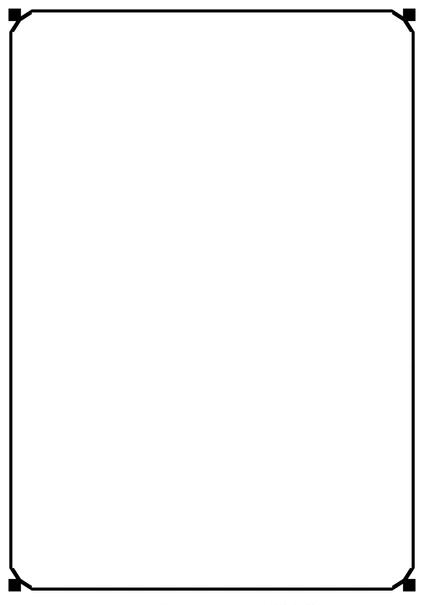
**Học viện kỹ thuật mật mã phân hiệu Tp. Hồ Chí Minh**

**Khoa: An toàn thông tin**

————————

**BÁO CÁO NGHIÊN CỨU**

**Môn: Cơ sở an toàn thông tin**

**Hệ mật trên đường cong Elliptic**

**và ứng dụng blockchain**

Giảng viên hướng dẫn: Th.S Đặng Thế Hùng  
 Sinh viên thực hiện:  
 Ngô Quang Khánh  
 Nguyễn Mạnh Tiến  
 Thân Minh Tuấn Phát

Tp. Hồ Chí Minh, tháng 8 năm 2022

**Mục lục**

[**Phần 1: Tổng quan cơ sở lý thuyết**](#_hxzubf912vt6) **5**

[**Chương I: Giới thiệu**](#_vkpanojv9l02) **5**

**1.1.** [**Đặt vấn đề**](#_cmm1h44lk1n9) **5**

**1.2.** [**Tính cấp thiết**](#_f0ty172d8u80) **5**

**1.3.** [**Mục tiêu nghiên cứu**](#_cihlxp6hut66) **5**

**1.4.** [**Giới hạn nghiên cứu**](#_uh1sgz5kx3p0) **6**

[**Chương II: Tìm hiểu về ECDSA**](#_36k2qlciwjwz) **7**

**2.1.** [**Khái niệm**](#_xm0sr7dsg4fm) **7**

**2.2.** [**Lịch sử hình thành và phát triển**](#_ni04m6fv10ku) **7**

**2.3.** [**Thuật toán**](#_xk8p42tc7d7e) **7**

**2.4.** [**Thế mạnh của ECDSA**](#_xgvlssjntu14) **14**

[**Chương III: Tìm hiểu về công nghệ Blockchain**](#_yj4rv8cfbalf) **16**

**3.1.** [**Khái niệm**](#_balze8djhoe1) **16**

**3.2.** [**Lịch sử hình thành và phát triển**](#_tfwcw52yuujz) **16**

**3.3.** [**Kiến trúc Blockchain**](#_5dof5k3e2clp) **17**

[**3.3.1 Chuỗi khối (block chain)**](#_wjh63vf1wflp) **17**

[**3.3.2 Nút (node)**](#_sksrwbv4d3kf) **20**

[**3.3.3 Khai thác**](#_e4cgxgz1a96q) **21**

**3.4.** [**Cách hoạt động của blockchain**](#_ebgaifyhcvcc) **22**

**3.5.** [**Ứng dụng**](#_i2af209rf8x1) **24**

**3.6.** [**Ưu và nhược điểm**](#_adod3ybwj1te) **25**

[**3.6.1 Ưu điểm công nghệ Blockchain**](#_2em2x95j16de) **25**

[**3.6.2 Nhược điểm công nghệ Blockchain**](#_xsvnlsidb56q) **26**

[**Phần 2: Triển khai ứng dụng Blockchain sử dụng ECDSA**](#_j58o7y7pffnp) **27**

[**Chương I: Triển khai ECDSA**](#_vvw8ewmmmqs6) **27**

[**Chương II: Triển khai Blockchain**](#_ngsatine7qyb) **29**

[**Chương III: Triển khai ứng dụng Blockchain**](#_pqgwornaf682) **35**

[**Phần 3: Đánh giá và kết luận nghiên cứu**](#_73ukxivc7h2u) **54**

[**Chương I: Đánh giá kết quả**](#_3jo5pq6febsh) **54**

[**Chương II: Kết luận đề tài**](#_bxubzungbhv) **54**

[**Bảng phân công:**](#_qwtyn3hgexys) **55**

[**Tài liệu tham khảo**](#_twf6y9ajui7r) **56**

# **Phần 1: Tổng quan cơ sở lý thuyết**

# **Chương I: Giới thiệu**

## Đặt vấn đề

Xã hội loài người đã phát triển từ thời kỳ đồ đá cho đến hiện tại, thời kỳ công nghệ kỹ thuật số. Đời sống của con người từ đó được cải thiện nhiều hơn so với quá khứ nhưng không dừng lại ở đó, con người tiếp tục phát minh ra nhiều công nghệ và các sản phẩm công nghệ mới theo từng năm. Công nghệ số đã được ứng dụng vào nhiều ngành nghề khác nhau như nông nghiệp, sản xuất, thông kê, tài chính… Nhưng công nghệ càng phát triển, con người càng gặp nhiều mối nguy từ công nghệ của họ phát triển như bị mất quyền riêng tư, mất đi sự bảo mật thông tin và nhiều thứ khác đang và có thể sắp diễn ra.

Đặc biệt về vấn đề tài chính, con người thường gửi tiền vào các ngân hàng để bảo vệ tiền của họ khỏi trộm đột nhập vào nhà. Nhưng các ngân hàng hiện nay đã xuất hiện ở khắp nơi và các vụ tham nhũng, lừa đảo, phá sản diễn ra làm cho tâm lý của con người ngày càng bất an với các ngân hàng. Việc xuất hiện hình thức tài chính phi tập trung (Decentralized Finance - DeFi) dựa trên công nghệ Blockchain như một cách trấn an tâm lý người dùng. Với hình thức tài chính này, người dùng không còn phải nhờ bên thứ 3 bảo vệ tài sản nữa.

Nhưng để thuyết phục được người dùng sử dụng hình thức tài chính phi tập trung, công nghệ Blockchain phải thể hiện được tính bảo mật tuyệt đối đối với một tương lai dài, nhờ vậy kết hợp với hệ mật dựa trên đường cong Elliptic từ đó ra đời các loại tiền mã hoá, kỹ thuật số như Bitcoin, Ethereum.

## Tính cấp thiết

Với bài nghiên cứu này, các sinh viên thực hiện sẽ củng cố thêm kiến thức mật mã đã được học trên giảng đường và được ứng dụng thực tế để biết khả năng bảo mật thực tế của thuật toán. Ngoài ra, các sinh viên sẽ tìm hiểu công nghệ đang được bùng nổ gần đây là Blockchain.

## Mục tiêu nghiên cứu

Mục tiêu của bài nghiên cứu:

* Đầu tiên, hiểu và có thể áp dụng ECDSA một cách hiệu quả bằng ngôn ngữ lập trình Python
* Thứ hai, hiểu và tạo ứng dụng dựa trên công nghệ Blockchain bằng ngôn ngữ lập trình Python
* Thứ ba, kết hợp ECDSA và công nghệ Blockchain một cách hiệu quả bằng ngôn ngữ lập trình Python

## Giới hạn nghiên cứu

Do chủ đề của bài nghiên cứu rất rộng nên nhóm sẽ giới hạn một số thứ và cũng như sẽ nói một số giới hạn của bạn thân đối với bài nghiên cứu:

* ECDSA là một thuật toán phức tạp nên nhóm sẽ chỉ nghiên cứu về lý thuyết sau đó sẽ áp dụng thư viện có sẵn từ Python
* Blockchain có rất nhiều thứ cần phải quan tâm nên nhóm sẽ vẫn trình bày lý thuyết và viết ra một đồng tiền điện tử dùng Python nhưng sẽ không đầy đủ tất cả các chức năng cần có. Ít nhất nó sẽ thể hiện được tính chất của blockchain.
* Cuối cùng là thời gian làm bài hơn 1 tháng nên nhóm sẽ cố gắng để có đầy đủ hoặc hơn những mục tiêu đề ra.

# **Chương II: Tìm hiểu về ECDSA**

## **Khái niệm**

ECDSA là viết tắt của Elliptic Curve Digital Signature Algorithm thuật toán sinh chữ ký số dựa trên đường cong Elliptic.

ECDSA được sử dụng để tạo chữ kí số cho dữ liệu, giúp chống lại sự giả mạo cũng như làm sai lệch dữ liệu, cung cấp một phương pháp xác thực mà không ảnh hưởng đến tính bảo mật của dữ liệu gốc.

ECDSA được ứng dụng rộng rãi trong rất nhiều lĩnh vực cần tính bảo mật và sự riêng tư dữ liệu, đặc biệt như trong Blockchain.

## **Lịch sử hình thành và phát triển**

Thuật toán sinh chữ ký số dựa trên đường cong Elliptic là phép toán tương tự với đường cong elliptic của DSA. ECDSA lần đầu tiên được đề xuất vào năm 1992 bởi Scott Vanstone trong hồi đáp gửi tới NIST’s (Viện Tiêu chuẩn và Công nghệ Quốc gia) để yêu cầu lấy ý kiến ​​cộng đồng về đề xuất đầu tiên của họ đối với DSS (Digital Signature Standard). Nó đã được chấp nhận vào năm 1998 như là một tiêu chuẩn của ISO (Tổ chức Tiêu chuẩn Quốc tế) (ISO 14888-3), được chấp nhận vào năm 1999 như một tiêu chuẩn Tiêu chuẩn ANSI (Viện Tiêu chuẩn Quốc gia Hoa Kỳ) (ANSI X9.62), và được chấp nhận vào năm 2000 với tư cách là một tiêu chuẩn IEEE (Viện Kỹ sư Điện và Điện tử) (IEEE 1363-2000) và tiêu chuẩn FIPS (FIPS 186-2). Nó cũng đang được xem xét để đưa vào một số tiêu chuẩn ISO khác.

## **Thuật toán**

Như đã nêu ở khái niệm thì ECDSA là thuật toán sinh chữ ký số dựa trên đường cong Elliptic, nên trước hết ta sẽ tìm hiểu về các công thức và phép toán của đường cong Elliptic, cụ thể là trong blockchain.

### **Công thức**

Công thức của đường cong Elliptic là:



Bitcoin hay Ethereum sử dụng một đường cong theo tiêu chuẩn [secp256k1](https://en.bitcoin.it/wiki/Secp256k1) do *Viện Tiêu Chuẩn và Kỹ Thuật Quốc Gia Mỹ* (NIST) đặt ra.

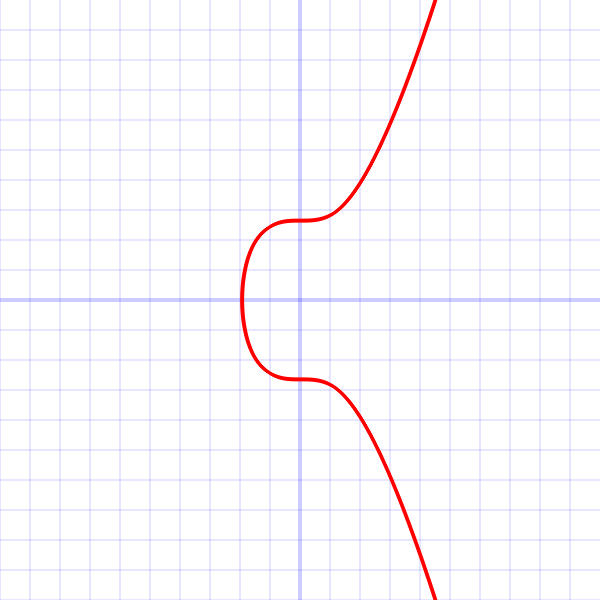
Đường cong này có công thức như sau:



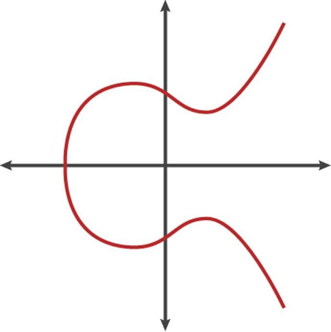
với p là một số nguyên tố rất lớn,



Hình dưới là minh họa cho đường cong elliptic được sử dụng trong Bitcoin và Ethereum:



Hình dưới minh hoạ đường cong elliptic:



### Các phép toán trên đường cong Elliptic

Có 2 phép toán quan trọng trên đường cong Elliptic: phép cộng và phép nhân

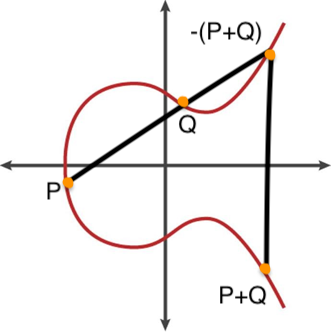
### Phép cộng

Đường cong Elliptic có một tính chất: "Nếu hai điểm P và Q nằm trên đường cong, thì điểm P+Q cũng sẽ nằm trên đường cong".

Điểm này được xác định như sau:

* Vẽ đường thẳng nối 2 điểm P và Q, đường thẳng này sẽ cắt đường cong tại một điểm nữa
* Lấy đối xứng của điểm này qua trục hoành, ta sẽ có được P+Q

Hình dưới mô tả phép cộng được tiến hành trong đường cong Elliptic thế nào:



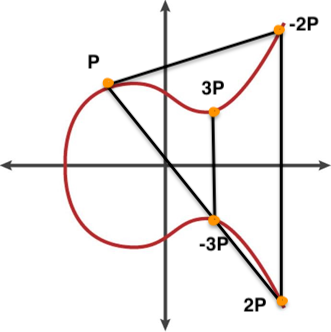
Chú thích: Nếu 3 điểm trên đường cong Elliptic là thẳng hàng, thì tổng của chúng bằng 0.

### Phép nhân

Trên đường cong Elliptic, việc nhân một điểm với một hằng số không đơn thuần chỉ là lấy từng toạ độ rồi nhân là xong. Thực chất, phép nhân ở đây vẫn là phép cộng, nhưng thực hiện nhiều lần mà thôi.

Ví dụ trong phép toán tính 3P, đầu tiên ta sẽ tính 2P bằng cách tính P+P. Theo cách cộng ở bên trên, ta vẽ đường thẳng nối P và P, ở đây chính là tiếp tuyến của đường cong, nó cắt đường cong tại điểm -2P, lấy đối xứng qua trục hoành ta có 2P. Tiếp tục vẽ đường thẳng nối giữa 2P và P , cắt đường cong tại -3P, lấy đối xứng ta có 3P.

Hình dưới mô tả phép nhân được tiến hành trong đường cong Elliptic thế nào:



### **Tạo private key**:

Một khoá bí mật - private key chỉ đơn thuần là một con số được chọn ra ngẫu nhiên. Đúng như cái tên của nó, private key cần được giữ bí mật, nên việc chọn ra số ngẫu nhiên phải vô cùng an toàn và đảm bảo tính thực sự ngẫu nhiên để tránh các cuộc tấn công vét cạn hay các cuộc tấn công khác nhằm lấy được private key.

