UNIT II

E-MAIL SECURITY & FIREWALLS

UNIT II

- PGP
- S/MIME
- Internet Firewalls for Trusted System
  - Roles of Firewalls
  - Firewall related terminology
  - Types of Firewalls
  - Firewall designs
- SET for E-Commerce Transactions

Electronic Mail Security

- **PGP**
- **S/MIME**

  - PGP and S/MIME are on an IETF standards track
  - PGP will remain the choice for **personnel e-mail** security
  - S/MIME will emerge as the industry standard for **commercial and organisational use**

Electronic Mail Security

- **PGP (**Pretty Good Privacy)
  - Invented by Philip Zimmermann
  - Version 1.0 in 1991
  - Subsequent versions 2.6.x and 5.x (or 3.0)
  - Individual and commercial versions
  - Runs on a variety of platforms
  - Uses a combination of symmetric secret-key and asymmetric public-key encryption
  - Provide security services and data integrity services for electronic mail and data files
  - Use
    - Digital signature
    - Encryption
    - Compression (zip)
    - Radix-64 conversion

Electronic Mail Security

- **S/MIME** (Secure/Multipurpose Internet Mail Extension)

- RFC 2822 specifies “Internet Message Format “
- It defines a format for text messages being sent using e-mail
- MIME is an extension to the RFC 2822 framework
- MIME address some of the problems and limitations of SMTP (Simple Mail Transfer Protocol)

- (S/MIME) is a security enhancement to the MIME Internet e-mail format standard, based on technology from RSA Data Security

PGP

- Notation

K_s = session key	H = hash function
KP_a = public key of user A	KP_b = public key of user B
KS_a = private key of user A	KS_b = private key of user B
E = conventional encryption	D = conventional decryption
E_p = public-key encryption	D_p = public-key decryption
Z = compression using zip algorithm	Z^{-1} = decompression
$\|\|$ = concatenation	

PGP

- **Confidentiality via Encryption**
  - In PGP data files stored or transmitted is provided confidentiality by encrypting messages
  - conventional encryption algorithm such as IDEA, 3DES or CAST- 128 are used
  - Symmetric key (session key)
    - used only once
    - A new session key (128 bit) is generated for each message
    - Since it is used only once, the session key is bound to the message and transmitted with it
    - Key is encrypted with the receiver's public key for security

PGP

- **Confidentiality via Encryption**
  - The sender creates a message
  - The sending PGP generates a session key (128 bit) for this message only
  - The session key is encrypted with RSA, using the recipient's public key
  - The sending PGP encrypts (CAST-128,IDEA, 3DES,) the message with the session key
  - Message is also compressed
  - The receiving PGP uses RSA with its private key to decrypt and recover the session key
  - The receiving PGP decrypts the message using the session key
  - If the message was compressed, it will be decompressed

PGP

Confidentiality via Encryption

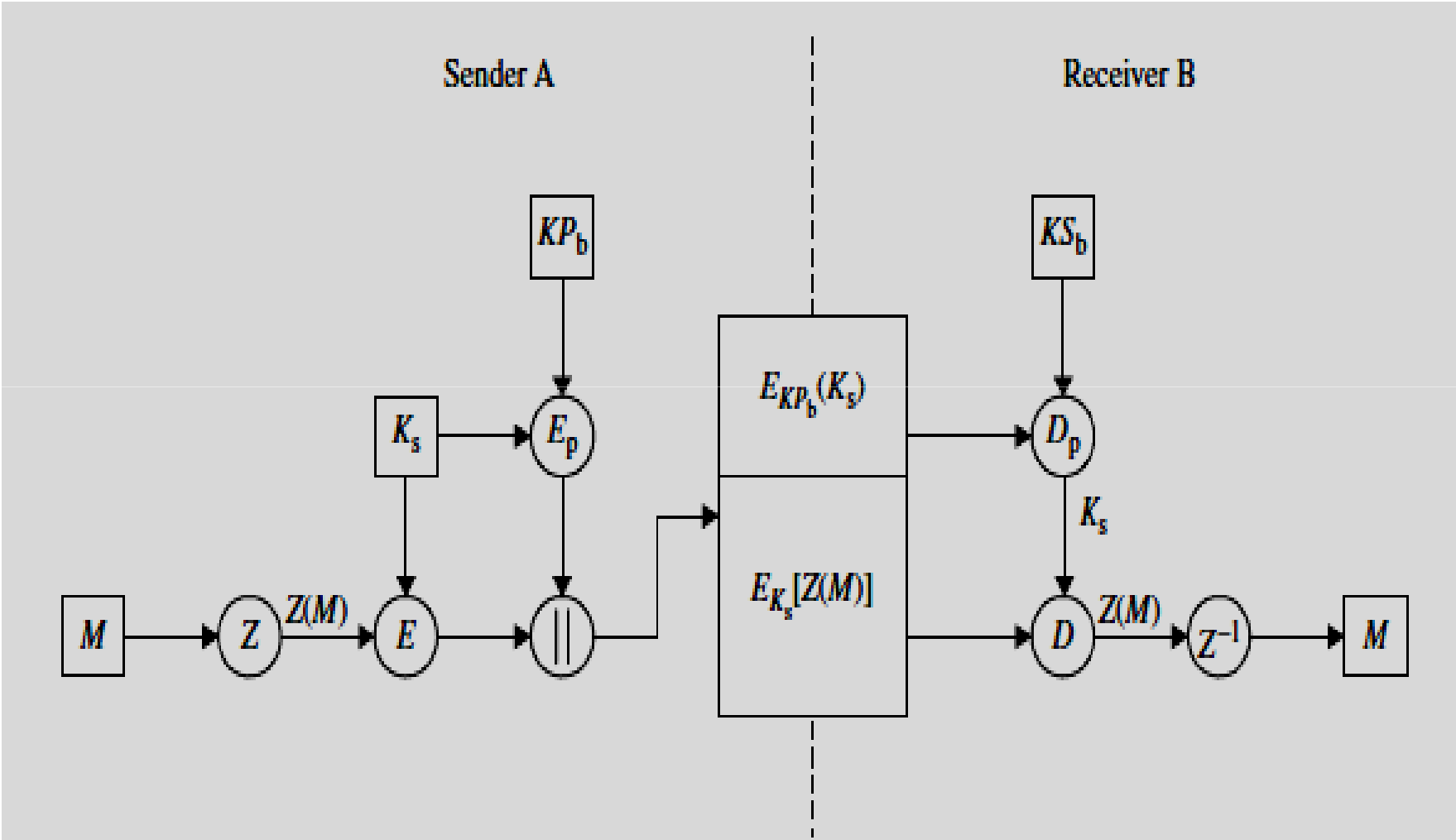


Figure 9.1 PGP confidentiality computation scheme with compression/decompression Algorithms.

