Why BIP39 Asks for Sorted Wordlists

November 7, 2026 · Document #421 · bip39-phrase.com

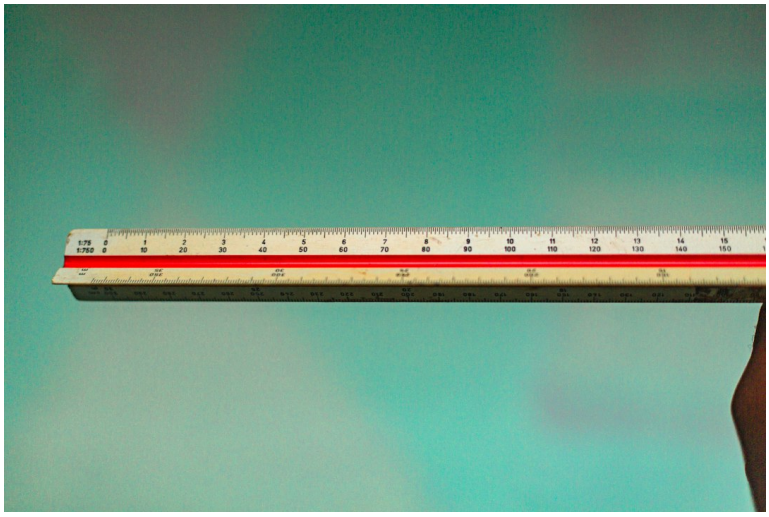


Photo by josh A. D. on Unsplash

BIP39’s sorted English wordlist makes word lookup efficient while preserving the fixed mapping between words and 11-bit values.

Contents

What the sorted-wordlists guidance means	2
How alphabetical position becomes an 11-bit value	2
Why sorting enables efficient lookup	3
Why reordering breaks the agreed encoding	3
FAQ	4
Sources	4
Example sequences from the BIP39 wordlist	4
Reading a BIP39 phrase	5
See also	6

BIP39’s English wordlist is an ordered encoding table: each word’s zero-based position identifies an 11-bit value. Alphabetical order makes that table easy to search, while its fixed ordering keeps implementations using that list interpreting the same words consistently. Changing the order changes the encoding, even when all 2,048 words remain present.
([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039.mediawiki))

What the sorted-wordlists guidance means

BIP39 lists sorting among the characteristics of an ideal wordlist. It explicitly gives efficient lookup as the reason: implementations can use binary search instead of scanning the list sequentially. It also mentions tries, or prefix trees, as useful for compression. These are implementation benefits, not additional sources of randomness.
([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039.mediawiki))

Two properties work together here. An ordered list provides the mapping between words and numbers. Sorting that list alphabetically makes the same structure searchable by spelling. An unsorted list could still encode numbers if everyone agreed on its exact order, but ordinary binary search by spelling would no longer work directly. This follows from the encoding and lookup rules.
([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039.mediawiki))

The published English file supplies the agreed order. Software should preserve that sequence rather than rebuild it using a preferred dictionary, spelling corrections or a different sorting rule.
([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039/english.txt))

How alphabetical position becomes an 11-bit value

There are `2^11 = 2048` possible 11-bit values, ranging from `0` through `2047`. BIP39 appends checksum bits to the initial entropy, divides the combined bitstream into 11-bit groups, and uses each group as a wordlist index.
([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039.mediawiki))

The English list begins with these mappings:

- `abandon`: index `0`, binary `00000000000`.
- `ability`: index `1`, binary `00000000001`.
- `able`: index `2`, binary `00000000010`.
- `about`: index `3`, binary `00000000011`.
  ([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039/english.txt))

Position means zero-based index. A printed list numbering its first entry `1` uses display numbers that are one greater than BIP39's encoded values. Confusing those conventions changes the recovered bits. Alphabetical order determines which English word occupies each position; the encoding rule makes that position its numeric value.
([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039.mediawiki))

For a 12-word mnemonic, `128` entropy bits receive `4` checksum bits, producing `132` bits. Dividing by `11` gives twelve indices. Validation reverses that mapping and checks the checksum against the recovered entropy. The checksum catches many errors, not all: exactly `1` in `16` possible 12-word sequences drawn from the list has a valid checksum.
([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039.mediawiki))

Why sorting enables efficient lookup

Converting an index into a word is straightforward with an array: retrieve the entry at that index. Sorting mainly helps the reverse operation, when software receives a word and needs its index. A linear search may inspect all `2048` entries; binary search repeatedly eliminates roughly half the remaining candidates. ([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039.mediawiki))

A binary-search implementation follows this sequence:

1. Compare the entered word with the middle entry.
2. If it matches, return that entry's original index.
3. If it sorts earlier, continue in the lower half.
4. If it sorts later, continue in the upper half.
5. Repeat until a match is found or no candidates remain. ([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039.mediawiki))

From the list size, `log2(2048) = 11`: the search needs roughly eleven halving stages. Exact comparison counts depend on the implementation. Binary search is an available optimization, not a requirement that every wallet use that algorithm. ([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039.mediawiki))

BIP39 separately recommends selecting words so that their first four letters suffice to identify them; shorter English words can be entered in full. That selection property helps prefix entry; sorting helps locate matching entries. A trie can share common prefixes, although constructing a trie does not inherently require sorted input. ([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039.mediawiki))

