

Desarrollo de software para blockchain 2024

Bienvenid@s!

Metodología

- Clases teórico-prácticas
- Comunicación y consultas por discord
- Entregas individuales parciales.
- Trabajo integrador grupal con 3 entregas parciales y entrega final.
- El TI consiste en el desarrollo de un proyecto completo.

Primeros conceptos

- Criptografía
- Blockchain
- Transacción
- Bitcoin
- Algoritmo ECDSA
- Algoritmo SHA-256
- Algoritmo de consenso: PoW
- Frase semilla
- UTXO

Criptografía

Criptografía

La criptografía es un campo de estudio que se ocupa de asegurar la comunicación y el almacenamiento de información mediante técnicas de codificación y decodificación.

Técnicamente, la criptografía se basa en algoritmos matemáticos que transforman la información original (texto plano o datos sin cifrar) en una forma ilegible llamada texto cifrado o criptograma.

Criptografía

Existen dos categorías principales de criptografía:

Criptografía simétrica: Utiliza la misma clave tanto para cifrar como para descifrar la información. Ambas partes (emisor y receptor) deben compartir y mantener en secreto esta clave. Ejemplos de algoritmos simétricos incluyen DES, AES y RC4.

Criptografía asimétrica (o de clave pública): Utiliza un par de claves (una pública y una privada). Lo que se cifra con una clave solo puede ser descifrado con la otra y viceversa. La clave pública puede ser compartida libremente, mientras que la clave privada se mantiene en secreto. Los sistemas de clave pública son esenciales para muchas aplicaciones en línea, incluidas las firmas digitales y el cifrado de correo electrónico. Ejemplos de criptografía asimétrica incluyen RSA, DSA y ECDSA.

Criptografía

Además de cifrar y descifrar información, la criptografía también se utiliza para:

Firmas digitales: Proporcionan una forma de verificar la autenticidad y la integridad de los datos. Una firma digital es un dato que se genera a partir de un mensaje y una clave privada, y puede ser verificado por cualquier persona que tenga la clave pública correspondiente y el mensaje original.

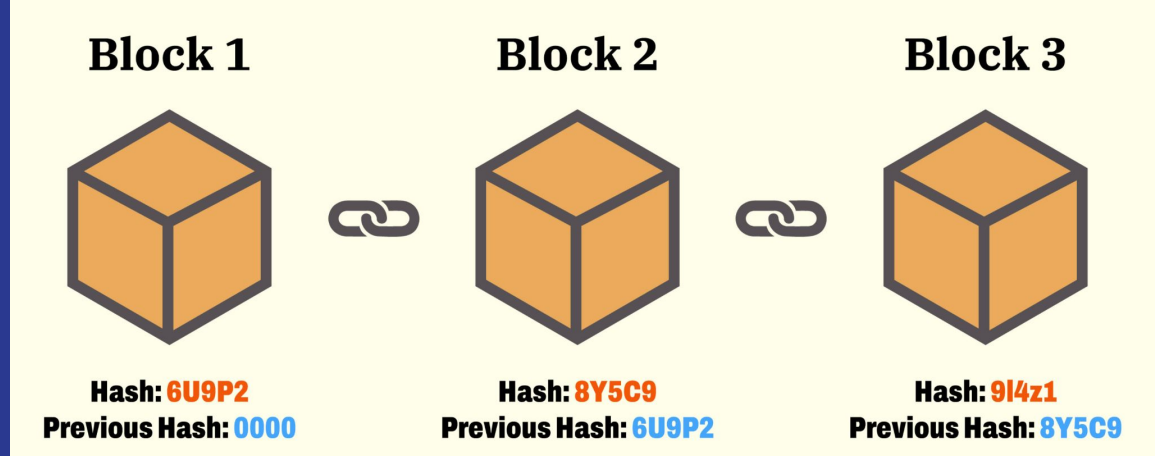
Funciones hash criptográficas: Estas funciones toman una entrada (o "mensaje") y devuelven una cadena fija de bytes. La salida, típicamente un valor de hash, debería ser única (o muy difícil de repetir) para cada entrada única. Es computacionalmente difícil generar la misma salida hash a partir de dos entradas diferentes. Ejemplos incluyen SHA-256 y RIPEMD-160.



