**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA AN TOÀN THÔNG TIN**

****

**BÁO CÁO BÀI TẬP LỚN**

**HỌC PHẦN: MẬT MÃ HỌC CƠ SỞ**

**ĐỀ TÀI: THÁM MÃ BLOCKCHAIN**

Các sinh viên thực hiện:

|  |  |
| --- | --- |
| Lê Tiến Dương | B22DCAT063 |
| Lê Thành Đạt | B22DCAT073 |
| Hoàng Văn Hướng | B22DCAT156 |
| Phạm Đức Quân | B22DCAT239 |
| Phạm Minh Tâm | B22DCAT255 |
| Nguyễn Khắc Trí | B22DCAT303 |

Tên nhóm: 05

Tên lớp: N4

Giảng viên hướng dẫn: PGS.TS. Đỗ Xuân Chợ

**HÀ NỘI 5-2025**

MỤC LỤC

[MỤC LỤC 2](#_Toc199331331)

[CHƯƠNG 1. Mở đầu 3](#_Toc199331332)

[1.1 Lý do chọn đề tài 3](#_Toc199331333)

[1.2 Báo cáo này được thực hiện nhằm mục đích 3](#_Toc199331334)

[CHƯƠNG 2. Giới thiệu tổng quan về Blockchain 4](#_Toc199331335)

[2.1 Giới thiệu chung về Blockchain 4](#_Toc199331336)

[2.1.1 Giới thiệu về Blockchain 4](#_Toc199331337)

[***2.1.2* Lịch sử ra đời** 5](#_Toc199331338)

[***2.1.3* Các phiên bản** 5](#_Toc199331339)

[CHƯƠNG 3. Tổng quan về bảo mật mật mã trong Blockchain 7](#_Toc199331340)

[3.1 Hàm băm mật mã (Cryptographic Hash Functions) 7](#_Toc199331341)

[3.2 Chữ ký số (Digital Signatures) 9](#_Toc199331342)

[3.3 Hệ mật mã khóa công khai: Đường cong Elliptic (ECC) 13](#_Toc199331343)

[3.4 Bằng chứng không tiết lộ (Zero-knowledge Proof – ZKP) 15](#_Toc199331344)

[3.5 Vai trò của mật mã học trong Blockchain 16](#_Toc199331345)

[CHƯƠNG 4. Mô phỏng tấn công qua Lab thực hành 19](#_Toc199331346)

[4.1 Tấn công đồng thuận / chèn block giả 19](#_Toc199331347)

[4.1.1 Tấn công vào đồng thuận BLS-BFT 19](#_Toc199331348)

[4.1.2 Tấn công Proof-of-Work giả (PoW-Fake) 22](#_Toc199331349)

[4.2 Thám mã chữ ký số ECDSA 24](#_Toc199331350)

[4.2.1 ECDSA Signature Malleability 24](#_Toc199331351)

[4.2.2 LAB: noncek-ecdsa 26](#_Toc199331352)

[4.3 Lỗ hổng và tấn công trong Smart Contract 29](#_Toc199331353)

[4.3.1 Mục đích bài thực hành 29](#_Toc199331354)

[4.3.2 Yêu cầu đối với sinh viên 30](#_Toc199331355)

[4.3.3 Nội dung bài thực hành 30](#_Toc199331356)

[Kết luận 33](#_Toc199331357)

[TÀI LIỆU THAM KHẢO 34](#_Toc199331358)

1. Mở đầu
   1. Lý do chọn đề tài

Trong thời đại công nghệ số phát triển mạnh mẽ, blockchain đã và đang trở thành một trong những nền tảng công nghệ cốt lõi, được ứng dụng rộng rãi trong nhiều lĩnh vực như tài chính, y tế, chuỗi cung ứng và quản trị dữ liệu. Với các đặc điểm nổi bật như phi tập trung, minh bạch và không thể thay đổi, blockchain được kỳ vọng sẽ thay đổi cách con người trao đổi và lưu trữ thông tin một cách an toàn và hiệu quả.

Tuy nhiên, để đạt được những tính năng đó, blockchain phụ thuộc rất lớn vào các cơ chế mật mã học như hàm băm, chữ ký số, mã hóa khóa công khai - khóa riêng, và các thuật toán đồng thuận. Mặc dù các kỹ thuật mật mã này đã được chứng minh là an toàn trong nhiều năm qua, nhưng chúng vẫn tồn tại những điểm yếu tiềm ẩn có thể bị khai thác thông qua các hình thức tấn công thám mã (cryptanalysis).