### Tạo public key:

Khác với private key, public key được công khai cho tất cả mọi người.

public key được tạo ra bởi phép nhân với private key trong đường cong Elliptic, ta sẽ nói rõ hơn ở phần tiếp theo.

Phép nhân đường cong Elliptic là một phép toán sử dụng hàm trapdoor (cửa lật), có nghĩa là nó dễ tính theo một chiều (phép nhân) và không thể tính được theo chiều ngược lại (phép chia).

Do đó người sở hữu private key có thể dễ dàng tạo ra public key và yên tâm chia sẻ với mọi người mà không lo lắng rằng ai đó có thể đảo ngược public key để chiếm lấy private key của mình.

Lý thuyết này tạo nên nên tảng cho các chữ ký số an toàn và không thể làm giả, ví dụ được dùng để chứng minh quyền sở hữu đối với Bitcoin hay Ethereum trong blockchain.

Ta đã có một private key là một số ngẫu nhiên 

Trên đường cong Elliptic ta chọn một điểm G, gọi là điểm sinh hay được gọi là điểm cơ sở.

Public key  được sinh ra bằng kết quả của phép nhân:



với Bitcoin hay Ethereum thì:

G = **04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8** (Ở dạng chưa nén)

G = **02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798** (Ở dạng nén)

theo tiêu chuẩn [secp256k1](https://en.bitcoin.it/wiki/Secp256k1).

Tất nhiên  cũng sẽ là một điểm trên đường cong Elliptic.

Mối quan hệ giữa  và  là cố định, và chỉ tính được theo một chiều từ  đến  . Đó là lý do tại sao ta có thể sinh ra khoá công khai từ khoá bí mật và có thể chia sẻ khoá công khai này với tất cả mọi người, mà không thể dùng khoá công khai để tìm ngược lại về khoá bí mật.

### Tạo chữ ký số:

Đầu tiên, bạn cần biết rằng chữ ký có kích thước 40 byte và được biểu diễn bằng hai giá trị mỗi giá trị 20 byte, giá trị đầu tiên được gọi là r và giá trị thứ hai được gọi là s. Vì vậy cặp (r, s) chính là chữ ký ECDSA của bạn.

Để tạo ra cặp (r, s) này đầu tiên ra sẽ phải chọn ra một số ngẫu nhiên k (lưu ý đây là một số ngẫu nhiên khác với private key). Sau đó nhân k với điểm sinh G:



khi này ta có được một điểm P(x, y), toạ độ x của P chính là giá trị r.

Để tính s, bạn phải tạo hàm băm SHA1 của message, nó cung cấp cho bạn một giá trị 20 byte mà bạn sẽ coi là một số nguyên rất lớn và ta sẽ gọi nó là ‘z‘. Bây giờ bạn có thể tính s bằng công thức:



Ta lưu ý ở đây  là nghịch đảo theo modulo của k, xem thêm tại ([modular multiplicative inverse](https://en.wikipedia.org/wiki/Modular_multiplicative_inverse)) chứ không phải nghịch đảo đơn thuần trong tính toán thập phân là  .

Nếu trong tính toán thập phân  thì theo modulo 

### Xác nhận chữ ký:

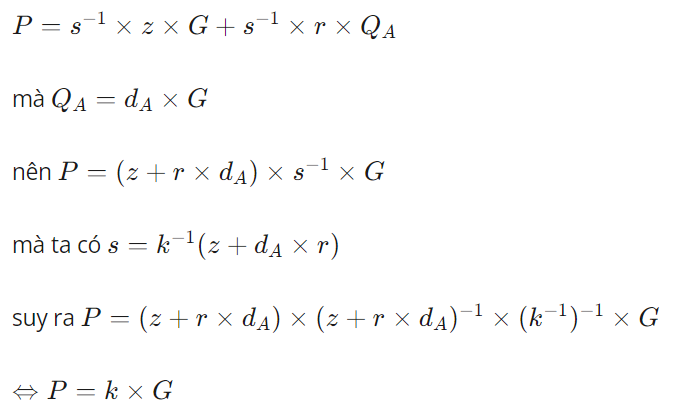
Để xác minh tính hợp lệ của chữ ký, ta chỉ cần public key là đủ:

Bằng cách tính:



Nếu toạ độ x của P bằng r, có nghĩa là chữ ký là hợp lệ. Ngược lại thì không hợp lệ.

Ta sẽ chứng minh điều này như sau:



Đây chính là công thức mà ta dùng để tính là P khi tạo ra chữ ký, vậy ta có điều phải chứng minh.

**Hàm trapdoor trong thuật toán ECDSA:**

**1.Giải thích đơn giản về trapdoor:**

Để giải thích đơn giản và dễ hiểu nhất thì lấy ví dụ sẽ là dễ nhất

1.1 Ví dụ không phải là Trapdoor:

Ta có A + B = C, nếu ta có được A và B thì ta dễ dàng tính được C. Tuy nhiên nếu có B và C thì ta cũng dễ dàng tính được A, thì qua ví dụ này ta sẽ thấy rõ ràng đây không phải là hàm Trapdoor.

1.2 Ví dụ là Trapdoor:

Ta có "I love Fox and Friends” + Public Key --> s80s1s9sadjds9s

Nếu được cung cấp "I love Fox and Friends" và khóa công khai, ta có thể tạo s80s1s9sadjds9s, nhưng nếu được cung cấp s80s1s9sadjds9s và Public Key, ta không thể tạo ra "I love Fox and Friends".

Trong RSA, được biết đến là hệ thống mật mã khóa công khai được sử dụng rộng rãi nhất, hàm cửa sập dựa vào mức độ khó để tính các số lớn thành thừa số nguyên tố của chúng.

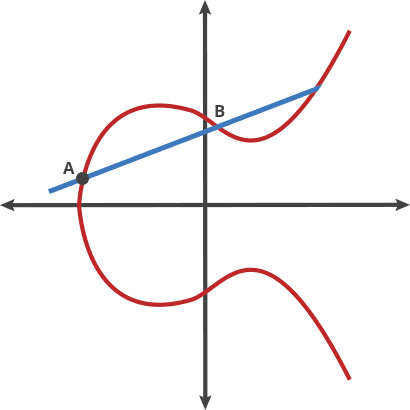
Public Key: 944,871,836,856,449,473

Private Key: 961,748,941 và 982,451,653

Trong ví dụ trên, Public Key là một số rất lớn và Private Key là hai yếu tố chính của hệ thống mật mã khóa công khai. Đây là một ví dụ điển hình về hàm Trapdoor vì rất dễ dàng nhân các số trong Private Key với nhau để có được Public Key, nhưng nếu tất cả những gì bạn có là Public Key thì sẽ mất rất nhiều thời gian để sử dụng máy tính và tạo ra được Private Key.

**2. Hàm Trapdoor trong ECDSA:**

Hàm Trapdoor là điều làm cho ECDSA trở nên đặc biệt và khác biệt so với RSA. Chức năng của cửa sập tương tự như một trò chơi bi-a.

Đầu tiên, ta bắt đầu với một điểm tùy ý trên đường cong. Tiếp theo, ta sử dụng **dot function** để tìm điểm mới. Cuối cùng, ta tiếp tục lặp lại **dot function** để nhảy xung quanh đường cong cho đến khi ta kết thúc ở điểm cuối cùng của ta. Hãy xem qua thuật toán. 

Bắt đầu từ A:

* A dot B = -C (Vẽ một đường thẳng từ A đến B và nó cắt nhau tại -C)
* Phản chiếu qua trục X từ -C đến C
* A dot C = -D (Vẽ một đường thẳng từ A đến C và nó cắt -D)
* Phản chiếu qua trục X từ -D đến D
* A dot D = -E (Vẽ một đường thẳng từ A đến D và nó cắt -E)
* Phản chiếu qua trục X từ -E đến E

Hàm trapdoor thật sự rất tuyệt vì nếu bạn biết điểm bắt đầu (A) ở đâu và cần bao nhiêu bước để đến điểm kết thúc (E), bạn sẽ rất dễ dàng tìm thấy điểm kết thúc. Mặt khác, nếu tất cả những gì bạn biết là điểm bắt đầu và điểm kết thúc ở đâu, thì gần như không thể tìm ra được cần phải mất bao nhiêu bước để đến đó.

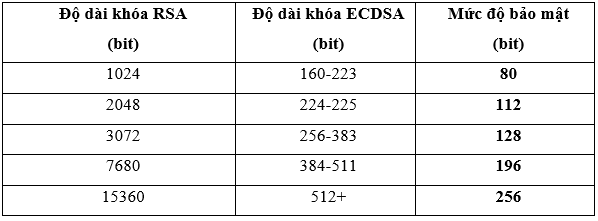
Public key: Điểm bắt đầu A, Điểm kết thúc E

Private key: Số bước nhảy từ A đến E

## Thế mạnh của ECDSA

Thuật toán ECDSA hiện nay đang được mọi người coi là có thể soán ngôi được RSA - một thuật toán mã hóa được sử dụng rất nhiều hiện nay, vậy chúng ta sẽ cùng làm một vài phép so sánh để thấy thế mạnh của ECDSA so với RSA.

### 1.Độ dài lưu khóa ngắn



Chú thích:

- Mức độ bảo mật ở đây được định lượng bởi số bit khóa tiêu chuẩn trên lý thuyết cần sử dụng để đảm bảo độ bảo mật của ‘n’ bit.

- Độ dài khóa công khai cũng được đo bằng bit, nhưng nó là một khái niệm hoàn toàn khác, đề cập đến độ dài vật lý của chìa khóa.

### 2.Tiêu tốn ít tài nguyên trên hệ thống

Như bảng trên, có thể thấy được mặc dù cùng một mức độ bảo mật nhưng độ dài của khóa sử dụng thuật toán ECDSA lại nhỏ hơn rất nhiều so với khóa sử dụng thuật toán RSA. Đối với các doanh nghiệp lớn, họ cần lưu trữ lại rất nhiều khóa, khi đó đòi hỏi doanh nghiệp cần có đủ nguồn lực về hạ tầng để đáp ứng nhu cầu lưu trữ trên. Giả dụ, có 1 thiết bị lưu trữ với dung lượng 2048bit, nếu sử dụng RSA, ta chỉ có thể lưu trữ được 1 khóa duy nhất, còn đối với ECDSA, ta có thể lưu tới 9 khóa cùng mức độ bảo mật, mở rộng không gian lưu trữ gấp 9 lần và còn tiết kiệm được chi phí.

### 3.Rút ngắn thời gian ký số

Đối với thuật toán ECDSA, quá trình mã hóa và giải mã được rút ngắn, đồng thời cũng giảm bớt năng lượng cần sử dụng trong quá trình, hơn hẳn so với việc sử dụng thuật toán RSA. Cụ thể, khóa của ECDSA nhỏ, nên việc tạo chữ ký số sẽ nhanh hơn, giảm tải chu kỳ xử lý của CPU, không chỉ vậy, với việc sử dụng khóa của thuật toán ECDSA, chứng chỉ an toàn sẽ nhỏ và ít dữ liệu hơn, điều này hỗ trợ đẩy nhanh quá trình kết nối, cải thiện tốc độ internet.

### 4.Tăng cường độ bảo mật

ECDSA sử dụng các thuật toán phức tạp trên đường cong elliptic vào các quá trình tạo khóa, mã hóa và giải mã, giúp cho thông tin được tăng tính bảo mật, làm hacker khó khăn khi đánh cắp thông tin và hầu như là không thể hack được. Điều này làm cho ECDSA trở thành thuật toán bảo mật mạnh mẽ.

Dễ dàng thấy được thuật toán ECDSA đáp ứng được các vấn đề mà RSA còn chưa được tối ưu như độ dài khóa, hiệu suất. Và không có gì ngạc nhiên khi ECDSA được dự đoán là soán ngôi của RSA.

# **Chương III: Tìm hiểu về công nghệ Blockchain**

## Khái niệm

Blockchain (chuỗi khối) là một mạng lưới cơ sở dữ liệu phi tập trung lưu trữ thông tin trong các khối thông tin được liên kết với nhau bằng mã hóa và mở rộng theo thời gian. Mỗi khối thông tin đều chứa thông tin về thời gian khởi tạo và được liên kết tới khối trước đó, kèm một mã thời gian và dữ liệu giao dịch. Blockchain được thiết kế để chống lại sự thay đổi của dữ liệu: Một khi dữ liệu đã được mạng lưới chấp nhận thì sẽ không có cách nào thay đổi được nó.

Blockchain được đảm bảo nhờ cách thiết kế sử dụng hệ thống tính toán phân cấp với khả năng chịu lỗi byzantine cao. Nhờ thế nên Blockchain có thể đạt được sự đồng thuận phân cấp. Vì vậy Blockchain phù hợp để ghi lại những sự kiện, hồ sơ y tế, xử lý giao dịch, công chứng, danh tính và chứng minh nguồn gốc. Việc này có tiềm năng giúp xóa bỏ các hậu quả lớn khi dữ liệu bị thay đổi trong bối cảnh thương mại toàn cầu.

## Lịch sử hình thành và phát triển

Năm 1991, Hệ thống Blockchain đã được mô tả bởi W. Scott Stornetta và cộng sự của ông là Stuart Haber. Với mục đích ban đầu là ghi chú thời gian vào các tệp tài liệu để dễ quản lý thông tin. Và các tệp dữ liệu này trở nên bất biến. Điều này có nghĩa là không ai có thể sửa đổi ngày với bất cứ hình thức nào.

Năm 2008, nền tài chính thế giới sụp đổ. Lần đầu tiên xuất hiện khái niệm về hệ thống tiền mặt điện tử ngang hàng-phi tập trung (khái niệm nguyên sơ của hệ thống Blockchain). Khái niệm này được trình bày bởi một người hay một nhóm ẩn danh có tên là Satoshi Nakamoto, tạo ra một giao thức mã nguồn mở có tên là Bitcoin (BTC).

Ngày 03/01/2009, Bitcoin lần đầu ra đời bởi Satoshi Nakamoto khi ông xuất bản 50 khối Bitcoin. Sau đó, ông bán lại 10 khối Bitcoin cho một người thương nhân có tên là Half Finney, giao dịch về Bitcoin lần đầu tiên trên thế giới được diễn ra vào ngày 12/01/2009.

Công nghệ blockchain đóng vai trò như là một cuốn sổ cái cho tất cả các giao dịch. Qua việc sử dụng mạng lưới ngang hàng và một hệ thống dữ liệu phân cấp, Bitcoin blockchain được quản lý tự động. Việc phát minh ra blockchain cho Bitcoin đã làm cho nó trở thành loại tiền tệ kỹ thuật số đầu tiên giải quyết được vấn đề double spending (chi tiêu gian lận khi 1 lượng tiền được dùng 2 lần). Công nghệ này của Bitcoin đã trở thành nguồn cảm hứng cho một loạt các ứng dụng khác.

