include that Content-MD5 amongst the headers that are signed. Doing it this way confers three advantages:
> a) The Content-MD5 header is constructed in such a way that it is immune to changes of Content-Transfer-Encoding to which an article, or its sub-parts, may be subjected during transport.
> b) Given that many user agents already routinely construct a Content-MD5 header, and verify it on receipt (a practice much to be commended), it should be possible to generate a Signed header without an extra pass through the entire body (especially in the common case where there are no sub-parts). This applies particularly in the case of additional signatures by moderators or mailing list managers, who may not need to examine the body at all.
> c) If a Content-MD5 header should fail to verify (perhaps because of some transmission error) the verification of a Signed header might still succeed, giving the recipient at least some

partial information as to where any problem might lie.

NOTE: If, at some future time, a Content-SHA1 header (or any similar header based upon a different hashing algorithm) should be invented, it could equally well be used for this purpose.

2.3. Syntax of the Verified header

```
Verified       = "Verified" ["-" DIGIT9] ":" 1*SP name-addr
                     *( ";" header-parameter ) CRLF
name-addr      ; see [RFC 2822]
header-parameter
               =/ verified-header-parameter
verified-header-parameter
               = signature-token "=" signature-value /
                 hashcheck-token "=" hashcheck-value
signature-token= [CFWS] "signature" [CFWS]
hashcheck-token= [CFWS] "hashcheck" [CFWS]
signature-value= [CFWS] ( "good" / "FAILED" ) [CFWS]
hashcheck-value= [CFWS] DQUOTE ( "good" / "FAILED" )
                    FWS header-ref-list DQUOTE [CFWS]
```

The Verified header is a "variant header" (i.e. it may be present or not, and in a different form, in different copies of the same article or message).

The use of a DIGIT9 in the Verified header allows for 10 distinct such headers in one article. Each Verified header MUST match some Signed header with the same DIGIT9 in that same set of headers. There MAY be more than one Verified header with the same DIGIT9 within one set of headers (but observe that it would not then be possible to include those headers in a further Signed header).

Tokens used for attributes are case-insensitive. The only parameters defined by this proposal are the "signature" and "hashcheck" parameters. Any other-header-parameter that is present SHOULD be ignored. The absence of a "signature" parameter should be taken as indicating that the verification had succeeded. The "hashcheck" parameter is to indicate that a Content-MD5 (or similar) header identified in the header-ref-list had been verified, or not as the case may be.

[Do we also want a "confidence" parameter for the verifier to express his certainty of the identity of the original Signer, and if so, what notation to use?]

2.4. Semantics of the Verified header

The Verified header is intended to be added to an article by an agent through which the article passes, and serves as an assertion that the corresponding Signed header has been cryptographically verified by the person or entity identified in the name-addr (or otherwise if the "FAILED" value is present). The addr-spec contained in that name-addr MUST be a valid email address by which that person or entity may be contacted. The original Signed header MUST NOT be removed from the article. The Verified header (supposing it is the only one present

with that particular DIGIT9, if any) MAY itself be included in a further Signed header added at the same time.

NOTE: The purpose of a Verified header is to save the ultimate recipient the trouble of verifying the cryptographic signature himself (which can be time consuming, and may require knowledge of public keys not in his possession). Such a verification, if performed close to the ultimate recipient (such as by the news or mail server to which he connects) could normally be regarded as adequate evidence of authenticity, even if not signed itself. It would be hard (certainly in the case of Netnews) for a malicious interloper to cause such a verification to appear bearing the identity of the local server of each ultimate recipient.

NOTE: The Verified header is also useful in the case that a gateway (or a moderator) makes some change to an article that renders an original Signed header invalid. Such a gateway can therefore certify that the original form of the Signed header had been verified, and can then re-sign the article (including the added Verified header). Likewise, a site (such as the originator's own server) with a well known public key can verify and resign an article whose originator's public key may be less well known. However, Verified headers SHOULD NOT be added as routine by other intermediate sites.

It is normally the business of the reading agent of the ultimate recipient to check the correctness of a Content-MD5 or similar header. Nevertheless, an earlier agent that has added a Verified header and also checked such a Content-MD5 header MAY so indicate by including a "hashcheck" parameter.

3. Protocol definition

3.1. Requirements for canonicalization algorithms

It is a sad fact of life that those implementing agents for handling Netnews and Email cannot resist the temptation to "improve" articles passed through them by rewriting headers that are thought not to conform to some real or supposed standard. Experience shows that, in the majority of cases, such tinkering makes matters worse rather than better, and for that reason [USEFOR] and, to a lesser extent, [RFC 2822] and [RFC 2821] try to forbid it, especially when perpetrated by relaying and transport agents (there are arguments in favour of allowing injecting agents and other agents close to the originator to do some limited cleanups, especially where it is impractical to return the article to the originator for correction).

Furthermore, in the case of Email it is often required for the transport protocols to modify articles en route, most notably when articles containing octets with the 8th bit set have to be passed through a channel that permits only 7bit.

It is a further sad fact of life that agents which make such changes are not going to go away just because some standard says so. Therefore, the canonicalization algorithm SHOULD endeavour to enable the headers of articles to be signed and verified in accordance with

this proposal in spite of such tinkerings, insofar as they can be anticipated. The following list indicates some common practices which are worth detecting and protecting against.

- o Headers may be re-folded to fit within some preferred overall line length. This may result in the creation of whitespace where none existed before.
- o Trailing whitespace may be removed, and line endings changed to/from CRLF.
- o Field-names may be converted into some usual canonical form (e.g. "Mime-Version" into "MIME-Version").
- o Phrases, or parts thereof, may be converted to or from quoted-strings.
- o Date-times may be rewritten in some preferred format, or into some preferred timezone.
- o Headers with non-ASCII characters may be converted to or from the notation defined in [RFC 2047].

  Observe that there is no canonical way to do this conversion and it is, moreover, frequently performed in contexts where it is not strictly allowed.

