UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

Bitcoin wallets

Esame di Sicurezza Informatica

A cura di

Gianluca Scatigna

[Matricola 652750]

ANNO ACCADEMICO 2018

Sommario

Introduzione

Bitcoin è la prima valuta digitale decentralizzata che non è sostenuta da alcun governo o organizzazione.

Nonostante la mancanza di una organizzazione centrale che la presiede è in grado di consentire transazioni istantanee peer-to-peer, non ha bisogno di una terza parte fiduciaria per operare, basa la sicurezza su protocolli crittografici e quindi rappresenta low-cost banking per tutti, ovunque.

Bitcoin ha avuto i suoi precursori, a partire dal 1982 con Ecash, ma nessun progetto ha avuto successo. L'annuncio da parte di uno sconosciuto, Satoshi Nakamoto, avvenne il 31 ottobre 2008 scrivendo a una mailing list di crittografia e pubblicando "Bitcoin: A Peer-to-Peer Electronic Cash System".

Nel documento sosteneva di aver lavorato e creato Bitcoin ma soprattutto di aver risolto il problema della doppia spesa, definito fino ad allora irrisolvibile.

Il 3 gennaio 2009 fu minato il blocco genesi dando vita a Bitcoin e poco dopo venne rilasciato il suo codice sorgente.

In questo testo verranno analizzati i principi sulla generazione dei numeri pseudocasuali e le strategie utilizzate che li applicano. Inoltre tre tipi di test statistici che vengono applicati agli algoritmi per testarne la loro casualità.

Verranno descritti i diversi tipi di portafogli Bitcoin e le loro differenze. Successivamente l'intero processo di creazione di un portafoglio deterministico a partire dalla generazione del seed, la derivazione della chiave privata e di quella pubblica. A seguire il processo di derivazione dell'indirizzo bitcoin e la codificazione in base58.

Un'analisi del portafoglio deterministico gerarchico e delle sue caratteristiche uniche quali le chiavi estese e la derivazione delle chiavi figlie.

Infine viene proposta una soluzione per generare un Paper Wallet in maniera assolutamente sicura.

Principi sulla generazione di numeri pseudocasuali

La chiave privata bitcoin è una stringa numerica di 256 bit che soddisfa due requisiti: randomness e unpredictability.

Randomness

I seguenti due criteri sono utilizzati per affermare che una sequenza di numeri sia casuale:

- **Distribuzione uniforme:** la distribuzione dei bit in una sequenza dovrebbe essere uniforme cioè la frequenza di occorrenze di zero e uno dovrebbe essere approssimativamente uguale.

- **Indipendenza:** nessuna sotto sequenza della sequenza dovrebbe poter esser dedotta dalle altre.

Sebbene ci siano test per determinare che una sequenza di bit corrisponda a una particolare distribuzione come quella uniforme non ve n'è nessuno che provi l'indipendenza.

Piuttosto un certo numero di test può essere applicato per dimostrare che la sequenza non mostra indipendenza. Solitamente vengono quindi applicati questi test fin quando la fiducia che l'indipendenza esista non sia sufficientemente forte.

Unpredictability

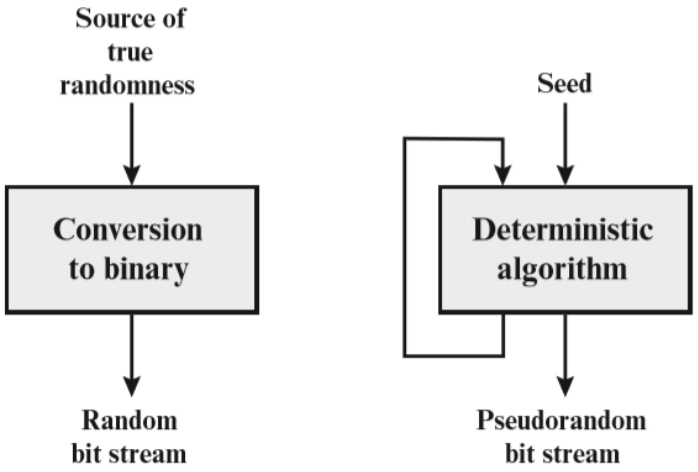
Un flusso di numeri pseudorandomici dovrebbe mostrare due forme di imprevedibilità:

- **Forward unpredictability:** se il seed è sconosciuto, il successivo bit della sequenza in output dovrebbe essere imprevedibile nonostante la conoscenza dei bit precedenti.

- **Backward unpredictability:** non dovrebbe essere possibile determinare il seed dalla conoscenza di un qualunque valore già generato.

Dunque non ci deve essere nessuna correlazione evidente tra il seed e i valori generati, ogni elemento della sequenza dovrebbe apparire come il risultato di un evento casuale la cui probabilità è di ½.

TRNG e PRNG



(a) TRNG (b) PRNG

TRNG = true random number generator
PRNG = pseudorandom number generator

Per generare sequenze casuali si fa uso di particolari tecniche algoritmiche distinte in **True random number generator (TRNG)** letteralmente generatore di numeri casuali veri e **Pseudorandom number generator (PRNG)** ossia generatore di numeri pseudocasuali.

Il TRNG ha come input una fonte non deterministica, cui si riferisce spesso come fonte d'entropia, per produrre la casualità. Il TRNG effettua una conversione binaria di processi naturali come rilevare gli impulsi di radiazioni ionizzanti o condensatori che perdono.

L'RFC 4086 lista le possibili sorgenti di casualità che possono essere utilizzate facilmente da un computer tra cui il rumore termico generato da una videocamera con la lente oscurata o le oscillazioni casuali nella velocità rotazionale degli hard drive disk dovuta alla caotica turbolenza d'aria.

Il PRNG è un algoritmo deterministico e dunque produce sequenze di numeri che non sono statisticamente casuali, tuttavia se è buono, le sequenze prodotte passeranno molti test di casualità.

Queste sequenze di numeri vengono chiamate numeri pseudocasuali.

Il PRNG ha come input un valore di lunghezza fissa chiamato seed che spesso viene generato da un TRNG. E' importante notare che il flusso di output è determinato solamente dal seed dunque un malintenzionato che conosce l'algoritmo e il seed può riprodurre l'intero flusso.

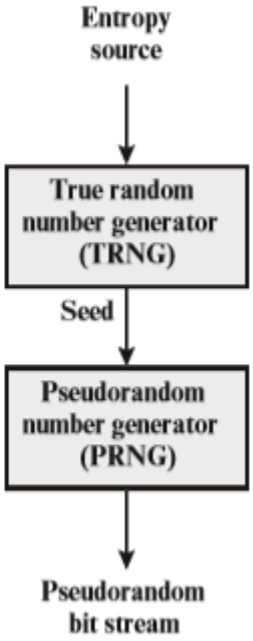


Figure 8.2 Generation of Seed Input to PRNG

Non esistono test per determinare se un PRNG genera numeri che abbiano le caratteristiche della casualità. Il meglio che si possa fare è di applicare più test al PRNG e se mostra la casualità sulla base di più test allora si considerano soddisfatti i suoi requisiti.

Il **NIST SP 800-22** specifica che i test dovrebbero cercare di determinare le seguenti tre caratteristiche:

- **Uniformity**: in qualsiasi punto nella generazione di una sequenza casuale o pseudocasuale, l'occorrenza di zero e uno è n/2, dove n è la lunghezza della sequenza.
- **Scalability**: qualsiasi test applicabile alla sequenza deve poter essere applicato a qualsiasi sotto sequenza. Se la sequenza è casuale allora qualsiasi sotto sequenza estratta dovrebbe essere casuale.
- **Consistency**: il comportamento di un generatore deve essere consistente dai valori iniziali.
  E' inadeguato quindi testare un PRNG basandosi sull'output di un singolo seed o testare un TRNG sull'output prodotto da una singola fonte.

Randomness tests proposti dal NIST

L' **SP 800-22** lista 15 test di casualità che richiedono una conoscenza basilare dell'analisi statistica, qui ne verranno analizzati solo alcuni.

Frequency (Monobit) Test

Il test determina la proporzione delle occorrenze di zero e uno in una sequenza di bit. Per un sequenza random ci si aspetta che il rapporto tra i numeri sia vicino a ½.

Si consideri una sequenza di bit $\boldsymbol{\varepsilon = \varepsilon_1, \ldots, \varepsilon_n}$ dove $\varepsilon_i = 0$ o $1, \forall i = 1, \ldots, n$

n è il numero di bit nella sequenza , $n \geq 100$ per avere risultati significativi.

I test trasformano la sequenza ε in una nuova sequenza X come $\boldsymbol{X_i = 2\varepsilon_i - 1 = \pm 1}$

La somma della sequenza è data da $\boldsymbol{S_n = X_1 + X_2 + \ldots + X_n}$

La statistica del test per la somma osservata $\mathbf{s_{obs}}$ è data da:

- $\boldsymbol{S_{obs} = \frac{|S_n|}{\sqrt{n}}}$

La distribuzione di riferimento per la statistica del test è quella normale con media zero per n grande. Se la sequenza è random, allora i +1 e i -1 tenderanno a cancellarsi così da avvicinare la statistica a 0.

Il P-value è invece dato da:

- $\boldsymbol{P\text{-}value = erfc\left(\frac{S_{obs}}{\sqrt{2}}\right)}$

Dove erfc(z) è la funzione di errore complementare:

- $\boldsymbol{erfc(z) = \frac{2}{\sqrt{\pi}} \int_z^{\infty} e^{-u^2} du}$

Se il numero di zero e uno nella sequenza tendono ad essere gli stessi, allora S_n sarà vicino a 0 e conseguentemente il P-value sarà vicino ad 1.

La sequenza testata sarà accetta come random se il P-value ≥ 0.01 altrimenti sarà non-random.