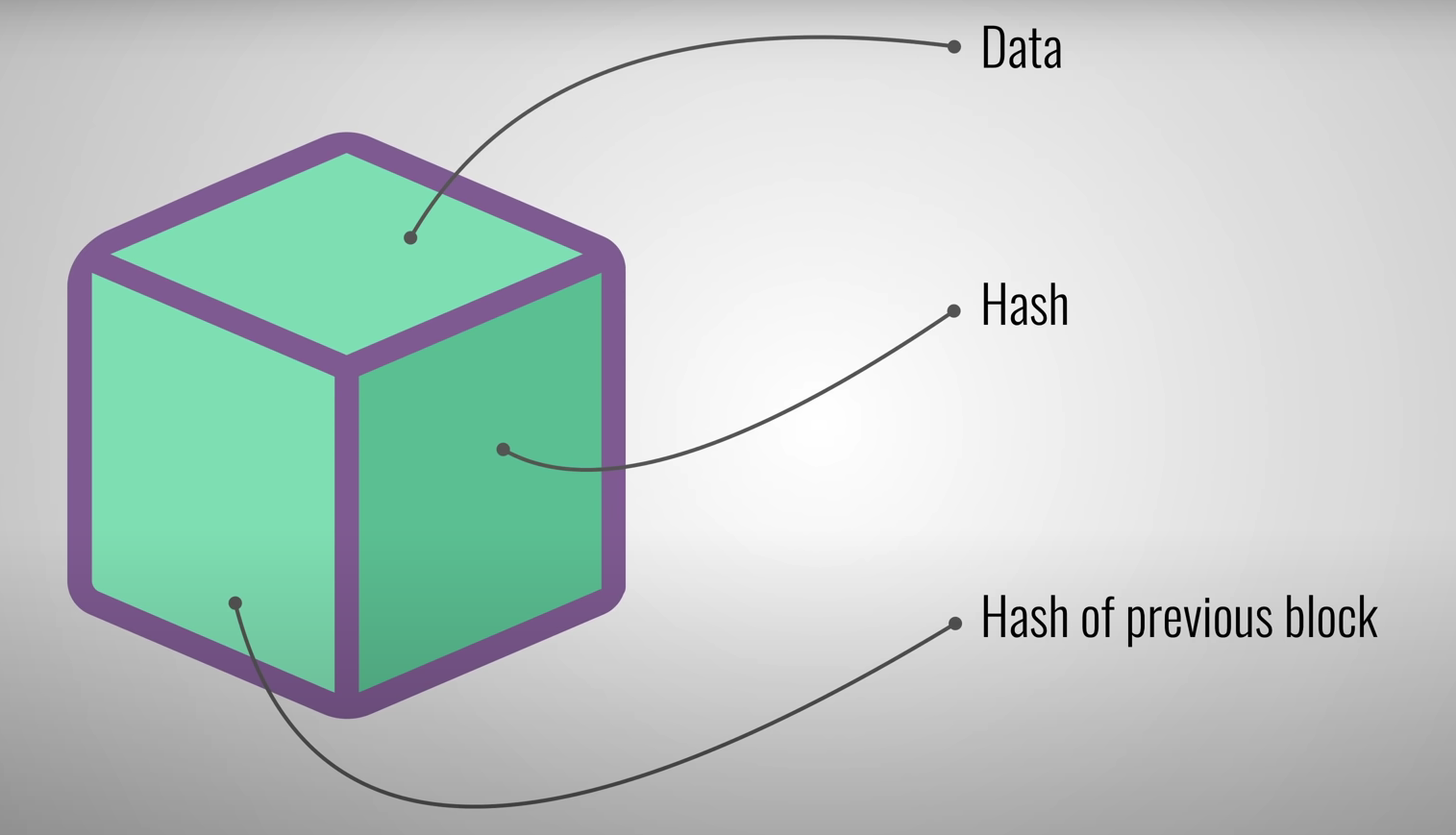
Blockchain cho phép thông tin kỹ thuật số được ghi lại và phân phối, nhưng không được chỉnh sửa. Theo cách này, blockchain là nền tảng cho các sổ cái bất biến hoặc các bản ghi của các giao dịch không thể bị thay đổi, xóa hoặc phá hủy. Đây là lý do tại sao blockchains còn được gọi là công nghệ sổ cái phân tán (DLT).

## Kiến trúc Blockchain

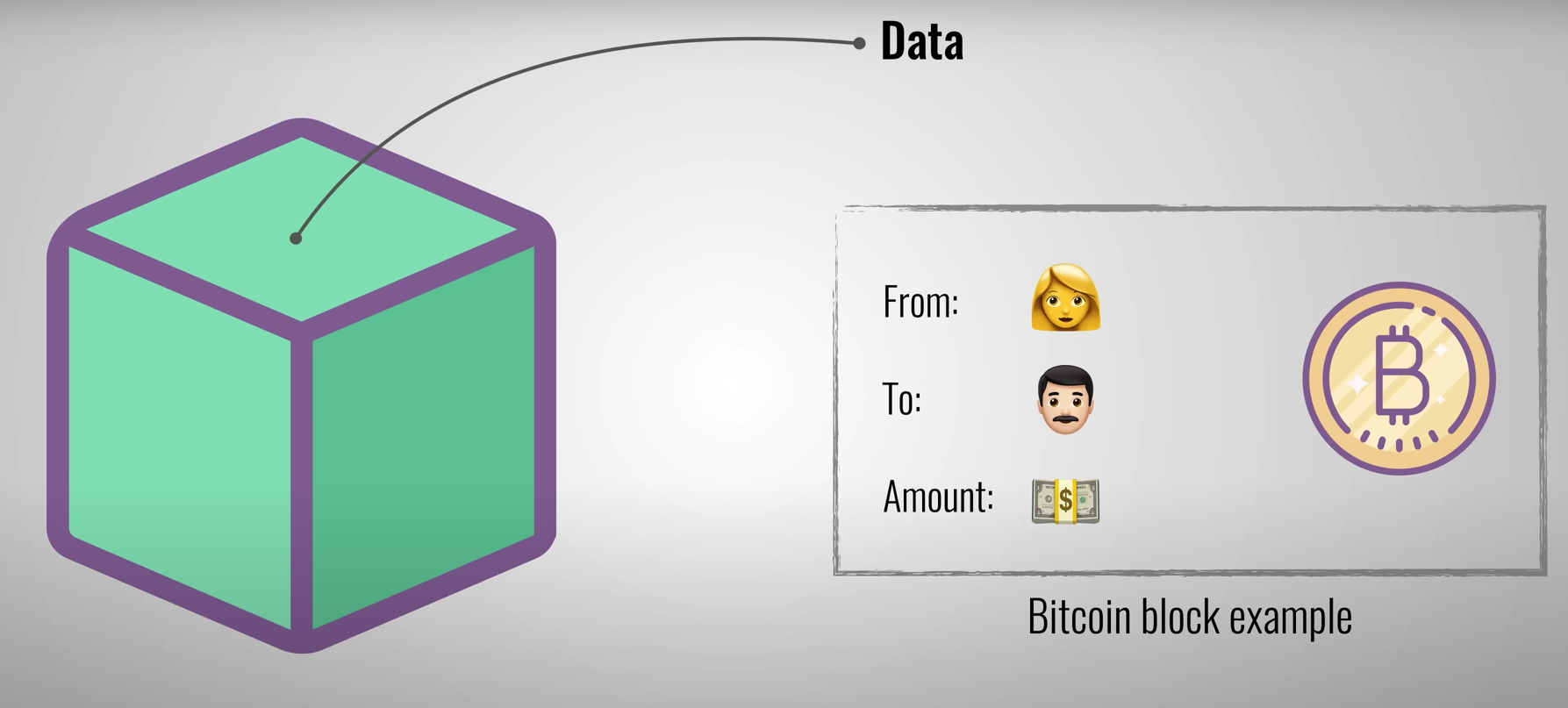
Chuỗi khối, nút và khai thác là ba ý tưởng chính trong Blockchain. Blockchain không lưu bất kỳ dữ liệu nào của nó ở một vị trí duy nhất. Thay vào đó, một mạng lưới các máy tính sao chép và lan truyền Blockchain. Mọi máy tính trên web đều cập nhật Blockchain của nó để phản ánh một khối mới vào Blockchain.

### 3.3.1 Chuỗi khối (block chain)

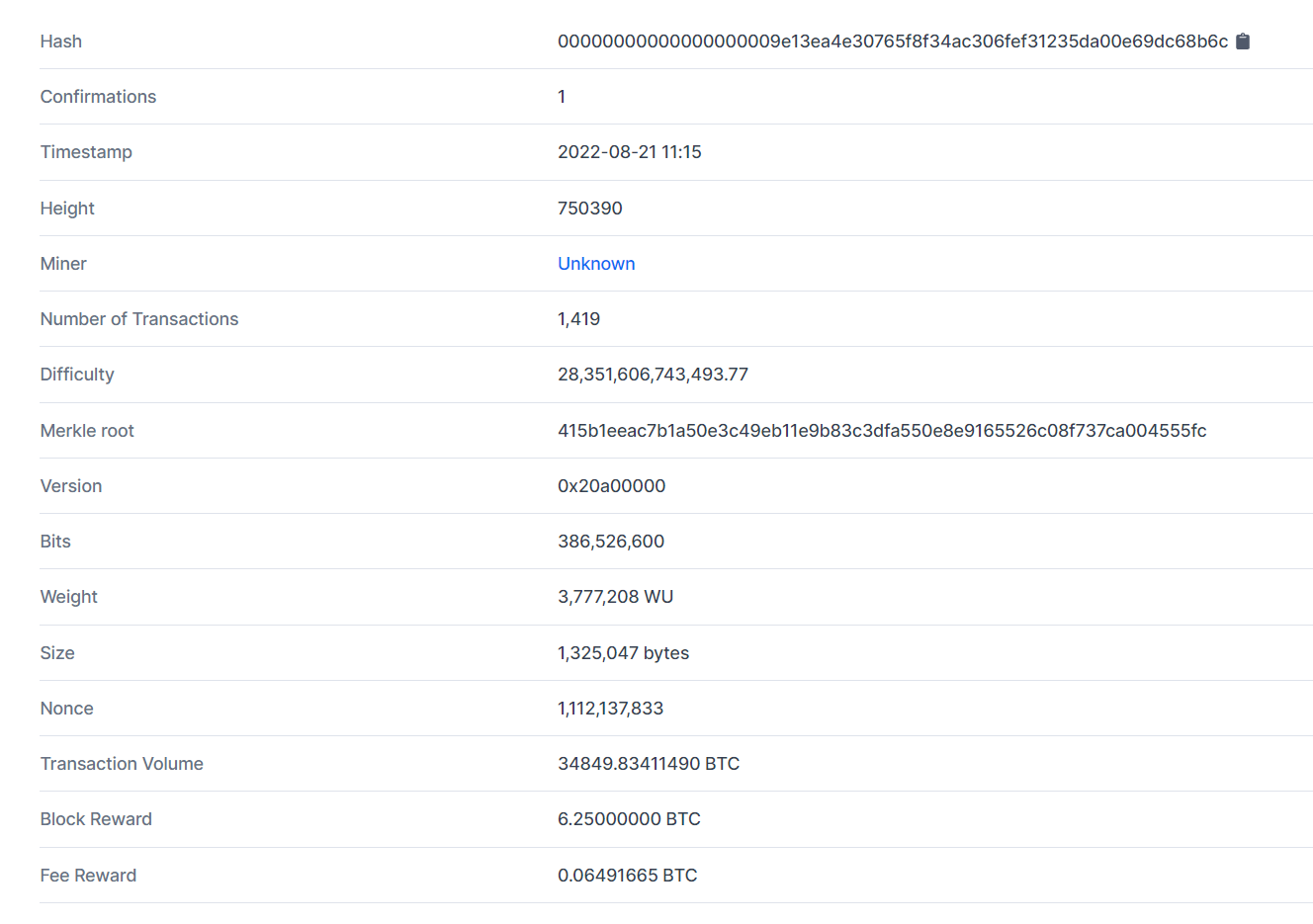
Về bản chất nó là các chuỗi khối liên kết với nhau như dạng danh sách liên kết nhưng có thể truy xuất ngược từ khối cuối (hiện tại) đến khối đầu tiên. Nó thực sự như một cuốn sổ cái phân tán (Distributed ledger) mà mỗi giao dịch (gọi là khối) trong sổ bao gồm các thông tin được lưu trữ như sau:

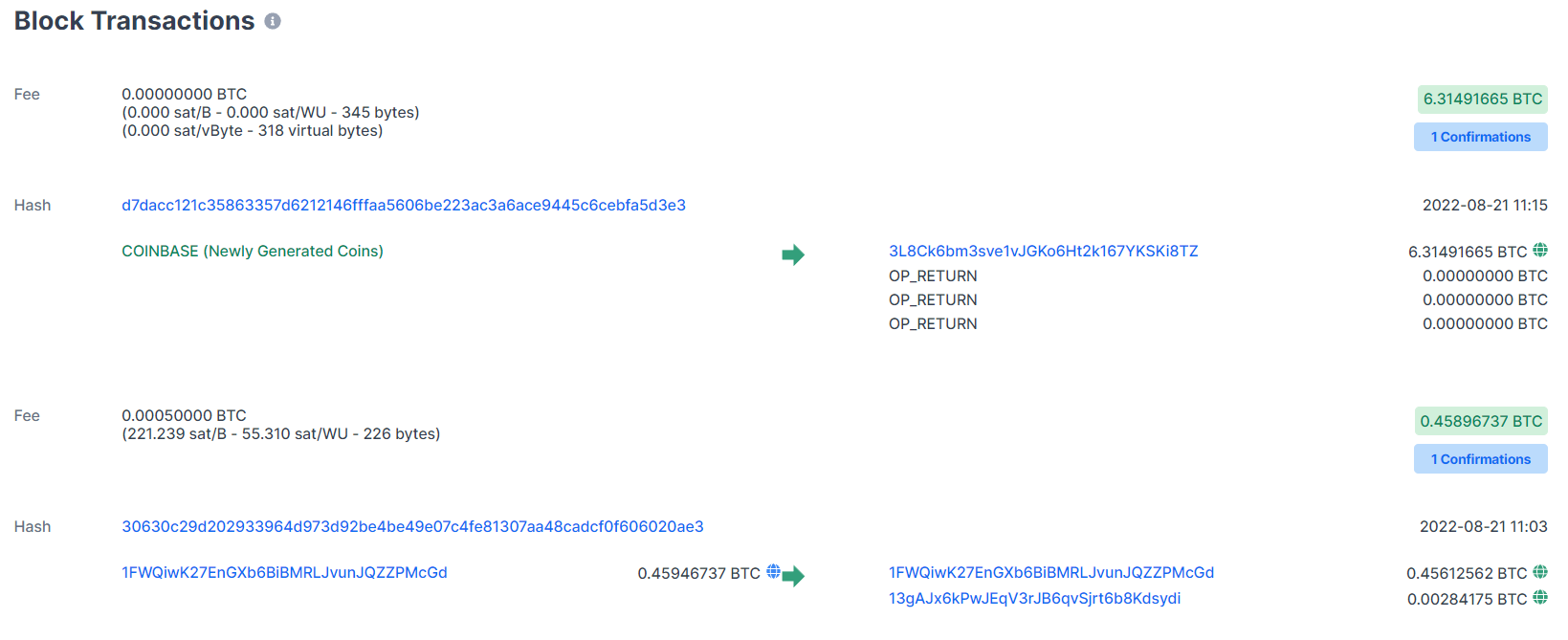


(1) Dữ liệu (Data):

