Binary-to-text encoding

By Contributors to Wikimedia projects

Published: 2005-11-30 · Archived: 2026-04-05 13:00:06 UTC

From Wikipedia, the free encyclopedia

A **binary-to-text encoding** is a data encoding scheme that represents binary data as plain text. Generally, the binary data consists of a sequence of arbitrary 8-bit byte (a.k.a. octet) values and the text is restricted to the printable character codes of commonly-used character encodings such as ASCII. In general, arbitrary binary data contains values that are not printable character codes, so software designed to only handle text fails to process such data. Encoding binary data as text allows information that is not inherently stored as text to be processed by software that otherwise cannot process arbitrary binary data. The software cannot interpret the information, but it can perform useful operations on the data such as transmit and store.

PGP documentation (RFC 9580) uses the term "ASCII armor" for binary-to-text encoding when referring to Base64.

Conceptually, binary-to-text encoding differs from numeric representation for a numeric base (radix). For example, decimal is a scheme for representing a value as base-10, but it is not a binary-to-text encoding. A binary-to-text encoding could be devised that uses decimal representation for encoded data, but such a system would use only 10 values of a 4-bit encoded sequence, leaving 6 values unused. A more efficient encoding would use all 16 values. This is Base16 which uses hexadecimal for encoding each 4-bit sequence. Notably, because 16 is a power of two, Base16 and hexadecimal are indistinguishable in practice even though they differ conceptually.

Escape encodings such as percent-encoding and quoted-printable also allow for representing arbitrary binary data as text, but in a significantly different way. A binary-to-text encoding involves encoding an entire input sequence whereas an escape encoding allows for embedding binary data in data that is already and inherently text.

Transmitting binary data as text

[edit]

A binary-to-text encoding enables transmitting data on a communication channel that does not allow arbitrary binary data (such as email or NNTP) or is not 8-bit clean. The encoding enables transmitting binary data over a communications protocol that is designed to carry human-readable (i.e. English language) text. Often such a protocol only supports 7-bit character values (and within that avoids certain control codes), and may require line breaks at certain maximum intervals, and may not maintain whitespace. Thus, only the 94 printable ASCII characters are safe to use to convey data.

The ASCII text-encoding standard uses 7 bits to encode characters. With this it is possible to encode 128 (i.e. 2^7) unique values (0–127) to represent the alphabetic, numeric, and punctuation characters commonly used in English, plus a selection of non-printable control characters. For example, the capital letter **A** is represented as 65 (41_{16},

100 0001_2), the numeral **2** is 50 (32_{16}, 011 0010_2), the right curly brace **}** is 125 ($7D_{16}$, 111 1101_2), and the carriage return control character **CR** is 13 ($0D_{16}$, 000 1101_2).

In contrast, most computers store data in memory organized in eight-bit bytes (a.k.a. octets). Files that contain machine-executable code and non-textual data typically contain all 256 possible eight-bit byte values. Many computer programs came to rely on this distinction between seven-bit *text* and eight-bit *binary* data, and would not function properly if non-ASCII characters appeared in data that was expected to include only ASCII text. For example, if the value of the eighth bit is not preserved, the program might interpret a byte value above 127 as a flag telling it to perform some function.

It is often desired to send non-textual data through a text-based system, such as attaching an image to an e-mail message. To accomplish this, the data is encoded in some way, such that 8-bit data is encoded as 7-bit ASCII characters (generally using only alphanumeric and punctuation characters—the ASCII printable characters). Upon arrival at its destination, it is then decoded back to its 8-bit form. This process is referred to as binary to text encoding. Many programs perform this conversion to allow for data-transport, such as PGP and GNU Privacy Guard.

Encoding plain text

[edit]

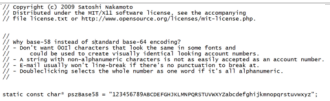
Binary-to-text encoding methods are also used as a mechanism for encoding plain text. Some systems have a more limited character set they can handle; not only are they not 8-bit clean, some cannot even handle every printable ASCII character. Other systems have limits on the number of characters that may appear between line breaks, such as the "1000 characters per line" limit of some Simple Mail Transfer Protocol software, as allowed by RFC 2821. Still others add headers or trailers to the text. A few poorly-regarded but still-used protocols use in-band signaling, causing confusion if specific patterns appear in the message. The best-known is the string "From " (including trailing space) at the beginning of a line, used to separate mail messages in the mbox file format.

By using a binary-to-text encoding on messages that are already plain text, then decoding on the other end, one can make such systems appear to be completely transparent. This is sometimes referred to as 'ASCII armoring'. For example, the ViewState component of ASP.NET uses base64 encoding to safely transmit text via HTTP POST, in order to avoid delimiter collision.

The table below describes notable binary-to-text encodings. The efficiency listed is the ratio between the number of bits in the input and the number of bits in the encoded output. For any encoding that maps *n* input possibilities into one 8-bit character, the efficiency is log2(*n*)/8.