[Other contributions to this list welcomed.]

Since the slightest change to a canonicalization algorithm will render it inoperable with previous versions, such an algorithm MUST NOT be changed once it has been defined by this proposal, or any extension thereof. In the event of some inadequacy being found, it would be necessary to devise and standardize a new algorithm, a task not to be undertaken lightly. For this reason, canonicalization algorithms SHOULD be designed to cope with the widest possible range of headers, including those not yet invented. Therefore, they SHOULD NOT, so far as possible, rely on the ability to parse any particular header.

> NOTE: A canonicalization algorithm is required simply to produce an octet stream for submission to the cryptographic algorithm. That stream does not have to be human readable, nor does it have to be a syntactically-correct header, nor does it have to be convertible back into the original header, or into any correct header at all. Insofar as many original headers can, in principle, be mapped into the same octet stream, this in no way reduces the utility of the algorithm, even though it might enable conspiracy theorists to imagine, and even implement, various sorts of covert channels for use by malicious interlopers.

3.2. The PGP-Head-1 protocol

The "The PGP-Head-1" protocol is comprised of a canonicalization algorithm, a cryptographic algorithm, and a macro definition.

3.2.1. The PGP-Head-1 canonicalization algorithm

For the purposes of this algorithm, the headers Subject, Comments, Organization and Summary, and all headers starting with "X-", are to be considered "unstructured" and all other headers "structured" (whether or not they were so described in any other standard). Headers are considered to be constrained to the following syntax:

```
structured-header
              = header-name ":"
                   1*SP structured-header-content CRLF
unstructured-header
              = header-name ":"
                   1*SP unstructured-header-content CRLF
header-name   = 1*name-character *( "-" 1*name-character )
name-character= ALPHA / DIGIT
structured-header-content
              = *structured-header-zone
unstructured-header-content
              = unstructured-header-zone
structured-header-zone
              = neutral-zone / quoted-zone / sharp-zone /
                square-zone / comment-zone
unstructured-header-zone
              = *( FWS / encoded-word / <any visible character> )
neutral-zone  = 1*( FWS / encoded-word /
                    <any character except DQUOTE, "<", "[", "("> )
quoted-zone   = DQUOTE *( FWS /
                          <any character except unquoted DQUOTE> )
                    DQUOTE
sharp-zone    = "<" *( FWS /
                       <any character except unquoted ">"> ) ">"
square-zone   = "[" *( FWS /
                       <any character except unquoted "]"> ) "]"
comment-zone  = "(" *( FWS / encoded-word / comment-zone /
                       <any character except unquoted ")"> ) ")"
encoded-word  = "=?" pure-token "?" pure-token "?"
                   1*<any printable US-ASCII character other than
                      "?" or SP> "?="
pure-token    = 1*<any (US-ASCII) CHAR except SP, CTLs,
                    or tspecials>
```

- where '<any visible character>' means any octet other than those representing the US-ASCII characters NULL, CR, LF, TAB and SP,
- where 'except unquoted "x"' means except any "x" not immediately preceded by a "\" and thus constituting a quoted-pair, and
- where an encoded-word does not include "(" or ")" when in a comment-zone, and does not include DQUOTE, "<", "[", or "(" when in a neutral-zone.

Observe that certain field-names containing non-alphanumeric characters, and permitted by [RFC 2822] (though never used in practice) are excluded from this protocol. Moreover, it is not assumed that this protocol will work on any of the obsolete syntax defined by [RFC 2822].

NOTE: All known Email and Netnews headers (and a lot more besides) are encompassed within this syntax. Observe that the various zones cannot possibly overlap, and that any encoded-word must be fully contained within its zone. All encoded-words permitted by [RFC 2047] (and more besides) are covered. The structure is easily parsed by a straightforward state machine (though the nesting of comment-zones is a nuisance, as is the impossibility to detect whether a sequence beginning "=?" was really an encoded-word until you get to the matching "?=").

Each header to be included in the algorithm, which will in general consist of several lines (those after the first commencing with whitespace), is processed as follows:

1. The header-name at the start of the header is converted to lowercase, the whitespace following it (if any) is removed, and a single SP is inserted.

2. Any date-time occurring in a Date, Resent-Date or Expires header (but not in any other header) is converted to UTC and written as a date-time in the format
   07 dec 2000 23:59:60 +0000
   Note absence of day-of-week, leading zero included in day, month-name in lower case, year as four digits, seconds included, and timezone 0000 preceded by a "+" as opposed to a "-", and observe that any FWS will be removed in the next step, giving
   07dec200023:59:60+0000

   NOTE: Observe that the effect is to treat "31 Dec 2000 23:59:60 +0000" (which is a legitimate date-time as defined by [RFC 2822]) as being different from "1 Jan 2001 00:00:00 +0000".

3. Within each unstructured-header-zone and each comment-zone, all instances of FWS are replaced by a single SP; within each neutral-, quote-, sharp- or square-zone, all instances of FWS are omitted (thus the header has now been unfolded into a single line). Any whitespace at the end of the header is removed, and it is ensured that the header ends with a single CRLF.

4. The DQUOTEs (ASCII '"') enclosing each quoted-zone are removed (but not any quoted DQUOTE or any DQUOTE within other zones so that, in particular, they are not removed within msg-ids).

5. Any encoded-word (where allowed by the above syntax, and whether or not its length is more than 75 characters) is replaced by the sequence of octets obtained by decoding it. Moreover, where two adjacent encoded-words are separated by whitespace, that whitespace is removed (see [RFC 2047]).

NOTE: The decoding of encoded-words must take place last, because it could produce arbitrary sequences of octets (when decoding into UCS-16, for example) which might then be confused with US-ASCII characters such as DQUOTE, etc. Whitespace needs

to be removed entirely from structured headers because it is possible it may have been introduced by folding in unexpected places en route, subsequent to the original signing.