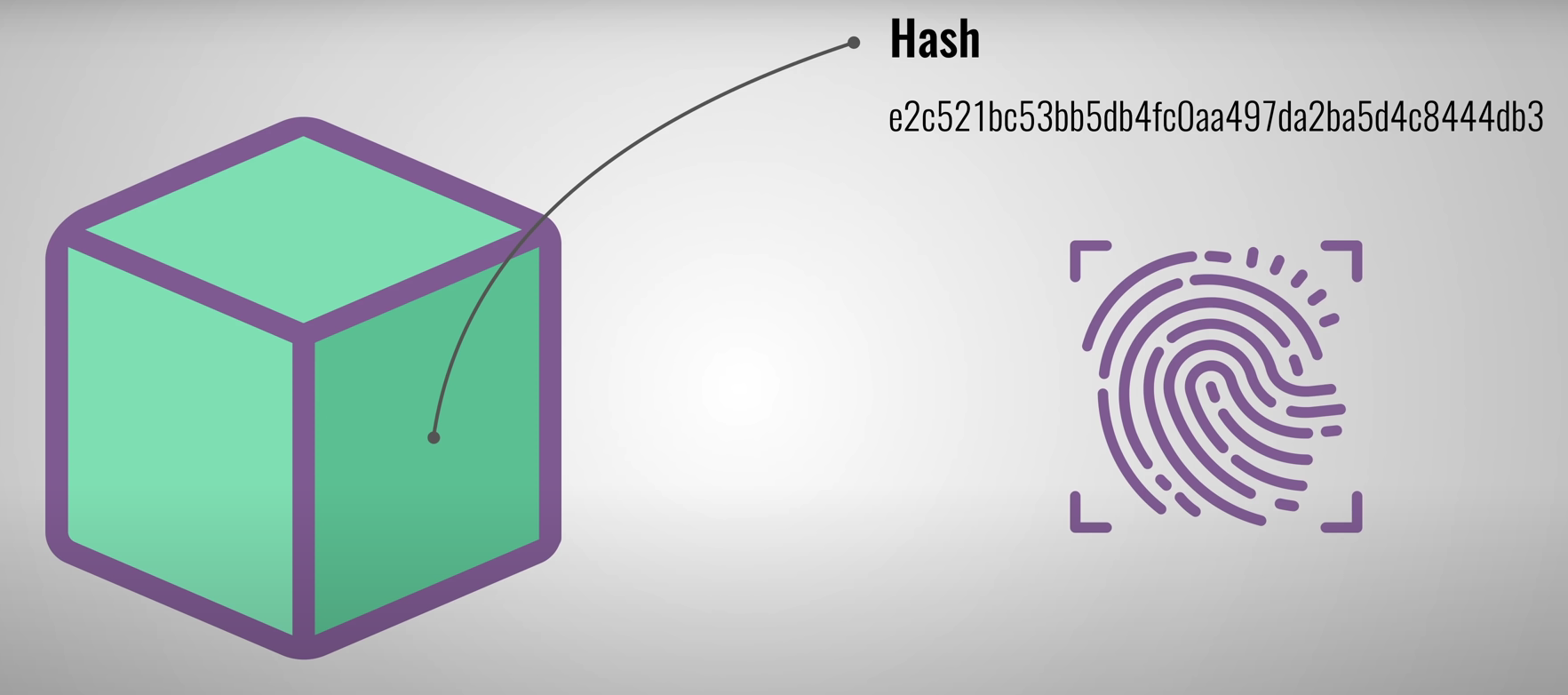


Dữ liệu trong mỗi khối phụ thuộc vào loại blockchain, ví dụ blockchain của bitcoin chứa thông tin về các giao dịch như thông tin người gửi, người nhận tiền và số bitcoin được giao dịch, blockchain của dịch vụ chăm sóc y tế thì dữ liệu sẽ là các thông tin sức khỏe của bệnh nhân,...



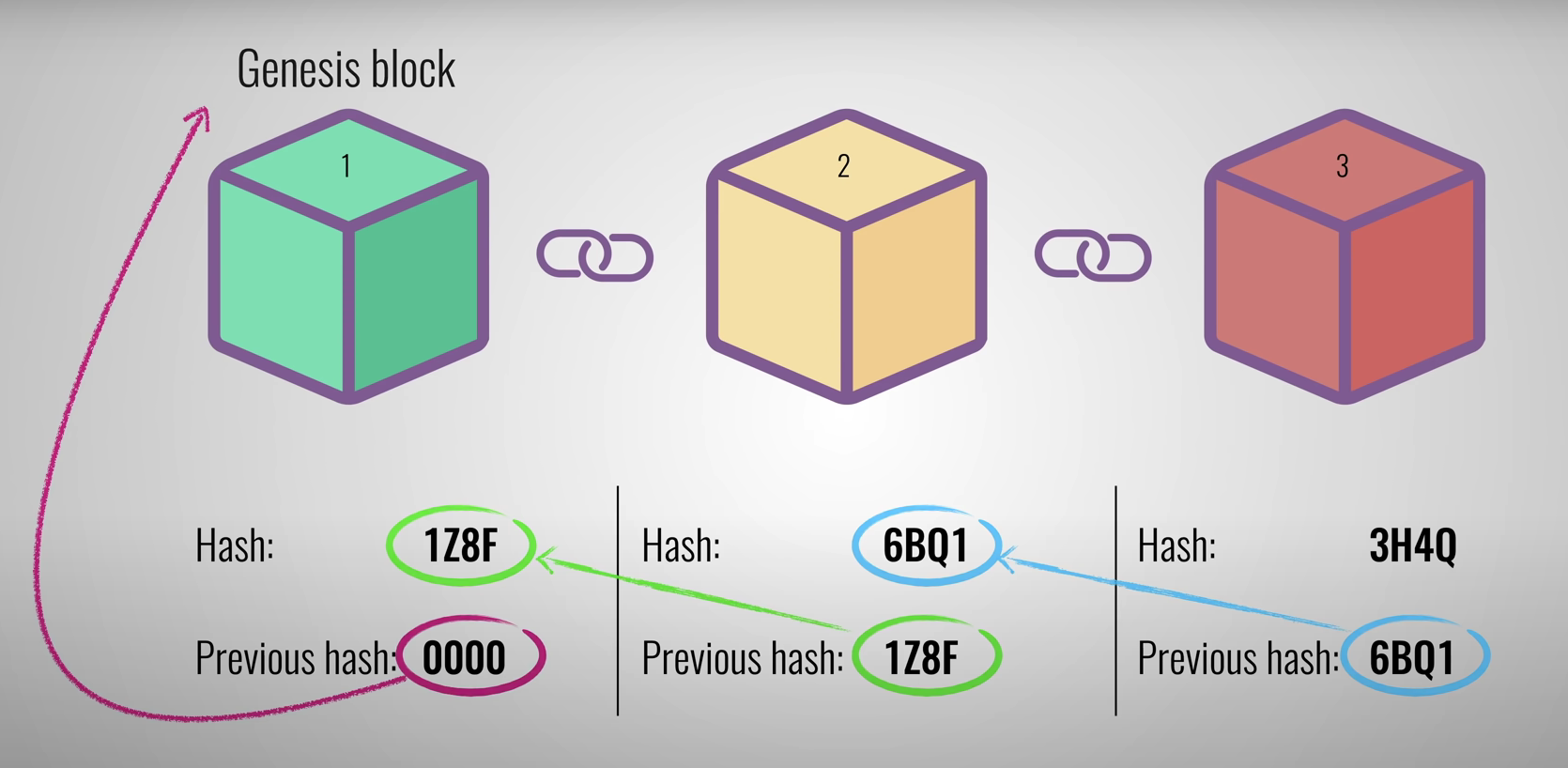


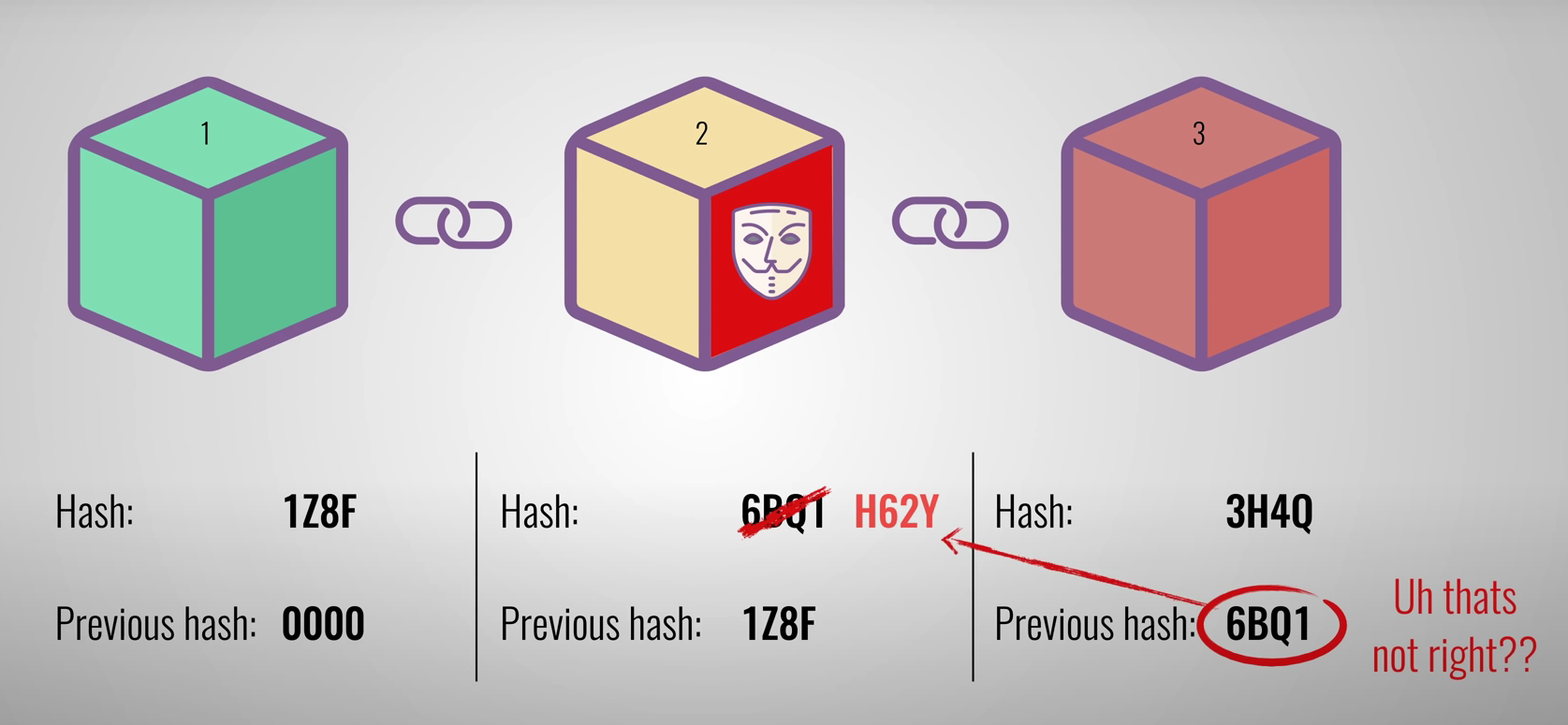
(2) Mã băm (Hash):



Dùng để nhận dạng một khối và các dữ liệu trong đó. Mã này là duy nhất, nó tương tự như dấu vân tay. Bất kỳ sự thay đổi nào trong khối thì mã băm cũng sẽ thay đổi.

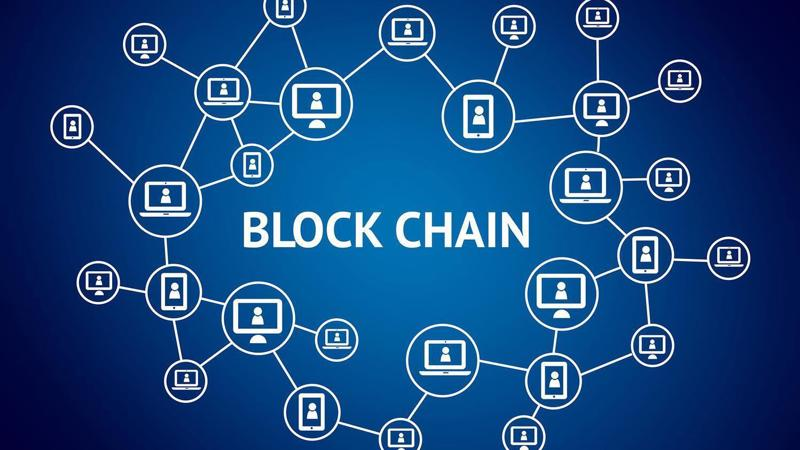
(3) Mã băm đối chiếu (Hash of previous block) sẽ tạo thành chuỗi. Bất cứ sự thay đổi một khối sẽ khiến các khối tiếp theo không phù hợp.





### 3.3.2 Nút (node)

Nút blockchain là một thiết bị, thường là một máy tính, tham gia vào mạng blockchain. Nó chạy phần mềm của giao thức blockchain, cho phép nó giúp xác thực các giao dịch và giữ cho mạng an toàn. Các nút chuỗi khối giao tiếp với nhau. Càng có nhiều nút, mạng càng được phân cấp.



Các vai trò chính của một nút blockchain là phát sóng và xác thực các giao dịch. Khi người dùng gửi một giao dịch, nó sẽ được nhận bởi một nút, nút này sẽ truyền nó đến phần còn lại của mạng. Tất cả các nút trong mạng kiểm tra giao dịch để đảm bảo rằng người gửi có sẵn tiền và được phép gửi chúng.

Thực tế là mọi nút đều xác minh các giao dịch giúp bảo mật mạng. Giao dịch không hợp lệ chỉ có thể được chấp thuận nếu 51% số nút xác nhận. Trong các mạng blockchain với hàng trăm hoặc hàng nghìn nút, rất khó có khả năng một kẻ xấu có thể chiếm 51%.

Sau khi các giao dịch mới được các nút xác thực, chúng được nhóm lại thành các khối. Mỗi khối mới được thêm vào blockchain tuân theo các quy tắc của cơ chế đồng thuận của nó, được thực thi bởi các nút đầy đủ (full node).

### 3.3.3 Khai thác

Khai thác hay đơn giản là thêm 1 khối mới vào chuỗi khối (blockchain) là việc dùng các hàm tính toán hash của khối mới dựa trên các thành phần: dữ liệu của nó, hash của khối trước nó,...

Vì là một hệ thống phân cấp phi tập trung nên việc thêm một khối cần sự chấp thuận của hơn 50% các nút, việc khai thác này giúp tránh các nguy cơ về giả mạo, double-spending,...

Việc khai thác các loại tiền điện tử sẽ có thêm mức độ khó, số ngẫu nhiên và target hash để việc hình thành khối không diễn ra quá nhanh.