Encoding	Efficiency	Programming language implementations	Comments
Ascii85	80%	awk Archived 2014-12-29 at the Wayback Machine, C, C (2), C#,	There exist several variants of this encoding, Base85, btoa, etc.

Encoding	Efficiency	Programming language implementations	Comments
		F#, Go, Java Perl, Python, Python (2)	
Base16	50%	Most languages	As it's based on hexadecimal, there are variants for upper, lower or either case
Base26	58.3%		Used to convert SHA-256 hash into all-uppercase strings in InChIKey (a standard indexing system of chemical structures)[1] and SID (sequence identification, an indexing system of PCR amplicons in forensics).[2] InChIKey specifically uses two kinds of mappings: 14b:3ch, 9b:2ch.
Base32	62.5%	ANSI C, Delphi, Go, Java, C# F#, Python	
Base36	64.6%	bash, C, C++, C#, Java, Perl, PHP, Python, Visual Basic, Swift, many others	Uses only numerals (0–9) and lowercase letters (a–z). Commonly used by URL redirection systems like TinyURL or SnipURL/Snipr as compact alphanumeric identifiers.
Base45	68.6% (97%[a])	Go, Python	Defined in IETF Specification RFC 9285 for including binary data compactly in a QR code.[3]
Base56	72.6%	PHP, Python, Go	Like Base58 but further excludes characters `1` and lowercase-O (`o`) in order to minimise the risk of fraud and human-error.[4]
Base58	73.2%	C, C++, Python, C#, Java	Like Base64 but excludes non-alphanumeric characters (`+` and `/`) and pairs of characters that often look ambiguous when rendered: zero (`0`) and capital-O (`O`), and capital-I (`I`) and lowercase-L (`l`). Base58 is used to represent bitcoin addresses.[*citation needed*] For SegWit, it was replaced by Bech32.

Encoding	Efficiency	Programming language implementations	Comments
			 Base58 in the original bitcoin source code
Base62	74.4%	Rust, Python	Like Base64 but contains only alphanumeric characters.
Base64	75.0%	awk Archived 2014-12-29 at the Wayback Machine, C, C (2), Delphi, Go, Python, many others	An early and still-popular encoding, first specified as part of RFC 989 in 1987
Base85	80.0%	C, Python, Python (2)	Revised version of Ascii85.
BaseXML[5]	80% ± 6 chars	C Python JavaScript	Encoding for stuffing data in XML.
Base91[6]	81.3%	C# F#	Constant width variant
basE91[7]	81.3%	C, Java, PHP, 8086 Assembly, AWK C#, F#, Rust	Variable width variant
Base94[8]	81.9%	Python, C, Rust	
Base122[9]	87.5%	JavaScript, Python, Java, Base125 Python and Javascript, Go, C	Encodes to UTF-8, hence a different efficiency claim from theoretical.
Bech32	62.5% - at least 8 chars overhead (label, separator, 6-char ECC)	C, C++, JavaScript, Go, Python, Haskell, Ruby, Rust	Specification.[10] Used in Bitcoin and the Lightning Network.[11] The data portion is encoded like Base32 with the possibility to check and correct up to 6 mistyped characters using the 6-character BCH code at the end, which also checks/corrects the Human Readable Part. The Bech32m variant has a subtle change that makes it more resilient to changes in length.[12]
BinHex	75%	Perl, C, C (2)	MacOS Classic

Encoding	Efficiency	Programming language implementations	Comments
Intel HEX	≲50%	C library, C++	Typically used to program EPROM, NOR flash memory chips
MIME	See Quoted-printable and Base64	See Quoted-printable and Base64	Encoding container for e-mail-like formatting
S-record (Motorola hex)	49.6%	C library, C++	Typically used to program EPROM, NOR flash memory chips. 49.6% assumes 255 binary bytes per record.
Tektronix hex	≲50%		Typically used to program EPROM, NOR flash memory chips.
TxMS	Variable	TypeScript, CLI, Dart	TxMS compresses binary data into a readable text format using Binary-to-Text encoding and allows reversible conversion back to hexadecimal.
uuencoding	~60% (up to 70%)	Perl, C, Delphi, Java, Python, probably many others	An early encoding developed in 1980 for Unix-to-Unix Copy. Largely replaced by MIME and yEnc
xxencoding	~75% (similar to Uuencoding)	C, Delphi	Proposed (and occasionally used) as replacement for Uuencoding to avoid character set translation problems between ASCII and the EBCDIC systems that could corrupt Uuencoded data
z85 (ZeroMQ spec:32/Z85)	80% (similar to Ascii85/Base85)	C (original), C#, Dart, Erlang, Go, Lua, Ruby, Rust and others	Specifies a subset of ASCII similar to Ascii85, omitting a few characters that may cause program bugs (`` ` \ " ' _ , ; ``). The format conforms to ZeroMQ spec:32/Z85.
RFC 1751 (S/KEY[13])	33%	C,Python	"A Convention for Human-readable 128-bit Keys". A series of small English words is easier for humans to read, remember, and type in than decimal or other binary-to-text encoding systems.[14] Each 64-bit number is mapped to six short words, of

Encoding	Efficiency	Programming language implementations	Comments
			one to four characters each, from a public 2048-word dictionary.[13]

Some older and today uncommon formats include BOO (a base64),[15] BTOA (vaguely-defined "binary to ascii", historically base85, today in JavaScript base64), and USR encoding.