Thám mã trong blockchain là việc nghiên cứu, phân tích và khai thác các điểm yếu trong việc triển khai các thuật toán mật mã hoặc giao thức liên quan đến blockchain nhằm mục đích làm lộ khóa bí mật, giả mạo giao dịch hoặc phá hoại tính toàn vẹn của hệ thống. Đây là mối đe dọa nghiêm trọng, đặc biệt trong bối cảnh các giao dịch tiền mã hóa mang giá trị tài chính cao, và việc mất mát tài sản số có thể xảy ra trong tích tắc.

* 1. Báo cáo này được thực hiện nhằm mục đích
* Trình bày tổng quan về các kỹ thuật mật mã được sử dụng trong blockchain.
* Phân tích các lỗ hổng bảo mật đã từng bị khai thác trong thực tế như: Signature Malleability, Reuse-k Attack, Timing Attack, và các tấn công vào giao thức đồng thuận BFT.
* Mô phỏng một số cuộc tấn công điển hình bằng các bài thực hành Labtainer.
* Đề xuất các phương án phòng chống và khuyến nghị an toàn.

Thông qua báo cáo, người đọc sẽ hiểu rõ rằng: blockchain không miễn nhiễm với tấn công, và việc nắm vững kỹ thuật thám mã không chỉ giúp phát hiện lỗ hổng mà còn là tiền đề để thiết kế hệ thống an toàn hơn trong tương lai.

1. Giới thiệu tổng quan về Blockchain
   1. Giới thiệu chung về Blockchain
      1. Giới thiệu về Blockchain

Blockchain là cuốn sổ cái kỹ thuật số chống giả mạo được triển khai theo mô hình phân tán (tức là không có kho lưu trữ trung tâm) và thường không cần một đơn vị đáng tin cậy chứng thực (như ngân hàng, công ty, chính phủ). Ở mức độ cơ bản, nó cho phép một cộng đồng người dùng ghi các giao dịch vào cuốn sổ cái chia sẻ, mà trong đó, với sự điều hành bình thường của mạng Blockchain thì không giao dịch nào có thể bị thay đổi sau khi xuất bản. Vào năm 2008, ý tưởng Blockchain được kết hợp với một vài công nghệ và khái niệm điện toán khác để tạo ra đồng tiền mã hóa hiện đại: tiền điện tử được bảo vệ bởi các cơ chế mật mã học thay vì nhờ vào bên chứng thực hoặc kho lưu trữ trung tâm.



1. Giới thiệu về Blockchain

Công nghệ này được biết đến rộng rãi vào năm 2009 với sự ra đời của mạng Bitcoin – một trong những đồng tiền mã hóa hiện đại đầu tiên. Ở hệ thống Bitcoin và các hệ thống tương tự, việc chuyển thông tin kỹ thuật số với đại diện là tiền điện tử diễn ra trong một hệ thống phân tán. Người dùng Bitcoin có thể ký chữ ký số và chuyển tài sản của mình sang người khác và Bitcoin ghi lại các giao dịch này công khai, cho phép những người tham gia mạng xác minh độc lập tính hợp lệ của giao dịch. Công nghệ Blockchain do đó được xem là giải pháp chung cho các đồng tiền mã hóa sau này.

Blockchain có thể được định nghĩa như sau: “Blockchain là cuốn sổ cái kỹ thuật số của các giao dịch được ký bằng mật mã. Mỗi khối được liên kết mã hóa với khối trước nó sau khi được xác thực thì trải qua một quyết định đồng thuận. Khi một khối mới thêm vào, khối cũ hơn trở nên khó bị chỉnh sửa. Cuốn sổ cái sau đó được sao chép đến toàn bộ mạng và bất kỳ xung đột nào được giải quyết tự động thông qua các quy tắc được thiết lập.”

**Lịch sử ra đời**

Ý tưởng chính đứng sau công nghệ Blockchain này nổi lên vào cuối những năm 1980, đầu năm 1990. Vào năm 1989, Leslie Lamport đã phát triển giao thức Paxos. Năm 1990, ông có bài báo The Part-Time Parliament được gửi đến ACM Transaction on Computer Systems; bài báo được phát hành lần cuối vào năm 1998. Bài báo miêu tả một mô hình đồng thuận giúp đạt được thỏa thuận trên một kết quả của mạng lưới máy tính – nơi mà các máy tính hoặc bản thân mạng có thể không ổn định. Năm 1991, một chuỗi thông tin được ký đã được dùng như một cuốn sổ cái điện tử cho các tài liệu có chữ ký kỹ thuật số theo cách mà không dễ dàng để tài liệu được ký nào bị thay đổi. Các khái niệm này được kết hợp và áp dụng vào tiền điện tử năm 2008 và được miêu tả trong bài báo, Bitcoin: A Peer-to-Peer Electronic Cash System, được xuất bản giả bởi Satoshi Nakamoto, và sau đó năm 2009 với sự ra đời của tiền điện tử Bitcoin.

