Independent
Internet-Draft
Intended status: Standards Track
Expires: 12 October 2026

K. R. Davis
D. Brown
Cisco Systems

10 April 2026

A Sortable Base64 Alphabet
draft-brown-davis-base64-sort-01

Abstract

This document details a formal specification for a new Base64 alphabet which follows the US-ASCII ordering and sorts the same as binary. This serves as an alternative variant to Base64 and Base64url when lexicographically sortable outputs are required. The alphabet re-uses the Base64url special characters and is designed to be compatible with existing Base64 implementations, while providing a new option for applications that require sorted output.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://raw.githubusercontent.com/kyzer-davis/base64-sort-ietf-draft/refs/heads/main/draft-brown-davis-base64-sort.md. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-brown-davis-base64-sort/.

Discussion of this document takes place on the Revise Universally Unique Identifier Definitions (uuidrev) Working Group mailing list (mailto:uuidrev@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/uuidrev/. Subscribe at https://www.ietf.org/mailman/listinfo/uuidrev/.

Source for this draft and an issue tracker can be found at https://github.com/kyzer-davis/base64-sort-ietf-draft.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Base64 ([RFC4648], Section 4) and Base64url ([RFC4648], Section 5) have been widely implemented and tested across the industry; however, if one requires a lexicographically sortable output they will need to use either Base32hex, Base36, or [Base58].

There is no standardized sortable Base64 alphabet. Since the Base64 alphabets from RFC4648 see far more industry usage than the other previously mentioned alphabets there is a real need and requirement for a sortable Base64 Alphabet in the industry. See: [RFC4648_Usage_Report] for the breakdown of RFC citations for RFC4648 alphabets.

This gap was made further apparent while examining Base64 usage with Universally Unique IDentifiers (UUIDs) defined in [RFC9562] to create [new-uuid-encoding-techniques-ietf-draft] which asserts that UUIDv6 and UUIDv7 benefit greatly from lexicographically sortable alphabets like Base64 Sortable.

This document serves as a standards-track RFC to provide a new alphabet that follows the US-ASCII ([RFC20]) ordering, re-uses the Base64url special characters and sorts the same as binary. This document is created to serve as a reference for new libraries and for future RFCs to cite.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. The Alphabet

This Base64 Sortable alphabet, which can be referenced as "Base64sort" or sometimes "Base64lex", is a variant of the Base64url alphabet ([RFC4648], Section 5).

While it utilizes the same set of 64 URL-safe characters as Base64url, its fundamental distinction lies in the assignment of values to these characters by aligning the character values with their US-ASCII ([RFC20]) ordering to ensure that the encoded output sorts lexicographically in the same order as the underlying binary data.

Specifically, the numeric characters (0-9) have been moved before the uppercase and lowercase characters while the special symbol characters, hyphen (-) and underscore (_), are placed at the beginning and middle of the alphabet as they appear in the US-ASCII character set.

Table 1 details the alphabet characters along with their respective decoding and encoding values.

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

Table 1: The Base64sort Alphabet as table

The Alphabet as a continuous text input can be found in Figure 1.

```
-0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ_abcdefghijklmnopqrstuvwxyz
```

Figure 1: The Base64sort Alphabet as text

3.1. Padding

Base64sort follows the same padding logic as Base64url ([RFC4648], Section 5). That is, padding MAY be used with Base64sort.

Although the equal sign (=) padding character is positioned between the numeric characters and the uppercase letters in the US-ASCII character set; it does not cause any issues with sorting since the padding is always in the right-most, least significant position of the encoded string.

Note that this assumes the sorted data is either entirely all padded or un-padded encodings. Mixing padded and un-padded encodings in the same sorted set may lead to unexpected results since the padding character would be treated as a significant character in the sort order.

Implementations MAY utilize a padding character that is not part of the Base64sort alphabet and does not interfere with sorting, such as a tilde (~), to maintain the integrity of the sort order of mixed datasets while still indicating padding.

4. Decoding

The decoding process for Base64sort follows the principles outlined in ([RFC4648], Section 5), for decoding Base64url.

5. Security Considerations

This document has no security considerations.

6. IANA Considerations

This document has no IANA actions.

7. References

7.1. Normative References

[RFC20] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/rfc/rfc20>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/rfc4648>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

7.2. Informative References

[Base58] Bitcoin, "Bitcoin Base58 Encoding", commit fae71d3, November 2008, <https://github.com/bitcoin/bitcoin/blob/master/src/base58.cpp>.

