**BÀI TẬP BUỔI 1**

# **Bài 1.** Trong blockchain có nhiều thuật toán chữ ký số và hàm băm được sử dụng, em hãy tìm hiểu các thuật toán này và thống kê lại trong bảng dữ liệu sau:

|  |  |  |
| --- | --- | --- |
| TT | Thuật toán băm  (mô tả rõ ứng dụng, tham số của các thuật toán) | Thuật toán chữ ký số  (mô tả rõ ứng dụng, tham số của các thuật toán) |
| 1 | SHA-256 (Secure Hash Algorithm 256-bit)  **Ứng dụng**: Được sử dụng trong Bitcoin và nhiều blockchain khác.  **Tham số**:  -Đầu vào: Một chuỗi dữ liệu có độ dài bất kỳ.  -Đầu ra: Một giá trị băm 256-bit (64 ký tự hex).  **Đặc tính**:  -Rất khó để tìm hai đầu vào khác nhau có cùng giá trị băm (collision resistance).  -Khả năng chống tiền ảnh và tiền ảnh thứ hai. | ECDSA (Elliptic Curve Digital Signature Algorithm)  **Ứng dụng**: Được sử dụng rộng rãi trong các blockchain như Bitcoin và Ethereum.  **Tham số**:  -Khóa riêng (private key): Một số nguyên ngẫu nhiên trong khoảng từ 1 đến 𝑛−1, trong đó n là số nguyên tố lớn.  -Khóa công khai (public key): Được tính từ khóa riêng và điểm sinh G trên đường cong elliptic.  -Thông điệp: Dữ liệu cần ký. |
| 2 | RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest)  **Ứng dụng**: Được sử dụng trong Bitcoin để tạo địa chỉ từ khóa công khai.  **Tham số**:  -Đầu vào: Một chuỗi dữ liệu có độ dài bất kỳ.  -Đầu ra: Một giá trị băm 160-bit (40 ký tự hex).  **Đặc tính**:  -Tương tự như SHA-256 nhưng với đầu ra ngắn hơn, phù hợp cho việc tạo địa chỉ. | RSA (Rivest-Shamir-Adleman)  **Ứng dụng**: Sử dụng trong nhiều hệ thống bảo mật, nhưng ít phổ biến hơn trong blockchain so với ECDSA.  **Tham số**:  -Khóa riêng (private key): Một cặp số nguyên (𝑑,𝑛)  -Khóa công khai (public key): Một cặp số nguyên (e,n).  -Thông điệp: Dữ liệu cần ký. |
| 3 | Keccak-256  **Ứng dụng**: Được sử dụng trong Ethereum.  **Tham số**:  -Đầu vào: Một chuỗi dữ liệu có độ dài bất kỳ.  -Đầu ra: Một giá trị băm 256-bit.  **Đặc tính**:  -Được thiết kế như một phần của gia đình SHA-3 nhưng với một vài khác biệt. | DSA (Digital Signature Algorithm)  **Ứng dụng**: Được tiêu chuẩn hóa bởi NIST và sử dụng trong một số hệ thống bảo mật.  **Tham số**:  -Khóa riêng (private key): Một số nguyên ngẫu nhiên.  -Khóa công khai (public key): Được tính từ khóa riêng và các tham số hệ thống p,q,g.  -Thông điệp: Dữ liệu cần ký. |
| 4 | Blake2  **Ứng dụng**: Blake2 được thiết kế như một thuật toán băm nhanh hơn và an toàn hơn SHA-2 và MD5. Nó có thể được sử dụng trong các ứng dụng cần tốc độ cao và độ an toàn cao.  **Tham số**:  Blake2b, Blake2s: Các biến thể với Blake2b tối ưu cho nền tảng 64-bit và Blake2s tối ưu cho nền tảng 8-bit đến 32-bit. | EdDSA (Edwards-curve Digital Signature Algorithm)  **Ứng dụng**: Được sử dụng trong một số blockchain như Monero, Stellar, và Cardano.  **Tham số**:  -Khóa riêng (private key): Một số ngẫu nhiên.  -Khóa công khai (public key): Được tính từ khóa riêng và điểm sinh trên đường cong elliptic Edwards (Ed25519 là một biến thể phổ biến).  -Thông điệp: Dữ liệu cần ký. |
| 5 | Skein  **Ứng dụng**: Skein là một thành viên của gia đình hàm băm được đề xuất trong cuộc thi SHA-3 và được sử dụng trong một số hệ thống bảo mật.  **Tham số**:  Skein-256, Skein-512, Skein-1024: Các biến thể với kích thước đầu ra khác nhau (256-bit, 512-bit, và 1024-bit). | Schnorr Signature  **Ứng dụng**: Được đề xuất để sử dụng trong Bitcoin (BIP-340) và một số blockchain khác.  **Tham số**:  -Khóa riêng (private key): Một số nguyên ngẫu nhiên.  -Khóa công khai (public key): Được tính từ khóa riêng và điểm sinh trên đường cong elliptic.  -Thông điệp: Dữ liệu cần ký. |
| 6 | Whirlpool  **Ứng dụng**: Whirlpool là một thuật toán băm mật mã được thiết kế bởi Vincent Rijmen và Paulo S. L. M. Barreto, sử dụng trong một số ứng dụng bảo mật dữ liệu.  **Tham số**:  Đầu ra: 512-bit. | BLS (Boneh-Lynn-Shacham) Signature  **Ứng dụng**: Sử dụng trong các hệ thống yêu cầu tổng hợp chữ ký (aggregate signatures), ví dụ như Dfinity và Algorand.  **Tham số**:  -Khóa riêng (private key): Một số nguyên ngẫu nhiên.  -Khóa công khai (public key): Được tính từ khóa riêng và một điểm trên đường cong elliptic đặc biệt (pairing-based cryptography).  -Thông điệp: Dữ liệu cần ký. |
| 7 | Streebog  **Ứng dụng**: Streebog là một hàm băm mật mã được thiết kế bởi Trung tâm Nghiên cứu Quốc gia về Truyền thông và Công nghệ Thông tin Liên bang Nga và được sử dụng trong tiêu chuẩn GOST R 34.11-2012.  **Tham số**:  Streebog-256, Streebog-512: Các biến thể với kích thước đầu ra khác nhau (256-bit và 512-bit). | Lamport Signature  **Ứng dụng**: Một dạng chữ ký số một lần (one-time signature) sử dụng trong một số hệ thống yêu cầu tính bảo mật cao như các hệ thống hậu lượng tử.  **Tham số**:  -Khóa riêng (private key): Một tập hợp các giá trị ngẫu nhiên.  -Khóa công khai (public key): Được tính từ khóa riêng và hàm băm.  -Thông điệp: Dữ liệu cần ký. |

