Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: October 2, 2021

S. Nakamoto
Bitcoin
M. Sporny
Digital Bazaar
March 31, 2021

The Base58 Encoding Scheme
draft-msporny-base58-03

Abstract

This document specifies the base 58 encoding scheme, including an introduction to the benefits of the approach, the encoding and decoding algorithm, alternative alphabets, and security considerations.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 2, 2021.

Copyright Notice

Table of Contents

1. Introduction

When trasmitting data, it can be useful to encode the data in a way that survives lower fidelity transmission mechanisms. For example, encoding data using a human alphabet in a way that a person can visually confirm the encoded data can be more beneficial than encoding it in binary form. The Base58 encoding scheme is similar to the Base64 encoding scheme in that it can translate any binary data to a text string. It is different from Base64 in that the conversion alphabet has been carefully picked to work well in environments where a person, such as a developer or support technician, might need to visually confirm the information with low error rates.

Base58 is designed with a number of usability characteristics in mind that Base64 does not consider. First, similar looking letters are omitted such as 0 (zero), O (capital o), I (capital i) and l (lower case L). Doing so eliminates the possibility of a human being mistaking similar characters for the wrong character. Second, the non-alphanumeric characters + (plus), = (equals), and / (slash) are omitted to make it possible to use Base58 values in all modern file systems and URL schemes without the need for further system-specific encoding schemes. Third, by using only alphanumeric characters, easy double-click or double tap selection is possible in modern computer interfaces. Fourth, social messaging systems do not line break on alphanumeric strings making it easier to e-mail or message Base58 values when debugging systems. Fifth, unlike Base64, there is no byte padding making many Base58 values smaller (on average) or the same size as Base64 values for values up to 64 bytes, and less than 2% larger for larger values. Finally, Base64 has eleven encoding variations that lead to confusion among developers on which variety of Base64 to use. This specification asserts that there is just one

simple encoding mechanism for Base58, making implementations and developer interactions simpler.

While Base58 does have a number of beneficial usability features, it is not always a good choice for an encoding format. For example,

when encoding large amounts of data, it is 2% less efficient than base64. Developers might avoid Base58 if a 2% increase in efficiency over large data sets is desired.

This document specifies the base 58 encoding scheme, including an introduction to the benefits of the approach, the encoding and decoding algorithm, alternative alphabets, and security considerations.

1.1. Requirements Language

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 RFC 2119.

2. The Base58 Alphabet

The Base58 alphabet consists of the following characters:

123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz

Each byte, interpreted as a decimal value, from 0 to 57 maps to the alphabet above in the following way:

Decimal	Character	Decimal	Character
0	1	29	W
1	2	30	X
2	3	31	Y
3	4	32	Z
4	5	33	a
5	6	34	b
6	7	35	c
7	8	36	d
8	9	37	e
9	A	38	f
10	B	39	g
11	C	40	h
12	D	41	i
13	E	42	j
14	F	43	k
15	G	44	m
16	H	45	n
17	J	46	o
18	K	47	p
19	L	48	q
20	M	49	r
21	N	50	s
22	P	51	t
23	Q	52	u
24	R	53	v
25	S	54	w

26	T	55	x
27	U	56	y
28	V	57	z

Table 1: Base58 Mapping Table

Other application-specific alphabets for Base58, such as the Ripple alphabet and the Flickr alphabet exist. Those alphabets, while valid in their own application spaces, are not valid encoding formats for this specification and MUST NOT be used. Supporting more than one Base58 encoding alphabet would harm interoperability.

3. The Base58 Encoding Algorithm

To encode an array of bytes to a Base58 encoded value, run the following algorithm. All mathematical operations MUST be performed using integer arithmetic. Start by initializing a 'zero_counter' to zero (0x0), an 'encoding_flag' to zero (0x0), a 'b58_bytes' array, a

'b58_encoding' array, and a 'carry' value to zero (0x0). For each byte in the array of bytes and while 'carry' does not equal zero (0x0) after the first iteration:

1. If 'encoding_flag' is not set, and if the byte is a zero (0x0), increment the value of 'zero_counter'. If the value is not zero (0x0), set 'encoding_flag' to true (0x1).

2. If 'encoding_flag' is set, multiply the current byte value by 256 and add it to 'carry'.

3. Set the corresponding byte value in 'b58_bytes' to the value of 'carry' modulus 58.

4. Set 'carry' to the value of 'carry' divided by 58.

Once the 'b58_bytes' array has been constructed, generate the final 'b58_encoding' using the following algorithm. Set the first 'zero_counter' bytes in 'b58_encoding' to '1'. Then, for every byte in 'b58_array', map the byte value using the Base58 alphabet in the previous section to its corresponding character in 'b58_encoding'. Return 'b58_encoding' as the Base58 representation of the input array

of bytes.

4. The Base58 Decoding Algorithm

To decode a Base58 encoded array of bytes to a decoded array of bytes, run the following algorithm. All mathematical operations MUST be performed using integer arithmetic. Start by initializing a 'raw_bytes' array, and a 'carry' value to zero (0x0). For each input byte in the array of input bytes:

1. Set 'carry' to the byte value associated with the input byte character. If a mapping does not exist, return an error code.

2. While 'carry' does not equal zero and there are input bytes remaining:

    1. Multiply the input byte value by 58 and add it to 'carry'.

    2. Set the output byte value to 'carry' modulus 256.

    3. Set 'carry' to the value of 'carry' divided by 256.

3. Set the corresponding byte value in 'raw_bytes' to the value of 'carry' modulus 58.

4. Set 'carry' to the value of 'carry' divided by 58.

5. Test Vectors

The following examples can be used as test vectors for the algorithms in this specification:

The Base58 encoded value for "Hello World!" is:

2NEpo7TZRRrLZSi2U

The Base58 encoded value for "The quick brown fox jumps over the lazy dog." is:

USm3fpXnKG5EUBx2ndxBDMPVciP5hGey2Jh4NDv6gmeo1LkMeiKrLJUUBk6Z

The Base58 encoded value for 0x0000287fb4cd is:

11233QC4

6. Acknowledgements

Thanks to Satoshi Nakamoto for inventing the Base58 encoding format and the Bitcoin community for popularizing its usage.

7. Security Considerations

Authors' Addresses

Satoshi Nakamoto
Bitcoin

Email: satoshin@gmx.com

Manu Sporny
Digital Bazaar

Email: msporny@digitalbazaar.com