Blockchain

Blockchain

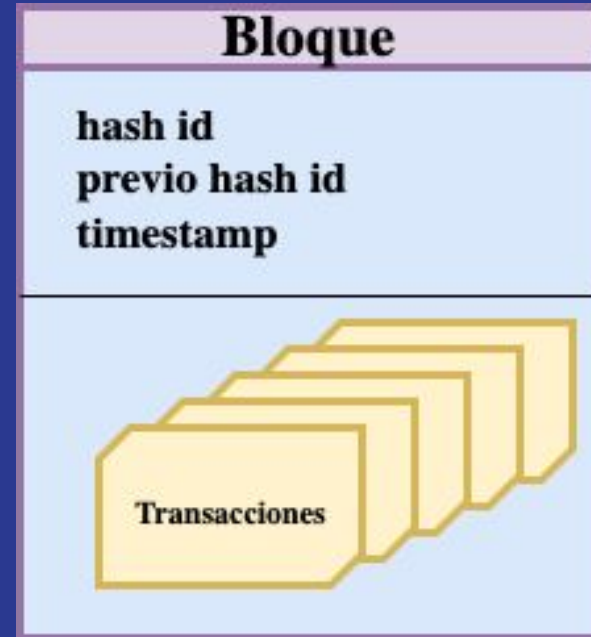
Es una estructura de datos en la que los bloques se enlazan secuencialmente. Cada bloque tiene una referencia al bloque anterior mediante un hash criptográfico. Esto crea una cadena de bloques que proporciona integridad y seguridad a los datos almacenados, ya que cualquier modificación en un bloque afectaría a todos los bloques siguientes.



Blockchain: Bloque

Un bloque contiene la siguiente información:

- ❖ Un hash que lo identifica.
- ❖ Una marca de tiempo.
- ❖ Referencia del bloque anterior.
- ❖ Transacciones del bloque.



Blockchain: Transacción

Una transacción contiene la siguiente información:

- ❖ Un hash que la identifica.
- ❖ El bloque al que pertenece.
- ❖ Un remitente.
- ❖ Un receptor.
- ❖ Valor enviado.
- ❖ Una marca de tiempo.
- ❖ Está firmada criptográficamente.

Blockchain: Transacción

Transaction Details

< >

SALES! Get 15% off (one-time) for any new API Pro subscription. [Code:ESFP15Q223](#)

Overview

State

Comments

Transaction Hash:

0xb56e9c2949a80c866dd3bead6ad78c5a2b84f46fe1dfcfe9858ac23647daf9d6

Status:

Success

Block:

17417514

1 Block Confirmation

Timestamp:

13 secs ago (Jun-05-2023 11:18:59 PM +UTC)

Sponsored:

Se quitó el anuncio. [Detalles](#)

From:

< rsync-builder.eth (rsync-builder)

To:

0xcba0074a77A3aD623A80492Bb1D8d932C62a8bab

Value:

0.076820177147786288 ETH (\$139.18)

Transaction Fee:

0.000416763474603 ETH (\$0.76)

Gas Price:

