

PBKDF2 vs HMAC: Speed Choices in BIP39 and BIP32

October 23, 2026 · Document #406 · bip39-pharse.com



Photo by taro ohtani on Unsplash

BIP39 uses 2048 PBKDF2 rounds to resist passphrase guessing, while BIP32 relies on a single HMAC round for fast child key derivation.

Contents

Why BIP39 Employs Slow Key Stretching	2
Why BIP32 Uses a Single Fast HMAC Round	2
Cryptographic Comparison: BIP39 vs BIP32 Derivation	3
Hardware Wallet Performance Implications	3
FAQ	3
Sources	4
Example sequences from the BIP39 wordlist	4
Reading a BIP39 phrase	5
See also	6

Deriving cryptocurrency keys balances two conflicting operational goals: brute-force resistance against human-readable secrets and computational throughput when scanning addresses. BIP39 and BIP32 resolve this conflict by splitting key generation into two fundamentally different cryptographic phases.

BIP39 converts a mnemonic phrase and optional passphrase into a 512-bit binary seed using **PBKDF2** (Password-Based Key Derivation Function 2) set to 2048 iterations of HMAC-SHA512. In contrast, BIP32 derives child keys from that seed down a derivation tree using a single iteration of HMAC-SHA512 per step. The difference in execution speed is deliberate and directly reflects the

cryptographic entropy present at each stage.

Why BIP39 Employs Slow Key Stretching

A standard BIP39 mnemonic phrase contains between 128 and 256 bits of entropy. However, BIP39 also supports an optional user-selected salt component commonly called the "passphrase" or "13th/25th word." Users frequently select short, memorable phrases or alphanumeric passwords that lack high cryptographic entropy.

To mitigate dictionary and brute-force attacks against weak passphrases, BIP39 applies key stretching via PBKDF2:

- **PRF:** HMAC-SHA512
- **Password:** The NFKD-normalized UTF-8 string of the mnemonic sentence
- **Salt:** The string literal "mnemonic" concatenated with the NFKD-normalized UTF-8 passphrase
- **Iteration count:** 2048
- **Output key length:** 512 bits (64 bytes)

Because PBKDF2 runs HMAC-SHA512 across 2048 chained rounds, computing the seed requires 2048 sequential HMAC evaluations. An attacker testing candidate passphrases against an extracted wallet seed or known public key must pay this computational cost for every single guess.

This delay occurs only once per wallet initialization. When an application imports or restores a seed phrase, computing PBKDF2 takes tens of milliseconds on modern consumer processors and roughly 0.5 to 2.0 seconds on low-power hardware wallet microcontrollers (such as ARM Cortex-M processors). Because this calculation runs only when unlocking or importing the device, the execution latency does not impair ongoing user experience.

Why BIP32 Uses a Single Fast HMAC Round

Once the 512-bit seed exists, BIP32 takes over to establish a Hierarchical Deterministic (HD) key tree. The root step splits the 512-bit output of an initial HMAC-SHA512 (Key = "Bitcoin seed", Data = seed) into a 256-bit master private key and a 256-bit master chain code. Every subsequent child key derivation (CKD) computes another HMAC-SHA512:

- 1 For hardened child derivation: HMAC-SHA512(Key = parent_chain_code, Data = 0x00 || parent_private_key || child_index)
- 2 For normal child derivation: HMAC-SHA512(Key = parent_chain_code, Data = parent_public_key || child_index)
- 3 The resulting 64-byte payload is split: the left 32 bytes modify the parent key, and the right 32 bytes form the child chain code.

BIP32 executes this derivation exactly once per tree level with zero iteration looping. Repeating HMAC here would provide no practical security benefit because the inputs—the parent key and the chain code—already possess full 256-bit cryptographic entropy. When the search space is 2^{256} , exhaustive search is mathematically infeasible regardless of whether derivation takes one microsecond or one second.

Furthermore, HD wallets must frequently derive hundreds or thousands of child keys in sequence during balance checks and UTXO discovery. If BIP32 child key derivation used 2048 PBKDF2 iterations per step, scanning an account with an address gap limit of 20 addresses across multiple derivation paths would require millions of hash operations, making wallet restoration on mobile devices or hardware signing devices completely impractical.