Việc sử dụng Blockchain cho phép Bitcoin được triển khai theo kiểu phân tán, như vậy không có người dùng đơn lẻ điều khiển được tiền điện tử và không có khuyết điểm tồn tại đơn lẻ. Lợi ích chính là cho phép các giao dịch trực triếp giữa những người dùng mà không cần bên thứ ba đáng tin cậy. Nó cũng cho phép phát hành tiền mới theo cách được định nghĩa đến những người quản lý việc xuất bản các khối mới và duy trì bản sao của sổ cái, những người đó được gọi là miners ở Bitcoin. Bằng cách sử dụng cơ chế đồng thuận để duy trì và một cơ chế tự kiểm soát được tạo ra để đảm bảo rằng chỉ có các giao dịch và các khối hợp lệ mới được thêm vào Blockchain.

**Các phiên bản**

**A diagram of a blockchain

AI-generated content may be incorrect.**

1. Các phiên bản của Blockchain
2. **Blockchain 1.0: Tiền mã hoá (Cryptocurrency)**

Mục tiêu của công nghệ này là cung cấp cách thức giao dịch, thanh toán qua không gian mạng trực tiếp và an toàn. Việc triển khai công nghệ sổ cái phân tán (Distributed Ledger Technology - DLT) đã dẫn đến ứng dụng đầu tiên là Cryptocurrency (tiền mã hóa). Bitcoin chính là ví dụ nổi bật nhất của nền tảng này.

1. **Blockchain 2.0: Hợp đồng thông minh (Smart Contract)**

Hợp đồng thông minh được phát triển dựa trên nền tảng Blockchain 2.0. Đây là các chương trình máy tính miễn phí thực thi tự động và kiểm tra các điều kiện được xác định trước đó như hỗ trợ, xác minh.  Ví dụ điển hình cho nền tảng này là Ethereum – một giao thức cho phép người dùng tạo ra những hợp đồng thông minh thay thế những phiên bản truyền thống.

A blue and white text with a cellphone in the middle

AI-generated content may be incorrect.

1. Hợp đồng thông minh (Smart Contract).
2. **Blockchain 3.0: Ứng dụng phi tập trung (DApps)**

Phiên bản này mở rộng ứng dụng của Blockchain ra ngoài tài chính, bao gồm y tế, bầu cử, quản lý tài nguyên, logistics và nhiều lĩnh vực khác. Mục tiêu của Blockchain 3.0 là tận dụng tính minh bạch và phi tập trung của công nghệ chuỗi khối để cải thiện hiệu quả và an toàn trong công việc.

1. **Blockchain 4.0: Công nghệ chuỗi khối cho ngành công nghiệp (Blockchain For Industry)**

Mục tiêu của Blockchain 4.0 là giải quyết toàn bộ vấn đề của ba thế hệ trước. Nền tảng này sẽ hỗ trợ doanh nghiệp giải thích các chiến lược và phương pháp để cải thiện hiệu quả kinh doanh.

1. Tổng quan về bảo mật mật mã trong Blockchain

Blockchain hiện đại sử dụng nhiều kỹ thuật mật mã để đảm bảo tính an toàn, minh bạch và tin cậy cho hệ thống. Dưới đây là các kỹ thuật mật mã phổ biến cùng giải thích chi tiết về định nghĩa, cơ chế hoạt động, ứng dụng trong blockchain, cũng như ưu nhược điểm của chúng.

* 1. Hàm băm mật mã (Cryptographic Hash Functions)

**Định nghĩa:** Hàm băm mật mã là một hàm toán học một chiều, nhận đầu vào có độ dài tùy ý và tạo ra đầu ra là một chuỗi bit có độ dài cố định (gọi là giá trị băm hoặc hash). Tính chất quan trọng của hàm băm là tính một chiều (từ đầu ra rất khó hoặc không thể suy ngược ra đầu vào) và nhạy cảm với thay đổi đầu vào (chỉ cần thay đổi nhỏ ở dữ liệu đầu vào sẽ dẫn đến giá trị băm đầu ra thay đổi hoàn toàn).