PGP

- **Confidentiality via Encryption**
  - Key
    - key encryption
      - RSA
      - Variant of Diffie–Hellman (known as ElGamal)
    - To reduce encryption time combination of conventional and public-key encryption is used
    - CAST-128 and other conventional algorithms are substantially faster than RSA or ElGamal
    - Public-key algorithms is used to exchange session key
    - PGP provide the user with a range of key size options from 768 to 3072 bits
  - Message
    - Both digital signature and confidentiality services may be applied to the same message
    - First, a signature is generated from the message and attached to the message
    - Then the message plus signature are encrypted using a symmetric session key
    - Finally, the session key is encrypted using public-key encryption and prefixed to the encrypted block

PGP

- RSA
  - prime no's **P = 53, Q = 59**
  - **n = P*Q = 3127**
  - Exponent **e**: **1 < e < Φ(n) e=3**
  - Φ(n) = (P-1)(Q-1) so, **Φ(n) = 3016**
  - Private Key, **d : d = (k*Φ(n) + 1) / e**
  - Integer k: k = 2, value of d is 2011
  - Public Key (n = 3127 and e = 3) and Private Key(d = 2011)
  - Eg:
    - H = 8 and I = 9
      Encryption: **$c = 89^e \bmod n$** = 1394
    - Decrypt 1349 : **Data= $c^d \bmod n$**= 89
    - 8 = H and I = 9 i.e. "HI".

PGP

- Diffie–Hellman key exchange algorithm
  - Provides a mechanism that allows two users to agree on a shared secret key without requiring encryption
    - $Yi \equiv \alpha^{Xi} \pmod q$
    - $Yj \equiv \alpha^{Xj} \pmod q$
    - $Kij \equiv \alpha^{XiXj} \pmod q$
  - Kij is common secret key
  - This shared key is immediately available for use in encrypting subsequent data transmission
  - Allows two users to agree on a shared secret key without requiring encryption

PGP

- ElGamal Cryptosystem
  - Prime number **p =17**
  - Generator **g =6**
  - Alice **a =5** and computes **A = g^a**
    - **g^a mod p = 6^5 mod 17 = 7**
  - Alice's public key is **(17, 6, 7): (p, g, g^a)** and her private key is **a**
  - Bob picks a number **k** which is smaller than p and message **m** and computes **c_1** and **c_2** and send it to Alice

$$c_1 = g^k \bmod p$$
$$c_2 = A^k * m \bmod p$$

  - Alice compute $c_1^{-a} * c_2 \bmod p = m$ $(g^k)^{-a} * (g^a)^k * m = m$

PGP

- **Authentication via Digital Signature**
  - The digital signature uses a hash code of the message digest algorithm, and a public-key signature algorithm
  - Digital signature service is provided by PGP
    - The sequence is as follows:
      - The sender creates a message
      - SHA-1 is used to generate a 160-bit hash code of the message
      - The hash code is encrypted with RSA using the sender's private key and a digital signature is produced
      - The binary signature is attached to the message
      - The receiver uses RSA with the sender's public key to decrypt and recover the hash code
      - The receiver generates a new hash code for the received message and compares it with the decrypted hash code. If the two match, the message is accepted as authentic

PGP

- **Authentication via Digital Signature**

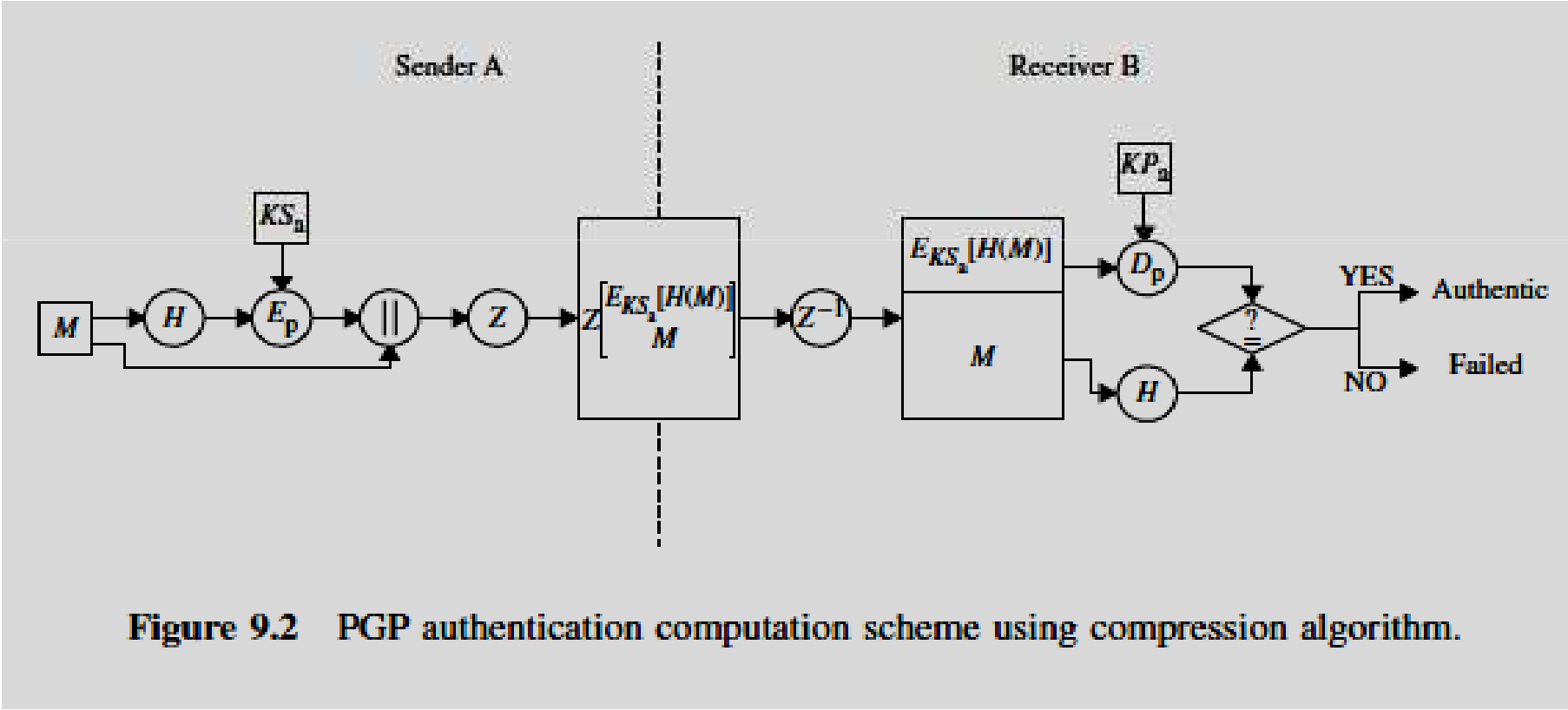


Figure 9.2 PGP authentication computation scheme using compression algorithm.