# **Bài 2.** Lập trình mô phỏng quá trình ký và kiểm tra chữ ký một giao dịch trong blockchain với lược đồ chữ ký ECDSA.

a) Để lập trình mô phỏng quá trình ký và kiểm tra chữ ký một giao dịch trong blockchain với lược đồ chữ ký ECDSA (Elliptic Curve Digital Signature Algorithm), chúng ta có thể sử dụng Python và thư viện ecdsa. Sau đây là các bước chi tiết để thực hiện:

1-Tạo khóa riêng và khóa công khai ECDSA

2-Ký một giao dịch

3-Kiểm tra chữ ký

b) Dưới đây là mã nguồn mô phỏng chi tiết:

Cài đặt thư viện cần thiết

Trước tiên, bạn cần cài đặt thư viện ecdsa nếu chưa có:

|  |
| --- |
| pip install ecdsa |

Mã nguồn:

|  |
| --- |
| import hashlib  from ecdsa import SigningKey, VerifyingKey, SECP256k1  def sha256(message):  return hashlib.sha256(message.encode('utf-8')).digest()  # 1. Tạo khóa riêng và khóa công khai  private\_key = SigningKey.generate(curve=SECP256k1)  public\_key = private\_key.get\_verifying\_key()  print("Khóa riêng:", private\_key.to\_string().hex())  print("Khóa công khai:", public\_key.to\_string().hex())  # 2. Ký một giao dịch  transaction = "Gửi 1 BTC từ A đến B"  transaction\_hash = sha256(transaction)  signature = private\_key.sign(transaction\_hash)  print("Giao dịch:", transaction)  print("Hash của giao dịch:", transaction\_hash.hex())  print("Chữ ký:", signature.hex())  # 3. Kiểm tra chữ ký  try:  public\_key.verify(signature, transaction\_hash)  print("Chữ ký hợp lệ")  except:  print("Chữ ký không hợp lệ") |

c) Giải thích mã nguồn

1-Tạo khóa riêng và khóa công khai:

-Sử dụng SigningKey.generate(curve=SECP256k1) để tạo khóa riêng dùng đường cong elliptic SECP256k1.

-Lấy khóa công khai từ khóa riêng bằng phương thức get\_verifying\_key().