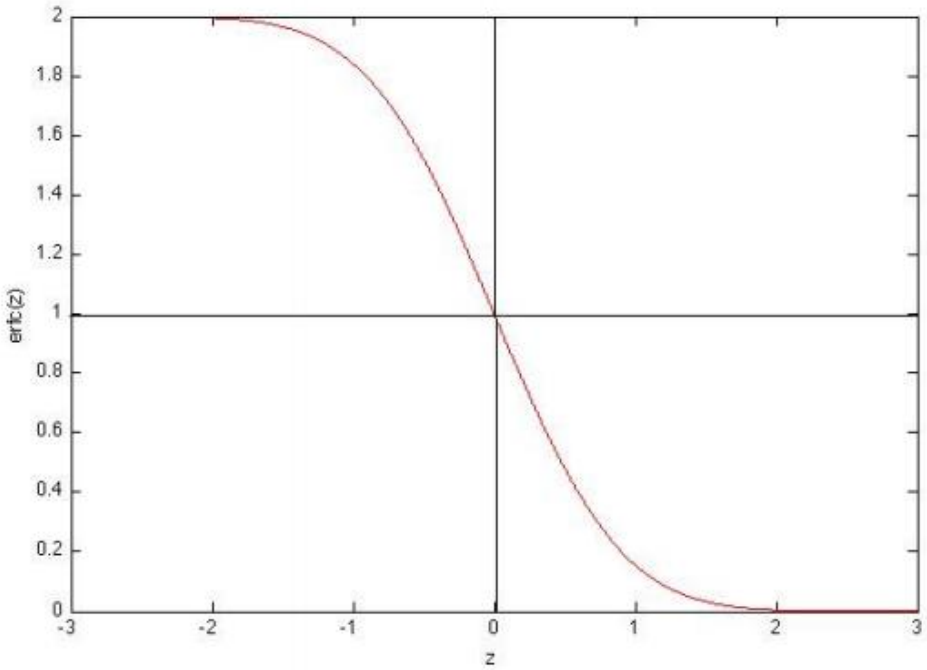


Figure 1: The complementary error function erfc(z)

Lempel-Ziv Compression Test

Il test **Lempel-Ziv**, seguendo le orme del **Maurer's Universal Statistal Test**, determina quanto una sequenza di bit può essere compressa.

La complessità di una sequenza fu definita nel 1976 da Lempel e Ziv. Questa misura conta il numero di differenti pattern in un sequenza, scansionata da sinistra verso destra.

Ad esempio la complessità Lempel-Ziv di s = 1010010100101111110 è 8 poiché i pattern osservati in s sono 1|0|10|01|010|0101|11|110|. La complessità Lempel-Ziv è la base dell'algoritmo di compressione LZ77.

Ci si aspetta che una sequenza random, di lunghezza n, abbia una complessità Lempel-Ziv vicina al valore atteso di complessità di una sequenza di lunghezza n.

Il test è stato usato nel NIST test suite tuttavia ha mostrato alcune debolezze e infatti è stato escluso recentemente per i seguenti motivi.

Anzitutto il test può essere applicato a dati di una specifica lunghezza ossia 10^6 bits.

Inoltre il test usa dati empirici generati da SHA-1 per stimare il valore atteso della complessità Lempel-Ziv di una sequenza di lunghezza 10^6 bits. Apparentemente i dati generati da SHA-1 riportano una stima non tanto buona ad esempio i primi 10^6 bits dell'espansione binaria di *e* falliscono il test di randomicità.

Mr. Ali Doganaksoy e Mr. Faruk Gologlu dell'Università tecnica del Medio Oriente (METU) di Ankara offrono una nuova e più robusta variante del test in "**On Lempel-Ziv Complexity of Sequences**".

Runs test

Viene definita come **run** una sequenza ininterrotta di bits identici. In particolare una run di lunghezza k è formata da k bits identici legati prima e dopo da un bit di valore opposto.

Per esempio la sequenza 0011101011010111 è formata dalle run: 00, 111, 0, 1, 0, 11, 0, 1, 0, 111.

Lo scopo dei run test è di determinare se il numero di run di zero e di uno di varia lunghezza è quello che ci si aspetta da una sequenza casuale. Più concretamente il test determina se le oscillazioni fra zero e uno sono troppo veloci o troppo lente.

PRNG VS TRNG

Table 8.5 Comparison of PRNGs and TRNGs

	Pseudorandom Number Generators	True Random Number Generators
Efficiency	Very efficient	Generally inefficient
Determinism	Deterministic	Nondeterministic
Periodicity	Periodic	Aperiodic

La tabella riassume le principali differenze tra PRNG e TRNG.

I PRNG sono efficienti quindi possono generare molti numeri in tempi brevi, deterministici quindi una data sequenza può essere riprodotta in un altro momento se si conosce il seed. Entrambe le caratteristiche sono molto vantaggiose per la maggior parte delle applicazioni che richiedono molti numeri e la necessità di ripetere la sequenza in momenti diversi, tuttavia sono soggetti alla periodicità.

Per **periodicità** si intende che la sequenza tende a ripetersi ma nei moderni algoritmi è così alta da poter essere ignorata.

I TRNG sono generalmente inefficienti se confrontati con i PRNG il che potrebbe essere un problema in una sistema bancario che potrebbe richiedere milioni di bit casuali al secondo. Inoltre non sono deterministici e sono **aperiodici**.

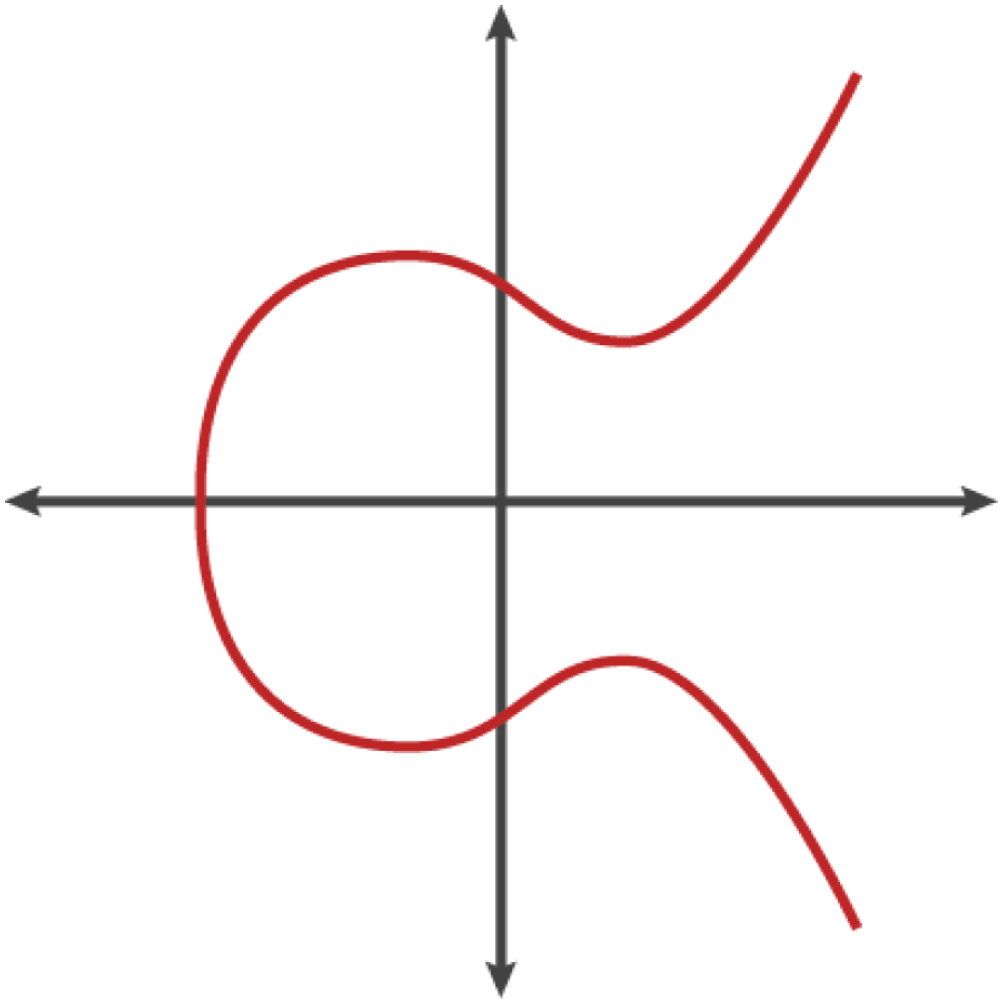
Generare la chiave pubblica

Analizzerò più avanti come la chiave privata viene generata in bitcoin, per il momento ci basta sapere che si tratta di un numero pseudocasuale di 256 bit.

La chiave pubblica è derivata dalla chiave privata utilizzando la moltiplicazione della curva ellittica, una operazione irreversibile:

K = k * G dove k è la chiave privata, G è un punto costante chiamato punto generatore e K è la chiave pubblica.

Curve ellittiche



Una curva ellittica

Bitcoin usa una specifica curva ellittica e un insieme di costanti matematiche definite nello standard **secp256k1**, stabilito dal National Institute of Standards and Technology (**NIST**).

