Writing Smart Contracts
03 Accounts

Peter H. Gruber

Supported by the Algorand Foundation

Algorand Adesses

(1) Private key

- A very long number ...
- 256 Bit $= 2^{256} \approx 10^{77}$ different possibilities
- "Master password to account", "Single Factor Authentication"

(2) Mnemonic = representation of private key

- 25 words out of a list of $2048 = 2^{11}$ words
- 1 word = 11 Bits
- 24 words $= 264 > 256$ Bits
- Algorand uses 25th word as checksum

(3) Address = public key

- Hash of private key
- Algorand: 256 Bit + 32 Bit Checksum
- Easy: private → public
- (Almost) impossible: public → private

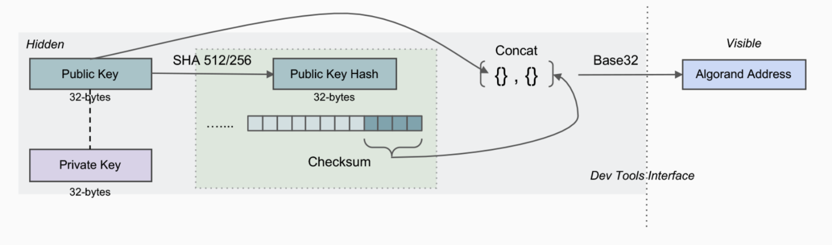
(4) Wallet = collection of keys

Public Key ∼ Address

From public key to address

- Public key = 256 Bit
- Add hash of 32 Bit length (4 Bytes)
- Encode as numbers/letters for readability (Base 32)
- 58 numbers/letters, 5 Bytes each = 290 Bits > 256+32

```
N72FLVBF2PW6SKXNDW6JLZT5WUACHGIDVZI3OPUCK2ALFUHO3KURCNRODE
```



Public Key to Algorand Address

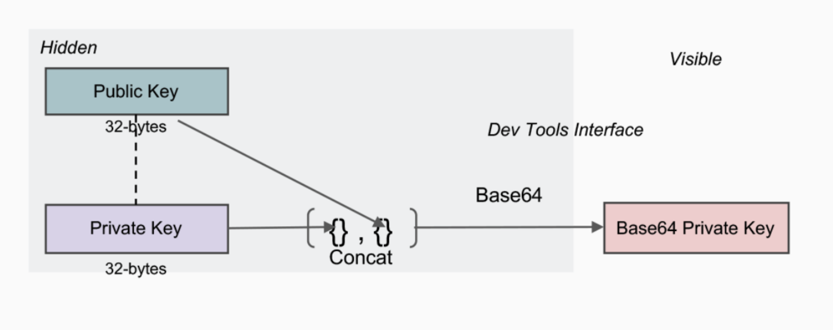
Private Key

Transformations

- Store Public and Private Keys
- Encode as numbers/letters for readability (Base 64)
- 80 numbers/letters, 6 Bytes each = 480 Bits
- For developpers only

```
VwrmAkisLya/0H+HALB13XRpLNGfkoMY4mgUXYL6FURv

9FXUJdPt6Srt HbyV5n21AC0ZA65Rtz6CVoCy007aqQ==
```



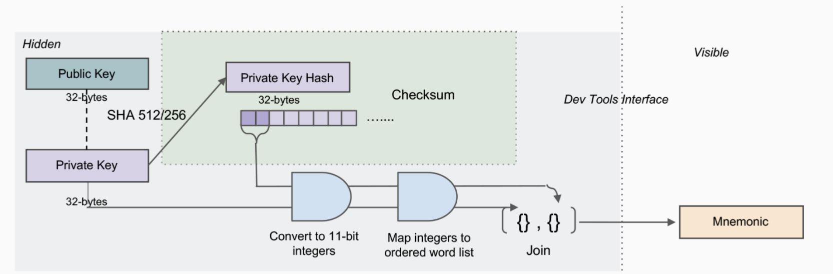
Base64 Private Key

Passphrase = Mnemonic

Representation of Private Key

- Encode 256Bit key as word sequence
- List of $2^{11} = 2048$ words
- Position of word in alphabetic list represents 11 Bit number
- 25 words, 11 Bits each = 275 Bits
- For end users

```
enough oblige accident setup gap sister magnet lemon axis scale river
evidence spray enrich write myth away mask crucial spend again leaf camera
able athlete
```



An Algorand transaction

```
{
  "txn": {
    "amt": 5000000,
    "fee": 1000,
    "fv": 6000000,
    "gen": "mainnet-v1.0",
    "gh": "wGHE2Pwdvd7S12BL5FaOP20EGYesN73ktiC1qzkkit8=",
    "lv": 6001000,
    "note": "SGVsbG8gV29ybGQ=",
    "rcv": "GD64YIY3TWGDMCNPP553DZPPR6LDUSFQOIJVFDPPXWEG3FVOJCCDBBHU5A",
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCCLIHZU6TBEOC7XRSBG4",
    "type": "pay"
  }
}
```

Life of a transaction

(1) Setup

- Create transaction in Python or (web) app
- Transaction is not yet signed

(2) Sign

- Sender uses private key to sign transaction
- Signature is added in `"sig"` field

(3) Submit

- Send to the blockchain via API or your own indexer

(4) Get accepted

- Relay nodes verify signature (using the public key of the sender)
- Consensus decides if a transaction is included in the next block

Accessing the blockchain

Where is the Algorand chain?

- On approx. 1600 relay nodes (Nov 2022) – one of them at USI
- Up-to-date: `https://algoscan.app`

How large is the Algorand Chain?

- Approx. 1010GB (Nov 2022)
- Up-to-date: `https://howbigisalgorand.com`

How can we access the chain?

- Set up our own indexer node
- Access via API, e.g.?`https://www.purestake.io`
- Explore using `https://algoscan.app` or `https://algoexplorer.io`

Python commands

Transactions

- Local
  1. Prepare/create transaction → `txn`
  2. Sign transaction → `stxn`
- On Chain
  3. Send transaction → `txid`
  4. Verify transaction → `txinfo`

Accounts

- Local
  - Create key pair
- On Chain
  - Get account balance