## Cách hoạt động của blockchain

Khi giao dịch/bản ghi/hợp đồng được hình thành các máy tính/nút trong hệ thống sẽ tính toán hash dựa trên khối cũ và thông tin giao dịch/bản ghi/hợp đồng trên để tạo ra một khối mới. Máy tính hay nút tham gia tính toán được hash hợp lệ sẽ gửi khối mới đến các máy tính/nút trong hệ thống, nếu khối mới đó được tính toán một cách chính xác sẽ nhận được sự đồng thuận của các máy tính/nút trong hệ thống, và sẽ được thêm vào chuỗi khối.

Việc đồng thuận giữa các node về block mới là vấn đề khó, nó liên quan đến bài toán Byzantine fault.

**Byzantine fault**

Mặc dù bài toán về các vị tướng Byzantine đã được nghiên cứu trong khoa học máy tính trước năm 1982, đây là một trong những nỗ lực đầu tiên để chuyển nó thành các giải pháp song song và được đề xuất. Sự tương tự sau đây minh họa vấn đề các tướng Byzantine. Một số sư đoàn của quân đội Byzantine đóng quân ngay bên ngoài thành phố của kẻ thù, chuẩn bị cho chiến tranh. Cách duy nhất để các vị tướng khác nhau kết nối là thông qua một sứ giả hay người đưa tin. Họ phải đồng ý về một quá trình tấn công hay rút lui.

Tuy nhiên, chúng ta phải cho rằng một số vị tướng sẽ có ý định rút lui nhưng vẫn đồng thuận tấn công, là những kẻ có ý đồ phản bội. Để đảm bảo rằng một nhóm nhỏ những kẻ phản bội không thể làm gián đoạn liên lạc, một thuật toán được yêu cầu.

Để giải quyết vấn đề của các vị tướng Byzantine, các vị tướng trung thành cần một phương tiện an toàn để thống nhất một kế hoạch (được gọi là đồng thuận) và thực hiện nó (được gọi là phối hợp). Trong khi việc giải quyết vấn đề các tướng Byzantine là một nhiệm vụ khó khăn, giờ đây chúng ta đã hiểu rõ hơn về vấn đề cơ bản. Điều quan trọng cần lưu ý là, như ví dụ cho thấy, khái niệm này có thể được áp dụng cho thông tin liên lạc quân sự.

Tuy nhiên, vấn đề này ảnh hưởng đến tất cả các loại hệ thống máy tính phân tán, không chỉ những hệ thống được sử dụng trong các ứng dụng quân sự. Vấn đề chung của Byzantine phải được giải quyết nếu một nhóm nút phân tán (ví dụ: máy tính hoặc các thiết bị vật lý khác) cần đạt được thông tin liên lạc đáng tin cậy.

Vấn đề chung của Byzantine có thể được giải quyết với sự trợ giúp của blockchain. Tất cả là nhằm mang đến cho mọi người cách giao tiếp an toàn và bảo mật trong một thế giới không thể đoán trước. Trong thế giới thực, hầu hết các giao dịch xảy ra giữa những người lạ không quen biết hoặc không tin tưởng lẫn nhau.

Mỗi cá nhân giống như một vị tướng, chờ lệnh để tấn công hoặc bảo vệ vị trí của họ. Không có người trung gian nào thay mặt bạn phân xử cuộc tấn công; bạn hoàn toàn tự quyết định của mình.

Một blockchain tạo ra một lớp có thể được tin cậy mà không cần phải tin tưởng vào từng cá nhân. Điều này được thực hiện bởi một mạng lưới các nút kết hợp với nhau để thống nhất về sự thật trước khi nó được ghi lại. Nếu vị tướng đó không chắc chắn về nội dung của giao tiếp, các vị tướng khác có thể xác minh điều đó bằng cách sử dụng những gì họ biết là đúng.

Khi một nút đã ghi lại nó, một bản sao sẽ được gửi đến tất cả các nút khác trong mạng. Thuật toán đồng thuận PoW được thiết kế để đạt được mục tiêu này. Những kẻ xấu vẫn sẽ cố gắng đánh lừa hệ thống vì thông tin không phải lúc nào cũng chính xác.

Vì hệ thống được thiết kế để được sử dụng bởi công chúng, các cơ chế chịu lỗi và bảo mật được đặt trong một chuỗi khối. Trong trường hợp này, mật mã được yêu cầu để đảm bảo rằng các thông điệp không thể bị thay đổi.

Hệ thống cung cấp các cặp khóa để ký điện tử một thông tin liên lạc nhằm xác minh danh tính làm bằng chứng rằng nó đến từ những người được cho là đã gửi nó. Khi tin nhắn đã được xác thực, chúng sẽ được ghi lại để minh bạch và bằng chứng lịch sử về trách nhiệm giải trình.

Việc tham gia tính toán sẽ tuân theo các nguyên tắc để giải quyết vấn đề trên.

**Proof of Work (PoW):**

Mọi khối trong mạng đến từ khối đầu tiên, còn được gọi là khối gốc, đều là khối hợp lệ. Những người khai thác tính toán mã băm cho khối mới, người đầu tiên tìm ra được khối mới sẽ gửi đến toàn bộ các nút, sau đó các nút sẽ tính toán lại khối đó để xác thực, nếu hơn 50% các nút chấp thuận, khối đó sẽ được thêm vào chuỗi.

Bằng cách sử dụng cơ chế đồng thuận PoW, Bitcoin đã vượt qua vấn đề chung của Byzantine và thiết lập một quy tắc rõ ràng, khách quan cho blockchain. Để thêm thông tin vào chuỗi khối, được gọi là khối, một thành viên mạng phải công bố bằng chứng rằng họ đã nỗ lực rất nhiều để tạo khối. Công việc này có giá cao đối với người sáng tạo, khuyến khích họ chia sẻ thông tin chính xác.

Không thể có bất đồng hoặc giả mạo thông tin trên mạng Bitcoin vì các quy tắc là khách quan. Hệ thống để chọn ai có thể khai thác Bitcoin mới và luật quy định giao dịch nào hợp lệ hoặc không hợp lệ đều là hai mục tiêu. Hơn nữa, không thể xóa một khối khỏi chuỗi khối sau khi nó đã được thêm vào, làm cho lịch sử của Bitcoin không thể thay đổi.

Do đó, vấn đề về các vị tướng Byzantine được giải quyết bởi những người khai thác tương tự như các vị tướng trong phiên bản blockchain của Satoshi. Mỗi nút chịu trách nhiệm xác thực các giao dịch, tương tự như các thông điệp được gửi đến các vị tướng. Các tác nhân xấu (ví dụ, tin tặc) nhằm đánh cắp tin nhắn hoặc gây hại cho hệ thống mạng có thể được coi là kẻ thù.

**Proof of Stake (PoS) và Delegated proof-of-stake (DPoS):**

PoS là một cơ chế đồng thuận blockchain khác tìm cách giải quyết vấn đề chung của Byzantine. Nó được triển khai lần đầu tiên vào năm 2012. Các mạng dựa trên PoS, không giống như các mạng dựa trên PoW, không phụ thuộc vào việc khai thác tiền điện tử. Thay vào đó, một kỹ thuật gọi là đặt cược được thực hiện.

Người dùng (được gọi là người xác nhận) đặt tiền vào hệ thống này. Người xác thực sở hữu nhiều đồng tiền hơn trên blockchain có thể xác nhận nhiều khối hơn và kiếm được phần thưởng lớn hơn. Người dùng cố gắng xác thực các giao dịch không chính xác có nguy cơ mất tiền đặt cọc của họ.

Người dùng có thể đặt cược bằng máy tính gia đình bình thường thay vì cần các máy chuyên dụng để khai thác trong mạng dựa trên PoW. Một số mạng dựa trên PoS đã tạo ra các cách để ngăn chặn các cuộc tấn công chi tiêu kép và các lỗ hổng bảo mật tiềm ẩn khác do lỗi của Byzantine gây ra. Ví dụ: Ethereum 2.0 (Serenity) sẽ sử dụng thuật toán Casper PoS, thuật toán này yêu cầu đa số hai phần ba các nút đồng ý về một khối trước khi nó có thể được tạo.

DPoS là một kỹ thuật đồng thuận blockchain hoạt động tương tự như PoS và được phát triển lần đầu tiên vào năm 2014. Cả hai đều yêu cầu người dùng đặt tiền. Chỉ một số người dùng (được gọi là đại biểu) có thể xác thực các giao dịch và tạo các khối trong mạng dựa trên DPoS.

Một vấn đề nữa sinh ra trong quá trình hoạt động của blockchain là sự tương đồng giữa các node, nếu sự nâng cấp, thay đổi giữa các node là không đồng thời thì chuỗi blockchain sẽ sinh ra nhánh mới và để giải quyết nhánh mới đó như thế nào chúng ta sẽ tìm hiểu về soft fork và hard fork.

**Hard fork và soft fork**

Hard fork và soft fork về cơ bản giống nhau theo nghĩa là khi mã nguồn hiện tại của nền tảng blockchain được thay đổi, một phiên bản cũ vẫn còn trên mạng trong khi phiên bản mới được tạo.

Với soft fork, chỉ một blockchain sẽ vẫn còn hiệu lực khi người dùng chấp nhận bản cập nhật. Trong khi với hard fork, cả blockchain cũ và mới đều tồn tại song song với nhau, có nghĩa là phần mềm phải được cập nhật để hoạt động theo các quy tắc mới. Cả hai fork tạo ra một sự phân tách, nhưng một hard fork tạo ra hai blockchains và một soft fork có nghĩa là tạo ra một.

Xem xét sự khác biệt về bảo mật giữa hard fork và soft fork, hầu như tất cả người dùng và nhà phát triển đều kêu gọi hard fork, ngay cả khi một soft fork có vẻ như nó có thể thực hiện được công việc. Việc đại tu các khối trong một chuỗi khối đòi hỏi sức mạnh tính toán rất lớn, nhưng quyền riêng tư thu được từ hard fork có ý nghĩa hơn so với việc sử dụng soft fork.

## Ứng dụng

Với những ưu điểm vô cùng lớn và quan trọng như tính toàn vẹn và minh bạch, công nghệ blockchain ngày càng được ứng dụng rộng rãi vào trong đời sống con người.

Sau đây là một số ứng dụng phổ biến:

Ứng dụng vào tiền điện tử: với đặc tính phi tập trung, blockchain tạo nền tảng cho các loại hình tiền điện tử ra đời và phát triển, việc không thông qua bên thứ ba giúp cho việc giao dịch an toàn và ẩn danh, giải quyết được nguy cơ bị đóng băng tài khoản, chi phí giao dịch ngân hàng…

Ứng dụng trong lĩnh vực chăm sóc sức khỏe, y tế: xu hướng số hóa dữ liệu, thông tin người bệnh, đơn đặt hàng, quản lý kho, giao dịch cho các thiết bị y tế,… trong quá trình quản lý tài liệu đã trở nên phổ biến hơn. Do vậy, các thiết bị thông minh được trang bị trong phần lớn các bệnh viện để giám sát các dữ liệu này. Tuy nhiên các thiết bị này vẫn còn nhiều hạn chế về quyền riêng tư và bảo mật, cho nên công nghệ blockchain được sử dụng để khắc phục những vấn đề này.

Ứng dụng blockchain trong ngành tài chính – ngân hàng: trong lĩnh vực tài chính và ngân hàng, vấn đề bảo mật dữ liệu người dùng, tham nhũng, lạm quyền là vấn đề rất nan giải. Nhưng với quyết định ứng dụng công nghệ blockchain với các điểm nổi bật từ chính năng năng như bảo mật cao, giao dịch nhanh, tiết kiệm chi phí, tối thiểu hóa rủi ro.

Ứng dụng blockchain trong lĩnh vực bán lẻ, xản suất: quy trình quản lý chuỗi cung ứng, quản lý sản phẩm như quá trình phân phối, kiểm soát thông tin và số lượng hàng hóa cũng như các báo cáo tài chính, hợp đồng quan trọng,… tất cả sẽ đều trở nên dễ dàng và chính xác khi công nghệ blockchain được ứng dụng.

## Ưu và nhược điểm

### 3.6.1 Ưu điểm công nghệ Blockchain

### Tính toàn vẹn

Bởi vì được xây dựng trên tính đảm bảo sự toàn vẹn của dữ liệu nên đây cũng được xem là một trong những ưu điểm lớn của công nghệ Blockchain. Các khối dữ liệu khi được thêm mới vào luôn tự động nằm bên cạnh khối cuối cùng của chuỗi khối hiện tại, do đó khi muốn thay đổi dữ liệu của bất kì một khối nào trong chuỗi thì bạn sẽ phải sửa từ các khối liền trước cho tới khối đó.

### Tính phi tập trung

Công nghệ Blockchain ứng dụng trong việc giao dịch nhờ vào việc lưu trữ các thông tin giao dịch lên các khối dữ liệu trong chuỗi. Trong khi đó các hệ thống dữ liệu truyền thống thì cần một server để lưu trữ thì Blockchain cần một tập hợp các máy tính để lưu trữ các khối thống tin.

Vì vậy nên các máy tính này có thể nằm bất cứ đâu trên trái đất này chứ không bắt buộc phải là một server tập trung. Các Node có thể tự động tham chiếu với rất nhiều node khác để kiểm tra tính đúng đắn của dữ liệu.

### Tính minh bạch

Bời vì có rất nhiều bản Node rải trên khắp thế giới và mỗi bản node là một bản sao của blockchain nên có thể thấy được tính minh bạch của công nghệ này là rất cao.

Mỗi khi có sự thay đổi trên các chuỗi, đồng loạt các Node sẽ thay đổi ngay lập tức. Đặc điểm này ứng dụng vào giao dịch tiền ảo giúp người giao dịch có thể cập nhật ngay lập tức các giao dịch và thông tin luôn được update liên tục.

### 3.6.2 Nhược điểm công nghệ Blockchain

### Khó khăn trong sửa đổi dữ liệu

Một nhược điểm chính của Blockchain là việc khó khăn trong sửa đổi dữ liệu. Một khi dữ liệu được đưa vào Blockchain thì việc sửa đổi dường như là rất khó khăn.

Như đã đề cập ở trên thì việc sửa đổi dữ liệu trong một khối bất kì thì bạn sẽ phải sửa từ khối cuối cùng trước đó dần cho tới khối đó. Đây là một lợi thế bào mật của Blockchain song không phải khi nào nó cũng mang lại hiệu quả.

### Cá nhân hóa

Trong ứng dụng vào giao dịch tiền ảo mỗi một tài khoản cá nhân sẽ có 2 chìa khóa bao gồm chìa khóa cá nhân và chìa khóa chung. một khi người dùng làm mất chìa khóa cá nhân thì họ sẽ bị mất đi số tiền đó và không có cơ sở nào để có thể lấy lại tài sản của mình.

### Không hiệu quả

Một số Blockchain đang sử dụng Proof of work (một phương pháp bảo mật cho sổ cái tiền mã hóa) mang lại hiệu quả chưa thực sự cao. Lý giải cho việc này chính là vì sự cạnh tranh cao giữa các thợ đào sau một khoảng thời gian thì có một người chiến thắng. Việc này khiến cho công sức của các thợ khác là lãng phí.