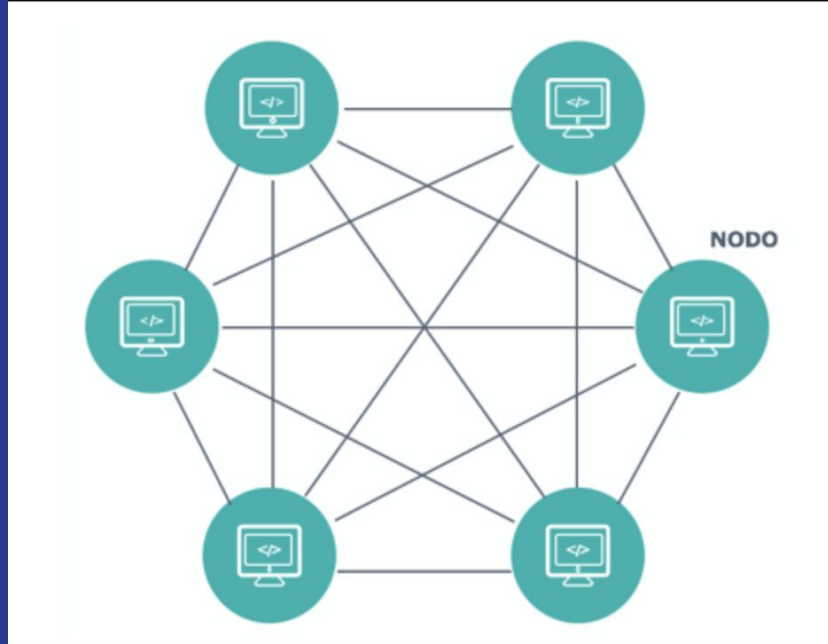
19.845879743 Gwei (0.000000019845879743 ETH)

Blockchain: Nodo

Un nodo se refiere a un dispositivo o computadora que participa en la red de blockchain. Cada nodo tiene una copia de la cadena de bloques completa o una parte de ella, dependiendo de la arquitectura de la red. Los nodos son responsables de mantener la integridad de la red y de validar las transacciones y los bloques.

Los nodos se comunican entre sí para compartir información y mantener la consistencia de la red. Su participación en la red permite la descentralización y la confianza en la seguridad y la validez de las transacciones en blockchain.

Blockchain: Nodo



Blockchain: Ciclo de vida de una txn

1. Creación y firma
2. Propagación a la red
3. Espera en el mempool
4. Un validador o minero la selecciona
5. Se agrega a un bloque y es validado.
6. Confirmación de la transacción

Blockchain: Validación de una txn

La validación de una transacción en blockchain es el proceso mediante el cual se verifica y confirma la autenticidad y la integridad de la transacción antes de ser registrada en la cadena de bloques. Los nodos de la red de blockchain, utilizando reglas y protocolos predefinidos, revisan la validez de la transacción, asegurándose de que cumpla con los requisitos y las reglas establecidas. Esto puede implicar verificar la firma digital, comprobar el saldo suficiente del remitente y aplicar las reglas de consenso para evitar transacciones fraudulentas o inválidas. Una vez validada, la transacción se agrega a un bloque y se propaga a través de la red para su posterior confirmación y consenso por parte de los nodos.



Bitcoin

Bitcoin(2009)

Blockchain inicialmente propuesta en el 2009: <https://bitcoin.org/bitcoin.pdf>

Características técnicas en su implementación:

- ❑ **Hashing SHA-256:** El algoritmo Secure Hash Algorithm 256-bit (SHA-256) es la función de hash principal utilizada en Bitcoin. Esta función toma datos de entrada y devuelve una cadena de 256 bits que parece aleatoria.
- ❑ **Criptografía de clave pública(asimétrica):** Cada usuario tiene un par de claves(una clave pública y una clave privada). Utiliza el algoritmo de firma digital Elliptic Curve Digital Signature Algorithm (ECDSA) para este propósito.

Pero, qué es una curva elíptica?

Una curva elíptica, en términos matemáticos, es el conjunto de puntos que satisfacen una ecuación específica de la forma:

$$y^2 = x^3 + ax + b$$

donde **a** y **b** son constantes y la curva está definida sobre un campo (como los números reales, números complejos o un campo finito).

Curva elíptica

La elección de una curva elíptica para uso criptográfico es esencial y no es trivial. Elegir una curva adecuada puede ser la diferencia entre un sistema seguro y un sistema vulnerable. La elección de la curva implica tanto criterios matemáticos como consideraciones prácticas.