Cryptographic Comparison: BIP39 vs BIP32 Derivation

The architectural division between the two standards reflects their operational differences:

Attribute	BIP39 Seed Derivation	BIP32 Child Derivation
Cryptographic Primitive	PBKDF2 (HMAC-SHA512)	Single-pass HMAC-SHA512
Iteration Count	2048 sequential loops	1 evaluation
Input Entropy	Variable (can include low-entropy passphrase)	256-bit pseudorandom chain code + key
Primary Vulnerability	Off-line dictionary guessing of passphrases	Exposure of extended keys
Frequency of Execution	Once per wallet setup/recovery	Repeated per transaction, account, and address

Hardware Wallet Performance Implications

Hardware signing devices use secure elements and general-purpose microcontrollers that operate at modest clock frequencies (often between 24 MHz and 120 MHz) to minimize power draw and physical attack surfaces.

A hardware device running a single round of HMAC-SHA512 takes mere microseconds to milliseconds. This allows the device to rapidly compute an address at path $m/44'/0'/0'/0/5$ by calculating five sequential HMAC operations and corresponding elliptic curve additions in a fraction of a second.

If child derivation required PBKDF2 stretching at each depth level, that same path would require $5 \times 2048 = 10,240$ HMAC evaluations. On resource-constrained hardware, this would introduce several seconds of latency for every single transaction signing request, without enhancing resistance against key recovery.

FAQ

Why does BIP39 use 2048 iterations instead of a higher count like modern password hashes?

BIP39 was finalized when hardware wallets possessed very limited processing power. A count of 2048 was chosen as a compromise between providing basic protection against low-effort passphrase guessing and allowing low-power microcontrollers to generate seeds in under two seconds.

Does the BIP39 iteration count protect the 12 or 24 mnemonic words themselves?

No. A standard 12-word phrase contains 128 bits of entropy and a 24-word phrase contains 256 bits. An adversary cannot brute-force a fully random 128-bit space regardless of iteration count. The 2048 PBKDF2 rounds specifically protect against guessing low-entropy, human-created passphrases appended to the phrase.

Could BIP32 be made more secure by adding key stretching to child derivation?

No. Key stretching only protects low-entropy inputs like passwords. Because BIP32 keys and chain codes are 256-bit cryptographically random values, stretching provides no mathematical security improvement while drastically slowing down address derivation and wallet synchronization.

Sources

- [BIP 32: Hierarchical Deterministic Wallets \(github.com\)](#)
- [BIP 39: Mnemonic code for generating deterministic keys \(github.com\)](#)
- [RFC 2898: PKCS #5: Password-Based Cryptography Specification Version 2.0 \(www.rfc-editor.org\)](#)

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	horn	2	neither
3	situate	4	payment
5	vacuum	6	hat
7	sketch	8	all
9	green	10	pistol
11	artist	12	birth

Example 2			
1	mercy	2	below
3	drum	4	benefit
5	pizza	6	special
7	knock	8	minimum
9	online	10	swamp
11	general	12	sketch

Example 3			
1	double	2	enrich
3	pen	4	illegal
5	mom	6	welcome
7	mean	8	when
9	power	10	file
11	arrive	12	album

Example 4

1 jar	2 swift	3 pencil	4 blind
5 soup	6 expand	7 aerobic	8 rural
9 spend	10 very	11 uncover	12 entry

Example 5

1 own	2 hand	3 okay	4 pudding
5 column	6 pudding	7 unfair	8 charge
9 mango	10 draft	11 antenna	12 monkey

Example 6

1 couple	2 cabin	3 churn	4 carbon
5 alarm	6 sun	7 help	8 loop
9 flower	10 purity	11 cousin	12 sound

Example 7

1 gas	2 dose	3 hazard	4 exotic
5 wrong	6 neck	7 seven	8 stay
9 spoil	10 danger	11 glimpse	12 situate

Example 8

1 want	2 autumn	3 enroll	4 rose
5 blossom	6 depth	7 spin	8 cable
9 brand	10 mom	11 debate	12 bean

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	horn	876	877	01101101100
2	neither	1185	1186	10010100001
3	situate	1613	1614	11001001101
4	payment	1293	1294	10100001101
5	vacuum	1924	1925	11110000100
6	hat	844	845	01101001100
7	sketch	1617	1618	11001010001
8	all	51	52	00000110011

#	Word	Index (0-2047)	Word no.	11 bits
9	green	818	819	01100110010
10	pistol	1323	1324	10100101011
11	artist	104	105	00001101000
12	birth	181	182	00010110101

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): 6d928726d0df08d33288336654ac340b

This sequence ends in the checksum bits 0101, but SHA-256 of its entropy starts with 0100, 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

- [Watch-Only Wallets: Tracking Funds Without Private Keys](#) · October 18, 2026
- [Mnemonic Seed Formats That Differ From BIP39](#) · October 21, 2026
- [Compiling Policies with rust-miniscript](#) · October 20, 2026
- [Why There Is No BIP39 Version 2](#) · October 16, 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-phrases.com/security/