PGP

- **Authentication via Digital Signature**
  - The combination of SHA-1 and RSA provides an effective digital signature scheme
  - As an alternative, signatures can be generated using DSS/SHA-1
  - The National Institute of Standards and Technology (NIST) has published Digital Signature Standard (DSS)
  - The DSS uses an algorithm that is designed to provide only the digital signature function
  - Although DSS is a public-key technique, it cannot be used for encryption or key exchange

PGP

- **Compression**
  - *Z compression*
  - *Z*−1 decompression
  - Saving space
    - e-mail transmission
    - file storage
  - Compression will add difficulty- different trade-offs in running speed versus compression ratio produce different compressed forms

PGP

- **Compression**
  - Compresses the message after applying the signature but before encryption
    - Encryption
      - Message encryption is applied after compression to strengthen cryptographic security
      - It makes the cryptanalysis more difficult
    - Signing
      - Signing an uncompressed original message
        - » Preferable
        - » Because the uncompressed message together with the signature are directly used for future verification
      - Signing compressed message
        - » one may consider two cases
          - Either to store a compressed message for later verification
          - To recompress the message when verification is required

PGP

- **Compression**
  - Compression package – “ZIP” is used by PGP
  - ZIP is functionally equivalent to PKZIP developed by PKWARE, Inc
  - ZIP is the most commonly used cross-platform compression technique
  - First proposed two main compression schemes
    - Abraham Lempel in 1977
    - Jakob Ziv in 1978
    - Text compression schemes
    - Lossless compression
    - Broadly used
    - Easy to implement and also fast
  -

PGP

- **Compression**
  - **Lempel–Ziv–Storer–Szymanski** (**LZSS**)
    - 1982
    - James Storer and Thomas Szymanski
    - Based on the work of Lempel and Ziv
    - *Sliding-window-based schemes*
  - *Variants*
    - Simple variable-length code (LZB)
    - Dynamic Huffman coding (LZH)
    - Shannon–Fano coding (ZIP 1.x)
    - All of them result in a certain degree of improvement over the basic scheme, especially when the data is rather random

PGP

- **Compression**
- Recently LZFG
  - Combines the idea behind LZ77 and LZ78
  - LZFG uses the standard sliding window
  - But stores the data in a modified tree data structure
  - Run faster
- Huffman compression
  - Statistical data compression technique
  - Reduces the average code length used to represent the symbols
  - Huffman encoding always generates optimal codes
- Shannon–Fano coding
  - Related to Huffman coding
  - This coding divides the set of symbols into two
  - The first subset is assigned a binary 0, the second a binary 1 and so on
  - Uses few more bits than Huffman
- Decompression
  - LZ77-simple and fast
    Whenever a (position, length) pair is encountered, one goes to that *position in that window and copies length* bytes to the output

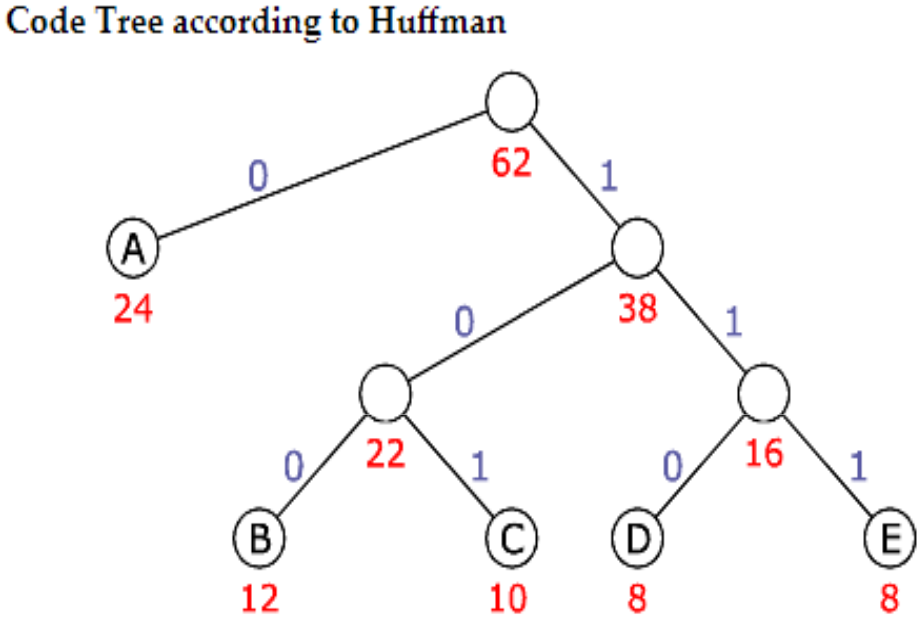
PGP

- **Huffman Code: Example**

Symbol	Frequency
A	24
B	12
C	10
D	8
E	8

----> total 186 bit

(with 3 bit per code word)



Symbol	Frequency	Code	Code Length	total Length
A	24	0	1	24
B	12	100	3	36
C	10	101	3	30
D	8	110	3	24
E	8	111	3	24
ges. 186 bit (3 bit code)				tot. 138 bit