Puntos importantes:

- **Número de puntos:** a mayor número de puntos mayor seguridad en la criptografía.
- **Propiedades matemáticas:** fundamental que no tenga propiedades matemáticas que la hagan vulnerable a ataques. Por ejemplo, algunas curvas tienen un número pequeño de puntos, lo que las hace inseguras. Otras pueden tener estructuras algebraicas que las hagan susceptibles a ciertos ataques.

Bitcoin: curva elíptica -> **secp256k1**

La ecuación de la curva **secp256k1** en su forma canónica es:

$$y^2 = x^3 + 7$$

Esta ecuación está definida sobre un campo primo de tamaño

$$2^{256} - 2^{32} - 977$$

Dentro del contexto de Bitcoin y la criptografía de curvas elípticas, esta curva se utiliza para el algoritmo de firma digital ECDSA (Elliptic Curve Digital Signature Algorithm). Las propiedades de la curva y el gran tamaño del campo subyacente contribuyen a la seguridad de las firmas y claves en el sistema de Bitcoin.

Bitcoin: curva elíptica -> **secp256k1**

La cantidad exacta de puntos N (orden de la curva) es un número primo muy grande que se utiliza en la criptografía de curvas elípticas para definir el grupo finito de puntos sobre el cual se realizan las operaciones.

El orden N de la curva **secp256k1** es:

N
=115792089237316195423570985008687907852837564279074904382605163141518161494337

Esto significa que hay exactamente N puntos en el grupo generado por el punto base de la curva. Este orden es fundamental para la seguridad del ECDSA y otros protocolos de criptografía basados en curvas elípticas, ya que garantiza que las operaciones se realicen en un grupo de tamaño conocido y suficientemente grande. Las operaciones matemáticas (como la suma y la multiplicación de puntos) dentro del contexto criptográfico se realizan módulo N .

Bitcoin: curva elíptica -> **secp256k1**

```
# ejemplo1.py
# pip install numpy
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-3, 3, 400)
y = np.sqrt(x**3 + 7)
plt.plot(x, y, label="y^2 = x^3 + 7", color='blue')
plt.plot(x, -y, color='blue')
plt.legend()
plt.title("Visualización de secp256k1 en el espacio real")
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)
plt.show()
```

Bitcoin: Generación de claves

- ❖ Se elige un punto base de la curva (G), que es fijo y conocido públicamente.
- ❖ Se selecciona un número aleatorio como clave privada en el rango $[1, N-1]$ con gran entropía.
- ❖ Se multiplica este número (la clave privada) por el punto base de la curva elíptica para obtener un nuevo punto: ese punto es la clave pública.

Bitcoin: Generación de claves

```
# ejemplo2.py
# pip install ecdsa
from ecdsa import SigningKey, SECP256k1, VerifyingKey
# Generando una clave privada para la curva secp256k1
sk = SigningKey.generate(curve=SECP256k1)
# Obteniendo la clave pública a partir de la clave privada
vk = sk.get_verifying_key()
# Imprimiendo la clave privada y pública
print("Private Key:", sk.to_string().hex())
print("Public Key:", vk.to_string().hex())
```

Bitcoin: SHA-256

Hashing SHA-256: El algoritmo Secure Hash Algorithm 256-bit (SHA-256) es la función de hash principal utilizada en Bitcoin. Esta función toma datos de entrada y devuelve una cadena de 256 bits que parece aleatoria. Cualquier cambio, por pequeño que sea, en los datos de entrada produce un hash completamente diferente. Las transacciones y bloques en la cadena de bloques de Bitcoin se identifican y verifican mediante este algoritmo de hash.