Typical results of the use of this canonicalization algorithm can be found with the example in section 5.2 below. Test data for use with implementations of this algorithm, and containing many obscure cases, can be found in Appendix B.

If, during signing, a header is found not to conform to the given syntax (in particular, if the closing delimiter of some zone is not found), then the signing MUST be aborted (and it MAY be aborted if the header is malformed for some other reason). When verifying a signature, however, an implementation MAY attempt to continue even when the final zone of a header has no closing delimiter.

NOTE: If an internet mail message in the format defined by [RFC 2822] is converted into X.400 mail by a gateway conforming to [RFC 1327] and then back into internet mail, then it is likely that any signature made in accordance with this proposal will fail to verify. For example, comments in headers containing addresses (such as in From, Reply-To, etc.) may be converted into phrases and moved in front of the addr-spec, or even removed entirely, and thus the canonicalized form of the message will have been changed. This old convention, for storing the Real Name of the person associated with the address in a following comment, is now deprecated by both [RFC 2822] and [USEFOR], but even where phrases are used for this purpose it is possible that other changes to the message will still render the signature unverifiable. Note that there is in any case no expectation that an internet mail message signed according to this proposal will ever be able to be verified once it has been passed permanently into an X.400 system, nor vice versa.

3.2.2. The PGP-Head-1 cryptographic algorithm

[Open PGP is the obvious choice for this, since it is widely available and is blessed by the IETF. My only reservation is that it comes with a rather poor certification system as compared with, say, SPKI. So this choice might yet have to be reviewed.]

The stream of octets resulting from the canonicalization algorithm is signed, in binary mode (signature type 0x00), in accordance with Open PGP [RFC 2440].

NOTE: The signature is made in binary mode just in case any [RFC 2047] decoding into UCS-16 has produced octets which might be mistaken for isolated CR, LF or trailing SP characters, which are treated specially in PGP text mode, and also because the

treatment of trailing whitespace differs between Open PGP and some earlier PGP implementations.

The output of the algorithm MUST be Ascii-armored [RFC 2440], but the Armor Header Line ("BEGIN PGP SIGNATURE"), the Armor Headers (e.g. "Version:"), the blank line following the Armor Headers, and the Armor Tail ("END PGP SIGNATURE") are to be omitted (thus yielding a sequence of base64 characters). Observe that these characters will include a CRC checksum, which SHOULD be on a separate line from the rest of the signature.

The signature included within the Ascii-armor MAY include certificates as evidence that the signing key has the necessary authorization to sign articles of that nature, but such usage is in general deprecated except between parties that have agreed otherwise or where, for some reason, an unusual signatory is signing and attaches a certificate from the usual signatory.

The signature SHOULD use the DSA public-key algorithm and the SHA-1 hashing algorithm, and be incorporated in a Version 4 Signature Packet in the new format. It MAY alternatively use the combination RSA/MD5 with Version 3 in the old format (for compatibility with PGP 2.6.x) and it MAY use the combination RSA/SHA-1 with Version 4 in the new format. Verifiers MUST be able to verify all of these forms.

3.2.3. The PGP-Head-1 Macro Definition

Two macro-names are defined, "$news-standard" and "$mail-standard".

"$news-standard" is a macro representing a set of common headers that SHOULD normally be included when signing the headers of a Netnews article, and is defined to expand into the header-ref-list

> Date, Newsgroups, Distribution, Message-ID, From, Reply-To, Followup-To, References, Subject, Keywords, Control, Content-Type, Content-ID

"$mail-standard" performs the same function for mail messages, and is defined to expand into the header-ref-list

> Date, From, Reply-To, To, Cc, In-Reply-To, References, Subject, Keywords, Content-Type, Content-ID

> NOTE: Those lists have carefully excluded those headers (such as Sender and Content-Transfer-Encoding) which are liable to be added or altered by sites downstream from the one which generated the Signed header.

4. Applications

It is anticipated that protocols for specific applications of the signature mechanisms described in this proposal will be devised, whether under the auspices of the IETF or otherwise. For example, the

need to be able to verify the origin of Control messages for creating and removing newsgroups and for cancelling articles was a prime motivation for creating this proposal.

It is up to each such application to specify appropriate mechanisms for establishing a Public Key Infrastructure suited to its purpose. Such an infrastructure would provide for the storing, distribution and authorization of the necessary public keys (and for revocations thereof). This proposal establishes no preferred mechanisms in this regard, except to draw attention to the possible usefulness of the Content-Type application/pgp-keys as defined in [RFC 2015] and [RFC 2015bis].

5. Examples

5.1. Newgroup Control message

A 'newgroup' control message in the format given in [USEFOR].

```
Newsgroups: comp.foo
From: "Charles Lindsey" <group-admin@isc.example>
Subject: cmsg newgroup comp.foo moderated
Control: newgroup comp.foo moderated
Approved: newgroups-request@isc.example
Message-ID: <919190727.4918@isc.example>
Date: Tue, 16 Feb 1999 18:45:27 -0000
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=88888888
Signed: $news-standard,+1:content-md5,+1:content-type,+3:content-md5,
   +3:content-type; protocol=pgp-head-1; key="0xA336D40C"
   (DSS-example);
   sig="
   iQA/AwUAO40EkyQRKsmjNtQMEQIwGgCfSYj8sgrRgRNQIwWKKnk+M9j0o+wAn2Mp
   fS2zwmmrA/KvCXyiTFsk35pr
   =8p5V"

This is a multipart message in MIME format.

--88888888
Content-Type: application/news-groupinfo
Content-MD5: 68BGYb5+8KAVeqno7Et7Ug==

For your newsgroups file:
comp.foo                For Foo discussions (Moderated)

--88888888
Content-Type: text/plain

comp.foo a moderated newsgroup which passed its vote for creation
by 424:8 as reported in news.announce.newgroups on 10 Feb 99.

--88888888
```

```
Content-Type: application/news-transmission
Content-MD5: cjeIxiGbPsrse1G/w9cfqQ==

Newsgroups: comp.foo
Path: not-for-relaying
Distribution: local
From: "Charles Lindsey" <group-admin@isc.example>
Message-ID: <919190727.4918$p=1@isc.example>
Date: Tue, 16 Feb 1999 18:45:27 -0000
Subject: Charter for newsgroup com.foo
Approved: newgroups-request@isc.example

The charter, culled from the call for votes:

   Comp.foo is a moderated newsgroup for discussing all manner of
   Foos.

Moderation submission address:
   comp-foo@bar.example

--88888888--
```

