TD2 - HD Wallet Programming

Lien du repository github: https://github.com/Pibastte/BlockChainProgramming-BIP39-32.git

Choix du langage

Pour ce projet j'ai choisi d'utiliser le langage javascript. Ce langage est prédominant aujourd'hui car c'est le langage utilisé par tous les sites et navigateurs web. Etant intéressé par l'application des technologies blockchain dans le web, j'ai voulu développer mes compétences en javascript.

Difficultées rencontrées

J'ai rencontré beaucoup de challenge en utilisant javascript. En effet il y'a sensiblement moins de documentation par rapport à python. J'ai trouvé peu de contenu qui n'utilisaient pas de dépendances.

J'ai eu des difficultées à implémenter la serisalisation des clées en format xpub ou xprv qui est un format plus standard que l'hexadécimal. J'ai cependant pu l'implémenter en utilisant cette section de la documentation officielle.

Serialization format

Extended public and private keys are serialized as follows:

- 4 byte: version bytes (mainnet: 0x0488B21E public, 0x0488ADE4 private; testnet: 0x043587CF public, 0x04358394 private)
- 1 byte: depth: 0x00 for master nodes, 0x01 for level-1 derived keys, ...
- 4 bytes: the fingerprint of the parent's key (0x00000000 if master key)
- 4 bytes: child number. This is $ser_{32}(i)$ for i in $x_i = x_{par}/i$, with x_i the key being serialized. (0x00000000 if master key)
- 32 bytes: the chain code
- 33 bytes: the public key or private key data ($ser_P(K)$ for public keys, 0x00 || $ser_{256}(k)$ for private keys)

This 78 byte structure can be encoded like other Bitcoin data in Base58, by first adding 32 checksum bits (derived from the double SHA-256 checksum), and then converting to the Base58 representation. This results in a Base58-encoded string of up to 112 characters. Because of the choice of the version bytes, the Base58 representation will start with "xprv" or "xpub" on mainnet, "tprv" or "tpub" on testnet.

Note that the fingerprint of the parent only serves as a fast way to detect parent and child nodes in software, and software must be willing to deal with collisions. Internally, the full 160-bit identifier could be used.

When importing a serialized extended public key, implementations must verify whether the X coordinate in the public key data corresponds to a point on the curve. If not, the extended public key is invalid.

J'ai également eu des difficultées avec la partie dérivation car elle implique des modulos. Javascript utilise par défaut des int 32 qui ne permettent pas de faire des modulos sur des entiers de grande taille. J'ai pu débloquer la situation en utilisant des BigInt.

Sources

J'ai été beaucoup aidé de la documentation officielle bip-32 (https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki), par cet article que j'ai trouvé tard mais qui explique très bien chaque étape (https://bitcointalk.org/index.php?topic=5316005.0). J'ai également pu tester mon code grace au site iancoleman (https://iancoleman.io/bip39/).