Introduction

The purpose of this section is to describe the algorithm with which the private key is calculated using the corresponding salt phrase, secret phrase, and currency type. The algorithm presented below is identical to the algorithm that is realized in Bitfi wallets. This same algorithm is realized in the private key genertion software found at the Bitfi Foundation website[1]. If you have any questions about the below content, please contact support@bitfi.com.

Algorithm

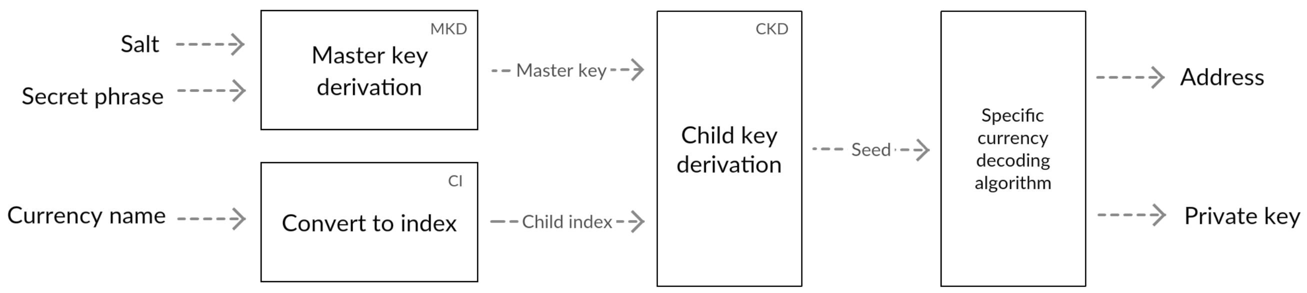

Figure 1: Bitfi key derivation algorithm.

The above diagram shows the algorithm at first glance. The given algorithm accepts textual data in the form of Salt, Secret Phrase, Currency name and returns Address and Private Key as the result. We will define functions MKD, CI, and CKD.

MKD(Master key derivation)

Before we can determine the function MKD, it is imperative to determine various assisting algorithms
Parameters:

H = A hash function
k = Length of output produced by H, in bits
$Integerify$ = A bijective function from $\{0,1\}^k \rightarrow \{0,...,2^k - 1\}$

Input:

B = Input of length k bits
N = Integer work metric $< 2^{k/8}$

Output:

B' = Output of length k bits

Algorithm 1 $ROMix_H(B, N)$

begin
  $X := B$
  for $i := 0$ to $N - 1$ do
    $V_i := X$
    $X := H(X)$
  end
  for $j := 0$ to $N - 1$ do
    $j := Intergify(X)\ mod\ N$
    $X := H(X \bigoplus V_j)$
  end
  $B' := X$
end

[1]https://www.btknox.org/calculate-your-private-keys

Parameters:

H = A hash function
r = blocksize parameter

Input:

$B_0 ... B_{2r-1}$ = Input vector of 2r k-bit blocks

Output:

$B_0^{'} ... B_{2r-1}^{'}$ = Output vector of 2r k-bit blocks

Algorithm 2 Algorithm $BlockMix_{H,r}(B)$

begin
  for $i := 0$ to $2r - 1$ do
    $X := H(X \bigoplus B_i)$
    $Y_i := X$
  end
  $B' := (Y_0, Y_2, ...Y_{2r-2}, Y_1, Y_3, ...Y_{2r-1})$
end

To determine function SMix_r

Definition 1. *The function* $SMix_r{}^2 : \{0,1\}^{1024r} \times \{0...2^{64} - 1\} \to \{0,1\}^{1024r}$ *is*

$$SMix_r(B, N) = ROMix_{BlockMix_{Salsa20/8,r}}(B, N)$$

where $Intergify(B_0, ...B_{2r-1})$ *is defined as the result of interpreting* B_{2r-1} *as a little-endian integer.*

Parameters:

PRF = A pseudorandom function
$hLen$ = Length of output produced by PRF, in octets
MF = A sequential memory-hard function from $\mathbb{Z}_{256}^{MFLen} \times \mathbb{N}$ *to* $\mathbb{Z}_{256}^{MFLen}$
$MFLen$ = Length of block mixed by MF, in octets.