5.2. Mail message re-signed by mailing list owner

```
received: from house.example by bar.example (8.8.8/AL/MJK-2.0)
        id XAA10880; Sat, 13 Feb 1999 23:00:14 GMT
Resent-From: "Example Mail Server" <majordomo@com.example>
Precedence: list
Received: (from list@localhost)
        by house.example (8.9.2/8.9.2) id OAA28279;
        Sat, 13 Feb 1999 14:59:56 -0800 (PST)
From: <"[john]"@
    temple.example> (John Smith)
Organization: http://www.temple.example/john
Subject: Submission to mailing list
      in connection with foo.
Message-ID: <19990213145946.20115@main.temple.example>
Date: Sat, 13 Feb 1999 22:59:46 +0000
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-MD5: ayoAIdYN8PZqpOgij7VG2Q==
Signed: $mail-standard,content-md5;
   protocol=PGP-Head-1; key="0xA336D40C" (DSS-example);
   sig="
   iQA/AwUAO40E1yQRKsmjNtQMEQLvzQCgtNnWdN2lwYtFoajEen96111IMboAn2hV
   z9edcA/oc2F6ui8nIj/X5/UW
   =buij"
Verified: majordomo-request@com.example; signature=good;
   hashcheck="good content-md5"
```

```
Signed-1: message-id,date,resent-from,
        verified,signed; protocol=PGP-HEAD-1; key="0xA336D40C";
   sig="
   iQA/AwUAO40GgCQRKsmjNtQMEQLsNACdFPk9gPtPq9qpWMLXlurvhBLqMbAAoLg0
   uOVRa6sHqBo2bVf+P/7qy0bF
   =FyFy"

Text of John's message.

--
John's signature.
```

Passing the original form of this through the PGP-Head-1 canonicalization algorithm produces the following, in the case of the "Signed:" header (observe lines folded for convenience of this document - the true line endings being indicated by "CRLF"):

```
signed: $mail-standard,content-md5;protocol=PGP-Head-1;key=0xA336
D40C(DSS-example)CRLF
date: 13feb199922:59:46+0000CRLF
from: <"[john]"@temple.example>(John Smith)CRLF
subject: Submission to mailing list in connection with foo.CRLF
content-type: text/plain;charset=us-asciiCRLF
content-md5: ayoAIdYN8PZqpOgij7VG2Q==CRLF
```

And here is the result of canonicalizing to produce the "Signed-1:" header:

```
signed-1: message-id,date,resent-from,verified,signed;protocol=PG
P-HEAD-1;key=0xA336D40CCRLF
message-id: <19990213145946.20115@main.temple.example>CRLF
date: 13feb199922:59:46+0000CRLF
resent-from: ExampleMailServer<majordomo@com.example>CRLF
verified: majordomo-request@com.example;signature=good;hashcheck=
goodcontent-md5CRLF
signed: $mail-standard,content-md5;protocol=PGP-Head-1;key=0xA336
D40C(DSS-example);sig=iQA/AwUAO40E1yQRKsmjNtQMEQLvzQCgtNnWdN2lwYt
FoajEen96111IMboAn2hVz9edcA/oc2F6ui8nIj/X5/UW=buijCRLF
```

> NOTE: the second signature signed only that which it had added itself, plus sufficient of the original headers to identify the original message. It did not need to scan the body to recompute the MD5 hash, but effectively included it by signing the original "Signed:" header.

6. Security

TBD
[What is there to say here?]

7. References

[PGPMOOSE] Greg Rose, [I need a URL for this], October 1995.

[PGPVERIFY] David Lawrence, <ftp://ftp.isc.org/pub/pgpcontrol/README.html>.

[RFC 1036] M. Horton and R. Adams, "Standard for Interchange of USENET Messages", RFC 1036, December 1987.

[RFC 1327] S. Hardcastle-Kille, "Mapping between X.400(1988) / ISO 10021 and RFC 822", RFC 1327, May 1992.

[RFC 1864] J. Myers and M. Rose, "The Content-MD5 Header Field", RFC 1864, October 1995.

[RFC 2015] M. Elkins, "MIME Security with Pretty Good Privacy (PGP)", RFC 2015, October 1996.

[RFC 2015bis] M. Elkins, D. Del Torto, R. Levien, and T. Roessler, "MIME Security with OpenPGP", draft-ietf-openpgp-mime-06.txt, April 2001.

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

[RFC 2047] K. Moore, "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.

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

[RFC 2234] D. Crocker and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[RFC 2440] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer, "OpenPGP Message Format", RFC 2440, November 1998.

[RFC 2821] John C. Klensin and Dawn P. Mann, "Simple Mail Transfer Protocol", RFC 2821, April 2001.

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

[USEFOR] Charles H. Lindsey, "News Article Format", draft-ietf-usefor-article-format-03.txt.

8. Acknowledgements

The author acknowledges the work of David Lawrence, as original author of "pgpverify", for many of the ideas contained herein, and also many contributions from members of the usenet-format mailing list.

9. Contact Address

Charles. H. Lindsey
5 Clerewood Avenue
Heald Green
Cheadle
Cheshire SK8 3JU
United Kingdom
Phone: +44 161 436 6131
Email: chl@clw.cs.man.ac.uk

Comments on this draft should preferably be sent to the mailing list of the Usenet Format Working Group at

usenet-format@landfield.com.