2-Ký một giao dịch:

-Tạo một thông điệp giao dịch (ví dụ: "Gửi 1 BTC từ A đến B").

-Tính băm của thông điệp giao dịch sử dụng hàm SHA-256.

-Sử dụng khóa riêng để ký băm của giao dịch.

3-Kiểm tra chữ ký:

-Sử dụng khóa công khai để kiểm tra chữ ký bằng phương thức verify().

-Nếu chữ ký hợp lệ, in ra "Chữ ký hợp lệ", ngược lại, in ra "Chữ ký không hợp lệ".

d) Chạy

Khi bạn chạy mã nguồn này, nó sẽ tạo ra một cặp khóa ECDSA, ký một giao dịch và kiểm tra tính hợp lệ của chữ ký. Bạn có thể thay đổi thông điệp giao dịch và thử lại để thấy kết quả thay đổi.

Đây là một cách đơn giản và hiệu quả để mô phỏng quá trình ký và kiểm tra chữ ký một giao dịch trong blockchain sử dụng ECDSA.

# **Bài 3.** Tìm hiểu về cơ chế sinh địa chỉ trên blockchain và lập trình mô phỏng cơ chế này

a) Cơ chế sinh địa chỉ trên blockchain

\*) Địa chỉ trên blockchain đóng vai trò quan trọng trong việc nhận và gửi các giao dịch. Chúng được sinh ra từ các khóa mật mã và đảm bảo rằng mỗi địa chỉ là duy nhất. Dưới đây là các bước chung trong quy trình sinh địa chỉ trên blockchain, bất kể loại blockchain nào.

1. Tạo Khóa Riêng (Private Key)

Khóa riêng là một chuỗi số ngẫu nhiên, thường là 256-bit, đại diện cho một giá trị duy nhất. Khóa riêng được dùng để ký các giao dịch và tạo khóa công khai.

Ngẫu nhiên: Khóa riêng phải được tạo một cách ngẫu nhiên và bảo mật để đảm bảo rằng không ai có thể đoán được nó.

2. Tạo Khóa Công Khai (Public Key)

Khóa công khai được tạo ra từ khóa riêng bằng cách sử dụng các thuật toán mật mã, chẳng hạn như Elliptic Curve Cryptography (ECC).

Bất đối xứng: Khóa công khai và khóa riêng tạo thành một cặp khóa bất đối xứng, trong đó khóa công khai có thể được chia sẻ công khai mà không ảnh hưởng đến bảo mật của khóa riêng.

Thuật toán: Thuật toán thường dùng là ECDSA (Elliptic Curve Digital Signature Algorithm) với các đường cong elliptic như secp256k1.

3. Băm Khóa Công Khai

Băm: Khóa công khai thường được băm để tạo ra một giá trị cố định độ dài. Các hàm băm phổ biến bao gồm SHA-256 và RIPEMD-160.

An toàn: Quá trình băm đảm bảo rằng dù khóa công khai có độ dài khác nhau, kết quả băm luôn có độ dài cố định và bảo mật.

4. Thêm Các Thông Tin Phiên Bản và Checksum

Phiên bản: Một byte phiên bản có thể được thêm vào trước giá trị băm để chỉ định loại địa chỉ hoặc mạng (ví dụ: mainnet, testnet).

Checksum: Một mã checksum được tạo ra bằng cách băm lại giá trị đã có phiên bản để phát hiện lỗi khi nhập địa chỉ. Checksum thường là một phần của giá trị băm cuối cùng.

5. Mã hóa Địa chỉ

Mã hóa Base58 hoặc Bech32: Địa chỉ thường được mã hóa thành một chuỗi ký tự dễ đọc và sử dụng. Base58 và Bech32 là các chuẩn phổ biến để mã hóa địa chỉ.

Base58: Loại bỏ các ký tự dễ gây nhầm lẫn như 0, O, I, l.

Bech32: Được thiết kế để giảm lỗi và hỗ trợ tốt hơn cho phân tách phần địa chỉ.

Ví dụ Khái Quát về Các Bước

Khóa riêng: Một số ngẫu nhiên 256-bit được tạo ra.

Khóa công khai: Sử dụng ECDSA và khóa riêng để tạo khóa công khai.

Băm khóa công khai: Sử dụng SHA-256 để băm khóa công khai.

Băm lần 2: Sử dụng RIPEMD-160 (hoặc một hàm băm khác) để băm kết quả của SHA-256.

Thêm phiên bản: Thêm một byte chỉ định phiên bản vào trước kết quả băm.

