 Date: 08 May 2019

Acknowledgement

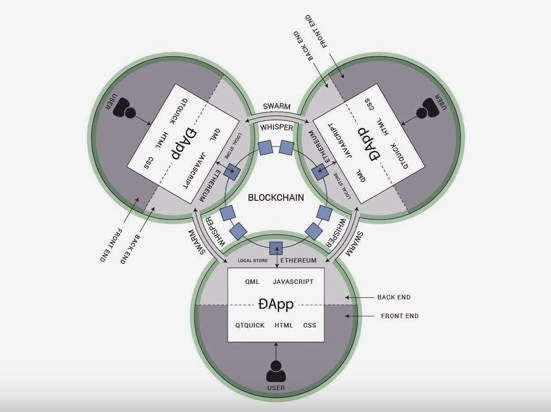
We gratefully acknowledge for the assistance, cooperation, guidance and clarifications provided during the development of the **Unicrypto : Bitcoin Wallet** website. My extreme gratitude to **Mr. Rahul Rastogi** who guided methroughout the project. Without his willing disposition, spirit of accommodation,frankness, timely clarification and above all faith in us, this project could not havebeen completed in due time.His readiness to discuss all important matters at work deserves special attention.

About Unicrypto

Unicrypto is a Decentralized Client Side Bitcoin wallet service:

Decentralized applications (dApps) are applications that run on a P2P network of computers rather than a single computer. dApps, have existed since the advent of P2P networks. They are a type of software program designed to exist on the Internet in a way that is not controlled by any single entity.

* Decentralized applications don’t necessarily need to run on top of a blockchain network. BitTorrent, Popcorn Time, BitMessage, Tor, are all traditional dApps that run on a P2P network, but not on a Blockchain (which is a specific kind of P2P network).
* As opposed to simple smart contracts, in the classic sense of Bitcoin, that sends money from A to B, dApps have an unlimited number of participants on all sides of the market.



### What is bitcoin?

Bitcoin is a digital currency already used all over the world. Unlike other currencies like Dollar and Euro, bitcoin does not exist in paper form and exist only digitally. It can be used as an investment opportunity or to purchase any goods similar to other currencies.

### Why use bitcoin?

Bitcoin price has been rising since its creation and is gaining popularity which in further increases it’s value. Bitcoins are becoming a mainstream currency which is not controlled by the government, therefore, makes you the true owner of this asset.

Process

To start, we’ll create a wallet. Bitcoin wallets are not stored in the blockchain. Rather, they are managed by each individual user and referenced in individual transactions. A wallet consists of the following parts, which are generated in the same order:

* Private key
* Public key
* Public key hash (PKH)
* Public address
* Private key WIF (wallet import format)

**Wallets Are Not Online**

An important concept to understand is that you do **not**need to be online to generate a Bitcoin wallet address. Because the methods to generate an address rely on math, there is no need to connect to a separate server, website or other service.

Let’s look at the code to generate a private key:

Api : SecureRandom/coinjs

let privateKey = secureRandom.randomBuffer(32);  
console.log('> Private key created: ', privateKey.toString('hex'));

Output : > Private key created: 66bbce88a18fb9170b89d372e9ec41e3dcf50e24ef8d8ab2c3bbe62571788e10

Now, what is it? Well, you might have been able to tell that it is a **32-byte**array in binary. Since there are 8 bits in a byte, that makes 256 bits.

If you want to know the probability of someone else generating the same private key as you, then it is 2²⁵⁶, which is an astronomically² large number

#### 2²⁵⁶, but not quite

#### Not every private key like this is valid. Only private keys that are less than the following value (in hexadecimal) work with Bitcoin: 0xFFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFE BAAE DCE6 AF48 A03B BFD2 5E8C D036 4140.

This is because Bitcoin uses elliptic curve cryptography (ECC) and can only accept private keys below that number. In case that wasn’t enough, the version of elliptic curve cryptography that Bitcoin uses is called secp256k

Our Revised Code :

const max = Buffer.from("0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364140", 'hex');   
let isInvalid = true;   
let privateKey;   
while (isInvalid) {   
 privateKey = secureRandom.randomBuffer(32);  
 if (Buffer.compare(max, privateKey) === 1) {   
 isInvalid = false;   
 }   
}  
console.log('> Private key: ', privateKey.toString('hex'));

Output : > Private key created: 9221592127fcd40dfbc3dce017bcee457e07555b1cf127987125c6de1de7926b

*Elliptic curve cryptography is a type of asymmetric or public key cryptography based on the discrete logarithm problem as expressed by addition and multiplication on the points of an elliptic curve.*

Now we’ll make use of the elliptic JavaScript library to generate our public key:

const keys = ecdsa.keyFromPrivate(privateKey);   
const publicKey = keys.getPublic('hex');   
console.log('> Public key created: ', publicKey);

const sha256 = require('js-sha256');  
const ripemd160 = require('ripemd160');

let hash = sha256(Buffer.from(msg, 'hex'));  
let publicKeyHash = new ripemd160().update(Buffer.from(hash, 'hex')).digest();

function createPublicAddress(publicKeyHash) {  
 // step 1 - add prefix "00" in hex  
 const step1 = Buffer.from("00" + publicKeyHash, 'hex');  
 // step 2 - create SHA256 hash of step 1  
 const step2 = sha256(step1);  
 // step 3 - create SHA256 hash of step 2  
 const step3 = sha256(Buffer.from(step2, 'hex'));  
 // step 4 - find the 1st byte of step 3 - save as "checksum"  
 const checksum = step3.substring(0, 8);  
 // step 5 - add step 1 + checksum  
 const step4 = step1.toString('hex') + checksum;  
 // return base 58 encoding of step 5  
 const address = base58.encode(Buffer.from(step4, 'hex'));  
 return address;  
}

Output : Public Address : 15pDZZny3jmbiJiTe4vcGg5aoWWsyJtiSg

Generating Private Key WIF

WIF stands for “wallet import format.” It was a standard introduced to make it easier and more secure for users to migrate wallets from different services.

The process of generating a private key WIF is not very different from generating a public address. Here are the steps:

* We add a prefix to the private key. In this case it is “80” (in hexadecimal)
* We derive the SHA-256 hash of the extended private key.
* We derive the SHA-256 hash of **that**, and then save the first byte as the **checksum**.
* We add the checksum to the extended private key and encode it to base 58

Code :

function createPrivateKeyWIF(privateKey) {  
 const step1 = Buffer.from("80" + privateKey, 'hex');  
 const step2 = sha256(step1);  
 const step3 = sha256(Buffer.from(step2, 'hex'));  
 const checksum = step3.substring(0, 8);  
 const step4 = step1.toString('hex') + checksum;  
 const privateKeyWIF = base58.encode(Buffer.from(step4, 'hex'));  
 return privateKeyWIF;  
}

Output:

Public Key : 15pDZZny3jmbiJiTe4vcGg5aoWWsyJtiSg

Private Key : 5Jga8XX1eZrXhi8abPyDhS89hUU9nfXv3AT38zk2whvuhbwP5ch