Bitcoin: SHA-256

```
import hashlib
#ejemplo3.py
def double_sha256 (data):
    """Aplica SHA-256 dos veces."""
    return hashlib.sha256 (hashlib.sha256 (data).digest()).digest ()

# Entrada ficticia: hash de transacción anterior y el índice de la salida
input_txid = "abcd1234abcd1234abcd1234abcd1234abcd1234abcd1234abcd1234"
input_vout = "00000000"

# Salida ficticia: cantidad en satoshis (0.1 BTC) y un scriptPubKey ficticio
output_value = "00ca9a3b00000000"

# Juntar todos los componentes para formar la estructura de la transacción serializada
raw_transaction = input_txid + input_vout + output_value

# Convertir la transacción en bytes
transaction_bytes = bytes.fromhex (raw_transaction)

# Generar el hash de la transacción
tx_hash = double_sha256 (transaction_bytes)

print (f"Hash de la transacción: {tx_hash.hex()}")
```

Algoritmo de consenso

Un algoritmo de consenso es un conjunto de reglas y mecanismos que permite a los participantes de la red llegar a un acuerdo sobre la validez y el orden de las transacciones. Estos algoritmos garantizan la seguridad, la integridad y la descentralización de la red blockchain, y pueden variar en términos de requerimientos computacionales, participación de los nodos y distribución de poder.

Bitcoin: Algoritmo de consenso - PoW

PoW (Proof of Work):

En este algoritmo, los nodos compiten para resolver un desafío criptográfico complejo, lo que requiere una gran cantidad de poder computacional. El primer nodo en encontrar la solución correcta tiene derecho a agregar un nuevo bloque a la cadena y recibir una recompensa. Este proceso se conoce como "minería" y ayuda a garantizar la seguridad y la integridad de la red blockchain.

Bitcoin: Algoritmo de consenso - PoW

PoW (Proof of Work):

Los mineros deben encontrar un valor de tal manera que el hash SHA-256 del encabezado del bloque resulte en un número que esté por debajo de un objetivo dado, conocido como "dificultad".

Esta tarea requiere una gran cantidad de intentos y consume mucha energía, pero una vez que se encuentra una solución, es fácil para otros nodos verificarla.

<https://blockchain-academy.hs-mittweida.de/2021/05/proof-of-work-simulator/>

Bitcoin: Algoritmo de consenso - PoW

```
import hashlib
#ejemplo4.py

def proof_of_work(block_data, difficulty="0000"):

    max_nonce = 10000000 # Ajustar para evitar bucles infinitos
    target = int(difficulty + "F" * (64 - len(difficulty)), 16)

    for nonce in range(max_nonce):
        data = f"{block_data}{nonce}".encode()
        hash_result = hashlib.sha256(data).hexdigest()
        if int(hash_result, 16) < target:
            return nonce, hash_result

    return None, None

# Datos del bloque
block_data = "Información del bloque 12345"
nonce, hash_result = proof_of_work(block_data)
print(f"Nonce: {nonce}")
print(f"Hash: {hash_result}")
```



UTXO

Bitcoin: Transacciones UTXO

UTXO:

Cuando se envía Bitcoin desde una wallet, los fondos no se transfieren en la misma forma en que se moverían en una cuenta bancaria tradicional. En su lugar, se utilizan las "salidas no gastadas" (UTXOs) de transacciones anteriores. Al realizar una nueva transacción, el remitente selecciona una o más de estas UTXOs como entrada.

Bitcoin: Transacciones UTXO

Cada transacción tiene al menos dos salidas:

- Dirección externa (receptor): Esta salida contiene los fondos que se envían al destinatario. La cantidad exacta se define en la transacción.
- Dirección interna (cambio): Cualquier cantidad de Bitcoin que no se haya transferido al destinatario (el cambio) se devuelve al remitente. Sin embargo, en lugar de volver a la misma dirección, las wallets generan automáticamente una nueva "dirección interna" para este propósito, lo que mejora la privacidad.

Bitcoin: Transacciones UTXO

Implicaciones de privacidad

Al usar direcciones internas para el cambio, se evita que sea evidente qué parte de una transacción corresponde al cambio y cuál al destinatario. Esto mejora la privacidad de los usuarios, ya que reduce la posibilidad de rastrear movimientos específicos de fondos.