La curva secp256k1 è definita dalla seguente funzione:

$$y^2 = (x^3 + 7) \text{ over } (\mathbb{F}_p)$$

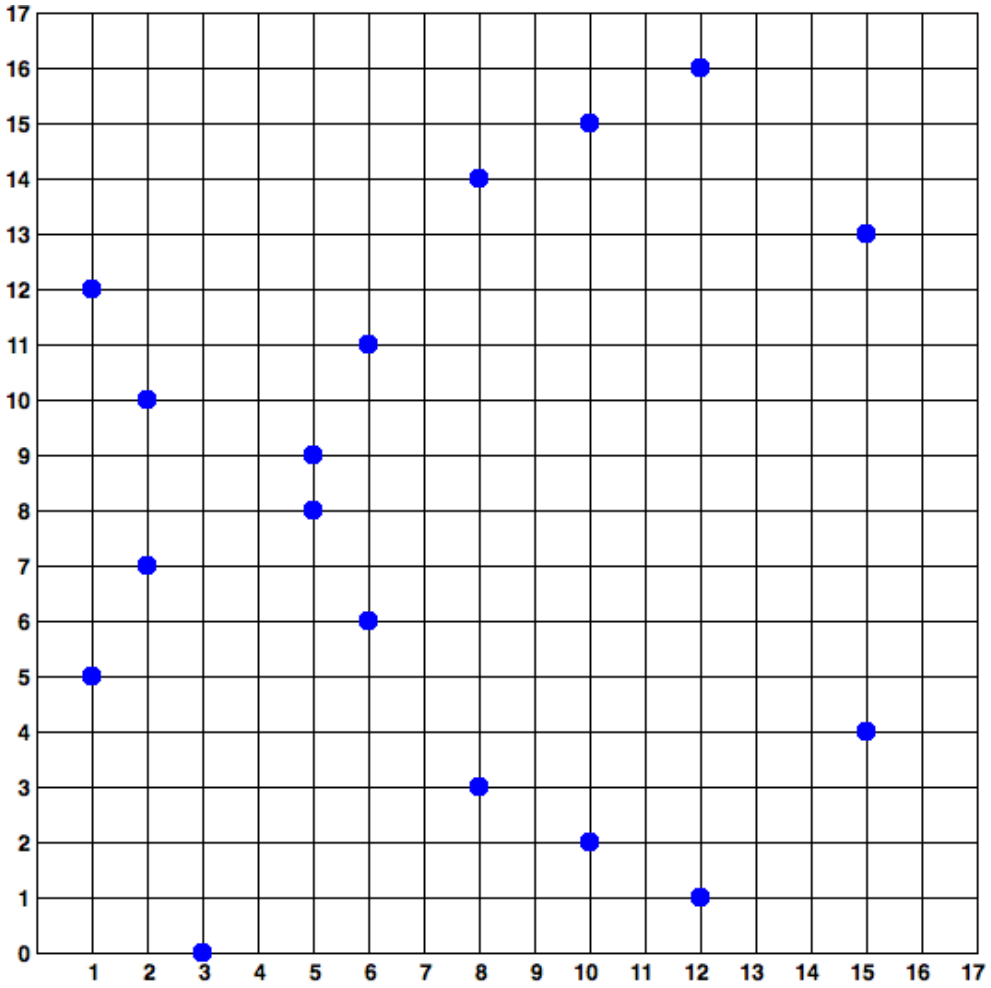
o

$$y^2 \bmod p = (x^3 + 7) \bmod p$$

Il *mod p* indica che questa curva è su un campo finito di ordine primo p scritto anche

$\mathbb{F}_p$ **dove p =** $2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$, un numero primo enorme.

Poiché questa curva è definita su un campo finito di ordine primo invece che sui numeri reali sembrerebbe come uno schema di punti sparsi in due dimensioni e sarebbe difficile da visualizzare. La figura sottostante mostra la stessa curva ellittica su un campo finito di ordine primo 17, molto più piccolo, mostrando uno schema di punti sulla griglia.



Curva ellittica su F(p) con p = 17

Nella matematica delle curve ellittiche c'è un punto chiamato punto all'infinito che corrisponde approssimativamente al ruolo dello zero nell'addizione. L'operatore di addizione ha proprietà simili a quelle dell'addizione tradizionale sui numeri reali.

Dati due punti P_1 e P_2 sulla curva ellittica c'è un terzo punto sulla curva tale che $\mathbf{P_3 = P_1 + P_2}$.

Geometricamente il terzo punto P_3 è calcolato tracciando una linea tra P_1 e P_2.

La linea interseca la curva in un punto $\mathbf{P_3' = (x, y)}$ che riflesso sull'asse x sarà il punto $\mathbf{P_3 = (x, -y)}$.

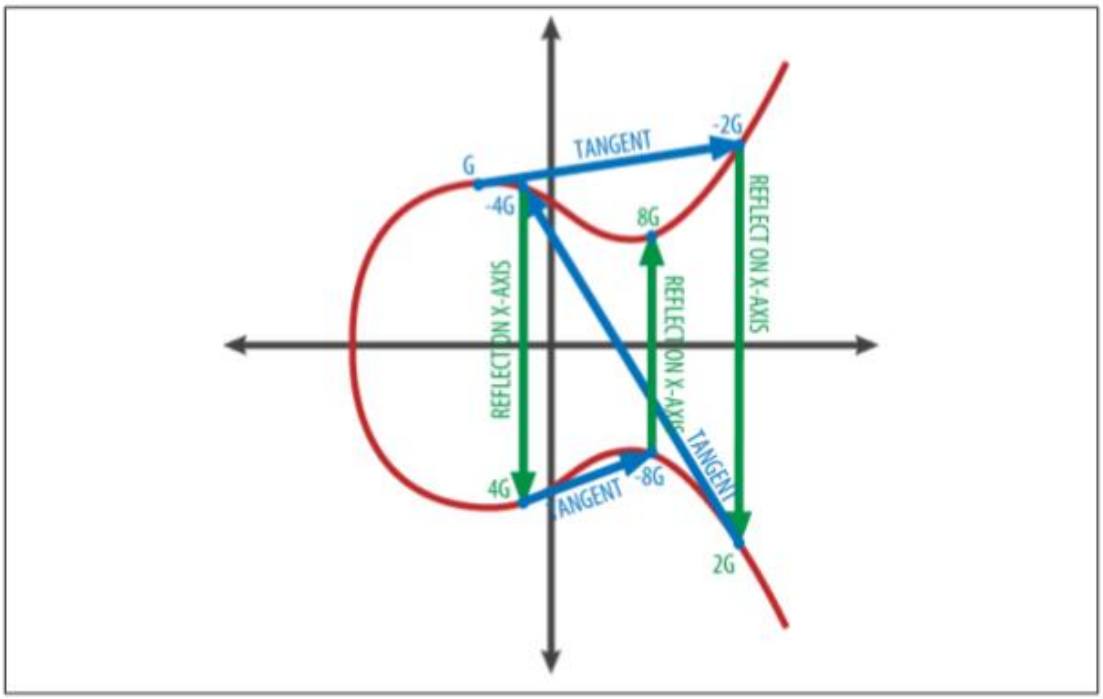
Definita l'addizione si può definire la moltiplicazione come:

dato un punto P sulla curva ellittica, se k è un numero intero allora

$\mathbf{kP = P + P + P + \ldots + P}$ **(k volte).**

Per visualizzare la moltiplicazione di un intero con un punto, userò la più semplice curva ellittica sui numeri reali. L'obiettivo è di trovare il multiplo kG del punto generatore G che è come sommare G a se stesso k volte.

Nelle curve ellittiche aggiungere un punto a stesso è l'equivalente di tracciare una tangente sul punto e trovare dove interseca di nuovo la curva, dopodiché riflettere quel punto sull'asse delle ascisse.



La moltiplicazione di un punto G per un intero k su una curva ellittica

L'operazione inversa conosciuta come "trovare il logaritmo discreto" calcolando k conoscendo K è difficile tanto quanto effettuare una ricerca brute-force di k.

Indirizzo Bitcoin

L'indirizzo Bitcoin è derivato dalla chiave pubblica attraverso l'uso di due funzioni di HASH.

Una funzione di HASH accetta come input un blocco di dati M di lunghezza variabile e produce un message digest di lunghezza fissa.

L'obiettivo principale di queste funzioni è di fornire un risultato per cui sia computazionalmente inattuabile di:

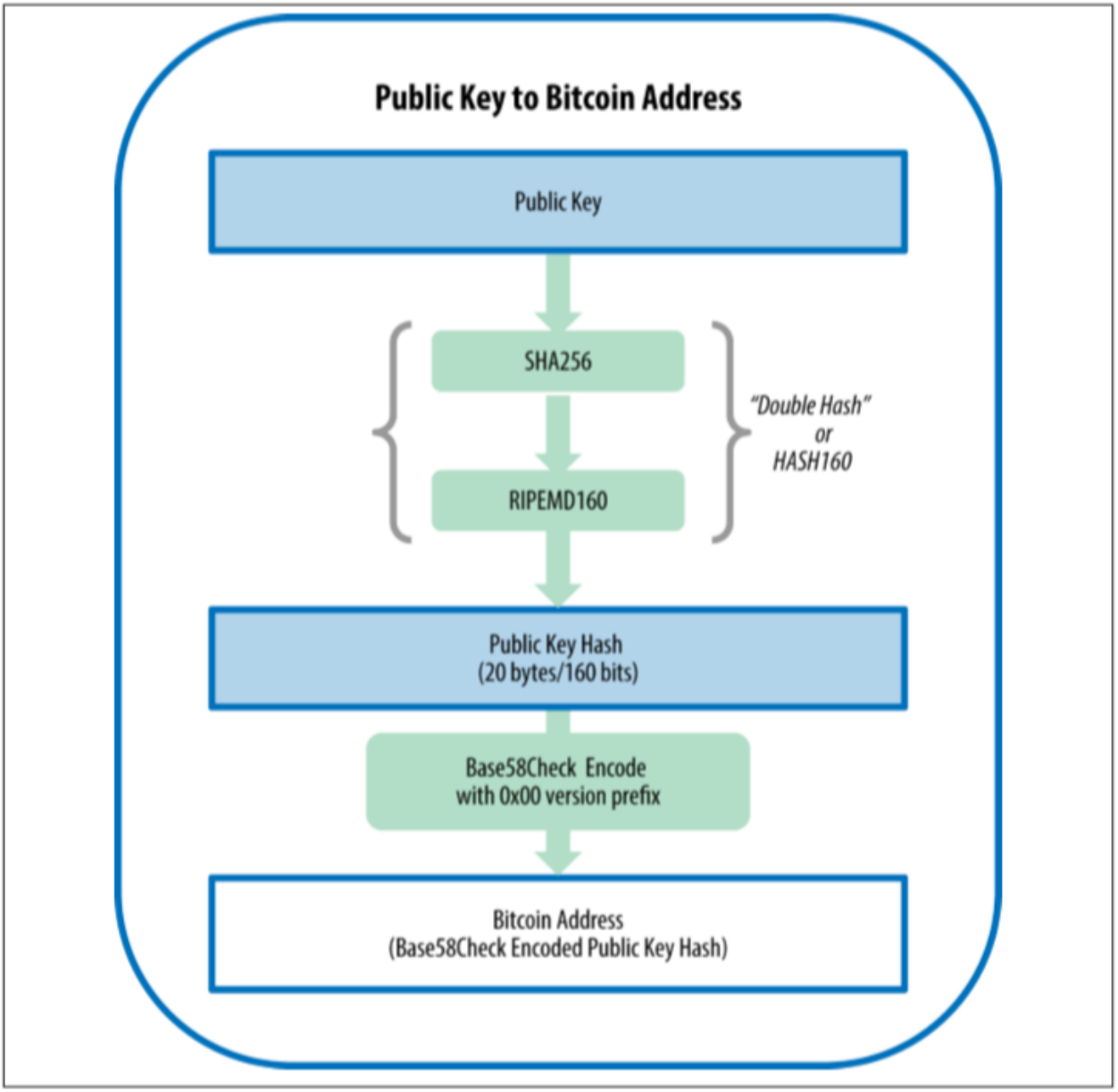
- Trovare un data object che fornisce il digest voluto (one way property)
- Trovare due data objects tali che forniscano lo stesso risultato di hash (collision-free property)

