---

title: Web3 secret storage definition description: Formal definition for web3 secret storage lang: en sidebarDepth: 2

To make your app work on Ethereum, you can use the web3 object provided by the web3.js library. Under the hood it communicates to a local node through RPC calls. web3 works with any Ethereum node which exposes an RPC layer.

`web3` contains the `eth` object - web3.eth.

```js var fs = require("fs") var recognizer = require("ethereum-keyfile-recognizer")

fs.readFile("keyfile.json", (err, data) => { var json = JSON.parse(data) var result = recognizer(json) })

/* *result * [ 'web3', 3 ] web3 (v3) keyfile * [ 'ethersale', undefined ] Ethersale keyfile * null invalid keyfile* / ```

This documents **version 3** of the Web3 Secret Storage Definition.

# Definition {#definition}

The actual encoding and decoding of the file remains largely unchanged from version 1, except that the crypto algorithm is no longer fixed to AES-128-CBC (AES-128-CTR is now the minimal requirement). Most of the meanings/algorithm are similar to version 1, except `mac`, which is given as the SHA3 (keccak-256) of the concatenations of the second-leftmost 16 bytes of the derived key together with the full `ciphertext`.

Secret key files are stored directly in `~/.web3/keystore` (for Unix-like systems) and `~/AppData/Web3/keystore` (for Windows). They may be named anything, but a good convention is `&lt;uuid&gt;.json`, where `&lt;uuid&gt;` is the 128-bit UUID given to the secret key (a privacy-preserving proxy for the secret key's address).

All such files have an associated password. To derive a given `.json` file's secret key, first derive the file's encryption key; this is done through taking the file's password and passing it through a key derivation function as described by the `kdf` key. KDF-dependent static and dynamic parameters to the KDF function are described in `kdfparams` key.

PBKDF2 must be supported by all minimally-compliant implementations, denoted though:

- `kdf`: `pbkdf2`

For PBKDF2, the kdfparams include:

- `prf`: Must be `hmac-sha256` (may be extended in the future);
- `c`: number of iterations;
- `salt`: salt passed to PBKDF;
- `dklen`: length for the derived key. Must be >= 32.

Once the file's key has been derived, it should be verified through the derivation of the MAC. The MAC should be calculated as the SHA3 (keccak-256) hash of the byte array formed as the concatenations of the second-leftmost 16 bytes of the derived key with the `ciphertext` key's contents, i.e.:

`js KECCAK(DK[16..31] ++ <ciphertext>)`

(where `++` is the concatenation operator)

This value should be compared to the contents of the `mac` key; if they are different, an alternative password should be requested (or the operation cancelled).

After the file's key has been verified, the cipher text (the `ciphertext` key in the file) may be decrypted using the symmetric encryption algorithm specified by the `cipher` key and parameterised through the `cipherparams` key. If the derived key size and the algorithm's key size are mismatched, the zero padded, rightmost bytes of the derived key should be used as the key to the algorithm.

All minimally-compliant implementations must support the AES-128-CTR algorithm, denoted through:
```

- `cipher: aes-128-ctr`

This cipher takes the following parameters, given as keys to the cipherparams key:

- `iv`: 128-bit initialisation vector for the cipher.

The key for the cipher is the leftmost 16 bytes of the derived key, i.e.`DK[0..15]`

The creation/encryption of a secret key should be essentially the reverse of these instructions. Make sure the`uuid`, `salt` and `iv` are actually random.

In addition to the `version` field, which should act as a "hard" identifier of version, implementations may also use `minorversion` to track smaller, non-breaking changes to the format.

Test Vectors {#test-vectors}

Details:

- `Address`: `008aeeda4d805471df9b2a5b0f38a0c3bcba786b`
- `ICAP`: `XE542A5PZHH8PYIZUBEJEO0MFWRAPPIL67`
- `UUID`: `3198bc9c-6672-5ab3-d9954942343ae5b6`
- `Password`: `testpassword`
- `Secret`: `7a28b5ba57c53603b0b07b56bba752f7784bf506fa95edc395f5cf6c7514fe9d`

PBKDF2-SHA-256 {#PBKDF2-SHA-256}

Test vector using `AES-128-CTR` and `PBKDF2-SHA-256`:

