Network Working Group
Request for Comments: 4648
Obsoletes: 3548
Category: Standards Track

S. Josefsson
SJD

October 2006

The Base16, Base32, and Base64 Data Encodings

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the “Internet Official Protocol Standards” (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This document describes the commonly used base 64, base 32, and base 16 encoding schemes. It also discusses the use of line-feeds in encoded data, use of padding in encoded data, use of non-alphabet characters in encoded data, use of different encoding alphabets, and canonical encodings.

Table of Contents

1. Introduction

Base encoding of data is used in many situations to store or transfer data in environments that, perhaps for legacy reasons, are restricted to US-ASCII [1] data. Base encoding can also be used in new applications that do not have legacy restrictions, simply because it makes it possible to manipulate objects with text editors.

In the past, different applications have had different requirements and thus sometimes implemented base encodings in slightly different ways. Today, protocol specifications sometimes use base encodings in general, and "base64" in particular, without a precise description or reference. Multipurpose Internet Mail Extensions (MIME) [4] is often used as a reference for base64 without considering the consequences for line-wrapping or non-alphabet characters. The purpose of this specification is to establish common alphabet and encoding considerations. This will hopefully reduce ambiguity in other documents, leading to better interoperability.

2. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [2].

3. Implementation Discrepancies

Here we discuss the discrepancies between base encoding implementations in the past and, where appropriate, mandate a specific recommended behavior for the future.

3.1. Line Feeds in Encoded Data

MIME [4] is often used as a reference for base 64 encoding. However, MIME does not define "base 64" per se, but rather a "base 64 Content- Transfer-Encoding" for use within MIME. As such, MIME enforces a limit on line length of base 64-encoded data to 76 characters. MIME inherits the encoding from Privacy Enhanced Mail (PEM) [3], stating that it is "virtually identical"; however, PEM uses a line length of 64 characters. The MIME and PEM limits are both due to limits within SMTP.

Implementations MUST NOT add line feeds to base-encoded data unless the specification referring to this document explicitly directs base encoders to add line feeds after a specific number of characters.

3.2. Padding of Encoded Data

In some circumstances, the use of padding ("=") in base-encoded data is not required or used. In the general case, when assumptions about the size of transported data cannot be made, padding is required to yield correct decoded data.

Implementations MUST include appropriate pad characters at the end of encoded data unless the specification referring to this document explicitly states otherwise.

The base64 and base32 alphabets use padding, as described below in sections 4 and 6, but the base16 alphabet does not need it; see section 8.

3.3. Interpretation of Non-Alphabet Characters in Encoded Data

Base encodings use a specific, reduced alphabet to encode binary data. Non-alphabet characters could exist within base-encoded data, caused by data corruption or by design. Non-alphabet characters may be exploited as a "covert channel", where non-protocol data can be sent for nefarious purposes. Non-alphabet characters might also be sent in order to exploit implementation errors leading to, e.g., buffer overflow attacks.

Implementations MUST reject the encoded data if it contains characters outside the base alphabet when interpreting base-encoded data, unless the specification referring to this document explicitly states otherwise. Such specifications may instead state, as MIME does, that characters outside the base encoding alphabet should simply be ignored when interpreting data ("be liberal in what you accept"). Note that this means that any adjacent carriage return/ line feed (CRLF) characters constitute "non-alphabet characters" and are ignored. Furthermore, such specifications MAY ignore the pad character, "=", treating it as non-alphabet data, if it is present before the end of the encoded data. If more than the allowed number of pad characters is found at the end of the string (e.g., a base 64 string terminated with "==="), the excess pad characters MAY also be ignored.