Gli algoritmi usati sono Secure Hash Algorithm (**SHA**) e RACE Integrity Primitives Evaluation Message Digest (**RIPEMD**), più precisamente SHA256 e RIPEMD160. SHA256 produce un digest di appunto 256 bits mentre RIPEMD160 di 160 bits.

Indicando con A l'indirizzo bitcoin e K la chiave pubblica:

$$A = RIPEMD160(SHA256(K))$$

Per rappresentare l'indirizzo di 20 bytes in maniera più compatta, aumentando la leggibilità all'occhio umano ma soprattutto per evitare ambiguità ed errori viene codificato in **base58**.



La figura mostra la conversione di una chiave pubblica in un indirizzo bitcoin

Base58

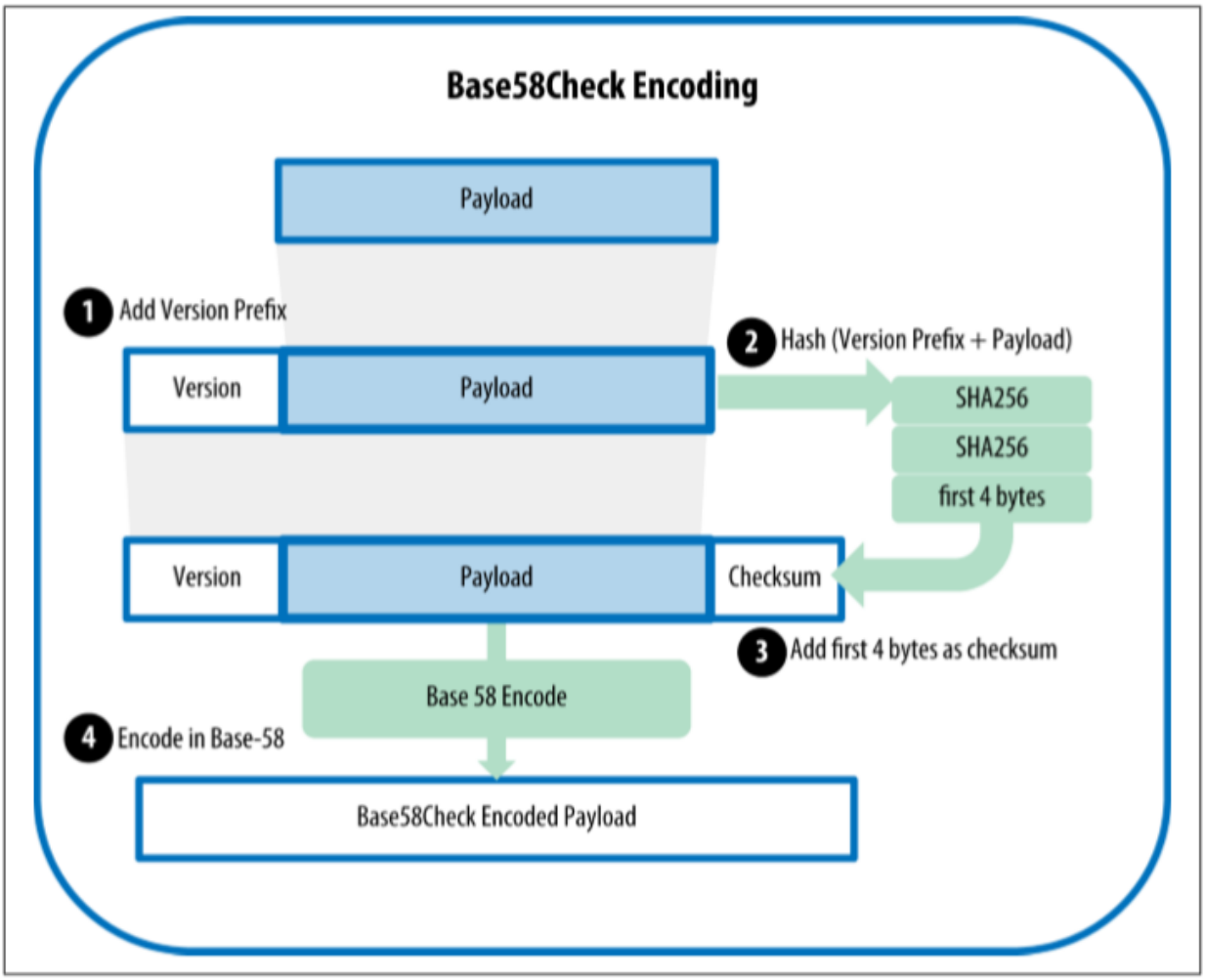
E' un formato di codifica binaria basato su testo sviluppato per l'uso in Bitcoin e molte altre criptomonete. Previene i più frequenti errori di scrittura causati spesso dai fonts che utilizzano simboli simili per caratteri diversi. Nello specifico Base58 è un sottogruppo di Base64 in quanto omette i simboli che indicano lo zero (0), la o maiuscola (O) , la L minuscola (l) , la i maiuscola (I) e i simboli " \ + / ".

Per aggiungere una sicurezza extra in caso di errori di battitura all'interno di **Base58Check** c'è un codice di controllo dell'errore. Il software di decodifica calcolerà il checksum dei dati e lo confronterà con quello incluso nell'indirizzo e se non corrispondono c'è un errore.

Per convertire dati in formato Base58Check si aggiunge un prefisso alla stringa detto "**version byte**" che serve ad identificare meglio i tipi di dati codificati, ad esempio per bitcoin il prefisso è zero (0x00 in esadecimale). Dopodiché si computa un doppio SHA256 alla stringa composta da prefisso e payload:

checksum = SHA256(SHA256(prefisso+dati))

Del checksum si prenderanno solo i primi 4 bytes e verranno concatenati alla fine della stringa formata dal prefisso e payload. La stringa ora viene codificata utilizzando l'alfabeto Base58 descritto precedentemente.



Wallets

Ad alto livello, il portafoglio è un'applicazione che funge da interfaccia utente. Controlla l'accesso al denaro dell'utente, gestisce chiavi e indirizzi, traccia il bilancio, crea e firma transazioni.

Per un programmatore il portafoglio è una struttura dati usata per memorizzare e gestire le chiavi dell'utente.

Un luogo comune errato su Bitcoin è che il portafoglio bitcoin contiene letteralmente bitcoin mentre invece le monete sono registrate sulla blockchain nella rete Bitcoin.

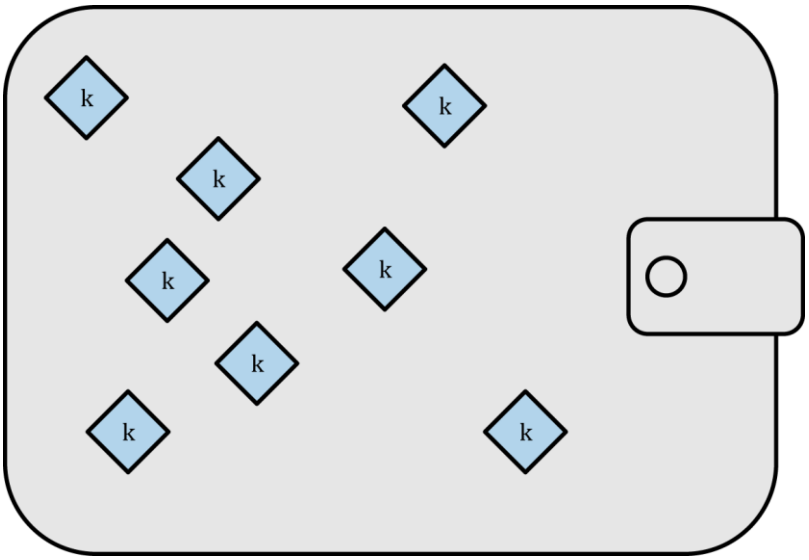
Esistono due tipi primari di portafogli distinti su come le chiavi sono legate tra loro.

