# NFKD Normalization in BIP39

November 1, 2026 · Document #415 · bip39-phrase.com



Photo by Magic Fan on Unsplash

BIP39 applies Unicode NFKD to the mnemonic and passphrase before UTF-8 encoding so canonically or compatibly equivalent text produces the same seed inputs.

**Contents**


Why the same character can have different bytes 1

What NFKD does in BIP39 2

Why this affects recovery 2

Implementation and recovery checks 3

FAQ 3

Sources 4

Example sequences from the BIP39 wordlist 4

Reading a BIP39 phrase 5

See also 5


A BIP39 wallet must normalize the mnemonic sentence and passphrase with Unicode NFKD before deriving the seed. This makes canonically or compatibly equivalent Unicode text produce the same byte sequence, so different representations of equivalent text do not create different wallet seeds.

## Why the same character can have different bytes

Unicode assigns a code point to each character, but a visible character can be represented by more than one sequence of code points. For example, “é” can be written as the single code point U+00E9, or as U+0065 (e) followed by U+0301, the combining acute accent. These sequences can look identical while their UTF-8 encodings contain different bytes.

A cryptographic function does not see the displayed glyph. It sees bytes. If a wallet encoded these two forms directly as UTF-8, it would give PBKDF2 different input and produce different seeds, even though the user might believe that the mnemonic or passphrase is the same.

Other Unicode differences involve compatibility characters. A fullwidth Latin letter, a ligature, or a character with a compatibility equivalent can look like ordinary text but have a different code point. Unicode normalization provides defined rules for converting canonical and compatibility equivalents into a consistent form.

Visual similarity alone is not enough. Two characters can look alike while having different meanings, and NFKD does not identify every visual confusable. The wallet must therefore normalize the exact Unicode text supplied by the user and preserve the resulting mnemonic and passphrase accurately.

## What NFKD does in BIP39

NFKD means **Normalization Form Compatibility Decomposition**. It performs canonical decomposition followed by compatibility decomposition. In practical terms, it expands characters into a standardized sequence of code points, including decomposing accented characters and selected compatibility characters.

BIP39 applies NFKD to both inputs used for seed derivation:

- The mnemonic sentence is normalized with NFKD and then encoded as UTF-8.
- The passphrase is combined with the ASCII string `mnemonic`, normalized with NFKD, and then encoded as UTF-8 to form the PBKDF2 salt.

The derivation is PBKDF2 using HMAC-SHA512, with 2048 iterations and a derived-key length of 64 bytes. Conceptually, it is equivalent to:

```
PBKDF2-HMAC-SHA512(password=UTF8(NFKD(mnemonic)), salt=UTF8(NFKD("mnemonic" + passphrase)), iterations=2048, dkLen=64)
```

The mnemonic sentence and the passphrase are separate inputs. Normalizing only the mnemonic, or only the passphrase, does not implement BIP39 correctly. A wallet that accepts a passphrase must apply the same NFKD rule to it, including spaces and non-ASCII characters.

## Why this affects recovery

The seed is determined by the exact normalized mnemonic bytes and normalized salt bytes. If two wallets use different normalization rules, they can derive different seeds from text that appears identical on screen. The resulting accounts may have no transaction history, even though the user entered what seems to be the correct phrase.

This is especially relevant when a phrase is copied between operating systems, keyboards, password managers, wallet applications, or hardware devices. Text can be converted into a different Unicode representation during input or storage. Normalization at the BIP39 boundary gives compliant implementations a common byte-level input.

The same rule applies to the BIP39 passphrase, sometimes called the “25th word.” It is not selected from the BIP39 word list and it is not checked by the mnemonic checksum. Every distinct normalized passphrase selects

a different seed. For example, a passphrase containing an accented character can change the wallet completely if one implementation receives a precomposed form and another receives a decomposed form without normalizing them.

NFKD also means that some compatibility distinctions are intentionally removed. A user should not rely on different compatibility spellings to create separate BIP39 wallets. Store and restore the exact original text, and test recovery with the wallet and derivation settings that will be used in an emergency.

## Implementation and recovery checks

A BIP39 implementation should make the normalization order explicit:

1. Accept the mnemonic sentence as Unicode text.
2. Apply Unicode NFKD to the complete sentence, including its spaces.
3. Encode the normalized sentence as UTF-8 for the PBKDF2 password.
4. Concatenate the ASCII text `mnemonic` with the original passphrase text.
5. Apply NFKD to that complete salt string.
6. Encode the normalized salt as UTF-8.
7. Run PBKDF2-HMAC-SHA512 with `2048` iterations and `64` output bytes.

Do not normalize individual words with custom rules, trim spaces, or apply a locale-specific case conversion. BIP39 word lists are language-specific, and the Japanese test vectors use a particular separator character; implementations should follow the BIP39 language and test-vector rules rather than substitute platform defaults.