[new-uuid-encoding-techniques-ietf-draft] Davis, K., "Alternate UUID Encoding Methods", n.d., <https://datatracker.ietf.org/doc/draft-davis-uuidrev-alt-uuid-encoding-methods/>.

[RFC4648_Usage_Report] "RFC4648 Alphabet Usage Report", commit de0a2760, November 2025, <https://gitlab.com/julian.reschke/base-encodings-terminology/-/blob/main/classifcation.md?ref_type=heads>.

[RFC9562] Davis, K., Peabody, B., and P. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <https://www.rfc-editor.org/rfc/rfc9562>.

Appendix A. Changelog

draft-00:

* Initial Release

draft-01:

* Update frontmatter venue and Alt encoding link

Appendix B. Test Vectors

The following test vectors start with a base of those found in [RFC4648] and then include additional values for testing the new alphabet and its sorting properties.

B.1. Encoding

Input	Encoded Data	Input Type
f	OV	Text
fo	Oaw	Text
foo	Oaxj	Text
foob	OaxjNV	Text
fooba	OaxjNa3	Text
foobar	OaxjNa4m	Text
test	S5KnS-	Text
Hello World	H5KgQ5wVKqxmQ5F	Text
-	AF	Hyphen
0	B-	Digit
_	Mk	Underscore
A	FF	Uppercase Letter
a	NF	Lowercase Letter
=	EF	Equal Sign
~	UV	Tilde
0123456789	B23mBnFpCYRsDF==	Input with Equal Padding
0123456789	B23mBnFpCYRsDF~~	Input with Tilde Padding

Table 2: Base64 Sortable Encode Table

B.2. Decoding

Input	Decoded Data
OV	f
Oaw	fo
Oaxj	foo
OaxjNV	foob
OaxjNa3	fooba
OaxjNa4m	foobar
S5KnS-	test
H5KgQ5wVKqxmQ5F	Hello World
AF	-
B-	0
Mk	_
FF	A
NF	a
EF	=
UV	~
B23mBnFpCYRsDF==	0123456789
B23mBnFpCYRsDF~~	0123456789

Table 3: Base64 Sortable Decode Table

Appendix C. Industry Tests

To validate the lexicographical sortability of the Base64sort alphabet in common computing environments, a series of tests were conducted across various programming languages and database systems.

The objective was to observe how a mixed list of characters from the proposed alphabet would sort using their default string comparison mechanisms.

C.1. Programming Language Results

Tests were performed using Python, C, C++, Java, JavaScript, Delphi/ Object Pascal, Perl, and Go.

In these languages, a list containing a subset of the Base64sort alphabet characters (A, a, b, 7, 3, B, C, c, E, z, -, _) was sorted.

The results consistently demonstrated sorting according to the US-ASCII character order: -, 3, 7, A, B, C, E, _, a, b, c, z.

This indicates that the chosen character set and their relative ASCII values are respected by the default string sorting algorithms in these environments.

However, C# and Visual Basic exhibited a different sorting behavior, placing _ before 3 and sorting uppercase letters after their lowercase counterparts (e.g., a before A).

This suggests that their default string comparison routines may employ locale-aware or cultural sorting rules rather than strict ordinal (ASCII) comparison.

C.2. Database Test Results

Database tests were conducted on PostgreSQL, MongoDB, and Redis.

With PostgreSQL, when using ORDER BY val ASC, the default collation resulted in underscore (_) appearing before the hyphen (-), and lowercase letters before uppercase letters. However, explicitly specifying ORDER BY val COLLATE "C" (which enforces a C locale, often equivalent to strict ASCII ordering) produced the desired sort order: -, 3, 7, A, B, C, E, _, a, b, c, z.

MongoDB’s default sort behavior for the test set aligned with the desired US-ASCII order, matching the results observed in most programming languages.

Redis’s SORT ... ALPHA command also showed a deviation, placing _ before 3 and lowercase letters before uppercase, similar to the default SQL, C#, and Visual Basic results.

C.3. Conclusion

The testing confirms that the Base64 Sortable alphabet generally achieves its goal of producing lexicographically sortable output when standard US-ASCII or C-locale string comparison rules are applied.

Implementers should be aware that some programming languages and database systems may employ locale-aware or alternative default collation rules that can alter the sort order.

To ensure consistent binary-equivalent sorting, explicit specification of ordinal or C-locale collation may be necessary in environments where such options are available.

Authors' Addresses

Kyzer R. Davis
Cisco Systems
Email: kydavis@cisco.com, kyzer.davis@outlook.com

Dillon Brown
Cisco Systems
Email: dillbrow@cisco.com