Network Working Group
Request for Comments: 1991
Category: Informational

D. Atkins
MIT
W. Stallings
Comp-Comm Consulting
P. Zimmermann
Boulder Software Engineering

August 1996

PGP Message Exchange Formats

Status of This Memo

Table of Contents

1. Introduction

PGP (Pretty Good Privacy) uses a combination of public-key and conventional encryption to provide security services for electronic mail messages and data files. These services include confidentiality and digital signature. PGP is widely used throughout the global computer community. This document describes the format of "PGP files", i.e., messages that have been encrypted and/or signed with PGP.

PGP was created by Philip Zimmermann and first released, in Version 1.0, in 1991. Subsequent versions have been designed and implemented by an all-volunteer collaborative effort under the design guidance of Philip Zimmermann. PGP and Pretty Good Privacy are trademarks of Philip Zimmermann.

This document describes versions 2.x of PGP. Specifically, versions 2.6 and 2.7 conform to this specification. Version 2.3 conforms to this specification with minor differences.

A new release of PGP, known as PGP 3.0, is anticipated in 1995. To the maximum extent possible, this version will be upwardly compatible with version 2.x. At a minimum, PGP 3.0 will be able to read messages and signatures produced by version 2.x.

2. PGP Services

PGP provides four services related to the format of messages and data files: digital signature, confidentiality, compression, and radix-64 conversion.

2.1 Digital signature

The digital signature service involves the use of a hash code, or message digest, algorithm, and a public-key encryption algorithm. The sequence is as follows:

- -the sender creates a message
- -the sending PGP generates a hash code of the message
- -the sending PGP encrypts the hash code using the sender's private key
- -the encrypted hash code is prepended to the message
- -the receiving PGP decrypts the hash code using the sender's public key
- -the receiving PGP generates a new hash code for the received message and compares it to the decrypted hash code. If the two match, the message is accepted as authentic

Although signatures normally are found attached to the message or file that they sign, this is not always the case: detached signatures are supported. A detached signature may be stored and transmitted separately from the message it signs. This is useful in several contexts. A user may wish to maintain a separate signature log of all messages sent or received. A detached signature of an executable program can detect subsequent virus infection. Finally, detached signatures can be used when more than one party must sign a document, such as a legal contract. Each person's signature is independent and therefore is applied only to the document. Otherwise, signatures would have to be nested, with the second signer signing both the document and the first signature, and so on.

2.2 Confidentiality

PGP provides confidentiality by encrypting messages to be transmitted or data files to be stored locally using conventional encryption. In PGP, each conventional key is used only once. That is, a new key is generated as a random 128-bit number for each message. Since it is to be used only once, the session key is bound to the message and transmitted with it. To protect the key, it is encrypted with the receiver's public key. The sequence is as follows:

- -the sender creates a message
- -the sending PGP generates a random number to be used as a session key for this message only
- -the sending PGP encrypts the message using the session key
- -the session key is encrypted using the recipient's public key and prepended to the encrypted message
- -the receiving PGP decrypts the session key using the recipient's private key

-the receiving PGP decrypts the message using the session key

Both digital signature and confidentiality services may be applied to the same message. First, a signature is generated for the message and prepended to the message. Then, the message plus signature is encrypted using a conventional session key. Finally, the session key is encrypted using public-key encryption and prepended to the encrypted block.

2.3 Compression

As a default, PGP compresses the message after applying the signature but before encryption.

2.4 Radix-64 conversion

When PGP is used, usually part of the block to be transmitted is encrypted. If only the signature service is used, then the message digest is encrypted (with the sender's private key). If the confidentiality service is used, the message plus signature (if present) are encrypted (with a one-time conventional key). Thus, part or all of the resulting block consists of a stream of arbitrary 8-bit bytes. However, many electronic mail systems only permit the use of blocks consisting of ASCII text. To accommodate this restriction, PGP provides the service of converting the raw 8-bit binary stream to a stream of printable ASCII characters, called ASCII Armor.

The scheme used for this purpose is radix-64 conversion. Each group of three bytes of binary data is mapped into 4 ASCII characters. This format also appends a CRC to detect transmission errors. This radix-64 conversion, also called Ascii Armor, is a wrapper around the binary PGP messages, and is used to protect the binary messages during transmission over non-binary channels, such as Internet Email.