Trong blockchain, hàm băm được dùng để tạo “dấu vân tay” cho dữ liệu, giúp phát hiện mọi sự thay đổi (dữ liệu bị sửa đổi sẽ cho giá trị băm khác, dễ bị phát hiện). Các blockchain phổ biến sử dụng nhiều hàm băm khác nhau, trong đó có hai hàm nổi bật là **SHA-256** và **Keccak-256 (SHA-3).**

1. **SHA-256**

**Định nghĩa:** SHA-256 (Secure Hash Algorithm 256-bit) là một thuật toán băm thuộc họ SHA-2 do Cơ quan An ninh Quốc gia Hoa Kỳ (NSA) thiết kế. SHA-256 tạo ra giá trị băm dài 256 bit (32 byte) cho mỗi dữ liệu đầu vào. Đây là hàm băm không khóa, nghĩa là không sử dụng khóa bí mật mà chỉ dựa trên tính chất toán học để đảm bảo an toàn.

**Cơ chế hoạt động:** SHA-256 sử dụng cấu trúc nén Merkle–Damgård. Dữ liệu đầu vào được bổ sung thêm bit đệm và chia thành các khối 512-bit. Thuật toán trải qua 64 vòng biến đổi, sử dụng các phép toán nhị phân (như phép xoay vòng bit, phép dịch chuyển và phép XOR) cùng với các hằng số được xác định trước. Kết quả cuối cùng sau tất cả các vòng là một giá trị băm 256 bit. Tính chất “một chiều” và “khuếch đại thay đổi” (avalanche effect) của SHA-256 đảm bảo rằng không thể tìm được hai dữ liệu khác nhau cho cùng một giá trị băm (tính chống va chạm), và thay đổi nhỏ ở đầu vào tạo ra giá trị băm hoàn toàn khác biệt.

**Ứng dụng trong blockchain:** SHA-256 được sử dụng rộng rãi trong Bitcoin và nhiều cryptocurrency khác để đảm bảo tính toàn vẹn và cơ chế đồng thuận. Cụ thể, Bitcoin áp dụng thuật toán SHA-256 kép (băm hai lần liên tiếp) cho bằng chứng công việc (Proof of Work): các “thợ đào” phải tìm một nonce sao cho giá trị SHA-256 của khối dữ liệu (bao gồm nonce) bắt đầu bằng một số lượng chữ số 0 nhất định (theo độ khó mạng quy định). Điều này đòi hỏi thử nhiều giá trị nonce khác nhau để tìm được giá trị băm thỏa mãn độ khó, đảm bảo việc thêm block mới tốn công sức tính toán. Ngoài ra, mỗi khối Bitcoin cũng chứa giá trị băm SHA-256 của khối trước đó (trong header), tạo nên chuỗi liên kết chặt chẽ – nếu ai đó muốn thay đổi dữ liệu của một khối, họ sẽ phải tính lại toàn bộ các băm khối sau nó, điều gần như không thể với mạng lưới lớn. Bên cạnh đó, Merkle Tree (cây Merkle) trong Bitcoin cũng dùng SHA-256 để tính các “merkle root” đại diện cho toàn bộ giao dịch trong khối.

**Ưu điểm:** SHA-256 có độ an toàn cao – chưa có trường hợp va chạm hoặc tiền ảnh (preimage) nào thực tế được tìm ra. Hàm có tính quyết định và nhanh chóng kiểm chứng: việc tính toán SHA-256 khá hiệu quả ngay cả trên phần cứng chuyên dụng (ASIC), và bất kỳ nút mạng nào cũng dễ dàng xác nhận được giá trị băm. SHA-256 cũng đã được kiểm chứng qua thời gian và được nhiều chuẩn an ninh sử dụng.

**Nhược điểm:** Nhược điểm chính của SHA-256 là tiêu tốn tài nguyên tính toán khi độ khó yêu cầu số lượng lớn phép thử (ví dụ trong khai thác Bitcoin). So với một số thuật toán băm hiện đại, SHA-256 có tốc độ băm chậm hơn (một phần do thiết kế an toàn hơn). Ngoài ra, về lý thuyết sự xuất hiện của máy tính lượng tử trong tương lai có thể giảm mức độ an toàn của SHA-256 (thuật toán Grover có thể tăng tốc độ tìm tiền ảnh, khiến bảo mật 256 bit bị giảm xuống mức 128 bit). Tuy nhiên, hiện tại điều này chưa phải mối đe dọa cấp bách. Một điểm nữa là kích thước đầu ra 256 bit của SHA-256 lớn hơn so với một số hàm băm khác (như RIPEMD-160 160 bit), dẫn đến tốn không gian lưu trữ hơn đôi chút – mặc dù điều này thường không quan trọng bằng độ an toàn.