Portafogli non deterministici

Nei primi bitcoin clients, i portafogli erano semplicemente una collezione di chiavi private generate randomicamente. Questo tipo di portafoglio è detto nondeterministic wallet Type-0.

Ad esempio, il bitcoin Core client genera 100 chiavi private casuali al primo utilizzo, usandone una sola per volta. Questo tipo di portafoglio è detto "Just a Bunch Of Keys" o JBOK ossia "un mucchio di chiavi". Lo svantaggio evidente nell'utilizzo di questi portafogli è la necessità di dover effettuare backup frequenti di tutte le chiavi. Va ricordato cha la chiave privata è l'unico modo per accedere ai fondi di un indirizzo bitcoin e dunque è fortemente consigliato di effettuare più backup. Invece utilizzare più volte lo stesso indirizzo riduce la privacy poiché vengono associate più transazioni allo stesso indirizzo.

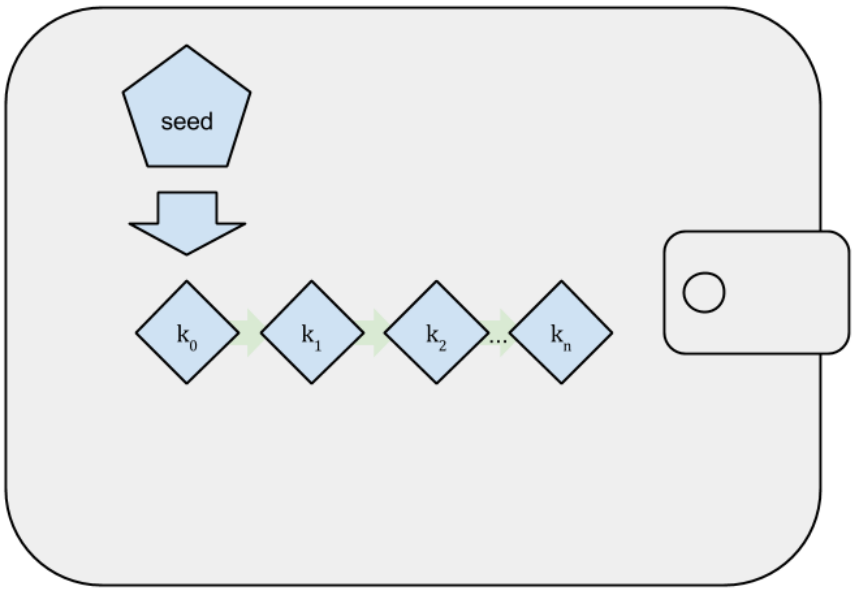
Sebbene il Bitcoin Core client include un wallet Type-0, il suo utilizzo è sconsigliato dagli sviluppatori stessi.



Portafoglio Type-0 non deterministico

Portafogli deterministici

I portafogli deterministici contengono le chiavi private derivate dallo stesso seed. Il seme permette di recuperare tutte le chiavi derivate rendendo la gestione del backup molto più semplice. Inoltre il seme è sufficiente per importare o esportare il portafoglio agevolando la migrazione di tutte le chiavi tra differenti implementazioni di portafogli.



Portafoglio Type-1 deterministico

Parole in codice mnemonico

Le parole in codice mnemonico sono sequenze di parole che rappresentano un numero usato come seme per derivare un portafoglio deterministico. La sequenza di parole è sufficiente per ricreare il seme e con esso di ricreare il portafoglio e tutte le chiavi derivate.

Una sequenza di parole è più facile da leggere, memorizzare e trascrivere senza commettere errori. E' possibile anche utilizzare un dizionario di parole di lingua diversa da quella inglese come quella italiana, spagnola, francese, cinese ecc.

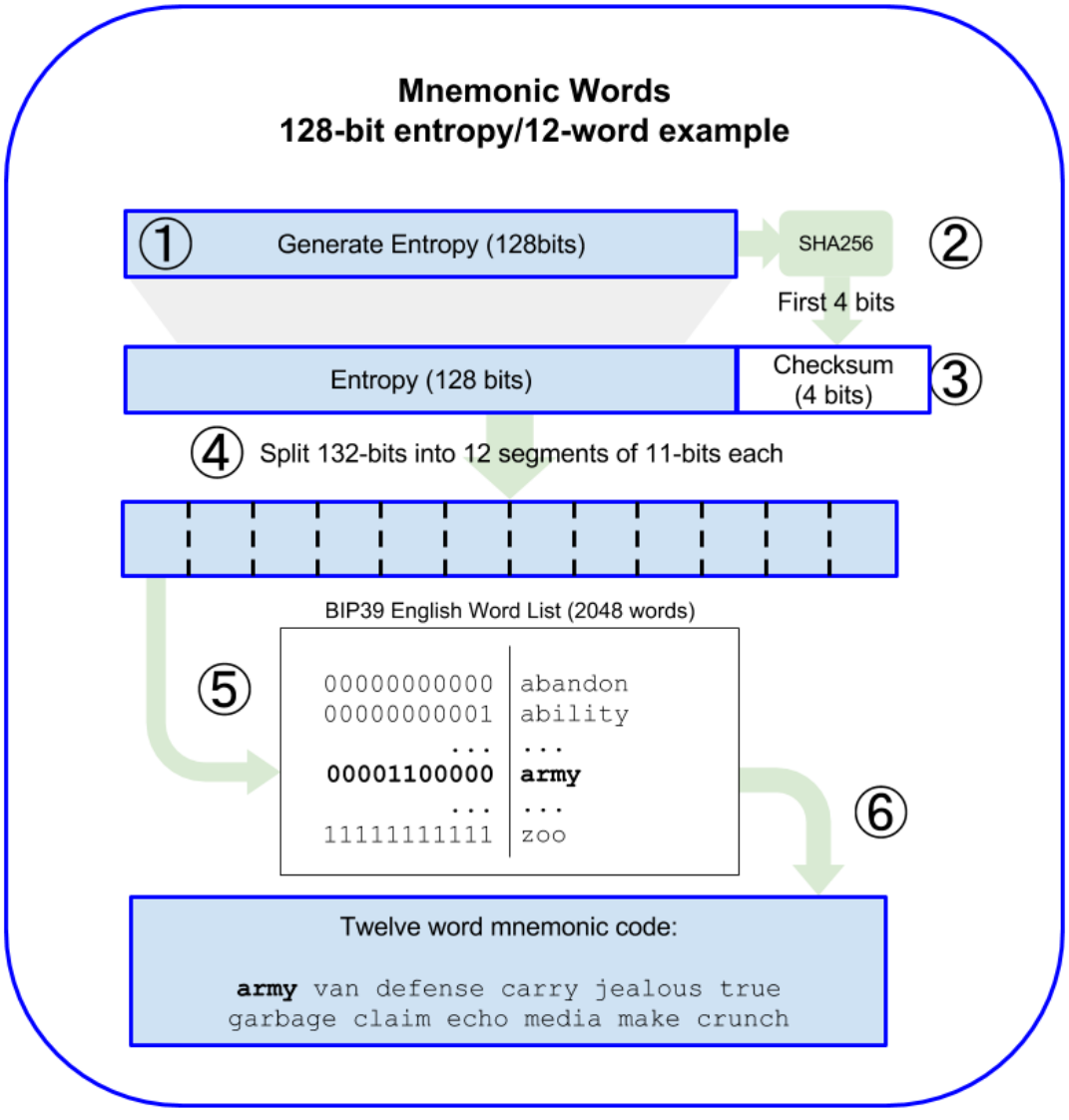
Le parole mnemoniche sono spesso confuse con i ***brainwallets***. Mentre le prime sono sequenze di parole scelte in maniera casuale da uno specifico dizionario, il brainwallet è una sequenza di parole scelte dall'utente. Sebbene per l'utente sia più facile da ricordare una sequenza da lui inventata avrà sicuramente una scarsa casualità.

I codici mnemonici sono definiti nel **BIP-39** tuttavia questa è una implementazione dello standard del codice mnemonico. Infatti c'è una implementazione differente con un differente insieme di parole usato dal portafoglio **Electrum**. Tuttavia BIP0039 fu proposto dalla compagnia alle spalle dell'hardware wallet **Trezor** e il suo successo lo ha reso uno standard de facto.

BIP-39

BIP0039 definisce la creazione del codice mnemonico e del seme come segue:

1. Generare una sequenza random dai 128 fino a 256 bits, in multipli di 32. Questa è l'entropia.
2. Creare un checksum della sequenza prendendo i primi N bits del suo hash SHA256.
   Dove N = lunghezza dell'entropia / 32.
3. Concatenare il checksum alla fine della sequenza.
4. Dividere la sequenza in sezioni da 11 bits.
5. Poiché 2^{11} allora ogni sezione è un numero da 0 a 2047 e ognuno sarà un indice del dizionario che corrisponderà ad una parola.
6. Il codice mnemonico è una sequenza di parole.



Generare entropia e codificarla come sequenza di parole

Table 5-2. Mnemonic codes: entropy and word length

Entropy (bits)	Checksum (bits)	Entropy + checksum (bits)	Mnemonic length (words)
128	4	132	12
160	5	165	15
192	6	198	18
224	7	231	21
256	8	264	24

La tabella mostra la relazione tra la dimensione dell'entropia e la lunghezza dei codici mnemonici

Caratteristiche del dizionario

Un dizionario ideale ha le seguenti caratteristiche:

- Una selezione smart delle parole:
  E' sufficiente scrivere le prime quattro lettere per identificare l'intera parola.
- Evitare le parole simili.
- Dizionario ordinato: aumenta l'efficienza nella ricerca.

Dalla mnemonica al seed

Le parole mnemoniche rappresentano una entropia con una lunghezza dai 128 ai 256 bits.