The following table defines the mapping. The characters used are the upper- and lower-case letters, the digits 0 through 9, and the characters + and /. The carriage-return and linefeed characters aren't used in the conversion, nor is the tab or any other character that might be altered by the mail system. The result is a text file that is "immune" to the modifications inflicted by mail systems.

6-bit character value	encoding	6-bit character value	encoding	6-bit character value	encoding	6-bit character value	encoding
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
1	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/
						(pad)	=

It is possible to use PGP to convert any arbitrary file to ASCII Armor. When this is done, PGP tries to compress the data before it is converted to Radix-64.

2.4.1 ASCII Armor Formats

When PGP encodes data into ASCII Armor, it puts specific headers around the data, so PGP can reconstruct the data at a future time. PGP tries to inform the user what kind of data is encoded in the ASCII armor through the use of the headers.

ASCII Armor is created by concatenating the following data:

- An Armor Headerline, appropriate for the type of data
- Armor Headers
- A blank line
- The ASCII-Armored data
- An Armor Checksum
- The Armor Tail (which depends on the Armor Headerline).

An Armor Headerline is composed by taking the appropriate headerline text surrounded by five (5) dashes (-) on either side of the headerline text. The headerline text is chosen based upon the type of data that is being encoded in Armor, and how it is being encoded. Headerline texts include the following strings:

```
BEGIN PGP MESSAGE -- used for signed, encrypted, or compressed files
BEGIN PGP PUBLIC KEY BLOCK -- used for transferring public keys
```

BEGIN PGP MESSAGE, PART X/Y -- used for multi-part messages, where the armor is split amongst Y files, and this is the Xth file out of Y.

The Armor Headers are pairs of strings that can give the user or the receiving PGP program some information about how to decode or use the message. The Armor Headers are a part of the armor, not a part of the message, and hence should not be used to convey any important information, since they can be changed in transport.

The format of an Armor Header is that of a key-value pair, the encoding of RFC-822 headers. PGP should consider improperly formatted Armor Headers to be corruption of the ASCII Armor. Unknown Keys should be reported to the user, but so long as the RFC-822 formatting is correct, PGP should continue to process the message. Currently defined Armor Header Keys include "Version" and "Comment", which define the PGP Version used to encode the message and a user-defined comment.

The Armor Checksum is a 24-bit CRC converted to four bytes of radix-64 encoding, prepending an equal-sign (=) to the four-byte code. The CRC is computed by using the generator 0x864CFB and an initialization of 0xB704CE. The accumulation is done on the data before it is converted to radix-64, rather than on the converted data. For more information on CRC functions, the reader is asked to look at chapter 19 of the book "C Programmer’s Guide to Serial Communications," by Joe Campbell.

The Armor Tail is composed in the same manner as the Armor Headerline, except the string "BEGIN" is replaced by the string "END".

3. Data Element Formats

3.1 Byte strings

The objects considered in this document are all "byte strings." A byte string is a finite sequence of bytes. The concatenation of byte string X of length M with byte string Y of length N is a byte string Z of length M + N; the first M bytes of Z are the bytes of X in the same order, and the remaining N bytes of Z are the bytes of Y in the same order.

Literal byte strings are written from left to right, with pairs of hex nibbles separated by spaces, enclosed by angle brackets: for instance, <05 ff 07> is a byte string of length 3 whose bytes have numeric values 5, 255, and 7 in that order. All numbers in this document outside angle brackets are written in decimal.

The byte string of length 0 is called "empty" and written <>.

3.2 Whole number fields

Purpose. A whole number field can represent any nonnegative integer, in a format where the field length is known in advance.

Definition. A whole number field is any byte string. It is stored in radix-256 MSB-first format. This means that a whole number field of length N with bytes b_0 b_1 ... b_{N-2} b_{N-1} in that order has value

b_0 * 256^{N-1} + b_1 * 256^{N-2} + ... + b_{N-2} * 256 + b_{N-1}.

Examples. The byte string <00 0D 64 11 00 00> is a valid whole number field with value 57513410560. The byte string <FF> is a valid whole number field with value 255. The byte string <00 00> is a valid whole number field with value 0. The empty byte string <> is a valid whole number field with value 0.