Ngoài ra việc các thợ đào sử dụng hệ thống tiêu thụ năng lượng điện lớn cũng được xem là một gánh nắng về năng lượng điện tiêu thụ tại một số quốc gia.

### Mạng lưới lưu trữ đang bị quá tải

Theo các chuyên gia, việc lưu trữ của Blockchain Bitcoin hiện tại cần rất nhiều dung lượng để có thể lưu trữ và tốc độ phát triển có thể tăng trưởng mạnh mẽ trong tương lai. Chính vì vậy nó có thể vượt xa dung lượng lưu trữ của các Node dẫn đến mất các node nếu như kích thước sổ cái lớn hơn các Node.

# 

# Phần 2: Triển khai ứng dụng Blockchain sử dụng ECDSA

Với phần triển khai ứng dụng, nhóm sẽ dùng phần mềm Python để thực hiện. Áp dụng từ các lý thuyết đã được tổng quan ở phần 1, nhóm sẽ tạo ra các module cho từng phần sau đó sẽ kết hợp lại để tạo thành 1 ứng dụng tiền điện tử có đầy đủ tính năng.

# Chương I: Triển khai ECDSA

Đầu tiên, nhóm sẽ thực hiện việc tạo ra các khoá để sinh chữ kỹ điện tử và cũng như có địa chỉ để gửi bằng việc sử dụng thư viện ecdsa trên python. Giới thiệu qua một chút về thư viện, ecdsa dễ sử dụng để thực hiện các ứng dụng với các hệ mật dựa trên đường cong Elliptic như là ECDSA (Elliptic Curve Digital Signature Algorithm), EdDSA (Edward-curve Digital Signature Algorithm), ECDH (Elliptic Curve Diffie-Hellman). Với thư viện, ta có thể dễ dàng tạo cặp chìa(chìa bí mật và chìa công khai), ký, xác minh tin nhắn. Ngoài ra còn có thêm nhiều tính năng khác.

Để cài đặt thư viện, dùng cú pháp:

#### pip install ecdsa

Tạo 1 file có tên *KeyGen.py*, bên trong ghi

#### from ecdsa import SigningKey, SECP256k1

Ở đây, chúng ta đã thêm các module của ecdsa gồm:

* *SigningKey*: Dùng để tạo ra khoá bí mật và khoá công khai
* *SECP256k1*: Đường cong Elliptic tiêu chuẩn
* SEC: **S**tandard for **E**fficient **C**ryptography: Tiêu chuẩn cho mật mã hiệu quả
* P: các ký tự đằng sau là các thông số (**P**arameter) của đường cong
* 256: Độ dài bit của kích thước trường
* k: Đường cong **K**olbitz
* 1: số thứ tự

Để tạo ra cặp chìa:

#### privateKey = SigningKey.generate(curve = SECP256k1)

#### publicKey = privateKey.verifying\_key

#### print(‘Private Key: ‘, privateKey.to\_string().hex())

#### print(‘Public Key: ‘,publicKey.to\_string().hex())

Lệnh 1 dùng method của *SigningKey* là *generate* áp thêm tên đường cong *SECP256k1* vào để tạo ra khoá bí mật tương ứng, lệnh 2 sẽ tạo ra khoá công khai từ biến *verying\_key* của *privateKey*. Sau đó in 2 khoá ra, *\*Key.to\_string().hex()* dùng để in khoá ra ở kiểu String và dạng Hex để dễ đọc.

*Các lệnh và kết quả*

# Chương II: Triển khai Blockchain

Với phần lý thuyết tổng quan ở Chương I, ta có thể từ đó tạo ra blockchain đơn giản để thể hiện những lý thuyết đã nói. Phần Blockchain này sẽ chỉ mang tính giáo dục và thể hiện như một ví dụ cho lý thuyết nên nó có thể khác với các kiến trúc blockchain khác. Ngoài ra để hiểu rõ được các dòng lệnh tiếp theo, người đọc cần có kiến thức về hướng đối tượng (OOP) của Python. Đầu tiên, ta thêm thư viện hashlib để dùng hàm sha256 để băm khối và thư viện datetime để thêm thời gian.

#### 

#### from hashlib import sha256 from datetime import datetime

Sau đó, ta định nghĩ Block:

#### 

| 1 | class Block: |
| --- | --- |
| 2 | def \_\_init\_\_(self, timestamp, data, previousHash = ‘’): |
| 3 | self.timestamp = timestamp |
| 4 | self.data = data |
| 5 | self.previousHash = previousHash |
| 6 | self.nonce = 0 |
| 7 | self.hash = self.calHash() |
| 8 | def calHash(self): |
| 9 | return sha256((self.previousHash + |
| 10 | str(self.timestamp) + |
| 11 | str(self.nonce) + |
| 12 | ‘’.join(self.data)) |
| 13 | .encode()).hexdigest() |
| 14 | def miningBlock(self, difficulty): |
| 15 | while(self.hash[:difficulty] != ''.join(['0' for i \ |
| 16 | in \ range(difficulty)])): |
| 17 | self.nonce += 1 |
| 18 | self.hash = self.calHash() |
| 19 | print(‘Block mined: “,self.hash) |

#### 

Dòng 1: Tạo lớp tên là Block

Dòng 2-7: Khởi tạo cái giá trị của Block:

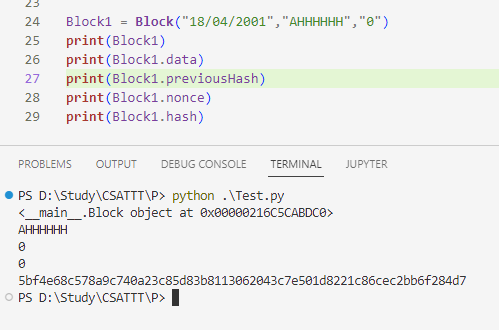
* self: giá trị trỏ về bản thân
* timestamp: thời gian Block được tạo ra
* data: Nội dung của Block
* previousHash: Giá trị băm của Block trước mặc định sẽ rỗng
* hash: Giá trị băm của Block hiện tại sẽ được gán với biến calHash() để tính giá trị của Block hiện tại

Dòng 8-13: Tạo hàm calHash() có nhiệm vụ tính hash, hàm sẽ dùng hàm sha256 băm các giá trị chuỗi (str) của Block nên cần phải có hàm str() để chuyển giá trị về chuỗi và trả về dạng hexa

Dòng 14-19: Tạo Hàm miningBlock() có nhiệm vụ đào Block hay xác thực Block cùng với độ khó (Difficulty). Độ khó giúp cho Block được tạo ra có ý nghĩa hơn và giúp tránh việc Block bị tạo ra quá nhiều dẫn đến có khả năng bị điều khiển.

Ta sẽ test thử xem class Block có hoạt động như mong đợi không bằng cách làm như hình bên dưới:

* Ở dòng đầu tiên: Python hiện ra địa chỉ của khối
* Dòng AHHHHHH: Dữ liệu lúc ta cho vào
* Dòng 0 đầu tiên: Là giá trị hash của khối trước nhưng hiện thời ta chưa dùng đến
* Dòng 0 tiếp theo: Là giá trị nonce ta cũng chưa dùng đến
* Dòng cuối cùng: Là giá trị băm của khối hiện tại

*Kết quả*

Tiếp theo là lớp BlockChain:

| 1 | class BlockChain: |
| --- | --- |
| 2 | def \_\_init\_\_(self, difficulty): |
| 3 | self.chain = [self.genesisBlock()] |
| 4 | self.difficulty = difficulty |
| 5 | self.pendingBlock = [] |
| 6 | def genesisBlock(self): |
| 7 | return Block(datetime(2022, 8,1),\ ”GenesisBlock”,”0”) |
| 8 | def getLastBlock(self): |
| 9 | return self.chain[len(self.chain)-1] |
| 10 | def minePendingBlock(self): |
| 11 | block = Block(datetime.now(), self.pendingBlock,\ self.getLastBlock().hash) |
| 12 | block.miningBlock(self.difficulty) |
| 13 | print(‘Block successfully mined!’) |
| 14 | self.chain.append(block) |
| 15 | self.pendingBlock = [] |
| 16 | def addData(self,data): |
| 17 | self.pendingBlock.append(data) |
| 18 | def printChain(self): |
| 19 | for i in range(len(self.chain)): |
| 20 | if i == 0: continue |
| 21 | print('Time: ',self.chain[i].timestamp) |
| 22 | print('Data: ',self.chain[i].data) |
| 23 | print('Nonce: ',self.chain[i].nonce) |
| 24 | print('PreviousHash: ',self.chain[i].previousHash) |
| 25 | print('Hash: ',self.chain[i].hash) |

Dòng 1: Tạo lớp BlockChain

Dòng 2-5: Khởi tạo giá trị cho lớp BlockChain:

* self.chain: tạo 1 danh sách khối, với hàm GenesisBlock (Khối khởi nguyên) để tạo khối đầu tiên trong danh sách
* self.difficulty: Tạo độ khó cho khối
* self.pendingBlock : Khối chờ, chứa thông tin

Dòng 6-7: Hàm tạo khối khởi nguyên, với thông tin do mình tự đặt

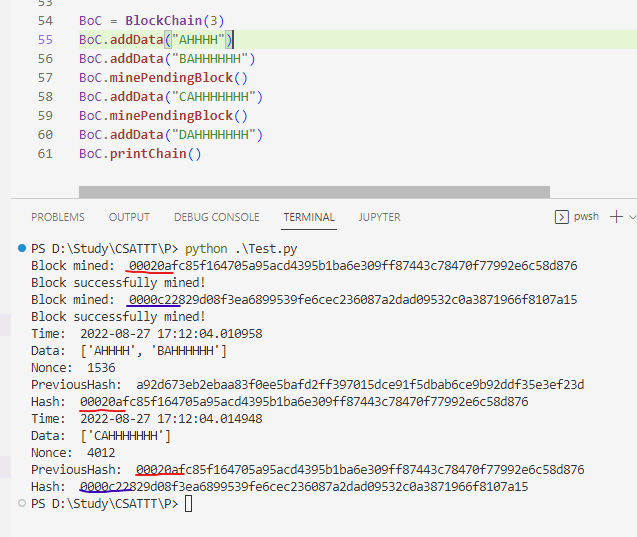
Dòng 8-9: Lấy khối ở cuối danh sách

Dòng 10-15: Hàm đào khối cho khối chờ. Khi tạo khối, dữ liệu sẽ được đưa vào danh sách chờ. Chờ đến khi có ai đó thực hiện hành động đào khối để tạo ra hàm băm tương ứng với hàm băm hiện tại của khối cộng với với độ khó. Khi khối đã được tạo ra thì tất cả dữ liệu trong khối chờ sẽ được đưa vào khối mới được tạo sau đó thêm vào danh sách và làm trống khối chờ cho dữ liệu tiếp theo.

Dòng 16-17: Thêm dữ liệu cho khối tiếp theo

Dòng 18-25: Xuất kết quả của tất cả các khối có trong danh sách trừ khối bắt đầu

Tiếp tục kiểm tra code xem nó có hoạt động hay không.



*Kết quả*

Ở dòng lệnh:

* Dòng 54: Ta triển khai BlockChain với độ khó là 3
* Từ dòng 55-61: Ta thêm dữ liệu và xuất ra tất cả các khối

Ở phần kết quả:

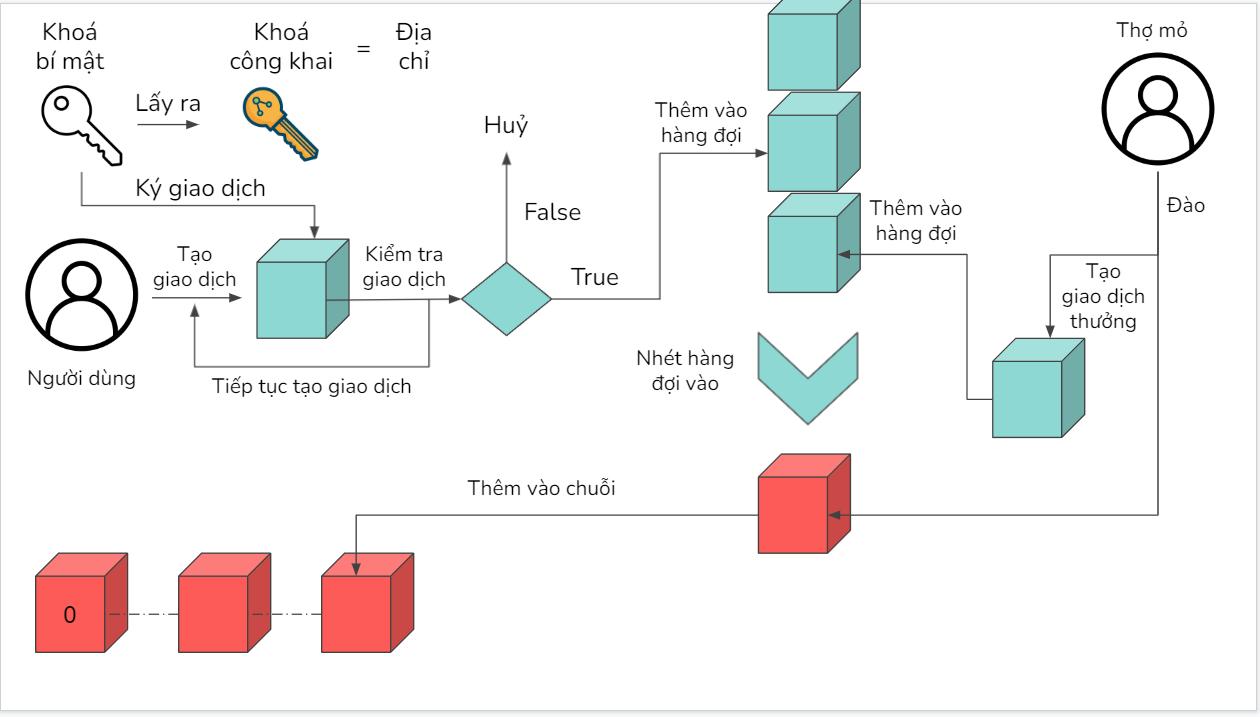
* Ở 4 dòng đầu: Ta có kết quả giá trị hash và thông báo đã đào thành công
* Ở 5 dòng kế:
* Thời gian tạo khối
* 2 phần dữ liệu: Ở đây có 2 phần dữ liệu do trước đó ta thêm 2 dữ liệu liên tiếp nên cả 2 đã vào khối chờ, sau khi có khối được đào cả 2 dữ liệu sẽ vào cùng 1 khối.
* Nonce: thể hiện số ngẫu nhiên để ra được giá trị hash phù hợp với độ khó.
* PreviousHash thể hiện giá trị hash của khối trước (Hiện tại khối trước là khối bắt đầu).
* Hash, giá trị băm của khối, Hash chắc chắn sẽ có 3 số 0 đứng trước thể hiện cho độ khó là 3 mà ta đã khai báo ở dòng 54.
* Ở 5 dòng cuối cùng như 5 dòng trên.