3.4. Choosing the Alphabet

Different applications have different requirements on the characters in the alphabet. Here are a few requirements that determine which alphabet should be used:

- Handled by humans. The characters "0" and "O" are easily confused, as are "1", "l", and "I". In the base32 alphabet below, where 0 (zero) and 1 (one) are not present, a decoder may interpret 0 as O, and 1 as I or L depending on case. (However, by default it should not; see previous section.)
- Encoded into structures that mandate other requirements. For base 16 and base 32, this determines the use of upper- or lowercase alphabets. For base 64, the non-alphanumeric characters (in particular, "/") may be problematic in file names and URLs.
- Used as identifiers. Certain characters, notably "+" and "/" in the base 64 alphabet, are treated as word-breaks by legacy text search/index tools.

There is no universally accepted alphabet that fulfills all the requirements. For an example of a highly specialized variant, see IMAP [8]. In this document, we document and name some currently used alphabets.

3.5. Canonical Encoding

The padding step in base 64 and base 32 encoding can, if improperly implemented, lead to non-significant alterations of the encoded data. For example, if the input is only one octet for a base 64 encoding, then all six bits of the first symbol are used, but only the first two bits of the next symbol are used. These pad bits MUST be set to zero by conforming encoders, which is described in the descriptions on padding below. If this property do not hold, there is no canonical representation of base-encoded data, and multiple base- encoded strings can be decoded to the same binary data. If this property (and others discussed in this document) holds, a canonical encoding is guaranteed.

In some environments, the alteration is critical and therefore decoders MAY chose to reject an encoding if the pad bits have not been set to zero. The specification referring to this may mandate a specific behaviour.

4. Base 64 Encoding

The following description of base 64 is derived from [3], [4], [5], and [6]. This encoding may be referred to as "base64".

The Base 64 encoding is designed to represent arbitrary sequences of octets in a form that allows the use of both upper- and lowercase letters but that need not be human readable.

A 65-character subset of US-ASCII is used, enabling 6 bits to be represented per printable character. (The extra 65th character, "=", is used to signify a special processing function.)

The encoding process represents 24-bit groups of input bits as output strings of 4 encoded characters. Proceeding from left to right, a 24-bit input group is formed by concatenating 3 8-bit input groups. These 24 bits are then treated as 4 concatenated 6-bit groups, each of which is translated into a single character in the base 64 alphabet.

Each 6-bit group is used as an index into an array of 64 printable characters. The character referenced by the index is placed in the output string.

Table 1: The Base 64 Alphabet

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

Special processing is performed if fewer than 24 bits are available at the end of the data being encoded. A full encoding quantum is always completed at the end of a quantity. When fewer than 24 input bits are available in an input group, bits with value zero are added (on the right) to form an integral number of 6-bit groups. Padding at the end of the data is performed using the '=' character. Since all base 64 input is an integral number of octets, only the following cases can arise:

(1) The final quantum of encoding input is an integral multiple of 24 bits; here, the final unit of encoded output will be an integral multiple of 4 characters with no "=" padding.

(2) The final quantum of encoding input is exactly 8 bits; here, the final unit of encoded output will be two characters followed by two "=" padding characters.

(3) The final quantum of encoding input is exactly 16 bits; here, the final unit of encoded output will be three characters followed by one "=" padding character.

5. Base 64 Encoding with URL and Filename Safe Alphabet

The Base 64 encoding with an URL and filename safe alphabet has been used in [12].

An alternative alphabet has been suggested that would use "~" as the 63rd character. Since the "~" character has special meaning in some file system environments, the encoding described in this section is recommended instead. The remaining unreserved URI character is ".", but some file system environments do not permit multiple "." in a filename, thus making the "." character unattractive as well.

The pad character "=" is typically percent-encoded when used in an URI [9], but if the data length is known implicitly, this can be avoided by skipping the padding; see section 3.2.

This encoding may be referred to as "base64url". This encoding should not be regarded as the same as the "base64" encoding and should not be referred to as only "base64". Unless clarified otherwise, "base64" refers to the base 64 in the previous section.

This encoding is technically identical to the previous one, except for the 62:nd and 63:rd alphabet character, as indicated in Table 2.

