



Dice Check – did the SeedSigner compute correctly?

Three ways to recompute dice mode independently · belongs with SeedSigner guide 1b and dice log sheet 1b · Chapter 3

⚠ **What is tested is the method, never the real seed.** The dice digits are the seed – whoever types them into a computer has typed the seed into a computer. So: roll a **throwaway seed**, recompute that one, then discard it. If the device computes practice digits correctly, it computes correctly every time. The productive seed is created on the device afterwards and never leaves it.

⚠ **The checking tool is attack surface too.** A manipulated check page could say "correct" where nothing is correct – or read the digits along. Therefore: compare the files against the checksums below before use, work offline, and when it matters, run two independent tools against each other.

WHAT YOU NEED

- ☐ One throwaway run on the SeedSigner: **write the digits down as you type them** (dice log sheet 1b) and note the words shown – with a practice seed that is allowed.
- ☐ From the workshop package: `dice-check.html` (way 1) and/or `verify_dice_seed.py` + `bip39_english.txt` + `SHA256SUMS` + `dice-check.sh` (way 2). Optionally Ian Coleman's `bip39-standalone.html` (way 3).
- ☐ A computer you can **disconnect from the network** – or a Tails stick (a bonus, not a must).

WAY 1 – DICE CHECK IN THE BROWSER (THE SIMPLEST)

A single HTML file, no server, no connection. Computes SHA-256 with two engines (the browser's WebCrypto + its own implementation), checks the built-in word list by hash and runs a self-test against the examples in the SeedSigner documentation on opening.

- 1 ☐ Save the file from the download page (right click → "Save link as"). **Compare the checksum** (below), value matches: _____
- 2 ☐ **Disconnect** (Wi-Fi off, cable out). Open the file by double-clicking – it runs from the file system, and the Content Security Policy forbids the browser to load anything.
- 3 ☐ Read the status line: **Connection: offline · Word list ... OK · SHA-256: engines agree · Self-test: 24 + 12 words OK**. If one of the four is red, do not use the file.
- 4 ☐ Enter the digits (99 for 24 words, 50 for 12, spaces are fine), optionally the words from the display below them, then **"Compute"**. Green = all words identical. Red = the differing words are marked.
- 5 ☐ **"Clear everything"**, close the browser window. For skeptics: F12 → network tab shows not a single request, not even while computing.

WAY 2 – CHECK SCRIPT IN THE TERMINAL (LINUX, MACOS, TAILS)

Around 90 lines of Python, standard library only plus the official word list. Before every start the wrapper `dice-check.sh` checks the hashes of both files against `SHA256SUMS` and only then launches.

- 1 ☐ Put the four files in one folder, check the hashes against the table below: `sha256sum -c SHA256SUMS` → "OK" twice. On Tails: create the folder in the persistent storage so the files survive a restart.
- 2 ☐ Self-test: `bash dice-check.sh --selftest` → the script answers in German: "Selbsttest 99 Würfe / 24 Wörter: OK" and "... 50 Würfe / 12 Wörter: OK" (self-test 99 rolls / 24 words, 50 rolls / 12 words).
- 3 ☐ Disconnect. `bash dice-check.sh` **without an argument** – the script asks for the digits. Never pass them as an argument, otherwise they end up in the shell history.
- 4 ☐ Compare the words printed with the display ("Seed Words: 1/6" to "6/6"). Afterwards `clear` or close the terminal – otherwise the words stay in the scrollbar.

Way 3 – second code base (Ian Coleman): load `bip39-standalone.html` from the official release, open it offline → "Show entropy details" → actively click **"Base 10"** → set **"Mnemonic Length" to "24 Words"** (or "12 Words", if "Use Raw Entropy" stays selected nothing is hashed and the words differ) → enter the digits. **Never the "Dice" mode** (it turns every 6 into a 0). The SeedSigner adopted Coleman's "Base 10" method (digits as text, SHA-256), which is why it matches the device.

Why three ways? A tool can be wrong or manipulated. Two independent code bases (browser JavaScript and Python, or one of them plus Coleman) arriving at the same words as the device are proof. One tool alone is an indication. **What it does not prove:** whether your dice are fair and whether your backup is readable – that is what the dice cup and the restore drill are for.

CHECKSUMS (SHA-256) – COMPARE AGAINST THE DOWNLOAD PAGE

<code>bip39_english.txt</code>	2f5eed53a4727b4bf8880d8f3f199efc90e58503646d9ff8eff3a2ed3b24dbda (official BIP39 list, identical with the Bitcoin BIPs repository)
<code>verify_dice_seed.py</code>	96bcf6a2062f5035ff9a3f2eeda455122db7f899d77a7d461f2f78c5e81f9b27
<code>dice-check.html</code>	see the download page (the file cannot contain its own hash) – enter the value here:
Check command	Linux/Tails/macOS: <code>sha256sum FILE</code> · Windows PowerShell: <code>Get-FileHash FILE</code>

RESULT

- ☐ **Passed:** tool(s) and device deliver the same words → the SeedSigner computes to the standard. Discard the practice seed, roll the real seed on the device, do not type those digits anywhere again.
- ☐ **Mismatch:** the most common cause is a typo between sheet and device. Check the digits, repeat the run. If two tools agree with each other but differ from the device: do not use the device, check the firmware version and where it came from.