Checksum: Băm SHA-256 hai lần lên kết quả đã thêm phiên bản và lấy 4 byte đầu tiên làm checksum.

Địa chỉ: Thêm checksum vào cuối kết quả băm đã thêm phiên bản và mã hóa toàn bộ thành chuỗi ký tự sử dụng Base58 hoặc Bech32.

\*) Đảm Bảo Bảo Mật và Độc Nhất

Bảo mật khóa riêng: Bảo mật của địa chỉ blockchain dựa trên việc bảo mật khóa riêng. Nếu khóa riêng bị lộ, tất cả tài sản liên quan đến địa chỉ đó đều có thể bị đánh cắp.

Độc nhất: Mỗi khóa riêng sẽ tạo ra một khóa công khai và địa chỉ duy nhất. Xác suất để hai khóa riêng khác nhau tạo ra cùng một địa chỉ là cực kỳ thấp.

\*) Các Thuật Toán và Tiêu Chuẩn Liên Quan

ECDSA và Elliptic Curve Cryptography: Được sử dụng rộng rãi trong các blockchain như Bitcoin, Ethereum và nhiều blockchain khác.

SHA-256 và RIPEMD-160: Các hàm băm phổ biến đảm bảo tính bảo mật và tính toàn vẹn dữ liệu.

Base58 và Bech32: Các phương pháp mã hóa giúp địa chỉ dễ đọc và giảm thiểu lỗi nhập liệu.

b) Lập trình mô phỏng

\*) Dưới đây là một ví dụ về cơ chế sinh địa chỉ cho Ethereum

Quy trình sinh địa chỉ Ethereum

-Tạo khóa riêng: Khóa riêng là một số ngẫu nhiên 256-bit.

-Tạo khóa công khai từ khóa riêng: Sử dụng thuật toán ECDSA với đường cong secp256k1.

-Tạo địa chỉ Ethereum từ khóa công khai:

+Băm khóa công khai bằng Keccak-256.

+Lấy 20 byte cuối cùng của kết quả băm làm địa chỉ Ethereum.

\*) Mã nguồn Python

Bạn cần cài đặt các thư viện cần thiết trước khi chạy mã này:

|  |
| --- |
| pip install ecdsa eth-hash |

Mã nguồn:

|  |
| --- |
| import os  from ecdsa import SigningKey, SECP256k1  from eth\_hash.auto import keccak  def generate\_ethereum\_address():  # 1. Tạo khóa riêng  private\_key = SigningKey.generate(curve=SECP256k1)  private\_key\_bytes = private\_key.to\_string()  print("Khóa riêng:", private\_key\_bytes.hex())  # 2. Tạo khóa công khai từ khóa riêng  public\_key = private\_key.get\_verifying\_key()  public\_key\_bytes = b'\04' + public\_key.to\_string()  print("Khóa công khai:", public\_key\_bytes.hex())  # 3. Băm khóa công khai bằng Keccak-256  keccak\_hash = keccak(public\_key\_bytes[1:]) # Bỏ byte đầu tiên (\04)  print("Keccak-256 của khóa công khai:", keccak\_hash.hex())  # 4. Lấy 20 byte cuối cùng của Keccak-256 hash làm địa chỉ  address = keccak\_hash[-20:]  ethereum\_address = '0x' + address.hex()  print("Địa chỉ Ethereum:", ethereum\_address)  return ethereum\_address  # Gọi hàm để tạo địa chỉ Ethereum  ethereum\_address = generate\_ethereum\_address()  print("Địa chỉ Ethereum được tạo:", ethereum\_address) |

\*) Giải thích mã nguồn

-Tạo khóa riêng:

+Sử dụng SigningKey.generate(curve=SECP256k1) để tạo khóa riêng dùng đường cong elliptic SECP256k1.

-Tạo khóa công khai từ khóa riêng:

+Lấy khóa công khai từ khóa riêng bằng phương thức get\_verifying\_key() và định dạng lại để có prefix \04.

-Băm khóa công khai bằng Keccak-256:

+Băm khóa công khai bằng Keccak-256 (trong Ethereum, Keccak-256 được sử dụng thay vì SHA-256).

+Bỏ byte đầu tiên của khóa công khai trước khi băm.

-Tạo địa chỉ Ethereum:

+Lấy 20 byte cuối cùng của kết quả băm Keccak-256 để tạo địa chỉ Ethereum.

+Thêm prefix 0x để biểu thị đây là địa chỉ Ethereum.

\*) Khi chạy mã này, bạn sẽ nhận được một địa chỉ Ethereum mới được tạo ra từ khóa công khai. Quy trình này đơn giản và đảm bảo rằng địa chỉ là duy nhất và an toàn để chia sẻ công khai.