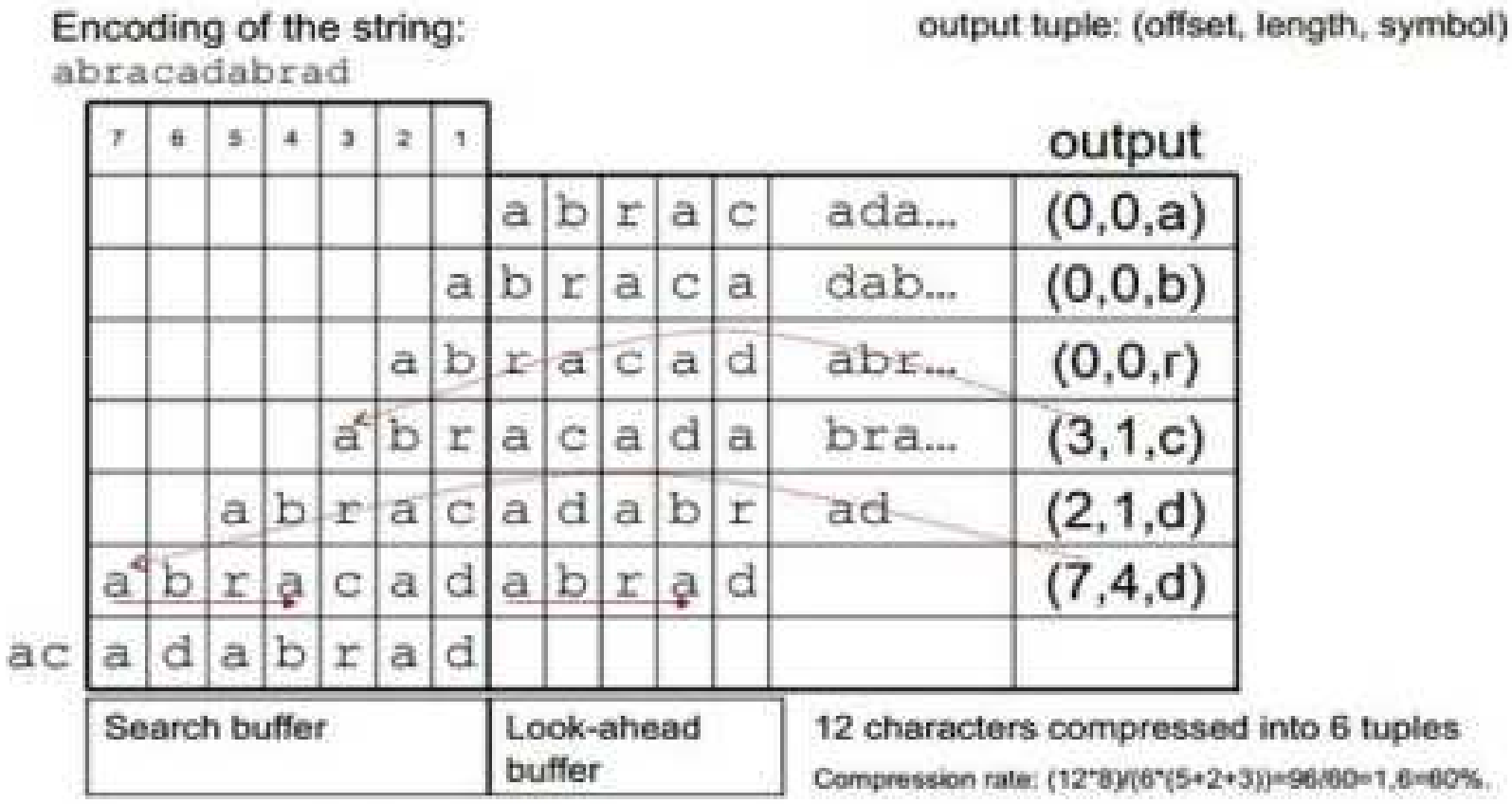
PGP

- Compression
  - Dictionary based
  - “ABABBABCABABBA”
  - output codes are:
    - 1 2 4 5 2 3 4 6 1

s	c	output	code	string
			1	A
			2	B
			3	C
A	B	1	4	AB
B	A	2	5	BA
A	B			
AB	B	4	6	ABB
B	A			
BA	B	5	7	BAB
B	C	2	8	BC
C	A	3	9	CA
A	B			
AB	A	4	10	ABA
A	B			
AB	B			
ABB	A	6	11	ABBA
A	EOF	1		

PGP

- Sliding window compression



PGP

- **Radix-64 Conversion**
  - 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 symmetric key)
  - Resulting block consists of a stream of arbitrary 8-bit octets
  - Encrypted or Compressed data will not be in a printable form
  - Radix 64 conversion convert data into printable form (ASCII)
  - Many mailing system supports only ASCII characters

PGP

- **Radix-64 Conversion**
  - Many electronic mail systems permit only ASCII text
  - PGP
    - Provides a service
    - Convert the raw 8-bit binary octets to a stream of printable 7-bit ASCII characters
    - Scheme called **radix-64 encoding or ASCII Armor**

PGP

- **Radix-64 Conversion**
  - Each group of three octets of binary data is **mapped** into four ASCII characters
  - Appends a **CRC** to detect transmission errors
  - This radix-64 conversion is a **wrapper** around the binary PGP messages
  - It is possible to use PGP to convert any arbitrary file to **ASCII Armor**
  - Used to **protect** the binary messages during transmission over non-binary channels, such as Internet e-mail

PGP

- **Radix-64 Conversion**
  - Table shows the mapping of 6-bit input values to characters
  - The character set consists
    - upper- and lower-case letters
    - digits 0–9,
    - characters '+' and *'/'.*
    - *The '=' character is used as the padding character*

Table 9.1 Radix-64 encoding

6-bit value	Character encoding	6-bit value	Character encoding	6-bit value	Character encoding	6-bit value	Character 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
10	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)	=

PGP

- **Radix-64 Conversion**

Table 9.1 Radix-64 encoding

6-bit value	Character encoding	6-bit value	Character encoding	6-bit value	Character encoding	6-bit value	Character 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
10	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)	=

PGP

ASCII	Hex	Symbol
0	0	NUL
1	1	SOH
2	2	STX
3	3	ETX
4	4	EOT
5	5	ENQ
6	6	ACK
7	7	BEL
8	8	BS
9	9	TAB
10	A	LF
11	B	VT
12	C	FF
13	D	CR
14	E	SO
15	F	SI

ASCII	Hex	Symbol
16	10	DLE
17	11	DC1
18	12	DC2
19	13	DC3
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RS
31	1F	US

ASCII	Hex	Symbol
32	20	(space)
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/

ASCII	Hex	Symbol
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

ASCII	Hex	Symbol
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O

ASCII	Hex	Symbol
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

ASCII	Hex	Symbol
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o

ASCII	Hex	Symbol
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	

PGP

- **Radix-64 Conversion**

Raw text	b2	63	29	
24-bit raw text:	10110010	01100011	00101001	
Arranging in blocks of 6 bits	101100	100110	001100	101001
Decimal values are	44,	38,	12,	41
Referring to Table				
Radix-64 encoding	s	m	M	p
ASCII format - hexadecimal	73	6d	4d	70
Binary	0111 0011	0110 1101	0100 1101	0111 0000

PGP

- **Radix-64 Conversion:** *ASCII Armor Format*
  - Specific headers are placed around the data
  - Helps to construct the data later
  - Headers informs the kind of data encoded in ASCII Armor
  - ASCII Armor:
    - Armor head line
    - Armor headers
    - Blank line
    - ASCII-Armored data
    - Armor checksum
    - Armor tail

PGP

- **Radix-64 Conversion:** *ASCII Armor Format*
- ASCII Armor:
  - Armor head line
  - Armor headers
  - Blank line
  - ASCII-Armored data
  - Armor checksum
  - Armor tail

Example of an ASCII Armored Message

```
-----BEGIN PGP MESSAGE-----
Version: OpenPrivacy 0.99

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END PGP MESSAGE-----
```