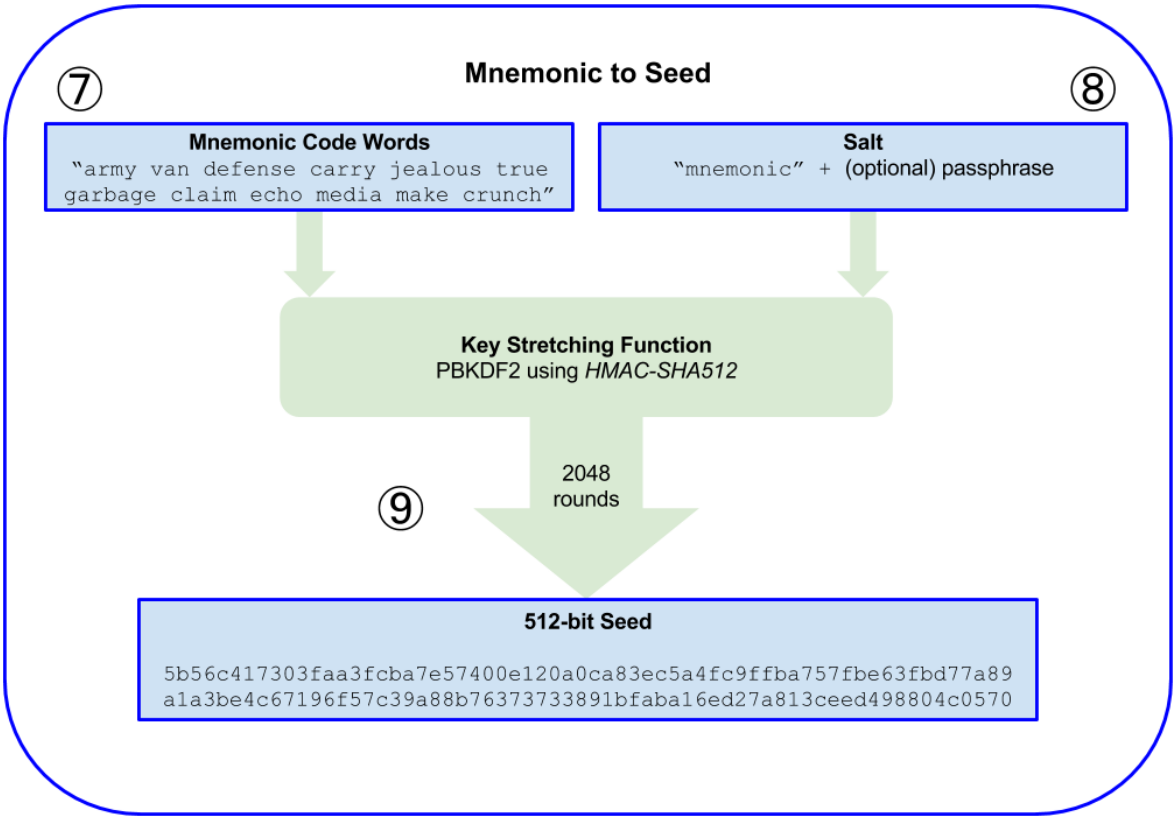
Per derivare da esse un seme di 512 bit viene utilizzata una particolare funzione ossia **PBKDF2**.

PBKDF2 prevede due parametri: la sequenza e il **salt value** letteralmente sale. Il sale ha come scopo quello rendere più difficile un attacco brute-force ma nel BIP-39 ne ha anche un altro. Il sale infatti consiste in una stringa costante "**mnemonic**" concatenata ad una frase opzionale scelta dall'utente.

Non c'è una frase opzionale sbagliata o meno casuale e tutte generano un seed diverso inoltre aggiunge un ulteriore protezione in quanto rende la sequenza mnemonica inutile senza di essa. Allo stesso tempo utilizzare una frase opzionale aumenta il rischio di perdita poiché è necessario memorizzare due stringhe ma soprattutto in luoghi separati.

Il processo fermatosi al punto 6 continua:

7. Il primo parametro per la funzione PBKDF2 è la sequenza mnemonica del punto 6
8. Il secondo parametro per la funzione PBKDF2 è il sale composto dalla stringa "mnemonic" concatenata con la frase opzionale.
9. PBKDF2 estende la mnemonica e il sale usando 2048 cicli di hashing con l'algoritmo **HMAC-SHA512** producendo un output di 512 bits che è il seed.



Dalla mnemonica al seed

Entropy input (128 bits)

```
0c1e24e5917779d297e14d45f14e1a1a
```

Mnemonic (12 words)

```
army van defense carry jealous true garbage claim echo media make crunch
```

Passphrase

(none)

Seed (512 bits)

```
5b56c417303faa3fcba7e57400e120a0ca83ec5a4fc9ffba757fbe63fbd77a89a1a3be4c67196f57c39
a88b76373733891bfaba16ed27a813ceed498804c0570
```

Seed generato da un'entropia di 128 bits senza frase opzionale

Entropy input (128 bits)

```
0c1e24e5917779d297e14d45f14e1a1a
```

Mnemonic (12 words)

```
army van defense carry jealous true garbage claim echo media make crunch
```

Passphrase

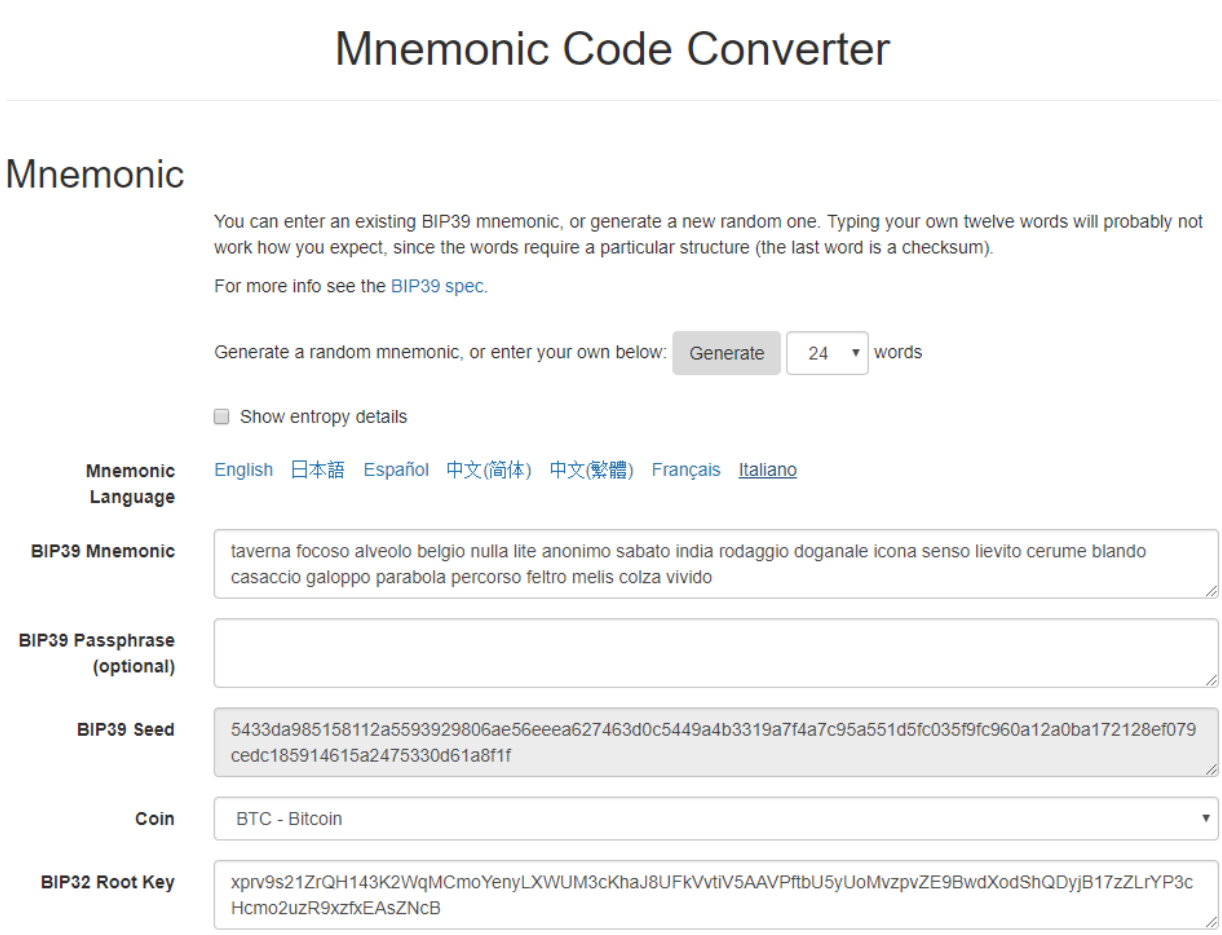
SuperDuperSecret

Seed (512 bits)

```
3b5df16df2157104cfdd22830162a5e170c0161653e3afe6c88defeefb0818c793dbb28ab3ab091897d0
715861dc8a18358f80b79d49acf64142ae57037d1d54
```