3.3 Multiprecision fields

Purpose. A multiprecision field can represent any nonnegative integer which is not too large. The field length need not be known in advance. Multiprecision fields are designed to waste very little space: a small integer uses a short field.

Definition. A multiprecision field is the concatenation of two fields:

(a) a whole number field of length 2, with value B;
(b) a whole number field, with value V.

Field (b) is of length [(B+7)/8], i.e., the greatest integer which is no larger than (B+7)/8. The value of the multiprecision field is defined to be V. V must be between 2^{B-1} and 2^B - 1 inclusive. In other words B must be exactly the number of significant bits in V.

Some implementations may limit the possible range of B. The implementor must document which values of B are allowed by an implementation.

Examples. The byte string <00 00> is a valid multiprecision integer with value 0. The byte string <00 03 05> is a valid multiprecision field with value 5. The byte strings <00 03 85> and <00 00 00> are not valid multiprecision fields. The former is invalild because <85> has 8 significant bits, not 3; the latter is invalid because the second field has too many bytes of data given the value of the first

field. The byte string <00 09 01 ff> is a valid multiprecision field with value 511. The byte string <01 00 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 07> is a valid multiprecision field with value 2^255 + 7.

3.4 String fields

Purpose. A string field represents any sequence of bytes of length between 0 and 255 inclusive. The length need not be known in advance. By convention, the content of a string field is normally interpreted as ASCII codes when it is displayed.

Definition. A string field is the concatenation of the following:

(a) a whole number field of length 1, with value L;
(b) a byte string of length L.

The content of the string field is defined to be field (b).

Examples: <05 48 45 4c 4c 4f> is a valid string field which would normally be displayed as the string HELLO. <00> is a valid string field which would normally be displayed as the empty string. <01 00> is a valid string field.

3.5 Time fields

Purpose. A time field represents the number of seconds elapsed since 1970 Jan 1 00:00:00 GMT. It is compatible with the usual representation of times under UNIX.

Definition. A time field is a whole number field of length 4, with value V. The time represented by the time field is the one-second interval beginning V seconds after 1970 Jan 1 00:00:00 GMT.

4. Common Fields

This section defines fields found in more than one packet format.

4.1 Packet structure fields

Purpose. The packet structure field distinguishes between different types of packets, and indicates the length of packets.

Definition. A packet structure field is a byte string of length 1, 2, 3, or 5. Its first byte is the cipher type byte (CTB), with bits labeled 76543210, 7 the most significant bit and 0 the least significant bit. As indicated below the length of the packet structure field is determined by the CTB.

CTB bits 76 have values listed in the following table:

10 - normal CTB
11 - reserved for future experimental work
all others - reserved

CTB bits 5432, the "packet type bits", have values listed in the following table:

0001 - public-key-encrypted packet
0010 - signature packet
0101 - secret-key certificate packet
0110 - public-key certificate packet
1000 - compressed data packet
1001 - conventional-key-encrypted packet
1011 - literal data packet
1100 - keyring trust packet
1101 - user id packet
1110 - comment packet (*)
all others - reserved

CTB bits 10, the "packet-length length bits", have values listed in the following table:

00 - 1-byte packet-length field
01 - 2-byte packet-length field
10 - 4-byte packet-length field
11 - no packet length supplied, unknown packet length

As indicated in this table, depending on the packet-length length bits, the remaining 1, 2, 4, or 0 bytes of the packet structure field are a "packet-length field". The packet-length field is a whole number field. The value of the packet-length field is defined to be the value of the whole number field.

A value of 11 is currently used in one place: on compressed data. That is, a compressed data block currently looks like <A3 01 . . .>, where <A3>, binary 10 1000 11, is an indefinite-length packet. The proper interpretation is "until the end of the enclosing structure", although it should never appear outermost (where the enclosing structure is a file).

Options marked with an asterisk (*) are not implemented yet; PGP 2.6.2 will never output this packet type.

4.2 Number ID fields

Purpose. The ID of a whole number is its 64 least significant bits. The ID is a convenient way to distinguish between large numbers such as keys, without having to transmit the number itself. Thus, a number that may be hundreds or thousands of decimal digits in length can be identified with a 64-bit identifier. Two keys may have the same ID by chance or by malice; although the probability that two large keys chosen at random would have the same ID is extremely small.