1. **Keccak-256 (SHA-3)**

**Định nghĩa:** Keccak-256 là thuật toán chiến thắng trong cuộc thi SHA-3 do NIST tổ chức và đã trở thành nền tảng cho chuẩn SHA-3. Về thực chất, Keccak-256 tạo ra giá trị băm 256 bit tương tự mục tiêu của SHA-256, nhưng dựa trên cấu trúc hàm bọt biển (sponge construction) thay vì Merkle–Damgård. Ethereum và nhiều ứng dụng blockchain khác sử dụng Keccak-256 làm hàm băm mặc định (Ethereum thường gọi keccak256 như là “SHA-3” của họ).

**Cơ chế hoạt động:** Không giống như SHA-256 hoạt động theo từng khối cố định, Keccak áp dụng cấu trúc bọt biển với trạng thái bên trong lớn (1600 bit) và hai pha “hấp thụ và vắt kiệt” (absorb và squeeze). Dữ liệu đầu vào được hấp thụ dần vào trạng thái bên trong qua các phép XOR, sau đó trạng thái được khuấy đảo bởi hàm chuyển vị (permutation) Keccak-f. Khi toàn bộ đầu vào đã được hấp thụ, hàm bọt biển sẽ “vắt” ra đầu ra 256 bit từ trạng thái. Kiến trúc này cho phép Keccak có tính linh hoạt cao – có thể tạo ra độ dài đầu ra tùy ý bằng cách vắt kiệt thêm – và tăng khả năng chống lại các dạng tấn công phân tích mật mã đã biết.

**Ứng dụng trong blockchain:** Ethereum sử dụng Keccak-256 rộng rãi ở nhiều chỗ trong hệ thống. Ví dụ, địa chỉ ví Ethereum được tạo ra bằng cách lấy 20 byte cuối của giá trị Keccak-256 của khóa công khai của người dùng. Hàm băm Keccak-256 cũng được sử dụng khi tính hash của block, tạo Merkle tree cho các giao dịch và nhật ký sự kiện, tính toán băm giao dịch và trong nhiều cơ chế khác của Ethereum. Solidity – ngôn ngữ lập trình smart contract – cung cấp hàm keccak256() như một hàm băm tích hợp. Ngoài Ethereum, một số blockchain khác cũng bắt đầu áp dụng các biến thể của SHA-3 (Keccak) vì tính an toàn và hiệu năng của nó. Điểm đáng chú ý: Ethereum sử dụng Keccak-256 không hoàn toàn giống với chuẩn SHA-3 của NIST (có một chút khác biệt ở phần padding), nhưng nhìn chung chức năng tương đương và thường vẫn được gọi là SHA-3.

**Ưu điểm:** Keccak-256/SHA-3 được thiết kế muộn hơn nên khắc phục được nhiều điểm yếu tiềm tàng của các hàm băm thế hệ trước. Thuật toán này có cấu trúc khác biệt giúp chống lại các phương pháp tấn công đã làm suy yếu MD5, SHA-1 (và có thể trong tương lai là SHA-2). Tính đến nay, chưa có lỗ hổng thực tế nào được công bố trên Keccak. Thuật toán cũng linh hoạt với cấu trúc bọt biển, có thể thay đổi độ dài đầu ra dễ dàng, hữu ích trong nhiều giao thức. Keccak-256 có tốc độ xử lý tốt trên phần mềm, và với phần cứng tối ưu, nó có thể đạt hiệu năng cao tương đương các hàm băm khác. Ethereum chọn Keccak một phần vì tính mới và an toàn dài hạn, tránh phụ thuộc quá nhiều vào SHA-256. Một ưu điểm khác là Keccak-256 không yêu cầu tham số khởi tạo bí mật – toàn bộ thiết kế minh bạch, giúp tăng niềm tin không có “cửa hậu”.

**Nhược điểm:** Do ra đời sau, Keccak-256 chưa được hỗ trợ rộng rãi bằng SHA-256 trong một số phần cứng truyền thống (ví dụ các bộ tăng tốc ASIC ban đầu tập trung cho SHA-256 của Bitcoin). Việc chuyển dịch sang SHA-3 trong một số hệ thống đòi hỏi thời gian và nỗ lực. Kích thước đầu ra 256 bit vẫn như SHA-256 nên không tiết kiệm không gian hơn. Trong Ethereum, tuy Keccak-256 an toàn, nhưng cơ chế Ethash (bằng chứng công việc của Ethereum trước đây) lại kết hợp thêm yếu tố bộ nhớ, do đó tốc độ băm thuần của Keccak không phải giới hạn duy nhất. Tóm lại, nhược điểm chủ yếu của Keccak có lẽ chỉ là nó tương đối mới, cần thời gian kiểm chứng lâu dài như SHA-2 đã có, dù hiện tại cộng đồng đánh giá rất cao độ an toàn của nó.