This draft expires six months after the date of publication (see Page 1) (i.e. in March 2002).

10. Intellectual Property Rights

[The usual texts from RFC 2026 to be inserted here.]

Appendix A - Model implementation

The following is written in PERL, with full use made of facilities provided by the Perl CPAN library.

Appendix A.1 - The PGP-Head-1 canonicalization

```
package Canon;

use MIME::Words qw(decode_mimewords);
use Date::Parse;
use Date::Format;
use Time::Local;
use Exporter ();
@ISA = qw(Exporter);
@EXPORT = qw(set_protocol $canonicalize %macros);

my $protocol;
my %unstructureds = ();
my %dates = ();
```

```
   my @ph1_news_standard = qw(date newsgroups distribution message-id
                              from reply-to followup-to references
                              subject keywords control content-type
```

```
                          content-id);
my @ph1_mail_standard = qw(date from reply-to to cc in-reply-to
                          references subject keywords content-type
                          content-id);

sub set_protocol {
    $_ = shift;
    SWITCH: {

        # PGP-Head-1 protocol
        /^pgp-head-1$/o && do {
            %macros = (
                'news-standard' => \@ph1_news_standard,
                'mail-standard' => \@ph1_mail_standard,
            );
            %unstructureds = ('subject', 1, 'comments', 1,
                              'organization', 1, 'summary', 1);
            %dates = ('date', 1, 'resent-date', 1, 'expires', 1);
            $canonicalize = \&ph1_canon;
            $protocol = $_;
            last SWITCH;
        };

        # other protocols go in here

        die "Unknown protocol $_\n";
    }
}



sub ph1_canon {
  my $tag = lc shift;
  my $line = shift;
  my $signing = shift;  # for more stringent checks when signing
  my ($ss,$mm,$hh,$day,$month,$year,$zone,$time,$dummy,@dateval);

  $is_structured = (not $unstructureds{$tag}) && $tag !~ m/^x-/o;
  $is_date = $dates{$tag};
  @outlist = ($tag, ': ');
  $outptr = \@outlist;  # will point to @encodelist during encoding
  $state = 0;           # for the state machine
  $encoding = 0;        # part of the state machine
  $pending = 0;         # to remember the FWS between encoded-words

  do {
    # lexical split of $line into plain ($x) + next delimiter ($y)
    $line =~ m/(.*?)    # anything except the following:
               ( \\\S      # quoted-pair
               | [][)><("] # various bracket delimiters
```

```
                | =\?(?!=) | \?=\s+=\? | \?=  # for encoded-words
                | \s*$      # trailing whitespace
                ) /sogx;
        $x = $1; $y = $2;
```

```
# convert $x into canonical form
if      ($is_date && $state == 0) {
  $x =~ s/(\S*)\s+/$1 /sog; # reduce FWS to SP
  if ($x !~ m/^\s*$/) {  # zone not empty
    if ($signing && $x !~ m/^\s?
            ((mon|tue|wed|thu|fri|sat|sun)\s?,\s?)?
            [0-9]{1,2}\s
            (jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec)\s
            [0-9]{4}\s
            [0-9]{2}:[0-9]{2}:[0-9]{2}\s
            [-+][0-9]{4}\s?
            /oix) {die "Bad Date '", $x, "'\n"}
    if (not (
        (($ss,$mm,$hh,$day,$month,$year,$zone) = strptime($x))
        && ($time = timegm(0,$mm,$hh,$day,$month,$year)) >= 0
                    # $ss not used, in case it is a leap second
        )) {die "Bad Date '", $x, "'\n"}
    ($dummy,$mm,$hh,$day,$month,$year) = gmtime($time - $zone);
    @dateval = ($ss,$mm,$hh,$day,$month,$year);
                    # $ss restored
    $x = lc strftime("%d%h%Y%T+0000", @dateval);
  }
} elsif ($is_structured && $state <= 0) {
  $x =~ s/(\S*)\s+/$1/sog;  # eliminate FWS
} else { # unstructured, or in a comment-zone
  $x =~ s/(\S*)\s+/$1 /sog; # reduce FWS to SP
}
push @$outptr, $x;

# state machine to process $y
if ($is_structured) {
  if      ($state == 0)  { # neutral-zone
    if    ($y eq '"')
          {$state = -1; _end_encoding()}
    elsif ($y eq '<')
          {$state = -2; push @$outptr, $y; _end_encoding()}
    elsif ($y eq '[')
          {$state = -3; push @$outptr, $y; _end_encoding()}
    elsif ($y eq '(')
          {$state =  1; push @$outptr, $y; _end_encoding()}
    elsif ($y eq '=?')
          {_start_encoding(); push @$outptr, $y}
    elsif ($y =~ m/\?=/o)
          {push @$outptr, $y; _end_encoding()}
    elsif ($y =~ m/^[])>]$/o) {
          if ($signing)   {die "Unbalanced '", $y, "'\n"}
          else            {push @$outptr, $y}
    }
    else  {$y =~ s/^\s*$/\r\n/o; push @$outptr, $y}
```

```
                # eliminate trailing WS; insert CRLF

          } else { # other zones
            if    ($y =~ s/^\s*$/\r\n/o && $signing)
                              {die "Unbalanced header ", $line}
```

```
        if    ($state == -1) { # in quoted-zone
              if    ($y eq '"') {$state = 0}
              else              {push @$outptr, $y}
        }
        elsif ($state == -2) { # in sharp-zone
              if    ($y eq '>') {$state = 0}
              push @$outptr, $y;
        }
        elsif ($state == -3) { # in square-zone
              if    ($y eq ']') {$state = 0}
              push @$outptr, $y;
        }
        elsif ($state > 0)   { # in comment-zone
              if    ($y eq '(')
                    {$state ++; push @$outptr, $y; _end_encoding()}
              elsif ($y eq ')')
                    {$state --; push @$outptr, $y; _end_encoding()}
              elsif ($y eq '=?')
                    {_start_encoding(); push @$outptr, $y}
              elsif ($y =~ m/\?=/o)
                    {push @$outptr, $y; _end_encoding()}
              else  {push @$outptr, $y}
        }
      }
    } else { # unstructured
      $y =~ s/^\s*$/\r\n/o;    # eliminate trailing WS; insert CRLF
      if    ($y eq '=?')
            {_start_encoding(); push @$outptr, $y}
      elsif ($y =~ m/\?=/o)
            {push @$outptr, $y; _end_encoding()}
      else  {push @$outptr, $y}
    }

  } until $y eq "\r\n";
  if ($encoding) {_end_encoding()}
  $line = join('', @outlist);
  return $line;
}

sub _start_encoding {     # entered at every '=?'
  @encodelist = ();
  $outptr = \@encodelist; # divert output during encoding
  $encoding = 1;
}

sub _end_encoding { # entered at every '?=' or unexpected delimiter
  my $token = "[^][()<>@,;:\"\?.=\x00-\x20\x7f-\xff]+";
  my $encoded_text = "[^\?\x00-\x20\x7f-\xff]+";
  if ($encoding) {
```

```
        $outptr = \@outlist;  # cease output diversion
        if ($y =~ m/^\?=/o) { # '?=' as expected
          $encodelist[$#encodelist] = '?='; # in case it was '?=\s=?'
          $x = join('', @encodelist);
          $genuine = $x =~ m/^=\?$token\?$token\?$encoded_text\?=$/o;
```

```
      if ($genuine)
        {$x = decode_mimewords($x)}     # dies if it fails
      if ($is_structured && $state <= 0) {
        if ($genuine) {$x =~ s/\s//go}  # eliminate FWS
      } else {
        if ($pending && not $genuine) {push @$outptr, ' '}
      }
      push @$outptr, $x;
    } else {  # unexpected delimiter during encoding
      if ($pending && (not $is_structured || $state > 0)) {
        push @$outptr, ' ';
      }
      push @$outptr, @encodelist;
    }
    $encoding = 0;
    if ($pending = $y =~ m/^\?=\s+=\?/o) {
      _start_encoding();
      push @$outptr, ('=?');
    }
  }
}
```