Seed generato da un'entropia di 128 bits con una frase opzionale

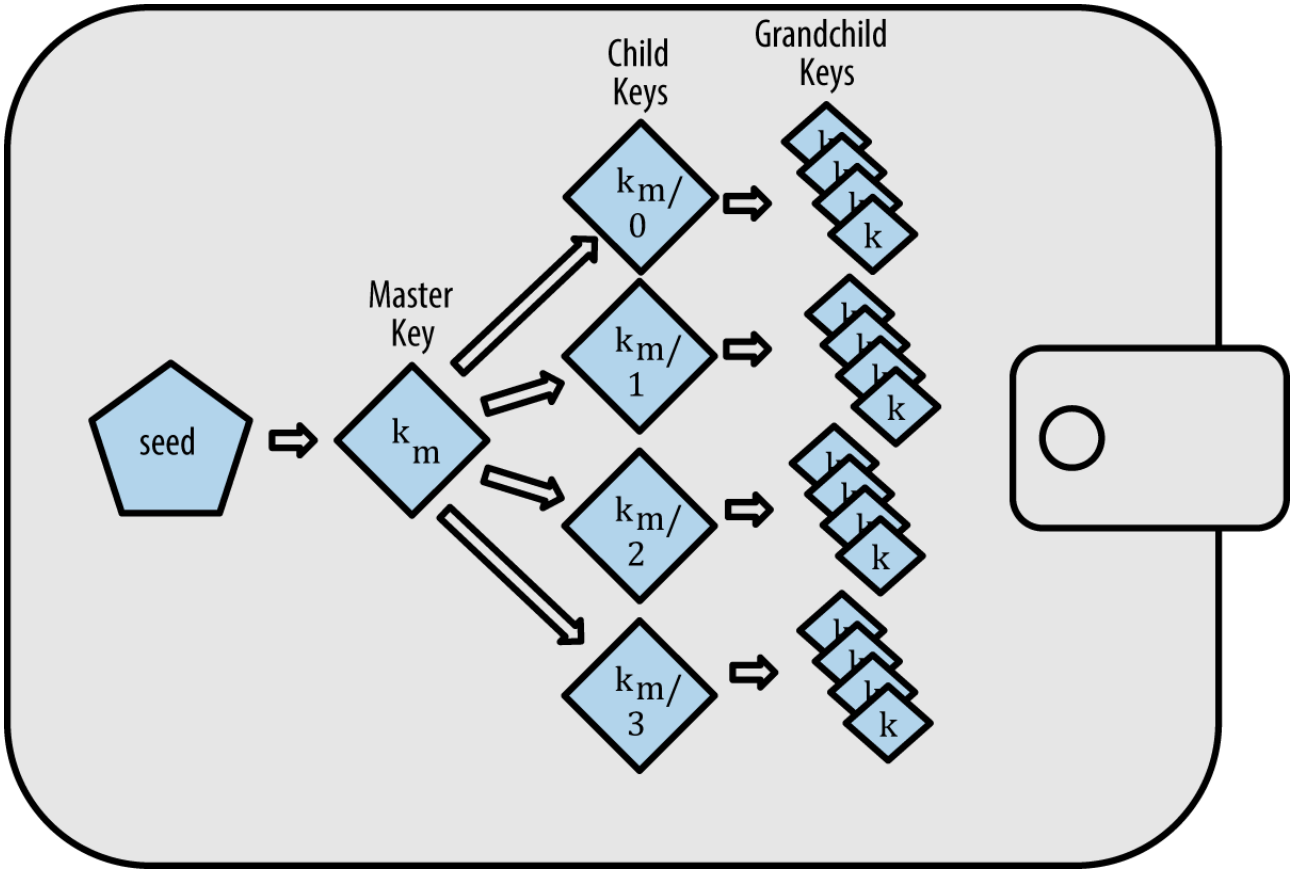
E’ stato implementato un generatore **BIP-39** in una pagina web, utilizzabile anche offline: https://iancoleman.io/bip39/



Portafogli deterministici gerarchici (BIP0032/BIP0044)

I portafogli deterministici gerarchici o **HD wallets** sono la forma più avanzata dei portafogli deterministici e sono definiti dallo standard **BIP0032**.

In questi portafogli le chiavi vengono derivate in una struttura ad albero e così da una chiave genitore si derivano le chiavi figlie e le chiavi da quest'ultime e così via con un'altezza infinita.

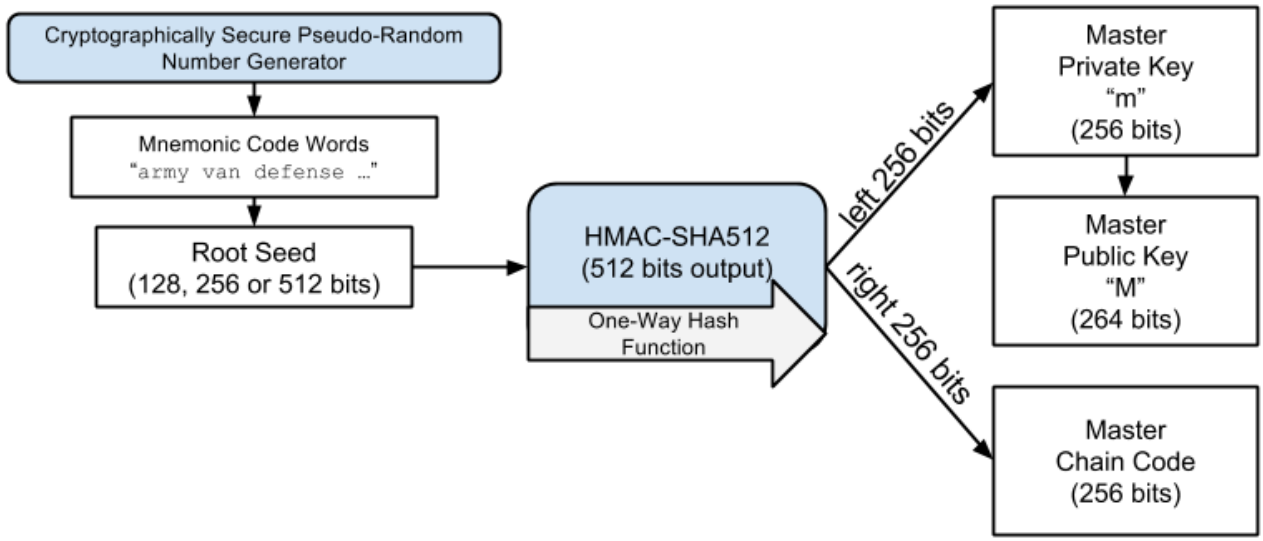


Portafoglio Type-2 deterministico gerarchico

Un HD wallet è generato da un seme, lo stesso analizzato in precedenza, che sarà l'input dell'algoritmo **HMAC-SHA512** il cui output è usato per creare la **master private key** (m) e il **master chain code** (c).

In particolare dal valore di hash di 512 bits i primi 256 bits a sinistra sono m e i 256 bits a destra sono c.

La **master public key** è derivata dalla master private key come abbiamo visto in precedenza.



Creare master key e chain code dal seed

Derivare le chiavi figlie

Per generare chiavi figlie un HD wallet utilizza una serie di funzioni dette **child key derivation (CKD)** che combina:

- Una chiave genitore privata o pubblica
- Un seme detto chain code (256 bits)
- Un numero **indice** (32 bits)

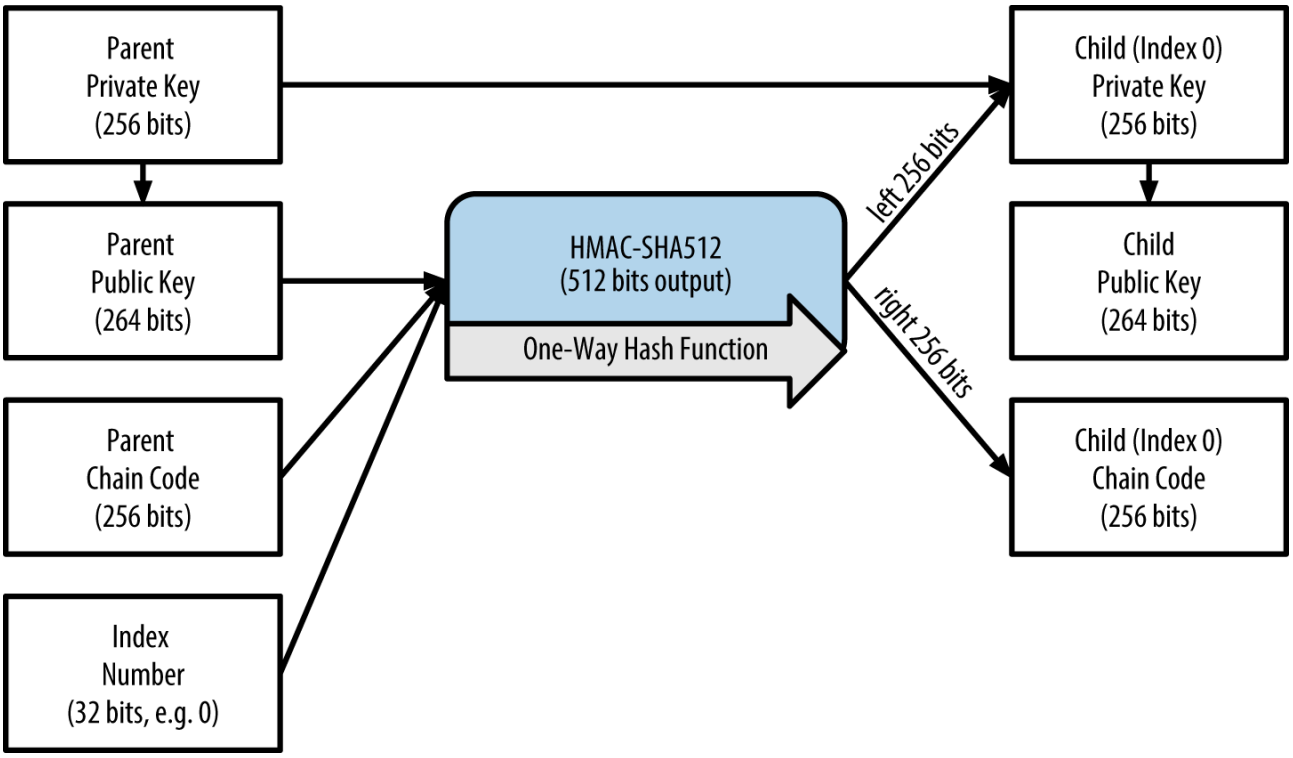
Il chain code è utilizzato per introdurre dati deterministici casuali al processo in modo che conoscendo l'indice e una chiave figlia non si possono derivare altre chiavi figlie.