Như mình đã đánh dấu trên hình, giá trị hash lúc khi đào phải giống với giá trị hash của khối và giá trị PreviousHash của khối sau phải giống giá trị Hash của khối trước. Nhưng hiện tại, ta không thấy phần của dữ liệu “DAHHHHHHH” bởi vì chưa có khối nào được đào nên hiện tại nó đang nằm ở khối chờ.

Đây chỉ là phần mình làm để thể hiện BlockChain 1 cách đơn giản, từ đây ta có thể chỉnh sửa, thêm bớt nhiều thứ để tạo ra 1 loại blockchain phù hợp với yêu cầu của cá nhân hay tổ chức. Kế tiếp mình sẽ kết hợp ECDSA và BlockChain lại với nhau.

# Chương III: Triển khai ứng dụng Blockchain

Bây giờ, mình sẽ tạo ra 1 ứng dụng tiền điện tử dùng công nghệ Blockchain và ECDSA tên là Coin. Nó sẽ vừa có khả năng xác thực vừa có khả năng mã hoá, ngoài ra còn có khả năng chuyển tiền, lấy số dư. Ứng dụng sẽ có quá trình hoạt động như sau:



Đầu tiên, mình sẽ tạo ra các tệp tin cần thiết trong 1 thư mục, gồm:

* BlockChain.py : Chứa mọi thứ liên quan đến blockchain
* KeyGen.py : Dùng để tạo ra khoá công khai và khoá bí mật có tác dụng như địa chỉ
* key.txt : Nơi mình lưu khoá bí mật
* main.py : Nơi để thực thi các tính năng

Sau khi đã tạo các tệp cần thiết, mình sẽ làm cái dễ trước đó là KeyGen.py

KeyGen.py

from **ecdsa** import **SigningKey**, SECP256k1

privateKey = **SigningKey**.**generate**(curve=SECP256k1)

publicKey = privateKey.verifying\_key

**print**('Private key: ', privateKey.**to\_string**().**hex**())

**print**('Public key: ', publicKey.to\_string().hex())

*# Tạo Key*

Sau khi chạy tệp KeyGen.py, chúng ta sẽ có 2 khoá là bí mật và công khai. Ta copy khoá bí mật và tệp key.txt đã tạo, còn khoá công khai thì không cần. Tiếp theo sẽ là tệp BlockChain.py

from **asyncio**.**windows\_events** import NULL

from **datetime** import **datetime**

from **hashlib** import **sha256**

import **json**

from **ecdsa** import **SigningKey**, **VerifyingKey**, SECP256k1, **keys**

Ở đây, mình thêm các thư viện cần thiết cho bài:

* NULL để cho python nhận biết giá trị NULL
* datetime để lấy thời gian
* sha256 là hàm băm
* json để mình xuất kết quả đẹp hơn
* ecdsa sẽ dành cho phần ký, xác thực và lấy địa chỉ công khai

class **Transaction**:

'''

Make a Transaction

'''

def **\_\_init\_\_**(self, fromAdd, toAdd : **VerifyingKey**, amount):

self.fromAdd = fromAdd

self.toAdd = toAdd

self.amount = amount

Lớp Transaction sẽ thể hiện cho 1 giao dịch, nó sẽ có địa chỉ nơi gửi, địa chỉ nơi nhận và số lượng.

def **calHash**(self):

'''

Make hash for signature

Hash make up of FromAddress, ToAddress, Amount

'''

if **type**(self.toAdd) == **str**:

return **sha256**(**str**(self.fromAdd.to\_string().hex()\

+ self.toAdd + **str**(self.amount)).**encode**('utf-8')).**hexdigest**()

return **sha256**(**str**(self.fromAdd.to\_string().hex() + self.toAdd.**to\_string**()\

.hex() + **str**(self.amount)).**encode**('utf-8')).**hexdigest**()

Hàm calHash() trong Transaction sẽ tạo ra giá trị hash phục vụ cho việc ký số, giá trị hash sẽ có địa chỉ tới, địa chỉ gửi và số lượng

def **signTrans**(self, sk):

'''

Sign the Transaction to authenticate the owner of the wallet/sender

'''

if (sk.verifying\_key != self.fromAdd): *#Verify owner/sender PublicKey*

raise **ValueError**('You can\'t sign transaction for other wallet!')

hashTrans = self.**calHash**() *# Make hash for signing*

sig = **SigningKey**.**sign**(sk,hashTrans.**encode**('utf-8')) *#Signing Transaction*

self.signature = sig

Hàm signTrans([Khoá bí mật]) trong Transaction sẽ có nhiệm vụ ký giao dịch. Hàm nhận khoá bí mật, sau đó sẽ lấy khoá công khai từ khoá bí mật so sánh với địa chỉ gửi để chắc chắn rằng người gửi là chủ của ví tiền

def **isValid**(self):

'''

Validate Transaction

'''

if(self.fromAdd == NULL): return True *#Validate for the mining reward*

if(not self.signature or **len**(self.signature) == 0): *#Check for Signature*

raise **ValueError**('No signature in this transaction')

pubkey = **VerifyingKey**.**from\_string**(self.fromAdd.to\_string(),\

SECP256k1) *#Extract PublicKey*

try:

return pubkey.**verify**(self.signature, \

self.**calHash**().**encode**()) *#Verifying signature*

except:

return False

Hàm isValid() trong Transaction sẽ xác nhận giao dịch này có hợp lệ hay không bằng cách chứng thực chữ ký số. Hết lớp Transaction.

class **Block**:

fm = '%d/%m/%Y'

def **\_\_init\_\_**(self, timestamp, transaction:**Transaction**, previousHash = ""):

'''

Make a block

'''

self.timestamp = timestamp.strftime(**Block**.fm)

self.transaction = transaction

self.previousHash = previousHash

self.nonce = **int**(0)

self.toAdress = []

self.fromAdress = []

self.amount = []

trans = self.transaction

if trans != 'GenesisBlock':

for i in trans:

self.toAdress.**append**(i.toAdd.to\_string().hex())

if **type**(i.fromAdd) != **int**:

self.fromAdress.**append**(i.fromAdd.to\_string().hex())

self.amount.**append**(**str**(i.amount))

self.hash = self.**calHash**()

Lớp Block sẽ thể hiện 1 khối trong blockchain trong đó có thời gian tạo khối, các giao dịch, giá trị hash trước, số nonce, địa chỉ tới, địa chỉ gửi, số lượng và giá trị hash của khối.

def **calHash**(self):

'''

Make block's hash

'''

return **sha256**((self.previousHash + self.timestamp + **str**(self.nonce) + \

', '.**join**(self.toAdress) + ', '.**join**(self.toAdress) + \

', '.**join**(self.amount)).encode()).**hexdigest**()

Hàm calHash() trong Block sẽ tính ra giá trị hash của khối từ các giá trị trong khối

def **\_\_str\_\_**(self) -> **dict**:

'''

print Block

'''

return {

'timestamp' : self.timestamp,

'previousHash' : self.previousHash,

'hash' : self.hash,

'nonce': self.nonce,

'FromAddress' : ', '.**join**(self.fromAdress),

'ToAddress' : ', '.**join**(self.toAdress),

'Amount' : ', '.**join**(self.amount)

}

Hàm \_\_str\_\_() trong Block sẽ thể hiện khối theo định dạng như trên

def **mineBlock**(self, difficulty):

'''

mine Block with a difficulty

ex: difficulty = 3 -> block's hash: 000+hash

'''

while(self.hash[:difficulty] != \

''.**join**(['0' for i in **range**(difficulty)])):

self.nonce += 1

self.hash = self.**calHash**()

**print**("Block mined: " + self.hash)

Hàm mineBlock([Độ khó]) trong Block sẽ tạo ra giá trị hash của khối phù hợp với độ khó

def **hasValidTrans**(self):

'''

Confirm the transaction

'''

trans = self.transaction

for tran in trans:

if (not tran.isValid()):

return False

return True

Hàm hasValidTrans() trong Block sẽ xác minh giao dịch một lần nữa

class **BlockChain**:

'''

Make a Chain of Block with difficulty of hash

'''

def **\_\_init\_\_**(self, difficulty):

self.chain = [self.**genesisBlock**()] *# Make a Chain with the first block is Genesis Block*

self.difficulty = difficulty

self.pendingTrans = [] *# Make a list of pending transaction*

self.miningReward = 100

Lớp BlockChain sẽ tạo ra chuỗi khối như khái niệm của nó, nó sẽ có:

* Danh sách khối trong đó có hàm genesisBlock() để tạo khối đầu tiên
* Độ khó của hash
* Khối chờ của giao dịch
* Phần thưởng cho người đào khối

def **genesisBlock**(self):

'''

make The First Block: The Beginning of the Chain

'''

return **Block**(**datetime**(2022, 8, 1),"GenesisBlock","0")

Hàm genesisBlock() trong BlockChain sẽ tạo khối đầu tiên

def **getLastBlock**(self):

'''

get The Last Block: The End of the Chain

'''

return self.chain[**len**(self.chain)-1]

Hàm getLastesBlock() trong BlockChain sẽ lấy khối cuối cùng danh sách

def **minePendingTrans**(self, miningRewardAdd):

'''

add a mined Block to a pending chain for a pending Transaction

'''

rewardTx = **Transaction**(NULL, miningRewardAdd, self.miningReward) *#This transaction will send reward to the miner*

self.pendingTrans.**append**(rewardTx) *#Send this transaction to pending chain*

block = **Block**(**datetime**.**now**(), self.pendingTrans,self.**getLastBlock**().hash) *# Make a block consist of the made time, list of pending Trans and the last block's hash*

block.**mineBlock**(self.difficulty)

**print**('Block successfully mined!')

self.chain.**append**(block) *# Add Block to Chain*

self.pendingTrans = [] *# Empty pending List*

Hàm minePendingTrans([Địa chỉ để gửi phần thưởng]) trong BlockChain, đầu tiên sẽ tạo 1 giao dịch để gửi phần thưởng cho địa chỉ đào, giao dịch sẽ được đưa vào khối chờ của giao dịch. 1 Khối sẽ được tạo chứa các giao dịch chờ và hash của khối trước nó. Khối này sẽ bắt đầu quá trình tạo ra giá trị hash cho bản thân nó với độ khó lúc đó. Sau khi hoàn thành, khối được thêm vào danh sách và làm trống khối chờ của giao dịch

def **addTrans**(self,trans):

'''

Add Transaction to pending list wait for a block

'''

if (not trans.fromAdd or not trans.toAdd): *# Check for address*

raise **ValueError**('Transaction must include from and to address')

if (not trans.isValid()): *# Check for Signature*

raise **ValueError**('Cant\' add invalid transction to chain')

self.pendingTrans.**append**(trans) *# Add transaction to a pending list*

Hàm addTrans([Giao dịch]) trong Blockchain sẽ kiểm tra giao dịch và thêm vào khối chờ của giao dịch

def **getBalance**(self, address):

'''

Get Balance of a Address

Because there is no saved file or database, we get balance by calculating how many Coins get in and out of the address

'''

balance = 0

for block in self.chain:

trans = block.transaction

if trans == 'GenesisBlock': continue

for add in trans:

if add.fromAdd == address:

balance -= add.amount

if add.toAdd == address:

balance += add.amount

return balance

Hàm getBalance([địa chỉ]) trong Blockchain trả về số dư của địa chỉ, do không có lưu trữ để có số dư nên việc tính số dư sẽ được thực bằng cách tính bao nhiêu đi ra và bao nhiêu vào địa chỉ

def **printChain**(self):

'''

Show all the block

'''

for i in **range**(**len**(self.chain)):

if i == 0: continue

**print**(**json**.**dumps**(self.chain[i].**\_\_str\_\_**(),separators=('',':'),indent=2))

Hàm printChain() trong Blockchain sẽ trả về tất cả khối có trong chuỗi dưới dạng json

def **isChainValid**(self):

'''

Check the Chain if:

There is invalid Transaction

There is Wrong Hash of the current Block

There is Wrong Hash of the previous Block

'''

for i in **range**(**len**(self.chain)):

if i == 0: continue

CurBlock = self.chain[i]

PreBlock = self.chain[i-1]

if not CurBlock.**hasValidTrans**():

**print**("Invalid transaction in Block[",i,"]")

return False

if CurBlock.hash != CurBlock.**calHash**():

**print**("Invalid hash in Block[",i,"]")

return False

if CurBlock.previousHash != PreBlock.hash: return False

return True

Hàm isChainValid() trong Blockchain sẽ kiểm tra xem chuỗi có hợp lệ không bằng cách kiểm tra giao dịch có hợp lệ, hash có hợp lệ.

Code hoàn chỉnh của BlockChain.py:

class **Transaction**:

'''

Make a Transaction

'''

def **\_\_init\_\_**(self, fromAdd, toAdd : **VerifyingKey**, amount):

self.fromAdd = fromAdd

self.toAdd = toAdd

self.amount = amount

def **calHash**(self):

'''

Make hash for signature

Hash make up of FromAddress, ToAddress, Amount

'''

if **type**(self.toAdd) == **str**:

return **sha256**(**str**(self.fromAdd.to\_string().hex()\

+ self.toAdd + **str**(self.amount)).**encode**('utf-8')).**hexdigest**()

return **sha256**(**str**(self.fromAdd.to\_string().hex() + self.toAdd.**to\_string**()\

.hex() + **str**(self.amount)).**encode**('utf-8')).**hexdigest**()

def **signTrans**(self, sk):

'''

Sign the Transaction to authenticate the owner of the wallet/sender

'''

if (sk.verifying\_key != self.fromAdd): *#Verify owner/sender PublicKey*

raise **ValueError**('You can\'t sign transaction for other wallet!')

hashTrans = self.**calHash**() *# Make hash for signing*

sig = **SigningKey**.**sign**(sk,hashTrans.**encode**('utf-8')) *#Signing Transaction*

self.signature = sig

def **isValid**(self):

'''

Validate Transaction

'''

if(self.fromAdd == NULL): return True *#Validate for the mining reward*

if(not self.signature or **len**(self.signature) == 0): *#Check for Signature*

raise **ValueError**('No signature in this transaction')

pubkey = **VerifyingKey**.**from\_string**(self.fromAdd.to\_string(),\

SECP256k1) *#Extract PublicKey*

try:

return pubkey.**verify**(self.signature, \

self.**calHash**().**encode**()) *#Verifying signature*

except:

return False

class **Block**:

fm = '%d/%m/%Y'

def **\_\_init\_\_**(self, timestamp, transaction:**Transaction**, previousHash = ""):

'''

Make a block

'''

self.timestamp = timestamp.strftime(**Block**.fm)

self.transaction = transaction

self.previousHash = previousHash

self.nonce = **int**(0)

self.toAdress = []

self.fromAdress = []

self.amount = []

trans = self.transaction

if trans != 'GenesisBlock':

for i in trans:

self.toAdress.**append**(i.toAdd.to\_string().hex())

if **type**(i.fromAdd) != **int**:

self.fromAdress.**append**(i.fromAdd.to\_string().hex())

self.amount.**append**(**str**(i.amount))

self.hash = self.**calHash**()

def **calHash**(self):

'''