# **Bài 4.** Tìm hiểu về Proof of Work và lập trình mô phỏng hoạt động của Proof of Work.

a) Proof of Work

Proof of Work (PoW) là một cơ chế đồng thuận được sử dụng trong nhiều blockchain, bao gồm Bitcoin, để ngăn chặn các hành vi gian lận như tấn công double-spend và đảm bảo tính toàn vẹn của blockchain. Nguyên lý cơ bản của PoW là yêu cầu các node (thợ mỏ) giải quyết một bài toán mật mã phức tạp để có quyền thêm một khối mới vào blockchain.

Thợ mỏ (Miners): Các node trong mạng lưới thực hiện công việc tính toán để giải bài toán PoW.

Bài toán PoW: Thường là tìm một giá trị nonce sao cho hàm băm của khối mới có giá trị nhỏ hơn một ngưỡng nhất định (target).

Hàm băm (Hash Function): Một hàm mật mã chuyển đổi dữ liệu đầu vào có kích thước tùy ý thành một giá trị băm có độ dài cố định. SHA-256 là một hàm băm phổ biến trong PoW.

Nonce: Một số ngẫu nhiên được thay đổi trong quá trình băm để tìm ra giá trị băm thỏa mãn yêu cầu.

Target: Mức độ khó khăn của bài toán PoW, được điều chỉnh để duy trì thời gian tạo khối ổn định.

Quy Trình Proof of Work:

Chuẩn bị khối: Thợ mỏ tập hợp các giao dịch hợp lệ thành một khối.

Tạo băm khối: Thợ mỏ băm khối bằng cách thay đổi giá trị nonce cho đến khi tìm ra một băm thỏa mãn điều kiện target.

Phát tán khối: Khi tìm ra băm hợp lệ, thợ mỏ phát tán khối mới vào mạng lưới để các node khác xác minh.

Xác minh: Các node khác xác minh tính hợp lệ của khối và nếu hợp lệ, khối được thêm vào blockchain.

b) Lập Trình Mô Phỏng Proof of Work

Dưới đây là một ví dụ đơn giản bằng Python để mô phỏng hoạt động của Proof of Work. Ví dụ này sử dụng thư viện hashlib để thực hiện băm SHA-256.

Mã Nguồn Python:

|  |
| --- |
| import hashlib  import time  def sha256(data):  return hashlib.sha256(data.encode('utf-8')).hexdigest()  def proof\_of\_work(block\_data, difficulty):  nonce = 0  target = '0' \* difficulty  start\_time = time.time()  while True:  data = f"{block\_data}{nonce}"  hash\_value = sha256(data)  if hash\_value.startswith(target):  end\_time = time.time()  print(f"Nonce: {nonce}")  print(f"Hash: {hash\_value}")  print(f"Time taken: {end\_time - start\_time} seconds")  return nonce, hash\_value  nonce += 1  if \_\_name\_\_ == "\_\_main\_\_":  block\_data = "Example block data"  difficulty = 4 # Number of leading zeros required in hash  nonce, hash\_value = proof\_of\_work(block\_data, difficulty)  print(f"Block data: {block\_data}")  print(f"Nonce found: {nonce}")  print(f"Hash of block: {hash\_value}") |

Giải Thích Mã Nguồn:

1-Hàm sha256(data):

-Nhận dữ liệu đầu vào và trả về giá trị băm SHA-256 của dữ liệu đó.

2-Hàm proof\_of\_work(block\_data, difficulty):

-block\_data: Dữ liệu khối cần được băm.

-difficulty: Độ khó của bài toán PoW, xác định số lượng chữ số 0 dẫn đầu cần có trong giá trị băm.

-nonce: Một giá trị thay đổi được để tìm giá trị băm phù hợp.

-target: Chuỗi ký tự chứa số lượng chữ số 0 dẫn đầu theo độ khó.

3-Quy trình:

-Bắt đầu từ nonce bằng 0 và tăng dần.

-Kết hợp block\_data và nonce, sau đó băm kết quả.

-Kiểm tra xem băm có bắt đầu với target (số lượng chữ số 0 dẫn đầu) hay không.

-Nếu tìm thấy băm phù hợp, in ra nonce, giá trị băm, và thời gian thực hiện.

Thử Nghiệm:

-Chạy mã nguồn và quan sát kết quả đầu ra.

-Thay đổi giá trị difficulty để thấy sự khác biệt về thời gian tìm nonce.