Bitcoin: Transacciones UTXO

Ejemplo:

- Si una wallet tiene 1 BTC disponible y se quiere enviar 0.6 BTC a un destinatario, la transacción podría verse así:
 - Entrada: 1 BTC (proveniente de una UTXO anterior).
 - Salida 1 (receptor): 0.6 BTC enviado a la dirección del destinatario.
 - Salida 2 (cambio): 0.399 BTC (menos comisiones de minería) enviado a una nueva dirección generada por la wallet del remitente.

Bitcoin: Transacciones UTXO



<https://mempool.space/es/tx/3fd926b2d5f03223311af35847cf5a7b8efb3285e1e9518a1e868ae57c2ada0a>



Frase semilla

Frase semilla

a menudo llamada "seed phrase", "recovery phrase" o "mnemonic phrase", es una representación legible para el ser humano de una semilla que se utiliza para generar determinísticamente un árbol de claves privadas. Esta frase semilla suele consistir en una lista de palabras, generalmente entre 12 y 24, tomadas de un diccionario específico.

La introducción de frases semilla surge del concepto de "carteras deterministas" (deterministic wallets) o HD (Hierarchical Deterministic) wallets en el espacio de criptomonedas. Una HD wallet permite generar un árbol infinito de claves privadas (y, por ende, públicas) a partir de una única semilla.

Frase semilla(usabilidad)

Derivación de Claves: A partir de esta semilla, se puede generar determinísticamente una serie infinita de claves privadas. Estas claves se generan en una estructura jerárquica, lo que significa que puedes tener claves "hijas" derivadas de claves "madres", y así sucesivamente.

Privacidad y Organización: Al utilizar HD wallets, se puede generar una nueva dirección de Bitcoin (o cualquier otra criptomoneda) para cada transacción, lo que mejora la privacidad. A pesar de generar muchas direcciones, todas pueden ser recuperadas a partir de la misma frase semilla.

Restauración: Si alguna vez se pierde el acceso a tu dispositivo o cartera, se puede usar la frase semilla para restaurar los fondos en otro dispositivo o software compatible. Al ingresar la frase semilla, el nuevo software puede regenerar todas tus claves privadas y, por ende, acceder a las direcciones y fondos asociados.

Frase semilla

```
#ejemplo5.py
#pip install mnemonic
#pip install git+https://github.com/primall00/pybitcointools
from cryptos import entropy_to_words, Bitcoin
import os
words = entropy_to_words(os.urandom(16))
print("mi frase semilla:", words)
coin = Bitcoin()
wallet = coin.wallet(words)
print("path de derivación:", wallet.keystore.root_derivation)
print("privada maestra:", wallet.keystore.xprv)
print("pública maestra:", wallet.keystore.xpub)
addr1 = wallet.new_receiving_address()# creando pública de recepción
print("addr1:", addr1)
print("privada de addr1:", wallet.privkey(addr1))
addr2 = wallet.new_change_address()# creando 2da pública de recepción
print("addr2:", addr2)
print("privada addr2:",wallet.privkey(addr2))
```

Frase semilla: derivation path = m/44'/0'/0'/0/0

La ruta `m/44'/0'/0'/0/0` es una especificación de derivación de claves en el sistema de derivación jerárquica propuesto por BIP32. Vamos a descomponerla para entender su significado:

`m/44'/0'/0'/0/0`

Representa la clave maestra o master key. Es la raíz desde donde se empieza a derivar todas las otras claves

Frase semilla: derivation path = m/44'/0'/0'/0/0

`m/**44'**/0'/0'/0/0`

El número 44 con el apóstrofo (') indica el propósito de esta derivación. El "44" es un número fijo que se ha asociado con BIP44, una extensión de BIP32 que define una estructura específica para la derivación de claves para diferentes criptomonedas y monederos. El apóstrofo indica que es una derivación "endurecida" (hardened), lo que significa que no puedes derivar claves públicas hijo a partir de claves públicas padre (necesitarías la clave privada padre).