Definition. A number ID field is a whole number field of length 8. The value of the number ID field is defined to be the value of the whole number field.

4.3 Version fields

Many packet types include a version number as the first byte of the body. The format and meaning of the body depend on the version number. More versions of packets, with new version numbers, may be defined in the future. An implementation need not support every version of each packet type. However, the implementor must document which versions of each packet type are supported by the implementation.

A version number of 2 or 3 is currently allowed for each packet format. New versions will probably be numbered sequentially up from 3. For backwards compatibility, implementations will usually be expected to support version N of a packet whenever they support version N+1. Version 255 may be used for experimental purposes.

5. Packets

5.1 Overview

A packet is a digital envelope with data inside. A PGP file, by definition, is the concatenation of one or more packets. In addition, one or more of the packets in a file may be subject to a transformation using encryption, compression, or radix-64 conversion.

A packet is the concatenation of the following:

(a) a packet structure field;
(b) a byte string of some length N.

Byte string (b) is called the "body" of the packet. The value of the packet-length field inside the packet structure field (a) must equal N, the length of the body.

Other characteristics of the packet are determined by the type of the packet. See the definitions of particular packet types for further details. The CTB packet-type bits inside the packet structure always indicate the packet type.

Note that packets may be nested: one digital envelope may be placed inside another. For example, a conventional-key-encrypted packet contains a disguised packet, which in turn might be a compressed data packet.

5.2 General packet structure

A pgp file consists of three components: a message component, a signature (optional), and a session key component (optional).

5.2.1 Message component

The message component includes the actual data to be stored or transmitted as well as a header that includes control information generated by PGP. The message component consists of a single literal data packet.

5.2.2 Signature component

The signature component is the signature of the message component, formed using a hash code of the message component and the public key of the sending PGP entity. The signature component consists of a single signature packet.

If the default option of compression is chosen, then the block consisting of the literal data packet and the signature packet is compressed to form a compressed data packet.

5.2.3 Session key component

The session key component includes the encrypted session key and the identifier of the recipients public key used by the sender to encrypt the session key. The session key component consists of a single public-key-encrypted packet for each recipient of the message.

If compression has been used, then conventional encryption is applied to the compressed data packet formed from the compression of the signature packet and the literal data packet. Otherwise, conventional encryption is applied to the block consisting of the signature packet and the literal data packet. In either case, the cyphertext is referred to as a conventional-key-encrypted data packet.

6. PGP Packet Types

PGP includes the following types of packets:

- -literal data packet
- -signature packet
- -compressed data packet
- -conventional-key-encrypted data packet
- -public-key-encrypted packet
- -public-key packet
- -User ID packet

6.1 Literal data packets

Purpose. A literal data packet is the lowest level of contents of a digital envelope. The data inside a literal data packet is not subject to any further interpretation by PGP.

Definition. A literal data packet is the concatenation of the following fields:

(a) a packet structure field;
(b) a byte, giving a mode;
(c) a string field, giving a filename;
(d) a time field;
(e) a byte string of literal data.

Fields (b), (c), and (d) suggest how the data should be written to a file. Byte (b) is either ASCII b <62>, for binary, or ASCII t <74>, for text. Byte (b) may also take on the value ASCII 1, indicating a machine-local conversion. It is not defined how PGP will convert this across platforms.

Field (c) suggests a filename. Field (d) should be the time at which the file was last modified, or the time at which the data packet was created, or 0.

Note that only field (e) of a literal data packet is fed to a message-digest function for the formation of a signature. The exclusion of the other fields ensures that detached signatures are exactly the same as attached signatures prefixed to the message. Detached signatures are calculated on a separate file that has none of the literal data packet header fields.

6.2 Signature packet

Purpose. Signatures are attached to data, in such a way that only one entity, called the "writer," can create the signature. The writer must first create a "public key" K and distribute it. The writer keeps certain private data related to K. Only someone cooperating with the writer can sign data using K, enveloping the data in a signature packet (also known as a private-key-encrypted packet). Anyone can look through the glass in the envelope and verify that the signature was attached to the data using K. If the data is altered in any way then the verification will fail.