File contents of `~/.web3/keystore/3198bc9c-6672-5ab3-d9954942343ae5b6.json`:


```
json { "crypto": { "cipher": "aes-128-ctr", "cipherparams": { "iv": "6087dab2f9fdbbfaddc31a909735c1e6" }, "ciphertext": "5318b4d5bcd28de64ee5559e671353e16f075ecae9f99c7a79a38af5f869aa46", "kdf": "pbkdf2", "kdfparams": { "c": 262144, "dklen": 32, "prf": "hmac-sha256", "salt": "ae3cd4e7013836a3df6bd7241b12db061dbe2c6785853cce422d148a624ce0bd" }, "mac": "517ead924a9d0dc3124507e3393d175ce3ff7c1e96529c6c555ce9e51205e9b2" }, "id": "3198bc9c-6672-5ab3-d995-4942343ae5b6", "version": 3 }
```


Intermediates:

`Derived key`: `f06d69cdc7da0faffb1008270bca38f5e31891a3a773950e6d0fea48a7188551` `MAC Body`: `e31891a3a773950e6d0fea48a71885515318b4d5bcd28de64ee5559e671353e16f075ecae9f99c7a79a38af5f869aa46` `MAC`: `517ead924a9d0dc3124507e3393d175ce3ff7c1e96529c6c555ce9e51205e9b2` `Cipher key`: `f06d69cdc7da0faffb1008270bca38f5`

Scrypt {#scrypt}

Test vector using AES-128-CTR and Scrypt:


```
json { "crypto": { "cipher": "aes-128-ctr", "cipherparams": { "iv": "740770fce12ce862af21264dab25f1da" }, "ciphertext": "dd8a1132cf57db67c038c6763afe2cbe6ea1949a86abc5843f8ca656ebbb1ea2", "kdf": "scrypt", "kdfparams": { "dklen": 32, "n": 262144, "p": 1, "r": 8, "salt": "25710c2ccd7c610b24d068af83b959b7a0e5f40641f0c82daeb1345766191034" }, "mac": "337aeb86505d2d0bb620effe57f18381377d67d76dac1090626aa5cd20886a7c" }, "id": "3198bc9c-6672-5ab3-d995-4942343ae5b6", "version": 3 }
```


Intermediates:

`Derived key`: `7446f59ecc301d2d79bc3302650d8a5cedc185ccbb4bf3ca1ebd2c163eaa6c2d` `MAC Body`: `edc185ccbb4bf3ca1ebd2c163eaa6c2ddd8a1132cf57db67c038c6763afe2cbe6ea1949a86abc5843f8ca656ebbb1ea2` `MAC`: `337aeb86505d2d0bb620effe57f18381377d67d76dac1090626aa5cd20886a7c` `Cipher key`: `7446f59ecc301d2d79bc3302650d8a5c`

Alterations from Version 1 {#alterations-from-v2}

This version fixes several inconsistencies with the version 1 published[here](). In brief these are:

- Capitalisation is unjustified and inconsistent (scrypt lowercase, Kdf mixed-case, MAC uppercase).
- Address unnecessary and compromises privacy.
- `Salt` is intrinsically a parameter of the key derivation function and deserves to be associated with it, not with the crypto in general.
- *SaltLen* unnecessary (just derive it from Salt).
- The key derivation function is given, yet the crypto algorithm is hard specified.
- `Version` is intrinsically numeric yet is a string (structured versioning would be possible with a string, but can be considered out of scope for a rarely changing configuration file format).
- `KDF` and `cipher` are notionally sibling concepts yet are organised differently.
- `MAC` is calculated through a whitespace agnostic piece of data(!)

Changes have been made to the format to give the following file, functionally equivalent to the example given on the previously linked page:

```
json { "crypto": { "cipher": "aes-128-cbc", "ciphertext":
"07533e172414bfa50e99dba4a0ce603f654ebfa1ff46277c3e0c577fdc87f6bb4e4fe16c5a94ce6ce14cfa069821ef9b",
"cipherparams": { "iv": "16d67ba0ce5a339ff2f07951253e6ba8" }, "kdf": "scrypt", "kdfparams": { "dklen": 32, "n":
262144, "p": 1, "r": 8, "salt": "06870e5e6a24e183a5c807bd1c43afd86d573f7db303ff4853d135cd0fd3fe91" }, "mac":
"8ccded24da2e99a11d48cda146f9cc8213eb423e2ea0d8427f41c3be414424dd", "version": 1 }, "id": "0498f19a-59db-4d54-
ac95-33901b4f1870", "version": 2 }
```

Alterations from Version 2 {#alterations-from-v2}

Version 2 was an early C++ implementation with a number of bugs. All essentials remain unchanged from it.