* 1. Chữ ký số (Digital Signatures)

**Định nghĩa:** Chữ ký số là một cơ chế mật mã khóa công khai cho phép một người (người ký) dùng khóa riêng của mình ký vào dữ liệu (ví dụ một giao dịch), tạo ra một chuỗi bit đặc trưng gọi là chữ ký. Người khác có thể dùng khóa công khai tương ứng để xác minh chữ ký – nếu chữ ký hợp lệ nghĩa là dữ liệu chưa bị thay đổi và thực sự do người có khóa riêng (chủ sở hữu hợp pháp) ký file. Chữ ký số đảm bảo hai thuộc tính quan trọng: toàn vẹn (dữ liệu không bị chỉnh sửa sau khi ký) và xác thực nguồn gốc (chỉ người có khóa bí mật mới tạo được chữ ký hợp lệ). Trong blockchain, chữ ký số được sử dụng để ủy quyền giao dịch – ví dụ, chỉ khi bạn ký một giao dịch bằng khóa riêng của ví bạn, mạng lưới mới chấp nhận giao dịch đó là hợp lệ. Hai thuật toán chữ ký số phổ biến trong blockchain là ECDSA và BLS.

1. **ECDSA (Elliptic Curve Digital Signature Algorithm)**

**Định nghĩa:** ECDSA là biến thể của thuật toán chữ ký số DSA, nhưng sử dụng mật mã đường cong elliptic (ECC) thay vì toán học trên trường số nguyên modulo như DSA cổ điển. Cụ thể, ECDSA sử dụng đường cong elliptic trên trường hữu hạn (Bitcoin và Ethereum dùng đường cong secp256k1). Đây là thuật toán chữ ký khóa công khai/khóa riêng, trong đó khóa riêng là một số nguyên bất kỳ, còn khóa công khai là một điểm trên đường cong elliptic được tính bằng khóa riêng nhân với điểm sinh $G$ của đường cong (tương tự như G^x trong toán học nhóm). ECDSA cung cấp mức bảo mật cao với độ dài khóa nhỏ – ví dụ khóa công khai 256 bit ECC có độ an toàn tương đương khóa RSA cỡ 3072 bit.

**Cơ chế ký và xác minh:** Quá trình tạo chữ ký số ECDSA bắt đầu từ việc người ký sử dụng khóa riêng d để tính khóa công khai *Q = d \* G,* trong đó G là điểm sinh đã được thỏa thuận trên đường cong elliptic, và Q là điểm thuộc đường cong đó.

Khi muốn ký một thông điệp m (chẳng hạn như dữ liệu giao dịch), người ký trước tiên sẽ băm thông điệp đó bằng một hàm băm mật mã học – thường dùng SHA-256 – để thu được một giá trị băm *e*. Sau đó, người ký chọn một số ngẫu nhiên k. Việc chọn k là cực kỳ quan trọng, vì nếu k bị lộ hoặc bị trùng lặp giữa các lần ký, khóa riêng d có thể bị suy ra.

Tiếp theo, người ký tính điểm *P = k \* G* trên đường cong, lấy hoành độ x của điểm P và tính *r = x mod n*, trong đó n là số nguyên lớn (bậc của nhóm sinh bởi G). Sau đó, người ký tính tiếp *s = k⁻¹ \* (e + d \* r) mod n*. Kết quả là một cặp *(r, s),* chính là chữ ký số ECDSA tương ứng với thông điệp m.

Quá trình xác minh chữ ký diễn ra như sau: người nhận sử dụng khóa công khai Q của người ký, đồng thời tính lại e = hash(m) từ thông điệp m gốc. Từ đó, người nhận tính hai giá trị trung gian:

*u₁ = e \* s⁻¹ mod n*

*u₂ = r \* s⁻¹ mod n*

Sau đó, tính điểm *X = u₁ \* G + u₂ \* Q,* tức là cộng hai điểm elliptic trên đường cong. Nếu hoành độ x của điểm X, khi lấy mod n, bằng với r trong chữ ký, thì chữ ký được xem là hợp lệ.

Nói cách khác, việc xác minh kiểm tra một phương trình trên đường cong elliptic, nhằm xác nhận rằng khóa công khai Q, chữ ký (*r, s),* và thông điệp đã băm e có mối liên hệ chính xác với nhau. Nếu chữ ký bị giả mạo hoặc dữ liệu thông điệp m đã bị thay đổi, phương trình sẽ không thỏa mãn, và chữ ký sẽ bị phát hiện là không hợp lệ.