PGP

- **Radix-64 Conversion:** *ASCII Armor Format*
  - *An Armor head line:*
    - *Header line text Surrounded by* five dashes (‘-’, 0x2D) on either side of the header line text
    - The header line text is chosen based upon the type of data that is being encoded in Armor, and how it is being encoded
    - Header line texts include the following strings:
      - **BEGIN PGP MESSAGE** – used for signed, encrypted or compressed files
      - **BEGIN PGP PUBLIC KEY BLOCK** – used for armouring public keys
      - **BEGIN PGP PRIVATE KEY BLOCK** – used for armouring private keys
      - **BEGIN PGP MESSAGE, PART X/Y** – used for multipart messages, where the armour is divided amongst Y parts, and this is the Xth part out of Y.
      - **BEGIN PGP MESSAGE, PART X** – used for multipart messages, where this is the Xth part of an unspecified number of parts
      - **BEGIN PGP SIGNATURE** – used for detached

PGP

- **Radix-64 Conversion:** *ASCII Armor Format*
  - *Armor headers:*
    - Give some information about **how to decode** or use the message
    - Part of the armour, **not a part of the message**
    - Hence are **not protected** by any signatures applied to the message

    - Pairs of strings
    - Pair of a **(key, value)** - A colon (‘:’ 0x38) and a single space (0x20) separate the key and value

    - PGP should consider improperly formatted Armor headers to be corruptions of ASCII Armor
    - Unknown keys should be reported to the user, but PGP should continue to process the message.

PGP

- **Radix-64 Conversion:** *ASCII Armor Format*
  - *Armor headers:*
    - Armor header keys include:
      - ***Version:*** *This states the PGP version used to encode the message.*
      - ***Comment****: This is a user-defined comment.*
      - ***MessageID:*** *a 32-character string of printable characters*
      - ***Hash:*** *This is a comma-separated list of hash algorithms used in the message.*
      - ***Charset:*** *This is a description of the character set that the plaintext is in - UTF-8 by default*

PGP

- **Radix-64 Conversion:** *ASCII Armor Format*
  - *A blank line:*
    - *This indicates zero length or contains only white space.*
  - *ASCII-Armoured data:*
    - *An arbitrary file can be converted to ASCII-Armoured data*
  - *Armor checksum:*
    - *This is a 24-bit CRC*
    - *Converted to four characters of radix-64 encoding preceded by an equals sign (=)*
    - *CRC generated on the data before it is converted to radix-64*
    - *The checksum with its leading equals sign may appear on the first line after the base 64 encoded data.*

PGP

- **Radix-64 Conversion:** *ASCII Armor Format*
  - Armor Tail Line
    - Composed in the same manner as the Armor Header Line
    - Except the string "BEGIN" is replaced by the string "END“
    - Eg: **END PGP MESSAGE**

PGP

- **Radix-64 Conversion:**
- *Encoding Binary in Radix-64*

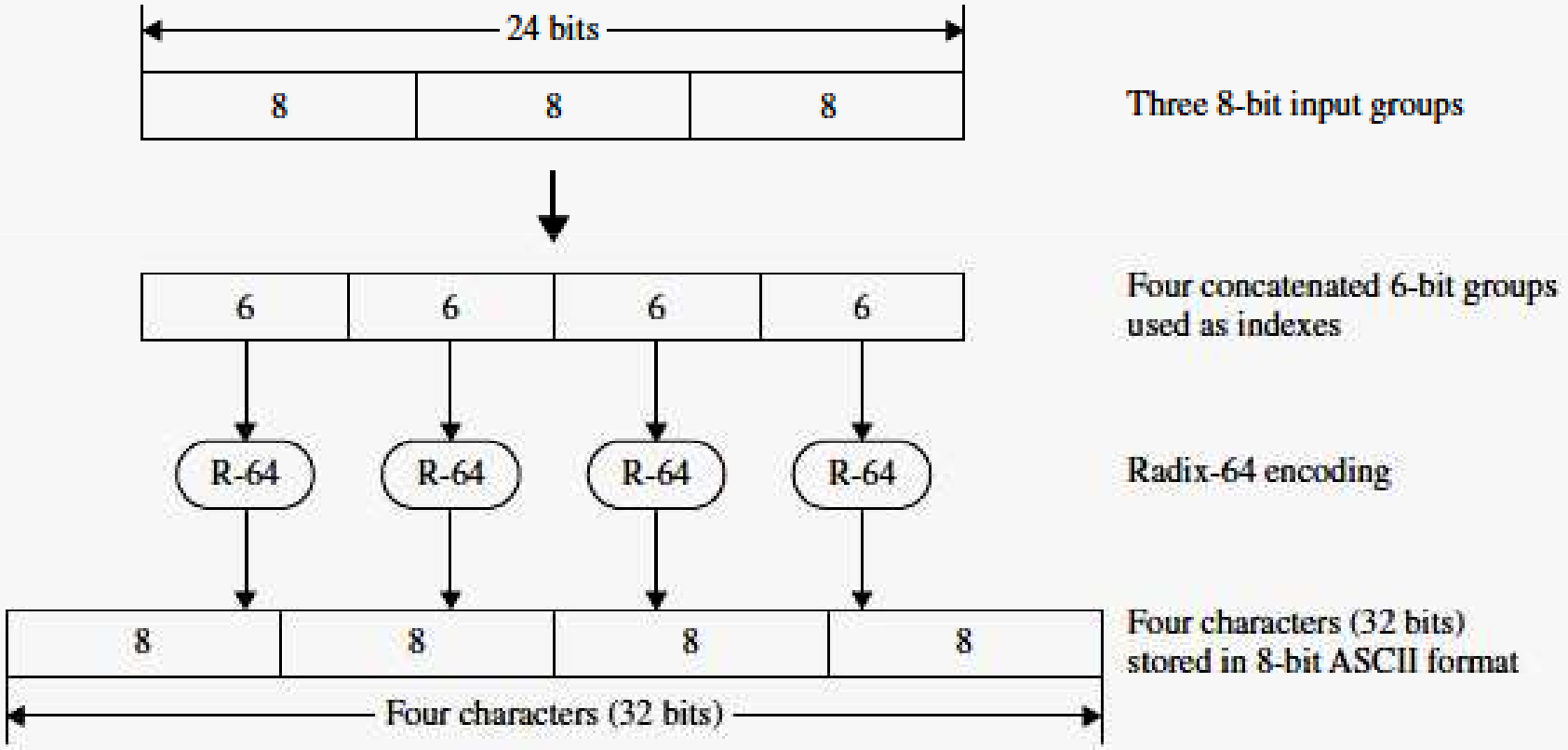


Figure 9.3 Radix-64 printable encoding of binary data.