For a diagnostic test, use the official BIP39 vector beginning `abandon abandon abandon abandon abandon abandon abandon abandon abandon abandon abandon about` and compare the derived result with an independent implementation. A test that uses only ASCII text cannot prove that Unicode normalization works, so production software should also test official non-ASCII vectors and verify both the mnemonic and passphrase paths.

A backup should record the mnemonic words, the passphrase if one is used, the wallet's derivation path and network, and any language or separator details required by the implementation. The passphrase is part of the secret: losing its exact text, including Unicode characters and spaces, can make the funds unrecoverable.

## FAQ

### Does NFKD change the visible mnemonic?

It can change the underlying code-point sequence while leaving the displayed text looking the same. Some compatibility characters may also display differently after decomposition, so the normalized result must be handled consistently.

### Does NFKD make similar-looking characters identical?

No. It handles defined canonical and compatibility equivalences. It does not turn every visual confusable into the same character.

### Must the BIP39 passphrase be normalized too?

Yes. BIP39 applies NFKD to the passphrase as part of the PBKDF2 salt, after adding the `mnemonic` prefix.

## Sources


- BIP-0039: Mnemonic code for generating deterministic keys (github.com)
- Unicode Standard Annex #15: Unicode Normalization Forms (www.unicode.org)
- RFC 8018: PKCS #5: Password-Based Cryptography Specification Version 2.1 (www.rfc-editor.org)



> 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 | figure | 2 | discover | 3 | bulb | 4 | child | 5 | silly | 6 | swing |
| 7 | fragile | 8 | person | 9 | awkward | 10 | unknown | 11 | elite | 12 | claim |

**Example 2**

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | soap | 2 | cattle | 3 | cricket | 4 | initial | 5 | process | 6 | sister |
| 7 | crumble | 8 | wood | 9 | myth | 10 | onion | 11 | tragic | 12 | hurdle |

**Example 3**

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | toddler | 2 | monitor | 3 | mention | 4 | flush | 5 | jealous | 6 | conduct |
| 7 | panda | 8 | elephant | 9 | gallery | 10 | only | 11 | skin | 12 | keen |

**Example 4**

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | gather | 2 | arrive | 3 | fresh | 4 | twice | 5 | diamond | 6 | prepare |
| 7 | donor | 8 | limit | 9 | forget | 10 | latin | 11 | water | 12 | hard |

**Example 5**

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | boat | 2 | project | 3 | doctor | 4 | company | 5 | polar | 6 | immense |
| 7 | connect | 8 | loud | 9 | peasant | 10 | hobby | 11 | metal | 12 | person |

**Example 6**

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | advice | 2 | rare | 3 | inch | 4 | oil | 5 | like | 6 | leader |
| 7 | awesome | 8 | cup | 9 | uphold | 10 | cradle | 11 | price | 12 | border |

| Example 7 | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | execute | 2 | genre | 3 | sniff | 4 | eagle | 5 | soap | 6 | dress |
| 7 | allow | 8 | spell | 9 | math | 10 | shield | 11 | that | 12 | fold |

| Example 8 | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | chest | 2 | giggle | 3 | stuff | 4 | tone | 5 | toy | 6 | fancy |
| 7 | sweet | 8 | over | 9 | robot | 10 | oil | 11 | cross | 12 | giraffe |

## 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 | figure | 688 | 689 | 01010110000 |
| 2 | discover | 503 | 504 | 00111110111 |
| 3 | bulb | 239 | 240 | 00011101111 |
| 4 | child | 319 | 320 | 00100111111 |
| 5 | silly | 1605 | 1606 | 11001000101 |
| 6 | swing | 1760 | 1761 | 11011100000 |
| 7 | fragile | 739 | 740 | 01011100011 |
| 8 | person | 1306 | 1307 | 10100011010 |
| 9 | awkward | 134 | 135 | 00010000110 |
| 10 | unknown | 1903 | 1904 | 11101101111 |
| 11 | elite | 576 | 577 | 01001000000 |
| 12 | claim | 333 | 334 | 00101001101 |

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): 5607dc7793fc8bb8171d1a10ddbd2014
```

This sequence ends in the checksum bits 1101, but SHA-256 of its entropy starts with 1010, 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

- BIPs, SLIPs, EIPs and BOLTs: Four Proposal Series · October 28, 2026
- Parse and Sign Bitcoin PSBTs with embit in Python · October 25, 2026
- BIP380 Descriptor Key Expressions: Hex, WIF, and BIP32 · October 24, 2026
- PBKDF2 vs HMAC: Speed Choices in BIP39 and BIP32 · October 23, 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/