Signatures have different meanings. For example, a signature might mean "I wrote this document," or "I received this document." A signature packet includes a "classification" which expresses its meaning.

Definition. A signature packet, version 2 or 3, is the concatenation of the following fields:

(a) packet structure field (2, 3, or 5 bytes);
(b) version number = 2 or 3 (1 byte);
(c) length of following material included in MD calculation (1 byte, always the value 5);
(d1) signature classification (1 byte);
(d2) signature time stamp (4 bytes);
(e) key ID for key used for singing (8 bytes);
(f) public-key-cryptosystem (PKC) type (1 byte);
(g) message digest algorithm type (1 byte);
(h) first two bytes of the MD output, used as a checksum (2 bytes);
(i) a byte string of encrypted data holding the RSA-signed digest.

The message digest is taken of the bytes of the file, followed by the bytes of field (d). It was originally intended that the length (c) could vary, but now it seems that it will alwaye remain a constant value of 5, and that is the only value that will be accepted. Thus, only the fields (d1) and (d2) will be hashed into the signature along with the main message.

6.2.1 Message-digest-related fields

The message digest algorithm is specified by the message digest (MD) number of field (g). The following MD numbers are currently defined:

1 - MD5 (output length 16)
255 - experimental

More MD numbers may be defined in the future. An implementation need not support every MD number. The implementor must document the MD numbers understood by an implementation.

A message digest algorithm reads a byte string of any length, and writes a byte string of some fixed length, as indicated in the table above.

The input to the message digest algorithm is the concatenation of some "primary input" and some "appended input."

The appended input is specified by field (c), which gives a number of bytes to be taken from the following fields: (d1), (d2), and so on. The current implementation uses the value 5 for this number, for fields (d1) and (d2). Any field not included in the appended input is not "signed" by field (i).

The primary input is determined by the signature classification byte (d1). Byte (d1) is one of the following hex numbers, with these meanings:

<00> - document signature, binary image ("I wrote this document")
<01> - document signature, canonical text ("I wrote this document")
<10> - public key packet and user ID packet, generic certification ("I think this key was created by this user, but I won't say how sure I am")
<11> - public key packet and user ID packet, persona certification ("This key was created by someone who has told me that he is this user") (#)
<12> - public key packet and user ID packet, casual certification ("This key was created by someone who I believe, after casual verification, to be this user") (#)
<13> - public key packet and user ID packet, positive certification ("This key was created by someone who I believe, after heavy-duty identification such as picture ID, to be this user") (#)
<20> - public key packet, key compromise ("This is my key, and I have revoked it")

<30> - public key packet and user ID packet, revocation ("I retract all my previous statements that this key is related to this user") (*)
<40> - time stamping ("I saw this document") (*)

More classification numbers may be defined in the future to handle other meanings of signatures, but only the above numbers may be used with version 2 or version 3 of a signature packet. It should be noted that PGP 2.6.2 has not implemented the packets marked with an asterisk (*), and the packets marked with a hash (#) are not output by PGP 2.6.2.

Signature packets are used in two different contexts. One (signature type <00> or <01>) is of text (either the contents of a literal packet or a separate file), while types <10> through <1F> appear only in key files, after the keys and user IDs that they sign. Type <20> appears in key files, after the keys that it signs, and type <30> also appears after a key/userid combination. Type <40> is intended to be a signature of a signature, as a notary seal on a signed document.

The output of the message digest algorithm is a message digest, or hash code. Field i contains the cyphertext produced by encrypting the message digest with the signer’s private key. Field h contains the first two bytes of the unencrypted message digest. This enables the recipient to determine if the correct public key was used to decrypt the message digest for authentication, by comparing this plaintext copy of the first two byes with the first two bytes of the decrypted digest. These two bytes also serve as a 16-bit frame check sequence for the message.

6.2.2 Public-key-related fields

The message digest is signed by encrypting it using the writer’s private key. Field (e) is the ID of the corresponding public key.

The public-key-encryption algorithm is specified by the public-key cryptosystem (PKC) number of field (f). The following PKC numbers are currently defined:

1 - RSA
255 - experimental

More PKC numbers may be defined in the future. An implementation need not support every PKC number. The implementor must document the PKC numbers understood by an implementation.