**Ứng dụng trong blockchain:** ECDSA (với đường cong secp256k1) là thuật toán chữ ký chính trong Bitcoin và Ethereum hiện nay. Mỗi giao dịch Bitcoin đều chứa chữ ký ECDSA do chủ sở hữu UTXO ký để chứng minh quyền sở hữu đầu ra (coin) đó và cho phép chi tiêu. Các thợ đào và node sẽ xác minh chữ ký ECDSA để đảm bảo giao dịch hợp lệ trước khi cho vào block. Tương tự, Ethereum dùng ECDSA để ký các giao dịch chuyển ETH hoặc thực thi smart contract – chữ ký đảm bảo chỉ chủ tài khoản (người nắm khóa riêng) mới có thể khởi tạo giao dịch từ địa chỉ của mình. Khóa công khai ECDSA trong Ethereum còn được dùng để tính địa chỉ ví (bằng cách lấy băm Keccak của khóa công khai rồi lấy 20 byte cuối). Ngoài ra, ECDSA/ECC cũng được sử dụng trong nhiều cơ chế khác như xác thực danh tính, trao đổi khóa trong các giao thức lớp dưới (TLS, v.v.).

**Ưu điểm:** ECDSA mang lại bảo mật cao với khóa ngắn. So với RSA, thuật toán ECC/ECDSA có hiệu suất tốt hơn cho cùng mức an toàn – một khóa 256-bit ECC đủ an toàn so với RSA 3072-bit. Điều này giúp giảm kích thước chữ ký và khóa công khai lưu trữ trên blockchain, tiết kiệm không gian khối và băng thông truyền tải. Quá trình xác minh ECDSA cũng khá nhanh, phù hợp để mỗi node mạng blockchain có thể xác thực hàng nghìn chữ ký mỗi giây. Thuật toán ECDSA đã được sử dụng hơn một thập kỷ trong Bitcoin/Ethereum và tỏ ra bền vững, chưa có lỗ hổng lý thuyết. Một ưu điểm nữa: nếu triển khai đúng, ECDSA cho phép tạo chữ ký định danh – tức cùng một thông điệp và khóa riêng sẽ luôn tạo ra một chữ ký duy nhất (theo RFC 6979) mà không cần phụ thuộc vào bộ sinh số ngẫu nhiên, giúp tránh rủi ro về sinh khóa kém.

**Nhược điểm:** ECDSA phức tạp hơn RSA trong triển khai, dễ mắc lỗi nếu không cẩn thận. Đặc biệt, yêu cầu bảo mật số ngẫu nhiên k mỗi lần ký. Nếu kbị lặp lại hoặc đoán được, khóa bí mật $d$ có thể bị lộ (đã có trường hợp máy chơi game Sony PlayStation bị hack do tái sử dụng nonce trong ECDSA). Kích thước chữ ký ECDSA (~64 byte) lớn hơn một số hệ thống chữ ký mới (như EdDSA có ~ 64 byte nhưng ưu việt hơn, hoặc BLS khi ký nhiều người có thể gộp lại chỉ còn 1 chữ ký). ECDSA cũng không hỗ trợ tự nhiên việc gộp chữ ký: nếu nhiều người cùng ký các thông điệp khác nhau, sẽ phải lưu nhiều chữ ký riêng rẽ trên chuỗi (có kỹ thuật Schnorr hay MuSig có thể gộp nhưng phức tạp và chưa được dùng rộng rãi trên chuỗi khối chính thống). Thêm vào đó, ECC nói chung sẽ bị đe dọa bởi máy tính lượng tử (Shor’s algorithm có thể giải bài toán logarit rời rạc), tức nếu máy tính lượng tử đủ mạnh ra đời, ECDSA sẽ không an toàn – mặc dù điều này cũng đúng cho RSA và hầu hết mật mã khóa công khai hiện tại.

1. **BLS (Boneh–Lynn–Shacham Signature)**

**Định nghĩa:** BLS là một thuật toán chữ ký số dựa trên đường cong elliptic và hàm song ánh (pairing), được Boneh, Lynn và Shacham giới thiệu vào năm 2004. Điểm độc đáo của BLS nằm ở khả năng gộp (aggregate) nhiều chữ ký thành một chữ ký duy nhất một cách tự nhiên và hiệu quả. BLS được triển khai trên các đường cong elliptic đặc biệt gọi là pairing-friendly curves, điển hình như đường cong BLS12-381 hiện đang được sử dụng trong Ethereum 2.0.