Why reordering breaks the agreed encoding

Consider reversing the English list. Every word moves to a different index, so every mnemonic generated from the same entropy changes. Reading existing words through that reversed mapping also produces different bits. More generally, any permutation changes the mapping for the words it moves. A swap of two entries leaves phrases containing neither entry unaffected, so saying every possible reorder changes every individual phrase would be too broad. These consequences follow directly from positional encoding. ([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039.mediawiki))

There is another essential distinction: BIP39 derives the wallet seed from the mnemonic text and optional passphrase using `PBKDF2-HMAC-SHA512`, with `2048` iterations and UTF-8 NFKD normalization. That operation does not look up word indices. Consequently, unchanged mnemonic text and passphrase produce the same seed even if a separate wordlist file is reordered, though wallet validation may reject the phrase. Generating different words changes the seed derivation input. ([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039.mediawiki))

For implementation checks, compare the complete wordlist sequence against the published file and verify encoding against the test vectors linked by BIP39. Checking only the number of entries cannot detect reordered words. For recovery, preserve the phrase's original word order; alphabetical sorting applies to the dictionary. ([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039.mediawiki))

FAQ

Should I alphabetize my recovery phrase?

No. Its word sequence carries the encoded bits and supplies the text used for seed derivation. Preserve its original order.
([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039.mediawiki))

Does alphabetical sorting make a phrase harder to guess?

Sorting improves lookup. It adds no entropy to the random input used to generate the mnemonic.
([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039.mediawiki))

Can software use a different internal storage order?

Yes, provided it preserves the original BIP39 indices explicitly. Using new storage positions as encoded values changes the mapping.
([raw.githubusercontent.com](https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039.mediawiki))

Sources

- BIP39: Mnemonic code for generating deterministic keys (github.com)
- BIP39 English wordlist (github.com)
- Trezor python-mnemonic reference implementation (github.com)

> Continue reading on bip39-phrase.com: BIP39 vs BIP32 vs BIP44: How the Seed Standards Fit Together
> More on this topic: BIP Standards

Example sequences from the BIP39 wordlist

Randomly generated sequences of words from the BIP39 English wordlist. None of them passes the BIP39 checksum, so no wallet will accept them: they show the format only. Never store funds with a phrase you found in a document.

Example 1

1	immense	2	nest	3	harbor	4	night	5	oak	6	arrange
7	often	8	six	9	matter	10	blossom	11	enforce	12	purchase

Example 2

1	sleep	2	pact	3	believe	4	ankle	5	lunch	6	decade
7	steel	8	sword	9	spare	10	ticket	11	fiscal	12	team

Example 3

1	assume	2	ancient	3	lemon	4	risk	5	solve	6	current
7	burden	8	gain	9	text	10	laundry	11	style	12	toilet

Example 4

1	access	2	youth	3	observe	4	carpet	5	kite	6	flower
7	argue	8	soap	9	into	10	bubble	11	recipe	12	candy

Example 5

1	emerge	2	shrug	3	silly	4	inquiry	5	lizard	6	sunny
7	lesson	8	trophy	9	leisure	10	fine	11	stomach	12	hidden

Example 6

1	essay	2	add	3	magic	4	bottom	5	pony	6	demise
7	coyote	8	lamp	9	license	10	such	11	device	12	message

Example 7

1	wall	2	jazz	3	drastic	4	panther	5	tag	6	tip
7	hair	8	knock	9	fossil	10	voyage	11	actress	12	blood

Example 8

1	photo	2	metal	3	common	4	first	5	happy	6	rapid
7	brave	8	expand	9	such	10	gun	11	stem	12	snap

Reading a BIP39 phrase

Every word stands for an 11-bit number: its position in the 2048-word list, counted from 0. Here is Example 1 decoded word by word. Each word links to its page on bip39-phrase.com, where the list is numbered from 1.

#	Word	Index (0-2047)	Word no.	11 bits
1	immense	908	909	01110001100
2	nest	1188	1189	10010100100
3	harbor	840	841	01101001000
4	night	1196	1197	10010101100
5	oak	1214	1215	10010111110
6	arrange	98	99	00001100010
7	often	1229	1230	10011001101
8	six	1614	1615	11001001110
9	matter	1098	1099	10001001010
10	blossom	192	193	00011000000
11	enforce	593	594	01001010001
12	purchase	1393	1394	10101110001

The 12 words carry 132 bits: 128 bits of entropy followed by a 4-bit checksum. The checksum must equal the first 4 bits of the SHA-256 hash of the entropy.

```
Entropy (hex): 719291a44ac97c18a66e4e89430128d7
```

This sequence ends in the checksum bits 0001, but SHA-256 of its entropy starts with 0011, so wallets reject it as an invalid phrase. That is the point of the checksum: about 15 of every 16 random or mistyped sequences fail it.

See also

- Taproot Key-Path Signing in bitcoinjs-lib · November 6, 2026
- The SLIP-39 Digest: Detecting a Bad Share Set · November 4, 2026
- NFKD Normalization in BIP39 · November 1, 2026
- BIPs, SLIPs, EIPs and BOLTs: Four Proposal Series · October 28, 2026

Educational document. The example sequences are random and fail the BIP39 checksum: they are not wallets and must never hold funds. Security notes: bip39-phrase.com/security/