Appendix A.2 - Parsing of the Signed header

```
# This module must be stored in Mail/Field/Signed.pm
# relative to the other programs in the suite
package Mail::Field::Signed;

use strict;
use vars qw(@ISA);
use MIME::Field::ParamVal;
use Canon;
use Carp;

@ISA = qw(MIME::Field::ParamVal);

INIT: {
  my $x = bless([]);

  $x->register('Signed');
  $x->register('Signed_1');
  $x->register('Signed_2');
  $x->register('Signed_3');
  $x->register('Signed_4');
  $x->register('Signed_5');
  $x->register('Signed_6');
  $x->register('Signed_7');
  $x->register('Signed_8');
  $x->register('Signed_9');
```

```
   }

   sub parse {
     my ($self, $string, $signing) = @_;
     my $clean_string = _skip_CFWS($string);
```

```
  my $macro;
  $self->set($self->parse_params($clean_string));
  $self->{string} = $string;
  $self->{header_refs} = ();
  set_protocol($self->protocol);
  do {
    if ($self->{_} =~
        m/\G(((\d+:)*)(\$)|([-+]?)((\d+:)*))([-\w]+)(?!:)/og) {
        #   ((---$2--)($4) (--$5-)(---$6--))(--$8--)
      if (defined $4) {
        if ($macro = $macros{$8})
            {$self->_incorporate_header('', $2, @{$macro})}
        else { die "Unknown macro ", $8, "\n" }
      } else  {$self->_incorporate_header($5, $6, ($8))}
    } else { die "Bad header-ref-list ", $string, "\n" }
  } while ($self->{_} =~ m/,/og);
  foreach ('protocol', 'key') {
    unless($self->param($_)) {die "$_ missing\n"};
  }
  return $self;
}

sub tag {
  my $self = shift;
  return Mail::Field::tag($self);
}

sub protocol {
  my $self = shift;
  return lc($self->param('protocol'));
}

sub key {
  my $self = shift;
  return lc($self->param('key'));
}

sub sig {
  my $self = shift;
  return lc($self->param('sig'));
}

sub stringify {
  my $self = shift;
  return $self->{string};
}

sub header_refs {
  my $self = shift;
```

```
    @{$self->{header_refs}};
  }

  sub _incorporate_header {
    my ($self, $plusminus, $level, @additions) = @_;
```

```
  my $refs = \@{$self->{header_refs}};
  foreach (@additions) {
    if ($plusminus eq '-') {
      # item to be removed from list
      for (my $i = 0; $i < @$refs; $i++)
        {if (@$refs[$i] eq $level.$_) {splice(@$refs, $i, 1)} }
     } else {
      # item to be added to list
     I: {
        for (my $i = 0; $i < @$refs; $i++)
            {if (@$refs[$i] eq $level.$_) {last I} }
        push (@$refs, $level.$_);  # only if not already present
      }
    }
  }
}

sub _skip_CFWS {
  my $line = shift;
  my $count = 0;
  my @buf = ();
  while ($line =~ m/\G([^\s\("]*)\s*|\G(\()|\G(")/sog) {
                  #   (---$1----)      ($2)   (3)
    if    ($1) {push @buf, ($1)}
    elsif ($2) {  # comment
      $count += 1;
      do {
        $line =~ m/\G[^()]*([()])/sog
           or die "Unclosed comment\n";
        $count += ($1 eq '(') ? +1 : -1;
      } until ($count == 0);
    } elsif ($3) {  # quoted-string
      push @buf, ('"');
      do {
        $line =~ m/\G([^\"\s]+)|\G(\s+)|\G(")/sog;
                 #   (---$1---)   (-$2)   (3)
        if    ($1) {push @buf, ($1)}
        elsif ($2) {push @buf, (' ')}
        elsif ($3) {push @buf, ('"'); last}
      }
    }
  }
  return join('', @buf);
}

sub extract_headers {
  my ($self, $article, $FH, $proc, $signing) = @_;
  my $ref;
  my $signed = $self->stringify;
```

```
    $signed =~ s/\s*;[^;]*\bsig\b[^;]*$//io;  # remove "; sig=..."
    print $FH (&$proc($self->tag, $signed, $signing));
    foreach $ref ($self->header_refs) {
      _extract_header($article, $ref, $FH, $proc, $signing);
    }
```

```
}

sub _extract_header {
  my ($article, $ref, $FH, $proc, $signing) = @_;
  $ref =~ m/((\d+):((\d+:)*))?([-\w]+)/o;
          # ((-$2) (---$3--)) (--$5--)
  if ($1)  # $ref of the form "1:header"; call ourself recursively
    {_extract_header($article->parts($2-1), $3.$5,
                     $FH, $proc, $signing)}
  else {   # $ref is a header at the current level
    if    ($article->head->count($5) > 1)
      {die "Cannot sign duplicated header ", $5, "\n"}
    elsif ($article->head->count($5) == 1) {
      print $FH (&$proc($5, $article->head->get($5), $signing));
    }
  }
}

1;
```