Table 2: The "URL and Filename safe" Base 64 Alphabet

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

6. Base 32 Encoding

The following description of base 32 is derived from [11] (with corrections). This encoding may be referred to as "base32".

The Base 32 encoding is designed to represent arbitrary sequences of octets in a form that needs to be case insensitive but that need not be human readable.

A 33-character subset of US-ASCII is used, enabling 5 bits to be represented per printable character. (The extra 33rd character, "=", is used to signify a special processing function.)

The encoding process represents 40-bit groups of input bits as output strings of 8 encoded characters. Proceeding from left to right, a 40-bit input group is formed by concatenating 5 8bit input groups. These 40 bits are then treated as 8 concatenated 5-bit groups, each of which is translated into a single character in the base 32 alphabet. When a bit stream is encoded via the base 32 encoding, the bit stream must be presumed to be ordered with the most-significant- bit first. That is, the first bit in the stream will be the high- order bit in the first 8bit byte, the eighth bit will be the low- order bit in the first 8bit byte, and so on.

Each 5-bit group is used as an index into an array of 32 printable characters. The character referenced by the index is placed in the output string. These characters, identified in Table 3, below, are selected from US-ASCII digits and uppercase letters.

Table 3: The Base 32 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	9	J	18	S	27	3
1	B	10	K	19	T	28	4
2	C	11	L	20	U	29	5
3	D	12	M	21	V	30	6
4	E	13	N	22	W	31	7
5	F	14	O	23	X		
6	G	15	P	24	Y	(pad)	=
7	H	16	Q	25	Z		
8	I	17	R	26	2		

Special processing is performed if fewer than 40 bits are available at the end of the data being encoded. A full encoding quantum is always completed at the end of a body. When fewer than 40 input bits are available in an input group, bits with value zero are added (on the right) to form an integral number of 5-bit groups. Padding at the end of the data is performed using the "=" character. Since all base 32 input is an integral number of octets, only the following cases can arise:

(1) The final quantum of encoding input is an integral multiple of 40 bits; here, the final unit of encoded output will be an integral multiple of 8 characters with no "=" padding.

(2) The final quantum of encoding input is exactly 8 bits; here, the final unit of encoded output will be two characters followed by six "=" padding characters.

(3) The final quantum of encoding input is exactly 16 bits; here, the final unit of encoded output will be four characters followed by four "=" padding characters.

(4) The final quantum of encoding input is exactly 24 bits; here, the final unit of encoded output will be five characters followed by three "=" padding characters.

(5) The final quantum of encoding input is exactly 32 bits; here, the final unit of encoded output will be seven characters followed by one "=" padding character.

7. Base 32 Encoding with Extended Hex Alphabet

The following description of base 32 is derived from [7]. This encoding may be referred to as "base32hex". This encoding should not be regarded as the same as the "base32" encoding and should not be referred to as only "base32". This encoding is used by, e.g., NextSECure3 (NSEC3) [10].

One property with this alphabet, which the base64 and base32 alphabets lack, is that encoded data maintains its sort order when the encoded data is compared bit-wise.

This encoding is identical to the previous one, except for the alphabet. The new alphabet is found in Table 4.

Table 4: The "Extended Hex" Base 32 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	0	9	9	18	I	27	R
1	1	10	A	19	J	28	S
2	2	11	B	20	K	29	T
3	3	12	C	21	L	30	U
4	4	13	D	22	M	31	V
5	5	14	E	23	N		
6	6	15	F	24	O	(pad)	=
7	7	16	G	25	P		
8	8	17	H	26	Q		

8. Base 16 Encoding

The following description is original but analogous to previous descriptions. Essentially, Base 16 encoding is the standard case- insensitive hex encoding and may be referred to as "base16" or "hex".

A 16-character subset of US-ASCII is used, enabling 4 bits to be represented per printable character.

The encoding process represents 8-bit groups (octets) of input bits as output strings of 2 encoded characters. Proceeding from left to right, an 8-bit input is taken from the input data. These 8 bits are then treated as 2 concatenated 4-bit groups, each of which is translated into a single character in the base 16 alphabet.