Base64 (with many variants including uuencoding) maps sequences of 6 bits to printable characters. Since there are more than $2^6 = 64$ printable characters, this is possible. A given sequence of bytes is translated by viewing it as a stream of bits, breaking this stream into chunks of 6 bits and generating the sequence of corresponding characters. The different encodings differ in the mapping between sequences of bits and characters and in how the resulting text is formatted.

Some encodings (the original version of BinHex and the recommended encoding for CipherSaber) use four bits instead of six, mapping all possible sequences of 4 bits onto the 16 standard hexadecimal digits. Using 4 bits per encoded character leads to a 50% longer output than base64, but simplifies encoding and decoding—expanding each byte in the source independently to two encoded bytes is simpler than base64's expanding 3 source bytes to 4 encoded bytes.

Out of PETSCII's first 192 codes, 164 have visible representations when quoted: 5 (white), 17–20 and 28–31 (colors and cursor controls), 32–90 (ascii equivalent), 91–127 (graphics), 129 (orange), 133–140 (function keys), 144–159 (colors and cursor controls), and 160–192 (graphics).[16] This theoretically permits encodings, such as base128, between PETSCII-speaking machines.

- Alphanumeric shellcode – Code intended as a payload to exploit a software vulnerability
- Character encoding – Using numbers to represent text characters
- Computer number format – Internal representation of numeric values in a digital computer
- Geocode – Code that represents a geographic entity (location or object)
- Numeral system – Notation for expressing numbers
- Punycode – Encoding for Unicode domain names

1. ^ Encoding for QR code generation automatically selects the encoding to match the input character set, encoding 2 alphanumeric characters in 11 bits, and Base45 encodes 16 bits into 3 such characters. The efficiency is thus 32 bits of binary data encoded in 33 bits: 97%.

1. ^ *"Technical FAQ - InChI Trust"*. *inchi-trust.org*. *Retrieved 2021-01-08.*
2. ^ *Young, Brian; Faris, Tom; Armogida, Luigi (2019). "A nomenclature for sequence-based forensic DNA analysis". Genetics. **42**. Forensic Science International: 14–20. doi:10.1016/j.fsigen.2019.06.001. PMID 31207427. "[...] 2) the hexadecimal output of the hash function is converted to hexavigesimal (base-26)"*

3. ^ *Fältström, Patrik; Ljunggren, Freik; Gulik, Dirk-Willem van (2022-08-11). "The Base45 Data Encoding". "Even in Byte mode, a typical QR code reader tries to interpret a byte sequence as text encoded in UTF-8 or ISO/IEC 8859-1. ... Such data has to be converted into an appropriate text before that text could be encoded as a QR code. ... Base45 ... offers a more compact QR code encoding."*
4. ^ *Duggan, Ross (August 18, 2009). "Base-56 Integer Encoding in PHP".*
5. ^ *"BaseXML - for XML1.0+". GitHub. 16 March 2019.*
6. ^ *Dake He; Yu Sun; Zhen Jia; Xiuying Yu; Wei Guo; Wei He; Chao Qi; Xianhui Lu. "A Proposal of Substitute for Base85/64 – Base91" (PDF). International Institute of Informatics and Systemics.*
7. ^ *"binary to ASCII text encoding". basE91. SourceForge. Retrieved 2023-03-20.*
8. ^ *"Convert binary data to a text with the lowest overhead". Vorakl's notes. April 18, 2020.*
9. ^ *Albertson, Kevin (Nov 26, 2016). "Base-122 Encoding".*
10. ^ *"bitcoin/bips". GitHub. 8 December 2021.*
11. ^ *Rusty Russell; et al. (2020-10-15). "Payment encoding in the Lightning RFC repo". GitHub.*
12. ^ *"Bech32m format for v1+ witness addresses". GitHub. 5 December 2021.*
13. ^ Jump up to: ***a*** ***b*** RFC 1760 "The S/KEY One-Time Password System".
14. ^ RFC 1751 "A Convention for Human-Readable 128-bit Keys"
15. ^ *"Boo File Format, Encoding and Decoding". Columbia University.*
16. ^ *"Commodore 64 PETSCII codes". sta.c64.org.*