A PKC number identifies both a public-key encryption method and a signature method. Both of these methods are fully defined as part of the definition of the PKC number. Some cryptosystems are usable only for encryption, or only for signatures; if any such PKC numbers are defined in the future, they will be marked appropriately.

6.2.3 RSA signatures

An RSA-signed byte string is a multiprecision field that is formed by taking the message digest and filling in an ASN structure, and then encrypting the whole byte string in the RSA key of the signer.

PGP versions 2.3 and later encode the MD into a PKCS-format signature string, which has the following format:

```
MSB                 .   .   .                    LSB
0   1   <FF>(n bytes)   0   ASN(18 bytes)   MD(16 bytes)
```

See RFC1423 for an explanation of the meaning of the ASN string. It is the following 18 byte long hex value:

<30 20 30 0C 06 08 2A 86 48 86 F7 0D 02 05 05 00 04 10>

Enough bytes of <FF> padding are added to make the length of this whole string equal to the number of bytes in the modulus.

6.2.4 Miscellaneous fields

The timestamp field (d2) is analogous to the date box next to a signature box on a form. It represents a time which is typically close to the moment that the signature packet was created. However, this is not a requirement. Users may choose to date their signatures as they wish, just as they do now in handwritten signatures.

If an application requires the creation of trusted timestamps on signatures, a detached signature certificate with an untrusted timestamp may be submitted to a trusted timestamp notary service to sign the signature packet with another signature packet, creating a signature of a signature. The notary’s signature’s timestamp could be used as the trusted "legal" time of the original signature.

6.3 Compressed data packets

Purpose. A compressed data packet is an envelope which safely squeezes its contents into a small space.

Definition. A compressed data packet is the concatenation of the following fields:

(a) a packet structure field;
(b) a byte, giving a compression type;
(c) a byte string of compressed data.

Byte string (c) is a packet which may be decompressed using the algorithm identified in byte (b). Typically, the data that are compressed consist of a literal data packet or a signature packet concatenated to a literal data packet.

A compression type selects a compression algorithm for use in compressed data packets. The following compression numbers are currently defined.

1 - ZIP
255 - experimental

More compression numbers may be defined in the future. An implementation need not support every MD number. The implementor must document the compression numbers understood by an implementation.

6.4 Conventional-key-encrypted data packets

Purpose. A conventional-key-encrypted data packet is formed by encrypting a block of data with a conventional encryption algorithm using a one-time session key. Typically, the block to be encrypted is a compressed data packet.

Definition. A conventional-key-encrypted data packet is the concatenation of the following fields:

(a) a packet structure field;
(b) a byte string of encrypted data.

The plaintext or compressed plaintext that is encrypted to form field (b) is first prepended with 64 bits of random data plus 16 "key check" bits. The random prefix serves to start off the cipher feedback chaining process with 64 bits of random material; this serves the same function as an initialization vector (IV) for a cipher-block-chaining encryption scheme. The key check prefix is

equal to the last 16 bits of the random prefix. During decryption, a comparison is made to see if the 7th and 8th byte of the decrypted material match the 9th and 10th bytes. If so, then the conventional session key used for decryption is assumed to be correct.

6.4.1 Conventional-encryption type byte

Purpose. The conventional-encryption type byte is used to determine what conventional encryption algorithm is in use. The algorithm type byte will also define how long the conventional encryption key is, based upon the algorithm in use.

Definition. A conventional-encryption type byte is a single byte which defines the algorithm in use. It is possible that the algorithm in use may require further definition, such as key-length. It is up to the implementor to document the supported key-length in such a situation.

1 - IDEA (16-byte key)
255 - experimental

6.5 Public-key-encrypted packets

Purpose. The public-key-encrypted packet is the format for the session key component of a message. The purpose of this packet is to convey the one-time session key used to encrypt the message to the recipient in a secure manner. This is done by encrypting the session key with the recipient’s public key, so that only the recipient can recover the session key.

Definition. A public-key-encrypted packet, version 2 or 3, is the concatenation of the following fields:

(a) a packet structure field;
(b) a byte, giving the version number, 2 or 3;
(c) a number ID field, giving the ID of a key;
(d) a byte, giving a PKC number;
(e) a byte string of encrypted data (DEK).