Input:

P = Passphrase, an octet string
S = Salt, an octet string
N = CPU/memory cost parameter
p = Parallelization parameter; a positive integer satisfying $p \leq (2^{32} - 1)hLen/MFLen$
$dkLen$ = Intended output length in octets of the derived key; a positive integer satisfying $dkLen \leq (2^32 - 1)hLen$

Output:

DK = Derived key, of length dkLen octets

Algorithm 3 $MFcrypt_{H,MF}(P, S, N, p, dkLen)$

$(B_0...B_{p-1}) := PBKDF2_{PRF}(P, S, 1, p * MFLen)$
begin
  for $i := 0$ to $p - 1$ do
    $B_i := MF(B_i, N)$
  end
  $DK := PBKDF2_{PRF}(P, B_0 \parallel B_1 \parallel B_{p-1}, 1, dkLen)$
end

To determine function MKD

Definition 2. *The MKD function*[3] *is defined as*

$$MKD(P, S) = MFcrypt_{HMAC-SHA256, SMix_4}(P, S, 32768, 4, 64)$$

[2] refer https://en.wikipedia.org/wiki/Salsa20 for more details regarding Salsa20
[3] Permanent parameters were selected specifically for the Bitfi wallet

CKD(Child key derivation)

Before proceeding, we need to define some convention functions

- point(p): returns the coordinate pair resulting from EC point multiplication (repeated application of the EC group operation) of the secp256k1 base point with the integer p.
- $ser_{32}(i)$: serialize a 32-bit unsigned integer i as a 4-byte sequence, most significant byte first.
- $ser_{256}(p)$: serializes the integer p as a 32-byte sequence, most significant byte first.
- $ser_P(P)$: serializes the coordinate pair $P = (x, y)$ as a byte sequence using SEC1's compressed form: $(0x02or0x03) \parallel ser_{256}(x)$, where the header byte depends on the parity of the omitted y coordinate.
- $parse_{256}(p)$: interprets a 32-byte sequence as a 256-bit number, most significant byte first.

We represent an extended private key as (k, c), with k the normal private key, and c the chain code. An extended public key is represented as (K, c), with $K = point(k)$ and c the chain code.
Parameters:

$i =$ Derivation index

Input:

$(k_{par}, c_{par}) =$ Extended private parent key

Output:

$(k_i, c_i) =$ Extended private child key, corresponding to derivation index i

Algorithm 4 CKD

begin
  if $i \geq 2^{31}$ // *key is hardened*
    $I := HMAC - SHA512(Key = C_{par}, Data = 0x00 \parallel ser_{256}(k_{par}) \parallel ser_{32}(i)$
  end
  if $i < 2^{31}$
    $I := HMAC - SHA512(Key = C_{par}, Data = ser_p(paint(k_{par})) \parallel ser_{32}(i)$
  end
  $I_L := I.SUBARRAY(0, 32)$
  $I_R := I.SUBARRAY(32, 32)$
  if $parse_{256}(I_L) \geq n$ *or* $k_i = 0$
    The resulting key is invalid, and one should proceed with the next value for i
  end
  $k_i := parse_{256}(I_L) + k_{par}(mod\ n)$
  $c_i := I_R$
end

CI (Convert to index)

Function CI is simple and linear. It provides the line segment figure based on the algorithm below
Input:

$cn =$ Currency name

Output:

$index =$ Child index

Algorithm 5 CI

```
begin
    acc := ""
    for i := 0 to LENGTH(cn) do
        acc := acc + (ASCII − NUMBER(cn[i]) − 64).ToString()
    end
    index := Intergify(acc)
end
```

And finally, we can now determine the function of the Bitfi key derivation

Bitfi key derivation algorithm

Before proceeding, we need to define some convention functions

- GETBYTES: converts input string to byte array

Input:

P = Password phrase, type string
S = Salt phrase, type string
C = Currency name, type string

Output:

S = Seed key

Algorithm 6 Bitfi key derivation

```
begin
    P_bytes = GETBYTES(P)
    S_bytes = GETBYTES(S)
    k_par := MKD(P_bytes, S_bytes)
    c_par := SHA − 256(S_bytes)
    index := CI(C) || 0x80000000    //we want to make this index hardened
    S := CKD((k_par, c_par), index)
end
```

At exit, the algorithm puts out the appropriate seed, which can be used for generation of the public and private keys. The process of generation of the private and public keys is outside the scope of this article. For each cryptocurrency a specific and custom algorithm is used. One can learn more about this process through official resources.[4]

[4]For example, for cryptocurrency Monero, please visit: https://github.com/monero-project/monero