Make block's hash

'''

return **sha256**((self.previousHash + self.timestamp + **str**(self.nonce) + \

', '.**join**(self.toAdress) + ', '.**join**(self.toAdress) + \

def **\_\_str\_\_**(self) -> **dict**:

'''

print Block

'''

return {

'timestamp' : self.timestamp,

'previousHash' : self.previousHash,

'hash' : self.hash,

'nonce': self.nonce,

'FromAddress' : ', '.**join**(self.fromAdress),

'ToAddress' : ', '.**join**(self.toAdress),

'Amount' : ', '.**join**(self.amount)

}

def **mineBlock**(self, difficulty):

'''

mine Block with a difficulty

ex: difficulty = 3 -> block's hash: 000+hash

'''

while(self.hash[:difficulty] != \

''.**join**(['0' for i in **range**(difficulty)])):

self.nonce += 1

self.hash = self.**calHash**()

**print**("Block mined: " + self.hash)

def **hasValidTrans**(self):

'''

Confirm the transaction

'''

trans = self.transaction

for tran in trans:

if (not tran.isValid()):

return False

return True

class **BlockChain**:

'''

Make a Chain of Block with difficulty of hash

'''

def **\_\_init\_\_**(self, difficulty):

self.chain = [self.**genesisBlock**()] *# Make a Chain with the first block is Genesis Block*

self.difficulty = difficulty

self.pendingTrans = [] *# Make a list of pending transaction*

self.miningReward = 100

def **genesisBlock**(self):

'''

make The First Block: The Beginning of the Chain

'''

return **Block**(**datetime**(2022, 8, 1),"GenesisBlock","0")

def **getLastBlock**(self):

'''

get The Last Block: The End of the Chain

'''

return self.chain[**len**(self.chain)-1]

def **minePendingTrans**(self, miningRewardAdd):

'''

add a mined Block to a pending chain for a pending Transaction

'''

rewardTx = **Transaction**(NULL, miningRewardAdd, self.miningReward) *#This transaction will send reward to the miner*

self.pendingTrans.**append**(rewardTx) *#Send this transaction to pending chain*

block = **Block**(**datetime**.**now**(), self.pendingTrans,self.**getLastBlock**().hash) *# Make a block consist of the made time, list of pending Trans and the last block's hash*

block.**mineBlock**(self.difficulty)

**print**('Block successfully mined!')

self.chain.**append**(block) *# Add Block to Chain*

self.pendingTrans = [] *# Empty pending List*

def **addTrans**(self,trans):

'''

Add Transaction to pending list wait for a block

'''

if (not trans.fromAdd or not trans.toAdd): *# Check for address*

raise **ValueError**('Transaction must include from and to address')

if (not trans.isValid()): *# Check for Signature*

raise **ValueError**('Cant\' add invalid transction to chain')

self.pendingTrans.**append**(trans) *# Add transaction to a pending list*

def **getBalance**(self, address):

'''

Get Balance of a Address

Because there is no saved file or database, we get balance by calculating how many Coins get in and out of the address

'''

balance = 0

for block in self.chain:

trans = block.transaction

if trans == 'GenesisBlock': continue

for add in trans:

if add.fromAdd == address:

balance -= add.amount

if add.toAdd == address:

balance += add.amount

return balance

def **printChain**(self):

'''

Show all the block

'''

for i in **range**(**len**(self.chain)):

if i == 0: continue

**print**(**json**.**dumps**(self.chain[i].**\_\_str\_\_**(),separators=('',':'),indent=2))

def **isChainValid**(self):

'''

Check the Chain if:

There is invalid Transaction

There is Wrong Hash of the current Block

There is Wrong Hash of the previous Block

'''

for i in **range**(**len**(self.chain)):

if i == 0: continue

CurBlock = self.chain[i]

PreBlock = self.chain[i-1]

if not CurBlock.**hasValidTrans**():

**print**("Invalid transaction in Block[",i,"]")

return False

if CurBlock.hash != CurBlock.**calHash**():

**print**("Invalid hash in Block[",i,"]")

return False

if CurBlock.previousHash != PreBlock.hash: return False

return True

Cuối cùng là tệp main.py, đây sẽ là tệp mà mình thực hiện cái thao tác minh hoạ cho bài.

from **BlockChain** import **BlockChain**, **Transaction**

from **ecdsa** import **SigningKey**, **VerifyingKey**, SECP256k1

Thêm thư viện ecdsa và module BlockChain, Transaction của tệp BlockChain đã làm ở trên.

def **main**():

def **mining**(address):

'''

Mining Block (Address to send reward)

'''

**print**('Starting the miner...')

Coin.**minePendingTrans**(address)

def **getBalance**(address):

'''

Show balance of Address

'''

**print**('Balance:', Coin.**getBalance**(address))

def **send**(frm, to, amount, key):

'''

Send Coin (frm: from address, to: to address, amount: amount of coin, key: Private key to sign)

'''

trans = **Transaction**(frm, to, amount)

trans.**signTrans**(key)

Coin.**addTrans**(trans)

def **privateKey**(key):

'''

Convert hexkey to bytes for computer

return -> Private Key

'''

return **SigningKey**.**from\_string**(**bytes**.**fromhex**(key),curve=SECP256k1)

def **publicKey**(key):

'''

Make Public key from Private key

return -> Public Key

'''

return key.verifying\_key

*# myKey = SigningKey.from\_string(bytes.fromhex('899cad35b4857b41b3e439d9b86d6e3654193859de9e19d5590ea85343f608ae'),curve=SECP256k1)*

*# myWalletAddress = myKey.verifying\_key*

Các hàm trong main là các hàm từ BlockChain nhưng được làm cho dễ dùng và dễ hiểu hơn. Sau đây sẽ là một vài ví dụ thử nghiệm.

Coin = **BlockChain**(4) *#Generate the chain*

myKey = **privateKey**('899cad35b4857b41b3e439d9b86d6e3654193859de9e19d5590ea85343f608ae')

myWalletAddress = **publicKey**(myKey)

p1key = **privateKey**('ed1b8435585667c3ee34807d3a9a3a3e074bf20f86a6aad82427daa2f3690805')

p1Wallet = **publicKey**(p1key)

p2key = **privateKey**('7fb4d4c835e9fe9a8a4105fc750649fcfb1c38e75d282d94becdf40a66ae77a9')

p2Wallet = **publicKey**(p2key)

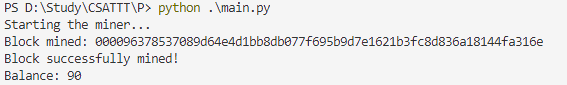
Khởi tạo Blockchain với độ khó là 4 nghĩa là giá trị hash sẽ bắt đầu bằng 4 số 0. Kế tiếp, mình sẽ thêm sẵn 3 khoá bí mật dùng để ký và 3 khoá công khai tương ứng làm địa chỉ ví

**send**(myWalletAddress, p2Wallet, 10, myKey)

**mining**(myWalletAddress)

**getBalance**(myWalletAddress)

Đầu tiên, mình sẽ lấy ví của mình gửi 10 coin cho ví p2, dùng khoá bí mật để ký giao dịch, rồi mình sẽ đào khối và lấy số dư được kết quả như hình dưới



Mình gửi 10 coin và mình đào thì nhận được 100 coin, số dư sẽ là 90 coin

**print**('------')

**send**(p1Wallet, myWalletAddress, 200, p1key)

**send**(p2Wallet, myWalletAddress, 10, p2key)

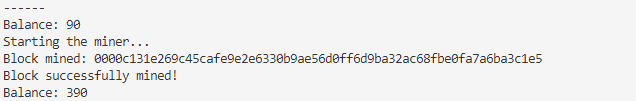
**send**(myWalletAddress, p1Wallet, 10, myKey)

**getBalance**(myWalletAddress)

**mining**(myWalletAddress)

**getBalance**(myWalletAddress)

Sau đó mình gửi đi gửi lại và xuất số dư



Kế tiếp mình sẽ kiểm tra chuỗi và tiến hành thay đổi 1 số thứ.

**print**('Is chain valid?', Coin.**isChainValid**()) *#Check valid of chain*

Mình sẽ kiểm tra chuỗi



Kết quả là chuỗi vẫn ổn, bây giờ mình sẽ thay đổi giá trị số lượng Coin để làm bản thân giàu hơn.

Coin.chain[1].transaction[0].amount = 5000

Và chạy lại lệnh xác minh



Kết quả là chuỗi đã bị xâm phạm. Có thể thấy bằng lệnh printChain()

Coin.**printChain**() *#Show all block*



Dù đã đổi amount thành 5000 nhưng trong chuỗi lại không tiếp nhận con số đó. Do con số bị sửa nằm trong transaction hiện tại, còn những thông tin trên là thông tin của quá khứ.

if \_\_name\_\_ == '\_\_main\_\_':

**main**()

2 câu lệnh trên chỉ để chạy hàm main().

Code của main.py:

from **BlockChain** import **BlockChain**, **Transaction**

from **ecdsa** import **SigningKey**, **VerifyingKey**, SECP256k1

def **main**():

def **mining**(address):

'''

Mining Block (Address to send reward)

'''

**print**('Starting the miner...')

Coin.**minePendingTrans**(address)

def **getBalance**(address):

'''

Show balance of Address

'''

**print**('Balance:', Coin.**getBalance**(address))

def **send**(frm, to, amount, key):

'''

Send Coin (frm: from address, to: to address, amount: amount of coin, key: Private key to sign)

'''

trans = **Transaction**(frm, to, amount)

trans.**signTrans**(key)

Coin.**addTrans**(trans)

def **privateKey**(key):

'''

Convert hexkey to bytes for computer

return -> Private Key

'''

return **SigningKey**.**from\_string**(**bytes**.**fromhex**(key),curve=SECP256k1)

def **publicKey**(key):

'''

Make Public key from Private key

return -> Public Key

'''

return key.verifying\_key

*# myKey = SigningKey.from\_string(bytes.fromhex('899cad35b4857b41b3e439d9b86d6e3654193859de9e19d5590ea85343f608ae'),curve=SECP256k1)*

*# myWalletAddress = myKey.verifying\_key*

Coin = **BlockChain**(4) *#Generate the chain*

myKey = **privateKey**('899cad35b4857b41b3e439d9b86d6e3654193859de9e19d5590ea85343f608ae')

myWalletAddress = **publicKey**(myKey)

p1key = **privateKey**('ed1b8435585667c3ee34807d3a9a3a3e074bf20f86a6aad82427daa2f3690805')

p1Wallet = **publicKey**(p1key)

p2key = **privateKey**('7fb4d4c835e9fe9a8a4105fc750649fcfb1c38e75d282d94becdf40a66ae77a9')

p2Wallet = **publicKey**(p2key)

**send**(myWalletAddress, p2Wallet, 10, myKey)

**mining**(myWalletAddress)

**getBalance**(myWalletAddress)

**print**('------')

**send**(p1Wallet, myWalletAddress, 200, p1key)

**send**(p2Wallet, myWalletAddress, 10, p2key)

**send**(myWalletAddress, p1Wallet, 10, myKey)

**getBalance**(myWalletAddress)

**mining**(myWalletAddress)

**getBalance**(myWalletAddress)

*# Coin.chain[1].transaction[0].amount = 5000 #Test*

**print**('Is chain valid?', Coin.**isChainValid**()) *#Check valid of chain*

Coin.**printChain**() *#Show all block*

if \_\_name\_\_ == '\_\_main\_\_':

**main**()

# Phần 3: Đánh giá và kết luận nghiên cứu

# Chương I: Đánh giá kết quả

Sau phần 1 và 2 của bài, nhóm đã đạt được những mục tiêu lúc đầu bắt đầu bài nghiên cứu. Hiểu về ECSDA, thấy được những điểm mạnh cũng như những điểm cần lo lắng trong ECDSA, 1 thuật toán dù có bảo mật tốt đến mức nào cũng sẽ có những điểm làm cho thuật toán đó mất an toàn. Tiếp theo là chủ đề về blockchain, dù không tạo giống được như Bitcoin, Ethereum nhưng nhóm đã hoàn thành trong việc tạo ra một đồng tiền điện tử của riêng nhóm dù không có tính kinh tế nhưng có tính giáo dục nhiều hơn. Qua việc tạo ra một đồng tiền điện tử, nhóm đã biết cách tạo ra các khối và liên kết các khối lại với nhau để hình thành một chuỗi để tạo thành blockchain, kết hợp thêm ECDSA để thêm tính năng chuyển, gửi tiền và ký giao dịch. Ngoài ra, nhóm còn biết được những điểm mạnh và yếu của công nghệ blockchain, cũng như tại sao lại có nhiều người lại cần nhiều mua card đồ hoạ về đào tiền. Sử dụng ngôn ngữ lập trình Python để thực hiện, nhóm được củng cố thêm về cách hoạt động của hướng đối tượng, cách tương tác và thể hiện dữ liệu trên Python.

# Chương II: Kết luận đề tài

Nhóm đã hoàn thành mục tiêu đề ra có thể thấy đó là một thành công nhưng vẫn còn chỗ cho phát triển:

* Tạo ra một giao diện có đầy đủ chức năng:
* Hiện địa chỉ của ví là khoá công khai khi ta thêm khoá bí mật vào.
* Kiểm tra số dư xem có đủ khả năng giao dịch không.
* Nút để đào tiền và hiện các dữ liệu liên quan.
* Kiểm tra lịch sử.
* Và nhiều thứ khác.
* Tạo ra mạng blockchain peer-to-peer như Bitcoin để có thể giao dịch toàn cầu.
* Chưa thể áp dụng với các vấn đề khác trên thế giới như y tế, sản xuất, dịch vụ,...

# Bảng phân công:

| **Thân Minh Tuấn Phát** | **Phần 1, chương II** |
| --- | --- |
| **Nguyễn Mạnh Tiến** | **Phần 1, chương III** |
| **Ngô Quang Khánh** | **Phần 2** |
| **Cả 3** | **Phần 1, chương I và Phần 3** |

# 

# Tài liệu tham khảo

“How To Time-Stamp a Digital Document”, *Stuar Haber and W. Scott Stornetta,* 1991

<https://link.springer.com/content/pdf/10.1007/BF00196791.pdf>

“Block: 750390.” *Blockchain.com*, 21 8 2022, <https://www.blockchain.com/btc/block/750390> .

“Blockchain – Wikipedia tiếng Việt.” *Wikipedia*, <https://vi.wikipedia.org/wiki/Blockchain> .

Curry, Caleb, performer. “Intro to Digital Signatures | ECDSA Explained.” 11 2 2020. *Intro to Digital Signatures*, <https://www.youtube.com/watch?v=f9eitAS1nsY> .

Hong, Euny. “How Does Bitcoin Mining Work? What Is Crypto Mining?” *Investopedia*, <https://www.investopedia.com/tech/how-does-bitcoin-mining-work/> .

Simply Explained, creator. *Building a blockchain with Javascript*. 2019. *Building a blockchain with Javascript*, <https://www.youtube.com/playlist?list=PLzvRQMJ9HDiTqZmbtFisdXFxul5k0F-Q4> .

snifikino. “Understanding How ECDSA Protects Your Data. : 15 Steps.” *Instructables*, <https://www.instructables.com/Understanding-how-ECDSA-protects-your-data/>