Each 4-bit group is used as an index into an array of 16 printable characters. The character referenced by the index is placed in the output string.

Table 5: The Base 16 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	0	4	4	8	8	12	C
1	1	5	5	9	9	13	D
2	2	6	6	10	A	14	E
3	3	7	7	11	B	15	F

Unlike base 32 and base 64, no special padding is necessary since a full code word is always available.

9. Illustrations and Examples

To translate between binary and a base encoding, the input is stored in a structure, and the output is extracted. The case for base 64 is displayed in the following figure, borrowed from [5].

```
+--first octet--+-second octet--+--third octet--+
|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
+-----------+---+-------+-------+---+-----------+
|5 4 3 2 1 0|5 4 3 2 1 0|5 4 3 2 1 0|5 4 3 2 1 0|
+--1.index--+--2.index--+--3.index--+--4.index--+
```

The case for base 32 is shown in the following figure, borrowed from [7]. Each successive character in a base-32 value represents 5 successive bits of the underlying octet sequence. Thus, each group of 8 characters represents a sequence of 5 octets (40 bits).

```
            1          2          3
 01234567 89012345 67890123 45678901 23456789
+--------+--------+--------+--------+--------+
|< 1 >< 2| >< 3 ><|.4 >< 5.|>< 6 ><.|7 >< 8 >|
+--------+--------+--------+--------+--------+
                                       <===> 8th character
                                  <====> 7th character
                             <===> 6th character
                        <====> 5th character
                   <====> 4th character
              <===> 3rd character
        <====> 2nd character
 <===> 1st character
```

The following example of Base64 data is from [5], with corrections.

```
Input data:  0x14fb9c03d97e
Hex:     1   4    f   b    9   c     | 0   3    d   9    7   e
8-bit:   00010100 11111011 10011100  | 00000011 11011001 01111110
6-bit:   000101 001111 101110 011100 | 000000 111101 100101 111110
Decimal: 5      15     46     28       0      61     37     62
Output:  F      P      u      c        A      9      l      +
```

```
Input data:  0x14fb9c03d9
Hex:     1   4    f   b    9   c     | 0   3    d   9
8-bit:   00010100 11111011 10011100  | 00000011 11011001
                                                 pad with 00
6-bit:   000101 001111 101110 011100 | 000000 111101 100100
Decimal: 5      15     46     28       0      61     36
                                                     pad with =
Output:  F      P      u      c        A      9      k      =
```

```
Input data:  0x14fb9c03
Hex:     1   4    f   b    9   c     | 0   3
8-bit:   00010100 11111011 10011100  | 00000011
                                       pad with 0000
6-bit:   000101 001111 101110 011100 | 000000 110000
Decimal: 5      15     46     28       0      48
                                            pad with =      =
Output:  F      P      u      c        A      w      =      =
```

10. Test Vectors

```
BASE64("") = ""

BASE64("f") = "Zg=="

BASE64("fo") = "Zm8="

BASE64("foo") = "Zm9v"

BASE64("foob") = "Zm9vYg=="

BASE64("fooba") = "Zm9vYmE="

BASE64("foobar") = "Zm9vYmFy"

BASE32("") = ""

BASE32("f") = "MY======"

BASE32("fo") = "MZXQ===="

BASE32("foo") = "MZXW6==="

BASE32("foob") = "MZXW6YQ="

BASE32("fooba") = "MZXW6YTB"

BASE32("foobar") = "MZXW6YTBOI======"

BASE32-HEX("") = ""

BASE32-HEX("f") = "CO======"

BASE32-HEX("fo") = "CPNG===="

BASE32-HEX("foo") = "CPNMU==="

BASE32-HEX("foob") = "CPNMUOG="

BASE32-HEX("fooba") = "CPNMUOJ1"

BASE32-HEX("foobar") = "CPNMUOJ1E8======"

BASE16("") = ""

BASE16("f") = "66"

BASE16("fo") = "666F"

BASE16("foo") = "666F6F"

BASE16("foob") = "666F6F62"

BASE16("fooba") = "666F6F6261"

BASE16("foobar") = "666F6F626172"
```

11. ISO C99 Implementation of Base64

An ISO C99 implementation of Base64 encoding and decoding that is believed to follow all recommendations in this RFC is available from:

http://josefsson.org/base-encoding/[1]

This code is not normative.