Byte string (e) represents the value of the session key, encrypted using the reader’s public key K, under the cryptosystem identified in byte (d).

The value of field (c) is the ID of K.

Note that the packet does not actually identify K: two keys may have the same ID, by chance or by malice. Normally it will be obvious from the context which key K was used to create the packet. But

sometimes it is not obvious. In this case field (c) is useful. If, for example, a reader has created several keys, and receives a message, then he should attempt to decrypt the message only with the key whose ID matches the value of field (c). If he has accidentally generated two keys with the same ID, then he must attempt to decrypt the message with both keys, but this case is highly unlikely to occur by chance.

6.5.1 RSA-encrypted data encryption key (DEK)

The Data Encryption Key (DEK) is a multiprecision field which stores an RSA encrypted byte string. The byte string is a PKCS encoding of the secret key used the encrypt the message, with random padding for each Public-Key encrypted packet.

PGP version 2.3 and later encode the DEK into an MPI using the following format:

```
  MSB                          .   .   .                          LSB
   0   2   RND(n bytes)   0  ALG(1 byte)  DEK(k bytes)  CSUM(2 bytes)
```

ALG refers to the algorithm byte for the secret key algorithm used to encrypt the data packet. The DEK is the actual Data Encryption Key, and its size is dependent upon the encryption algorithm defined by ALG. For the IDEA encryption algorithm, type byte 1, the DEK is 16 bytes long. CSUM is a 16-bit checksum of the DEK, used to determine that the correct Private key was used to decrypt this packet. The checksum is computed by the 16-bit sum of the bytes in the DEK. RND is random padding to expand the byte to fill the size of the RSA Public Key that is used to encrypt the whole byte.

6.6 Public Key Packet

Purpose. A public key packet defines an RSA public key.

Definition. A public key packet is the concatenation of the following fields:

(a) packet structure field (2 or 3 bytes);
(b) version number = 2 or 3 (1 byte);;
(c) time stamp of key creation (4 bytes);
(d) validity period in days (0 means forever) (2 bytes);
(e) public-key-cryptosystem (PKC) type (1 byte);
(f) MPI of RSA public modulus n;
(g) MPI of RSA public encryption exponent e.

The validity period is always set to 0.

6.7 User ID Packet

Purpose. A user ID packet identifies a user and is associated with a public or private key.

Definition. A user ID packet is the concatenation of the following fields:

(a) packet structure field (2 bytes);
(b) User ID string.

The User ID string may be any string of printable ASCII characters. However, since the purpose of this packet is to uniquely identify an individual, the usual practice is for the User ID string to consist of the user's name followed by an e-mail address for that user, the latter enclosed in angle brackets.

7. Transferable Public Keys

Public keys may transferred between PGP users. The essential elements of a transferable public key are

(a) One public key packet;
(b) One or more user ID packets;
(c) Zero or more signature packets

The public key packet occurs first. Each of the following user ID packets provides the identity of the owner of this public key. If there are multiple user ID packets, this corresponds to multiple means of identifying the same unique individual user; for example, a user may enjoy the use of more than one e-mail address, and construct a user ID packet for each one. Immediately following each user ID packet, there are zero or more signature packets. Each signature packet is calculated on the immediately preceding user ID packet and the initial public key packet. The signature serves to certify the corresponding public key and user ID. In effect, the signer is testifying to his or her belief that this public key belongs to the user identified by this user ID.

8. Acknowledgments

Philip Zimmermann is the creator of PGP 1.0, which is the precursor of PGP 2.x. Major parts of later versions of PGP have been implemented by an international collaborative effort involving a large number of contributors, under the design guidance of Philip Zimmermann.

9. Security Considerations

Security issues are discussed throughout this memo.

10. Authors' Addresses

Derek Atkins
12 Rindge Ave. #1R
Cambridge, MA

Phone: +1 617 868-4469
EMail: warlord@MIT.EDU

William Stallings
Comp-Comm Consulting
P. O. Box 2405
Brewster, MA 02631

EMail: stallings@ACM.org

Philip Zimmermann
Boulder Software Engineering
3021 Eleventh Street
Boulder, Colorado 80304 USA

Phone: +1-303-541-0140
EMail: prz@acm.org