Frase semilla: derivation path = m/44'/0'/0'/0/0

`m/44'/**0**'/0'/0/0`

Este es el identificador de criptomoneda. "0" se ha asignado a Bitcoin según el estándar SLIP-0044. Otros números se han asignado a otras criptomonedas (por ejemplo, Ethereum tiene el número 60). Al igual que antes, el apóstrofo indica una derivación endurecida.

Frase semilla: derivation path = m/44'/0'/0'/0/0

`m/44'/0'/**0**'/0/0`

Este es el identificador del monedero o account. Si tienes múltiples monederos Bitcoin bajo una misma semilla, puedes incrementar este número para obtener claves para cada monedero diferente. Una vez más, es una derivación endurecida.

Frase semilla: derivation path = m/44'/0'/0'/0/0

`m/44'/0'/0'/**0**/0`

Representa la cadena o chain. En BIP44, hay dos cadenas principales que se consideran: 0 para direcciones externas (aquellas que puedes compartir públicamente para recibir fondos) y 1 para direcciones internas (utilizadas principalmente para las transacciones de cambio en Bitcoin).

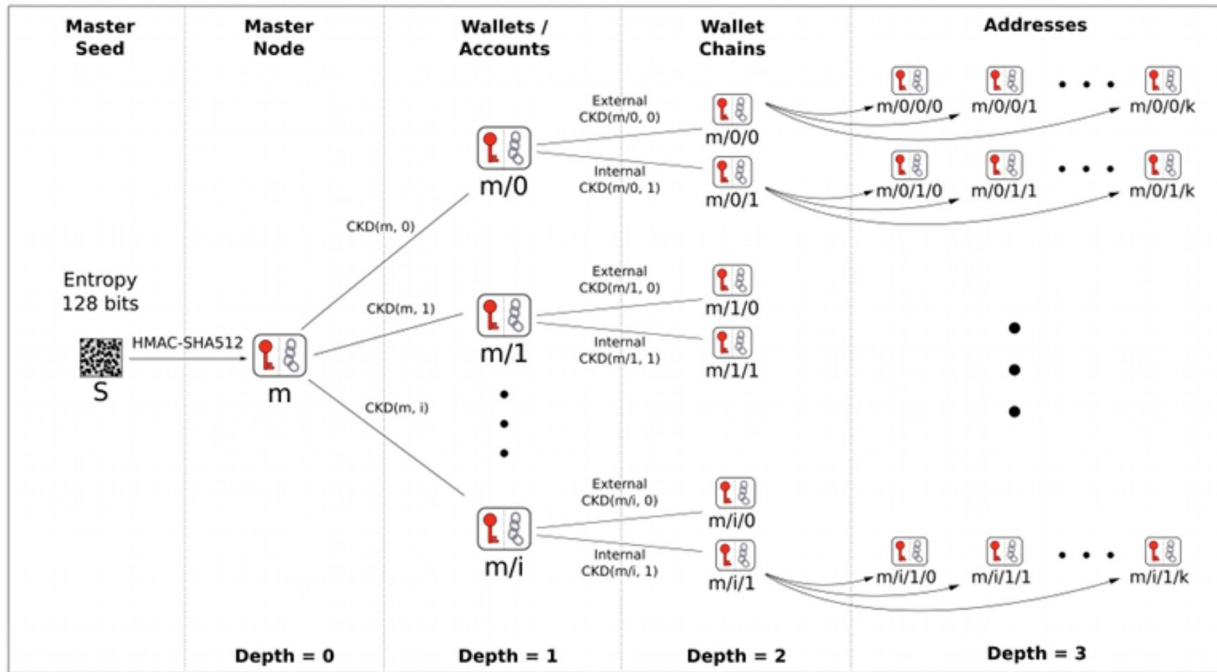
Frase semilla: derivation path = m/44'/0'/0'/0/0

`m/44'/0'/0'/0/**0**`

Este es el índice de la dirección. Si derivas muchas direcciones a partir de una misma cadena, este número se incrementará secuencialmente. Es decir, la primera dirección tendrá índice 0, la segunda índice 1, y así sucesivamente.