PGP

- **Radix-64 Conversion:**
- *Encoding Binary in Radix-64*
  - The last data group has 24 bits (3 octets).
    - No special processing is needed.
  - The last data group has 16 bits (2 octets).
    - The first two 6-bit groups are processed as above. The third (incomplete) data group has two zero-value bits added to it, and is processed as above.
    - A pad character (=) is added to the output.
  - The last data group has 8 bits (1 octet).
    - The first 6-bit group is processed as above.
    - The second (incomplete) data group has four zero-value bits added to it, and is processed as above
    - Two pad characters (=) are added to the output

PGP

1. Input raw text: 0x 15 d0 2f 9e b7 4c

8-bit octets:	00010101 11010000 00101111 10011110 10110111 01001100
6-bit index:	000101 011101 000000 101111 100111 101011 011101 001100
Decimal:	5 29 0 47 39 43 29 12
Output character: (radix-64 encoding)	F d A v n r d M
ASCII format (0x):	46 64 41 76 6e 72 64 4d
Binary:	01000110 01100100 01000001 01110110 01101110 01110010 01100100 01001101

PGP

2. Input raw text: 0x 15 d0 2f 9e b7

8-bit octets:	00010101 11010000 00101111 10011110 10110111
6-bit index:	000101 011101 000000 101111 100111 101011 011100
	Pad with 00 (=)
Decimal:	5 29 0 47 39 43 28
Output character:	F d A v n r c =

3. Input raw text: 0x 15 d0 2f 9e

8-bit octets:	00010101 11010000 00101111 10011110
6-bit index:	000101 011101 000000 101111 100111 100000
	Pad with 0000 (==)
Decimal:	5 29 0 47 39 32
Output character:	F d A v n g = =

PGP

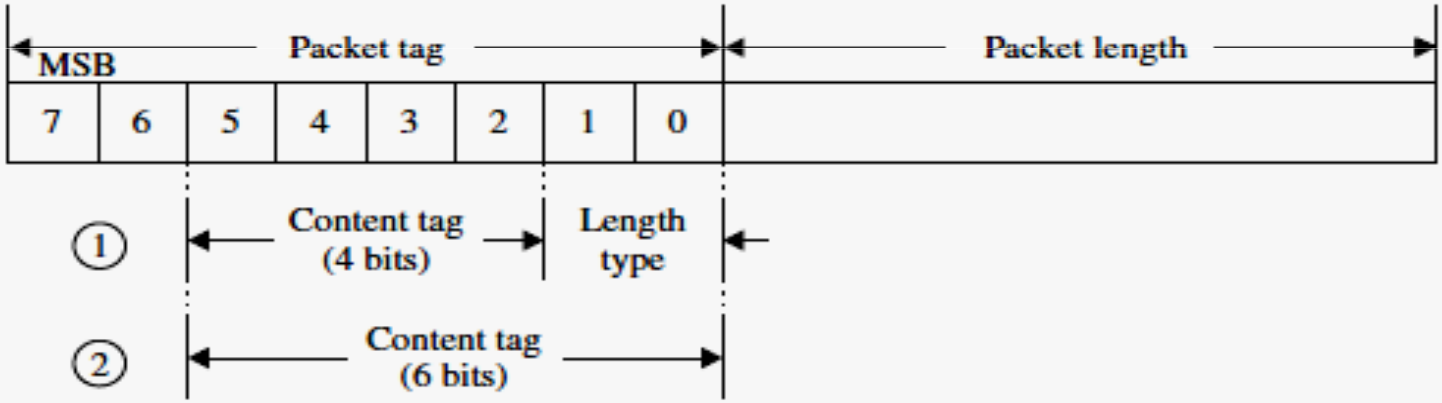
- **Packet**

Packet tag	Packet length	Packet body

Packet header

- **Packet Header**

PGP 2.6.x only uses old format packets



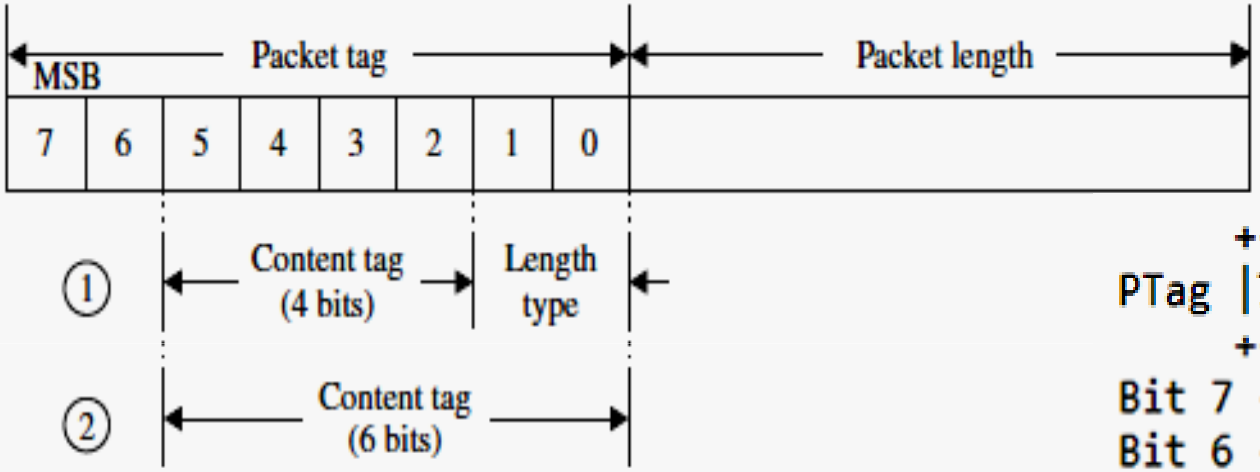
(1) Old format packets: content tag (bits 5, 4, 3, 2); length type (bits 1,0)

(2) New format packets: content tag (bits 5, 4, 3, 2, 1, 0)

Figure 9.4 Packet header.

