Learning Ethereum

Wallets

fausto.spoto@univr.it

https://bitbucket.org/spoto/learning-ethereum

Wallets

A wallet is a software application that keeps track of keys. It creates and broadcasts transactions signed with those keys.

Public key cryptography

Private and public keys come in pairs. Ethereum uses private keys to create digital signatures for transaction authentication. Hence EOAs are controlled by a private key.

Data are not natively encrypted in Ethereum!

Private/public keys from random numbers

Private key

In principle, just a random sequence of 256 bits.

Cryptographically-secure random generators

Private keys should be generated by using a cryptographically-secure random generator, or otherwise keys might not really be randomly spread and might be more easily guessed.

Public key

It is computed from the private key, through a one-way function called *elliptic curve multiplication* (ECDSA).

The generation of a private/public key pair does not require any centralized service. The probability of computing an already used key is negligeable.

Ethereum representation of the keys

- A private key are 256 random bits: 64 hexadecimal digits
- A public key is a couple of two 256 bit numbers: Ethereum uses `0x04` followed by 64 × 2 hexadecimal digits

There are libraries for computing public keys from private keys:

- OpenSSL (`https://www.openssl.org`)
- libsecp256k1 (`https://github.com/bitcoin-core/secp256k1`)

Cryptographic hash functions

Hash functions

Any function that can be used to map data of arbitrary size to data of fixed size.

Cryptographic hash function

A hash function with the following properties:

1. determinism
2. verifiability (in linear time)
3. noncorrelation (small change of input induces extensive change of hash)
4. irreversibility (*one-way*)
5. collision protection: difficult to compute two inputs that produce the same hash

Ethereum's cryptographic hash function

Keccak-256

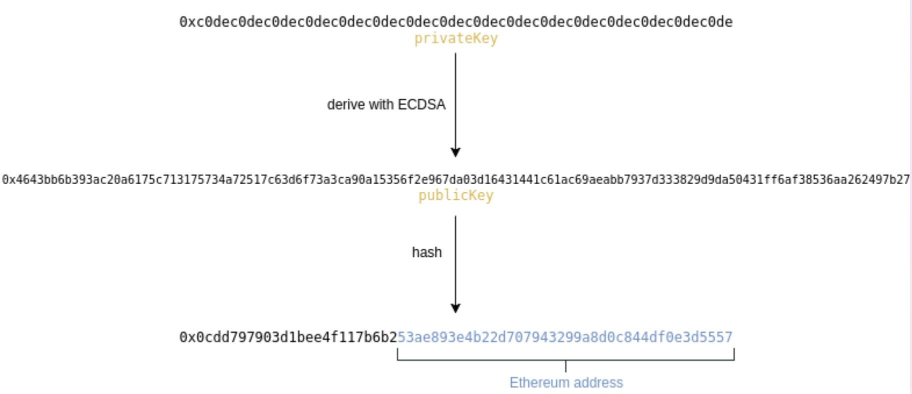
Ethereum uses the Keccak-256 cryptographic hash function. It is the original algorithm that won the SHA-3 Cryptographic Hash Function Competition of 2007. NIST standardized it in a slightly modified way as SHA-3. However, the Ethereum team never trusted such modification and used the original agorithm.

Keccak-256 or SHA-3?

You can still see blogs and even the source code of Ethereum refer to SHA-3. Despite of that, Etherem does not use SHA-3, but the original Keccak-256 algorithm!

Derivation of an Ethereum address from the private key

Compute the public key (without leading `0x04`), hash it with Keccak-256, keep only the last 20 bytes (160 bits).



0xc0dec0dec0dec0dec0dec0dec0dec0dec0dec0dec0dec0dec0dec0dec0dec0de

privateKey

derive with ECDSA

0x4643bb6b393ac20a6175c713175734a72517c63d6f73a3ca90a15356f2e967da03d16431441c61ac69aeabb7937d333829d9da50431ff6af38536aa262497b27

publicKey

hash

0x0cdd797903d1bee4f117b6b253ae893e4b22d707943299a8d0c844df0e3d5557

Ethereum address

No checksum!

Checksummed address format: ICAP

`XE` + 2 characters of checksum + base-36 integer of up to 30 digits

- the base-36 integer can represent up to 155 bits, hence ICAP can only be used for addresses starting with a zero byte
- ICAP is compatible with IBAN

Example

Ethereum address: `0x001d3f1ef827552ae1114027bd3ecf1f086ba0f9`

ICAP: `XE60 HAMI CDXS V5QX VJA7 TJW4 7Q9C HWKJ D`

Hex encoding with checksum in capitalization (EIP-55)

A backward compatible checksum injection

1. use Keccak-256 to compute 64 hex digits from the uncapitalized Ethereum address (40 hex digits)
2. capitalize each alphabetic address character if the corresponding hex digit of the hash is greater than or equal to `0x8`

Example

```
Address:    001d3f1ef827552ae1114027bd3ecf1f086ba0f9
Keccak256:  23a69c1653e4ebbb619b0b2cb8a9bad49892a8b9...
Result:     001d3F1ef827552Ae111402B7D3ECF1f086bA0F9
```

Checking procedure of an EIP-55 encoded address

- apply the above capitalization procedure to it
- the result must match the EIP-55 encoded address