The code could not be included in this RFC for procedural reasons (RFC 3978 section 5.4[2]).

[1] http://josefsson.org/base-encoding/
[2] https://tools.ietf.org/html/rfc3978#section-5.4

12. Security Considerations

When base encoding and decoding is implemented, care should be taken not to introduce vulnerabilities to buffer overflow attacks, or other attacks on the implementation. A decoder should not break on invalid input including, e.g., embedded NUL characters (ASCII 0).

If non-alphabet characters are ignored, instead of causing rejection of the entire encoding (as recommended), a covert channel that can be used to "leak" information is made possible. The ignored characters could also be used for other nefarious purposes, such as to avoid a string equality comparison or to trigger implementation bugs. The implications of ignoring non-alphabet characters should be understood in applications that do not follow the recommended practice. Similarly, when the base 16 and base 32 alphabets are handled case insensitively, alteration of case can be used to leak information or make string equality comparisons fail.

When padding is used, there are some non-significant bits that warrant security concerns, as they may be abused to leak information or used to bypass string equality comparisons or to trigger implementation problems.

Base encoding visually hides otherwise easily recognized information, such as passwords, but does not provide any computational confidentiality. This has been known to cause security incidents when, e.g., a user reports details of a network protocol exchange (perhaps to illustrate some other problem) and accidentally reveals the password because she is unaware that the base encoding does not protect the password.

Base encoding adds no entropy to the plaintext, but it does increase the amount of plaintext available and provide a signature for cryptanalysis in the form of a characteristic probability distribution.

13. Changes Since RFC 3548

Added the "base32 extended hex alphabet", needed to preserve sort order of encoded data.

Referenced IMAP for the special Base64 encoding used there.

Fixed the example copied from RFC 2440.

Added security consideration about providing a signature for cryptoanalysis.

Added test vectors.

Fixed typos.

14. Acknowledgements

Several people offered comments and/or suggestions, including John E. Hadstate, Tony Hansen, Gordon Mohr, John Myers, Chris Newman, and Andrew Sieber. Text used in this document are based on earlier RFCs describing specific uses of various base encodings. The author acknowledges the RSA Laboratories for supporting the work that led to this document.

This revised version is based in parts on comments and/or suggestions made by Roy Arends, Eric Blake, Brian E Carpenter, Elwyn Davies, Bill Fenner, Sam Hartman, Ted Hardie, Per Hygum, Jelte Jansen, Clement Kent, Tero Kivinen, Paul Kwiatkowski, and Ben Laurie.

15. Copying Conditions

16. References

16.1. Normative References

[1] Cerf, V., "ASCII format for network interchange", RFC 20, October 1969.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

16.2. Informative References

[3] Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures", RFC 1421, February 1993.

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

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

[6] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[7] Klyne, G. and L. Masinter, "Identifying Composite Media Features", RFC 2938, September 2000.

[8] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.

[9] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[10] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNSSEC Hash Authenticated Denial of Existence", Work in Progress, June 2006.

[11] Myers, J., "SASL GSSAPI mechanisms", Work in Progress, May 2000.

[12] Wilcox-O'Hearn, B., "Post to P2P-hackers mailing list", September 2001, <http://zgp.org/pipermail/p2p-hackers/2001-September/000315.html>.

Author's Address

Simon Josefsson
SJD
EMail: simon@josefsson.org

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).

[1] http://www.ietf.org/ipr
[2] mailto:ietf-ipr@ietf.org