PGP

- **Packet Header**



1. Old format packets: content tag (bits 5, 4, 3, 2); length type (bits 1,0)

2. New format packets: content tag (bits 5, 4, 3, 2, 1, 0)

Figure 9.4 Packet header.

```
      +---------------+
PTag |7 6 5 4 3 2 1 0|
      +---------------+
Bit 7 -- Always one
Bit 6 -- New packet format if set

Old format packets contain:

      Bits 5-2 -- packet tag
      Bits 1-0 -- length-type

New format packets contain:

      Bits 5-0 -- packet tag
```

PGP

- **Packet Header**
- *Packet Tags*
  - Denotes what type of packet the body holds
    - 0–Reserved
    - 1–Session key packet encrypted by public key *Tags*
    - 2–Signature packet
    - 3–Session key packet encrypted by symmetric key
    - 4–One-pass signature packet
    - 5–Secret-key packet
    - 6–Public-key packet
    - 7–Secret-subkey packet
    - 8–Compressed data packet
    - 9–Symmetrically encrypted data packet
    - 10–Marker packet
    - 11–Literal data packet
    - 12–Trust packet
    - 13–User ID packet
    - 14–Public subkey packet
    - 60 ~ 63–Private or experimental values

PGP

- **Packet Header**
- *Old-Format Packet Lengths*
  - *Bit 1 and 0*
  - Meaning of Length type
    - 0–The packet has a one-octet length. The header is two octets long.
    - 1–The packet has a two-octet length. The header is three octets long.
    - 2–The packet has a four-octet length. The header is five octets long.
    - 3–The packet is of indeterminate length.
      - The header is 1 octet long
      - The implementation must determine how long the packet is
      - If the packet is in a file, this means that the packet extends until the end of the file
      - In general, an implementation SHOULD NOT use indeterminate-length packets except where the end of the data will be clear from the context, and even then it is better to use a definite length, or a new format header.

PGP

- **Packet Header**
- *New-Format Packet Lengths*
  - Four possible ways of encoding length:

1. A one-octet Body Length header encodes packet lengths of up to 191 octets.
2. A two-octet Body Length header encodes packet lengths of 192 to 8383 octets.
3. A five-octet Body Length header encodes packet lengths of up to 4,294,967,295 (0xFFFFFFFF) octets in length.
4. Partial Body Length headers encode a packet of indeterminate length

   Encoded as one octet value that is greater than or equal to 224, and less than 255

PGP

- **Packet Header**
  - **Example**

1. Packet data length 100 *(decimal) = 01100100(binary) = 0x64 (hex)*
   - *Need* one octet
   - So packet header length is octet long
   - This header is followed by 100 octets of data
2. Length 1723
   - two octets: 0xc5, 0xfb
   - This header is followed by the 1723 octets of data
3. Length 100000
   - five octets: 0xff, 0x00, 0x01, 0x86, 0xa0.

PGP

- **PGP Packet Structure**
- A PGP file
  - Concatenation of one or more packets
  - Packets in a file may be subject to a transformation using
    - Encryption
    - Compression
    - Digital signature
    - Radix-64 conversion

PGP

- **PGP Packet Structure**
  - Message packet Signature packet Session key packet

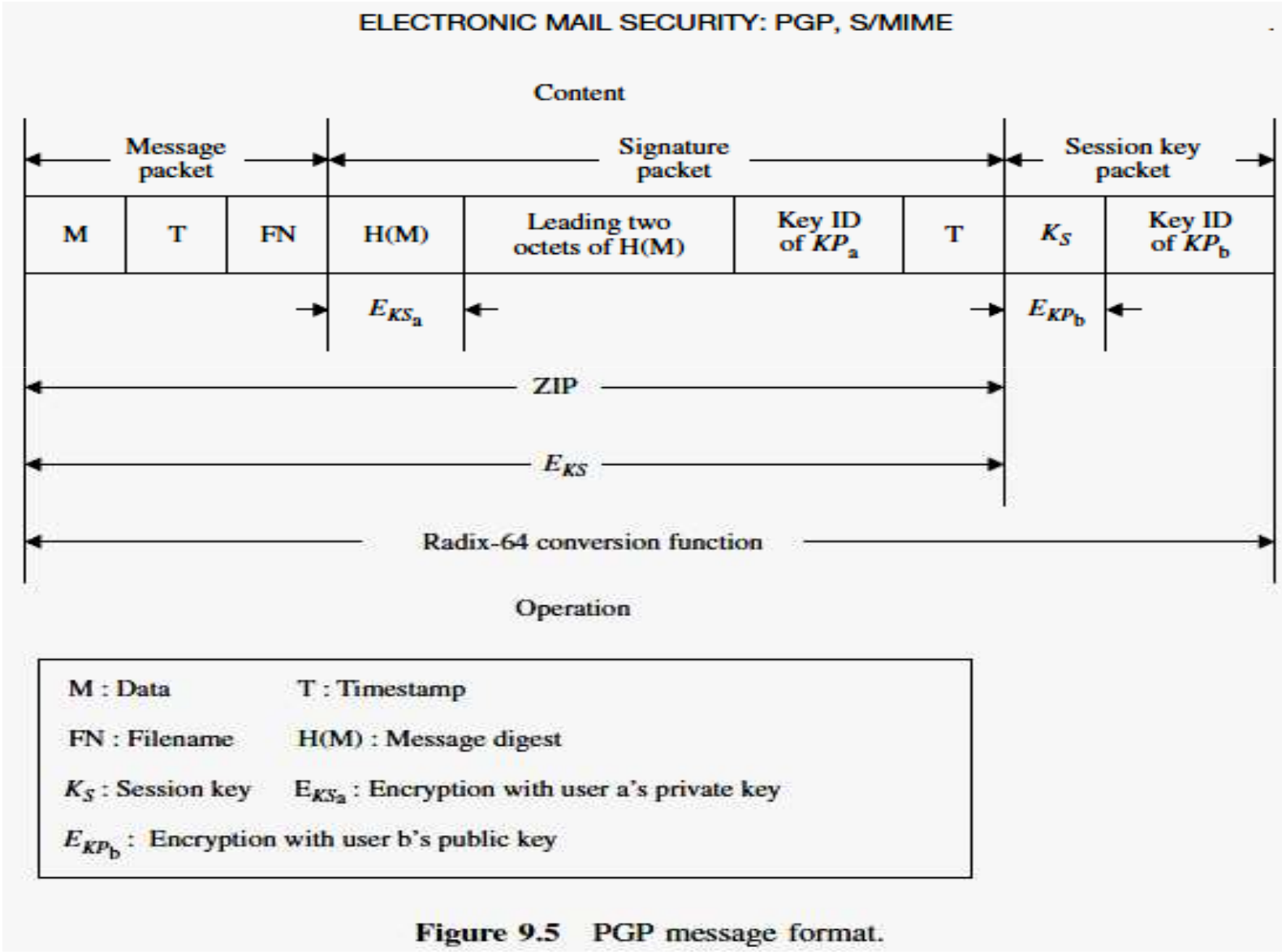


Figure 9.5 PGP message format.