Il processo di derivazione è il seguente:

1. La chiave genitore pubblica, il chain code e l'indice sono combinati e utilizzati come valori di input di HMAC-SHA512.
2. Il risultato dell'hash, 512 bits, viene diviso in due parti.
3. I 256 bits a destra diventano il chain code della chiave figlia.

4. I 256 bits a sinistra e l'indice sono aggiunti alla chiave privata genitore per produrre la chiave privata figlia.

   L'indice è di 32 bits ma ogni chiave genitore può avere 2^{31} chiavi figlie, quelle restanti sono riservate per una derivazione speciale.



Estendere una chiave pubblica genitore per creare una chiave pubblica figlia

Le chiavi private figlie sono indistinguibili da quelle non deterministiche e non possono essere utilizzate per trovare le chiavi sorelle. Ogni chiave figlia ha la sua chiave pubblica e il suo indirizzo e il fatto che siano parte di una sequenza non è visibile all'esterno dell'HD wallet che le ha generate.

Chiavi estese

Una **chiave estesa** è di 512 bits e vi sono due tipi:

- Una **chiave privata estesa** è la concatenazione della chiave privata e del suo chain code.
- Una **chiave pubblica estesa** è la concatenazione della chiave pubblica e del suo chain code.

Una chiave estesa può creare chiavi figlie e condividerla concede l'accesso all'intera diramazione.

Le chiavi estese codificate utilizzando Base58Check hanno il prefisso "**xprv**" per le private e "**xpub**" per le pubbliche, il tutto per essere facilmente riconoscibili.

Derivazione delle chiavi pubbliche figlie

Le chiavi pubbliche figlie possono essere derivate in due modi:

- In maniera tradizionale utilizzando la chiave privata figlia
- Utilizzando la **chiave pubblica genitore estesa**

Una chiave pubblica estesa può essere utilizzata per derivare tutte le chiavi pubbliche nel suo ramo. Questa proprietà può essere utilizzata per creare una sequenza infinita di chiavi pubbliche in server o applicazioni avendone una sua copia, senza la relativa chiave privata.
Una chiave privata estesa codificata in Base58Check:

- **xprv**9tyUQV64JT5qs3RSTJkXCWKMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9CAWrUE9i6GoNMKUga5biW6Hx4tws2six3b9c

Una chiave pubblica estesa codificata in Base58Check:

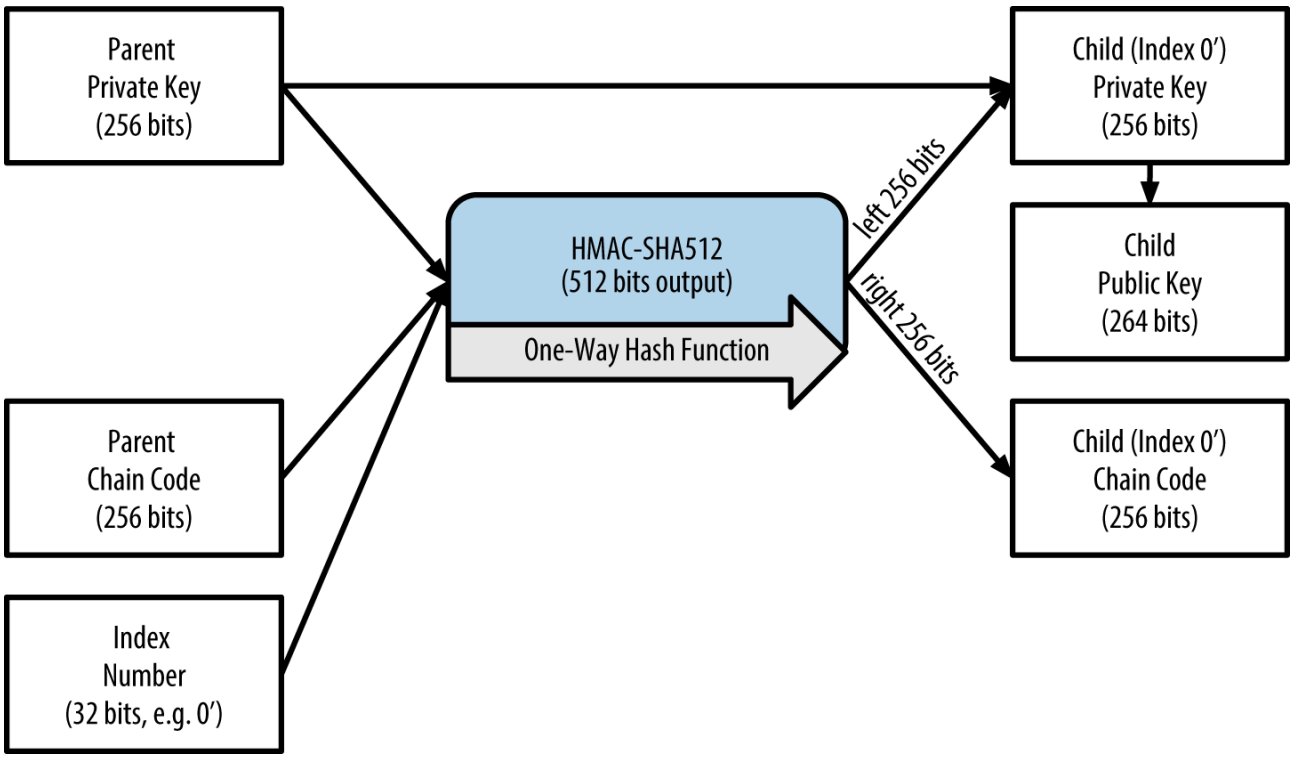
- **xpub**67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtjJ2tgypeQbBs2UAR6KECeeMVKZBPLrtJunSDMstweyLXhRgPxdp14sk9tJPW9

Derivazione delle chiavi figlie hardened (temprate)

La possibilità di poter derivare una ramificazione di chiavi pubbliche da una chiave estesa è molto utile ma c'è un potenziale rischio. La chiave xpub contiene il chain code e se anche una sola chiave privata figlia venisse diffusa può essere utilizzata con il chain code per derivare tutte le chiavi private sorelle. O peggio ancora una chiave privata figlia con un chain code genitore possono essere usati per dedurre la chiave privata genitore.

Per neutralizzare questo rischio c'è una funzione di derivazione alternativa che rompe il legame tra chiave pubblica genitore e chain code figlio.

La derivazione temprata usa la chiave privata per derivare il chain code figlio, creando un firewall nella sequenza genitore figlio. La risultante ramificazione di chiavi può essere usata per produrre chiavi pubbliche estese non vulnerabili ai rischi sopra citati poiché il chain code che contengono non può essere utilizzato per rivelare le chiavi private.



Derivazione temprata (hardened) di una chiave figlia

Per distinguere in maniera semplice le chiavi derivate normalmente da quelle temprate l'indice è stato diviso in due intervalli.

- Gli indici tra 0 e 2^{31}-1 sono utilizzati per le derivazioni normali
- Gli indici tra 2^{31} e 2^{32}-1 sono utilizzati per le derivazioni temprate

Paper Wallet

Una chiave privata stampata su carta è un paper wallet.

Definito anche come **cold storage** è un modo molto efficace per creare backup o un deposito di bitcoin offline. Se un paper wallet non mai stato memorizzato su un computer e le chiavi sono state generate offline è completamente immune ai problemi tecnologici dai guasti hardware ai key-loggers.

Bitaddress.org è uno strumento client-side JavaScript con licenza open-source che permette di generare wallet bitcoin anche offline. L'entropia viene generata dai movimenti eseguiti con il cursore.

Il modo più sicuro per generare un paper wallet è di avviare una macchina pulita, magari con un sistema operativo Linux da CD-ROM e utilizzare il tool bitaddress in locale stampando il wallet con una stampante collegata con cavo USB.

Paper wallet senza frase opzionale

Il tool permette anche di aggiungere una frase opzionale al wallet, che ovviamente non viene stampata, rendendolo immune anche ad un furto.

Paper wallet con frase opzionale

Utilizzando i fondi di un paper wallet si espone la chiave privata e alcuni portafogli potrebbero generare un nuovo indirizzo se non viene speso l'intero capitale. Per questo motivo è consigliabile inviare i fondi rimanenti ad un nuovo paper wallet.

Conclusioni

Bitcoin è una tecnologia completamente nuova, senza precedenti e altamente complessa. Con il passare del tempo sono state sviluppate nuove applicazioni software che permettono anche ai non esperti di farne uso.

SHA-256 e la crittografia ellittica forniscono una sicurezza molto resistente e ad oggi il protocollo Bitcoin non è mai stato penetrato. Solo la chiave privata consente l'accesso a Bitcoin e finché resterà al sicuro, la sicurezza di un portafoglio non sarà mai compromessa.

Bibliografia

Antonopoulos A. M., Mastering Bitcoin - Unlocking Digital Cryptocurrencies 2014

Antonopoulos A. M., Mastering Bitcoin - Programming the Open Blockchain 2E 2017

William Stallings, Cryptography and Network Security 7th edition

Test NIST, A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications

Ali Doganaksoy e Faruk Gologlu, On Lempel-Ziv Complexity of Sequences

Sitografia

Satoshi Nakamoto, “A Peer-to-Peer Electronic Cash System“, bitcoin.org/bitcoin.pdf

NIST SP 800-22, http://qrng.b-phot.org/static/media/NistTestsLongDescription.pdf

RFC4086, https://tools.ietf.org/html/rfc4086

SEC 2: Recommended Elliptic Curve Domain Parameters, http://www.secg.org/sec2-v2.pdf

Documentazione BIP-39, https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki

Documentazione BIP-32, https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

Generatore BIP-39, https://iancoleman.io/bip39/

Tool bitaddress, https://www.bitaddress.org