Appendix A.3 - The Signing program

```
use English;
use Mail::Header;
use Mail::Field;
use Mail::Field::Signed;
use MIME::Parser;
use Canon;

$signing = 1;  # This is a program to sign headers

# Read partial Signed header from file
open SIGNED, "<".$ARGV[0];
$signed = new Mail::Header \*SIGNED;
@names = $signed->tags;
$tag = $names[0];
if ($tag !~ m/^signed(-[1-9])?$/oi || $#names != 0)
  {die "Invalid SIGNED file ", $ARGV[0], "\n"}
$line = Mail::Field->extract($tag, $signed);

if ($line->sig) {die "'sig' already present\n"}

$parser = new MIME::Parser output_to_core=>'ALL';
$parser->parse_nested_messages('NEST');
   # special treatment for message/rfc822
$article = $parser->read(\*STDIN) or die "Malformed article\n";

if ($article->head->count($tag))
  {die "Message already signed\n"}
```

```
   $tmp = "/tmp/sign-$$";
   open(FH, "> $tmp") or die "Cannot open $tmp: $!\n";
   $line->extract_headers($article, \*FH, $canonicalize, $signing);
   close(FH);
```

```
# The remainder of this code is dependent upon the particular
# implementation of OpenPGP.

$key = $line->param('key');
$pgp =
  "pgps -fab +verbose=0 +textmode=off -u $key <$tmp 2>/dev/null |";
open(FH, $pgp) or die "Cannot open pipe from pgp: $!\n";

undef $INPUT_RECORD_SEPARATOR;
$_ = <FH>;  # The OpenPGP signature record
unlink $tmp;
s/^.*[^\w+\/=\n].*\n|^\n//mog;     # remove non-base64 lines
s/^/   /mog;                       # indent by 3 spaces
s/\A/;\n   sig="\n/mo; s/\Z/"/mo;  # enclose in '; sig="..."'

$article->head->add($tag, $line->stringify . $_);
$article->print;
```

Appendix A.4 - The Verification program

```
use English;
use Mail::Header;
use Mail::Field;
use Mail::Field::Signed;
use MIME::Parser;
use Canon;

$signing = 0;  # This is a program to verify signed headers
$parser = new MIME::Parser output_to_core=>'ALL';
$parser->parse_nested_messages('NEST');
   # special treatment for message/rfc822
$article = $parser->read(\*STDIN) or die "Malformed article\n";

$tag = $ARGV[0];
unless ($tag =~ m/^Signed(-[1-9])?/io)
  {die "Bad parameter ", $tag, "\n"}

$line = Mail::Field->extract($tag, $article);
unless ($line) {die $tag, " header not found\n"}
unless ($line->param('sig')) {die "Malformed Signed header\n"}

$tmp = "/tmp/sign-$$";
open(FH, "> $tmp") or die "Cannot open $tmp: $!\n";
$line->extract_headers($article, \*FH, $canonicalize, $signing);
close(FH);

# The remainder of this code is dependent upon the particular
# implementation of OpenPGP.

use IPC::Open2;
```

```
   $pgp = "pgpv -f --batchmode -o $tmp 2>&1";
   open2(\*PIPEOUT, \*PIPEIN, $pgp);

   $armour = $line->param('sig');
```

```
   $armour =~ s/\s//sog;
   $armour =~ s/([\w+\/=]{64})/$1\n/sog;
   $armour =~ s/(=[\w+\/]{4}\Z)/\n$1/so;
   print PIPEIN "-----BEGIN PGP SIGNATURE-----\n",
               "Charset: noconv\n\n",
               $armour, "\n",
               "-----END PGP SIGNATURE-----\n";
   close(PIPEIN);
   undef $INPUT_RECORD_SEPARATOR;
   $result = <PIPEOUT>;
   unlink $tmp;

   $result =~ s/^This signature applies to another message\n//mo;
   $result =~ m/Key ID +([0-9a-fA-F]+)/iom;
   unless ("0x" . $1 eq $line->param('key')) {
     print "Signature was for key ", $line->param('key'),
           ", not for 0x", $1, "\n";
     $badsig = 1;
   }
   $badsig |= ($result !~ m/Good signature/iom);
   print $result;
   exit $badsig;
```