Frase semilla: derivation path


BIP 32 - Hierarchical Deterministic Wallets




Child Key Derivation Function $\sim CKD(x,n) = HMAC-SHA512(x_{Chain}, x_{PubKey} || n)$


Frase semilla: generación

Técnicamente, se puede usar cualquier palabra que se desee para una "frase semilla". Sin embargo, hay consideraciones importantes:

 **Entropía:** Una de las razones por las que se utilizan frases semilla generadas de manera aleatoria es para garantizar una alta entropía. Esto significa que hay una amplia variedad de posibles combinaciones de palabras, lo que hace que sea extremadamente difícil para alguien adivinar o usar fuerza bruta para descifrar tu frase semilla. Si se elige palabras propias sin tener en cuenta la aleatoriedad, se podría terminar con una frase semilla con entropía baja y, por lo tanto, vulnerable.

Frase semilla: generación

 **Memorabilidad:** Aunque puede ser tentador usar una frase que tenga un significado personal y que sea fácil de recordar, esto puede comprometer la seguridad si la frase es algo que alguien podría adivinar o inferir conociendo detalles sobre el propietario.

 **Compatibilidad:** Al usar software o hardware de billetera que espera frases semilla BIP39, podrías haber problemas si se usa una frase semilla personalizada que no cumple con el estándar.

Frase semilla: generación

✅ **Diccionario:** Las frases semilla generadas a partir de herramientas como BIP39 (un estándar ampliamente adoptado para frases semilla) utilizan un diccionario específico de 2048 palabras. Estas palabras están diseñadas para ser distintivas y reducir la posibilidad de confusión. Si se eligen palabras propias fuera de este conjunto, podría haber problemas para restaurar la billetera con software o hardware que espera que una frase semilla cumpla con el estándar BIP39.

✅ **Checksum:** El BIP39 también introduce una suma de comprobación (checksum) en la última palabra de la frase semilla para verificar que las palabras anteriores no han sido alteradas o ingresadas incorrectamente. Si se eligen palabras propias, no se tendría este mecanismo de comprobación a menos que se implementes la suma de comprobación.

Pero donde se usa la curva **secp256k1** entonces?

1. Se genera la seed phrase con suficiente entropía
2. La privada maestra generada de la seed phrase se realiza de la siguiente manera:
 - 2.1. Se convierten en una secuencia de bytes utilizando el algoritmo PBKDF2 con HMAC-SHA512 aplicado a la seed. El resultado es una "seed" de 512 bits
 - 2.2. Los primeros 256 bits del hash resultante se toman como la clave privada maestra. Esto es un número que debe ser menor que el orden de la curva, si no lo es, se pueden tomar bits menos significativos o modificar la seed ligeramente.
 - 2.3. Los siguientes 256 bits se usan como la "cadena de derivación maestra" o "master chain code".
 - 2.4. La clave privada maestra, como un número, se multiplica por el punto base de la curva elíptica secp256k1, G. El resultado es un punto en la curva, que es la clave pública maestra.
3. A partir de la clave privada maestra se generan las claves privadas hijas

Generación de claves privadas hijas

Supongamos que deseamos derivar la clave privada hija en el índice i desde la clave privada maestra.

1. Se concatena la clave pública maestra con el índice i (para derivaciones no endurecidas) o la clave privada maestra con el índice i (para derivaciones endurecidas).
2. Se calcula el HMAC-SHA512 de la concatenación.
3. Los primeros 256 bits del resultado se suman (módulo el orden de la curva secp256k1) a la clave privada maestra para obtener la clave privada hija.

Wallets

bip39:

<https://github.com/bitcoin/bips/blob/master/bip-0039/bip-0039-wordlists.md>

ejemplos en python: <https://github.com/primal100/pybitcointools>

faucet: <https://testnet-faucet.com/btc-testnet/>