PGP

- **PGP Packet Structure**
- Message packet
  - Contains
    - Actual data
    - Filename
    - Timestamp

PGP

- **PGP Packet Structure**
- *Signature Packet (Tag 2)*
  - Describes a binding between some public key and some data
  - Components
    - Timestamp
    - Message digest
    - Leading two octets of hash code
    - Key ID of sender's public key

PGP

- **PGP Packet Structure**
- *Signature Packet (Tag 2)*

  - **Timestamp:** Time at which the signature was created
  - **Message digest (or hash code):**
    - 160-bit SHA-1 digest
    - Encrypted with sender private key
    - **H(Signature timestamp + data portion of the message)**
    - It protects against replay attacks
  - **Leading two octets of hash code:**
    - These enable the recipient to determine if the correct public key was used to decrypt the hash code for authentication
    - By comparing the plaintext copy of the first two octets with the first two octets of the decrypted digest
    - Two octets also serve as a 16-bit frame-check sequence for the message
  - **Key ID of sender's public key:**
    - This identifies the public key and the private key that was used to encrypt the hash code

PGP

- **PGP Packet Structure**
- Signature Packet (Tag 2)
  - Compressed data packet
    - Z(Message packet ||signature packet)
  - The message component and signature component (optional)
    - May be compressed using ZIP
    - May be encrypted using a session key

PGP

- ***PGP Packet Structure***
- Session Key Packets (Tag 1)
  - (E_KPb (Ks) || ID of receiver's public key)
  - The body of this session key component consists of:
    - A one-octet version number which is 3
    - An eight-octet - key ID of the public key
    - A one-octet number giving the public key algorithm used
    - A string of octets that is the encrypted session key

```
New: Public-Key Encrypted Session Key Packet(tag 1)(524 bytes)
    New version(3)
    Key ID - 0x2478DF98CED355D3
    Pub alg - RSA Encrypt or Sign(pub 1)
    RSA m^e mod n(4096 bits) - ...
        -> m = sym alg(1 byte) + checksum(2 bytes) + PKCS-1 block type 02
```

PGP

- **Key Material Packet**
  - Contains all the information about a public or private key
  - 4 variants
  - 2 versions : version 3 and version 4
  - *Key Packet Variants*
    - **Public-Key Packet (Tag 6)**
    - **Public-Subkey Packet (Tag 14)**
    - **Secret-Key Packet (Tag 5)**
    - **Secret-Subkey Packet (Tag 7)**

PGP

- **Key Material Packet**
  - **Public-key packet (tag 6):**
    - This packet starts a series of packets that forms a PGP 5.x key
  - **Public subkey packet (tag 14):**
    - This packet has exactly the same format as a publickey packet
    - Denotes a subkey
    - One or more subkeys may be associated with atop-level key
    - The top-level key provides signature services
    - The subkeys provide encryption services
  - **Secret-key packet (tag 5):**
    - This packet contains all the information that is found in a public-key packet
    - Also includes the secret-key material after all the public-key fields
  - **Secret-subkey packet (tag 7):**
    - It is the subkey analogous to the secret-key packet

PGP

- **Key Material Packet**
- *Public-key Packet Formats*
  - Version 2,3,4
  - A v3 key packet contains:
    - A one-octet version number (3)
    - A four-octet number denoting the time that the key was created
    - A two-octet number denoting the time in days that this key is valid
    - A one-octet number denoting the public-key algorithm of this key
    - A series of multiprecision integers (MPIs) comprising the key material: an MPI of RSA public module *n; an MPI of RSA public encryption exponent e*

PGP

- **Key Material Packet**
- *Public-key Packet Formats*
  - A v4 key packet contains:
    - A one-octet version number (4)
    - A four-octet number denoting the time that the key was created
    - A one-octet number denoting the public-key algorithm of this key
    - A series of MPIs comprising the key material

```
New: Public Key Packet(tag 6)(269 bytes)
    Ver 4 - new
    Public key creation time - Mon May 25 15:12:34 PDT 2015
    Pub alg - RSA Encrypt or Sign(pub 1)
    RSA n(2048 bits) - ...
    RSA e(17 bits) - ...
```

- **Key Material Packet**
- *Secret-key Packet format*
  - The secret-key and secret-subkey packets contain all the data of public-key and public subkey packets in encrypted form
  - And additional algorithm-specific key

PGP

- **Key Material Packet**
- *Secret-key Packet format*
- The secret-key packet contains:
  - A public-key or public-subkey packet
  - One octet indicating string-to-key (S2K) usage conventions
    - 0 indicates that the secretkey data is not encrypted
    - 255 indicates that an S2K specifier is being given

3.6.1 String-to-key (S2k) specifier types

There are three types of S2K specifiers currently supported, as follows:

3.6.1.1 Simple S2K

This directly hashes the string to produce the key data. See below for how this hashing is done.

```
    Octet 0:        0x00
    Octet 1:        hash algorithm
```

3.6.1.2 Salted S2K

This includes a "salt" value in the S2K specifier -- some arbitrary data -- that gets hashed along with the passphrase string, to help prevent dictionary attacks.

```
    Octet 0:        0x01
    Octet 1:        hash algorithm
    Octets 2-9:     8-octet salt value
```

3.6.1.3 Iterated and Salted S2K

This includes both a salt and an octet count. The salt is combined with the passphrase and the resulting value is hashed repeatedly. This further increases the amount of work an attacker must do to try dictionary attacks.

Octet 0:	0x04
Octet 1:	hash algorithm
Octets 2-9:	8-octet salt value
Octets 10-13:	count, a four-octet, unsigned value

PGP

- **Algorithms for PGP 5.x**

Public-Key Algorithms

ID	Algorithm
1	RSA (encrypt or sign)
2	RSA encryption only
3	RSA sign only
16	ElGamal (encrypt only)
17	DSA (DSS)
18	Reserved for elliptic curve
19	Reserved for ECDSA
20	ElGamal (encrypt or sign)
21	Reserved for Diffie–Hellman
100–110	Private/experimental algorithm

PGP

- **Algorithms for PGP 5.x**

Symmetric-Key Algorithms

ID	Algorithm
0	Plaintext or unencrypted data
1	IDEA
2	Triple DES (DES–EDE)
3	CAST 5 (128-bit key)
4	Blowfish (128-bit key, 16 rounds)
5	SAFER-SK128 (13 rounds)
6	Reserved for DES/SK
7	Reserved for AES (128-bit key)
8	Reserved for AES (192-bit key)
9	Reserved for ASE (256-bit key)
100–110	Private/experimental algorithm

PGP

- **Algorithms for PGP 5.x**

Compression Algorithm

ID	Algorithm
0	Uncompressed
1	ZIP (RFC 1951)
2	ZLIB (RFC 1950)
100–110	Private/experimental algorithm

PGP

- **Algorithms for PGP 5.x**

Hash Algorithms

ID	Algorithm
1	MD5
2	SHA-1
3	RIPE-MD/160
4	Reserved for double-width SAH (experimental)
5	MD2
6	Reserved for TIGER/192
7	Reserved for HAVAL (5 pass, 160-bit)
100–110	Private/experimental algorithm