Appendix B - Test cases

The following, believe it or not, is a valid email message. Note that there were various TABs and much trailing whitespace in it (but they are unlikely to have survived through to the published form of this document).

```
   Subject: Unstructured headers can contain unmatched (s and unescaped
           "s; (comments like this) and "quoted strings" are not
           treated specially.
   SUMMARY: Multiple         spaces,               tabs and foldings
    in unstructured headers are reduced to a single SP, and trailing
    whitespace (of which there is much in these examples)) is ignored.
   X-Header: All X headers are "treated "as unstructured")
   from: "Scooby Doo" <foo@bar.example>   (all FWS in
      structured headers is removed, except in comments)
   tO:     "John (the Boss) Smith"  <bar@foo.example> ,
      "Bill \"fingers\"
    Sykes" <"#*\"~"@twist.example> (Observe unescaped \( and escaped "
    within quoted strings, and (properly matched) parentheses within
    comments)
   rEPLY-tO:"#*\"~"@twist.example
    (Observe "s elided, since not in <...>)
   Message-ID: <"*\"~and-other-grunge)(]["@[127.0.0.1"Ugh!]>
    (Yes that is a legal msg-id, including the " in the domain-literal)
   Sender: foo@[127.0.0.1"Ugh!] (another " in a domain-literal)
```

```
Cc: foo@[127.0.0.1(this is not], bar@[a comment)127.0.0.1],
 "=?utf-8?Q?not_an_encoded_word?="
 <=?utf-8?Q?not_an_encoded_word?=@bar.example>,
 =?us-ascii?Q?Joe_D._Bloggs_=5Bwho=20else=5d?= <foo@bar.example>,
 =?us-ascii?Q?C&A?=@bar.example (treated as an encoded-word even
 though, syntactically, it isn't)
 (in comment but =?is0-8859-1?Q?not(an_encoded-word?=))
 (=?us-ascii?Q?encoded-word_split_into-?=
 =?us-ascii?b?cGFydHM=?=)
Comments: An unstructured encoded word can have
 =?us-ascii?Q?any_characters_in_it_<>()[]"?= =?bogus_e.w?=
Date: (pre comment) sAt, 13 fEb
         1999 14:59:56 -0800 (PST)
Keywords: (various illegal constructs which nevertheless get through)
 \(Not a comment\), \" (naked quoted-pair), \ (not a quoted-SP)

Comments: Various mismatches, which should be rejected.
Foo: ) (naked \))
Bar: ((mismatched parens)
Baz: <"mismatch"
Fred: ["mismatch"
Date: Sat, 13 Feb 1999 23:00:14 GMT
Date: 29 Feb 2001 23:00:14 +0000

The following is the result of applying the PGP-Head-1
canonicalization to it (lines folded for convenience, as before, and
blank lines inserted between headers for readability).

subject: Unstructured headers can contain unmatched (s and unesca
ped "s; (comments like this) and "quoted strings" are not treated
 specially.CRLF

summary: Multiple spaces, tabs and foldings in unstructured heade
rs are reduced to a single SP, and trailing whitespace (of which
there is much in these examples)) is ignored.CRLF

x-header: All X headers are "treated "as unstructured")CRLF

from: ScoobyDoo<foo@bar.example>(all FWS in structured headers is
 removed, except in comments)CRLF

to: John(theBoss)Smith<bar@foo.example>,Bill\"fingers\"Sykes<"#*\
"~"@twist.example>(Observe unescaped \( and escaped " within quot
ed strings, and (properly matched) parentheses within comments)CRLF

reply-to: #*\"~@twist.example(Observe "s elided, since not in <...
\&.>)CRLF

message-id: <"\*"~and-other-grunge)(]["@[127.0.0.1"Ugh!]>(Yes tha
t is a legal msg-id, including the " in the domain-literal)CRLF
```

```
   sender: foo@[127.0.0.1"Ugh!](another " in a domain-literal)CRLF
```

```
   cc: foo@[127.0.0.1(thisisnot],bar@[acomment)127.0.0.1],=?utf-8?Q?
   not_an_encoded_word?=<=?utf-8?Q?not_an_encoded_word?=@bar.example
   >,JoeD.Bloggs[whoelse]<foo@bar.example>,C&A@bar.example(treated a
   s an encoded-word even though, syntactically, it isn't)(in commen
   t but =?is0-8859-1?Q?not(an_encoded-word?=))(encoded-word split i
   nto-parts)CRLF

   comments: An unstructured encoded word can have any characters in
    it <>()[]" =?bogus_e.w?=CRLF

   date: (pre comment)13feb199922:59:56+0000(PST)CRLF

   keywords: (various illegal constructs which nevertheless get thro
   ugh)\(Notacomment\),\"(naked quoted-pair),\(not a quoted-SP)CRLF
```

Appendix C - PGP Public Key

For the benefit of those who would like to experiment with the examples given in section 5, the following is the Public Key for 0xA336D40C (DSS-example).

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: PGPfreeware 5.0i for non-commercial use

mQDiBDuB8NIRAgDeGDfg+ZlgHDZkkXDpeeaBJxIq9/pkuFL/6puw9j+k/JsKzLr9
ktqlFgkdnDyYbWm26lWAmjliZEeIyggBlxSlAKD/lbF/4JAJox/7xqW8fuSc9sPO
AwIA0rQJ1TEhIztyUYB5j4D9V7pHKyhbdifFEf1MwrYsnluiejd5/K623J4wQr/m
+zMzr7lnX6ZLPkITKgfgpjoAWQIAzg9BAYwHGVgjRg82MxxlP5737ihfa0yWMeVn
KTU1mToKMaokGMrMnvuOjvu6GmgHdbfgaFXThrnuerN8rRqVP7QLRFNTLWV4YW1w
bGWJAEsEEBECAAsFAjuB8NIECwMBAgAKCRAkESrJozbUDIgdAKCc4eqIbAFlOB6O
rWv8CzMPBNo2ZACeKOD6mS+GrEgQkD+cW1MytHVjFTE=
=Zl1B
-----END PGP PUBLIC KEY BLOCK-----
```

[Would it be useful to include descriptions of pgpverify and pgpmoose as additional appendices?]