Trong BLS, khóa riêng là một số nguyên x. Khóa công khai tương ứng là điểm *PK = x \* G*, trong đó G là một điểm thuộc nhóm G₂ trên đường cong elliptic. Khi muốn ký một thông điệp m, đầu tiên thông điệp đó được băm thành một điểm trên đường cong: *H(m)* thuộc nhóm G₁. Sau đó, chữ ký σ được tạo bằng cách nhân điểm *H(m)* với khóa riêng x*: σ = H(m) ^ x*. Kết quả σ là một điểm trên đường cong trong nhóm G₁. Để xác minh chữ ký, người nhận kiểm tra đẳng thức: pairing giữa σ và G có bằng với pairing giữa *H(m*) và *PK* hay không, cụ thể là:

*e(σ, G) = e(H(m), PK).*

Trong đó, e là một hàm song ánh được định nghĩa trên cặp nhóm (G₁, G₂). Nếu đẳng thức đúng, chữ ký được xem là hợp lệ. Một đặc điểm quan trọng của BLS là chữ ký có tính duy nhất và xác định – với mỗi thông điệp và một khóa riêng cho trước, chỉ tồn tại một chữ ký hợp lệ duy nhất.

Một trong những tính năng nổi bật nhất của BLS là khả năng gộp chữ ký. Giả sử có nhiều người dùng khác nhau, mỗi người sở hữu một khóa công khai riêng và muốn ký các thông điệp m₁, m₂, ..., mₙ tương ứng (hoặc cùng một thông điệp trong một số ứng dụng). Mỗi người tạo một chữ ký riêng lẻ *σᵢ = H(mᵢ) ^ xᵢ.* Khi đó, ta có thể gộp tất cả các chữ ký này lại bằng phép nhân trong nhóm G₁ để tạo thành một chữ ký tổng hợp:

*σₐgg = σ₁ × σ₂ × ... × σₙ.*

Điều đáng chú ý là chữ ký tổng hợp này vẫn có thể được xác minh chỉ bằng một phép kiểm duy nhất. Người xác minh chỉ cần kiểm tra đẳng thức:

*e(σₐgg, G) = e(H(m₁), PK₁) × e(H(m₂), PK₂) × ... × e(H(mₙ), PKₙ).*

Nhờ tính chất song ánh và song tuyến tính của hàm pairing, đẳng thức này đúng nếu và chỉ nếu từng chữ ký σᵢ đều hợp lệ với khóa công khai PKᵢ tương ứng.

Tính năng này giúp giảm đáng kể chi phí xác minh trong các hệ thống có số lượng chữ ký lớn, chẳng hạn như các mạng blockchain có hàng nghìn validator cùng ký vào một block. Thay vì phải xử lý và xác minh từng chữ ký riêng lẻ, BLS cho phép rút gọn tất cả xuống còn một chữ ký và một lần xác minh duy nhất, giúp tối ưu hóa hiệu suất hệ thống một cách vượt trội.

**Ứng dụng trong blockchain:** BLS đang được triển khai trong Ethereum 2.0 (Ethereum Beacon Chain) cho việc xác nhận khối và thông báo trạng thái của validator. Cụ thể, mỗi khối trong cơ chế đồng thuận PoS của Ethereum 2.0 phải có đủ chữ ký của một tỷ lệ lớn các validator (người xác thực). Thay vì đưa hàng trăm chữ ký ECDSA lên chuỗi, Ethereum 2 dùng BLS để các validator ký và sau đó gộp tất cả chữ ký lại thành một chữ ký BLS duy nhất trên mỗi khối – giúp giảm dữ liệu và tăng tốc độ xác minh. Ngoài Ethereum 2.0, Chia Network (một blockchain dùng thuật toán bằng chứng không gian) cũng sử dụng hoàn toàn BLS cho chữ ký giao dịch để cho phép gộp nhiều chữ ký một cách hiệu quả. Nhiều đề xuất cải tiến Bitcoin hay các hệ thống multi-sig cũng cân nhắc BLS như một giải pháp thay thế Schnorr/ECDSA trong tương lai, do khả năng gộp tự nhiên của nó.

**Ưu điểm:** Ưu điểm lớn nhất của BLS là hỗ trợ gộp chữ ký cực kỳ hiệu quả. Dù có 1 hay 1000 chữ ký BLS được gộp, kích thước chữ ký hợp nhất vẫn như một chữ ký (thường 48 byte nếu dùng BLS12-381). Điều này giúp giảm băng thông mạng và dung lượng lưu trữ khối khi có nhiều bên ký. Kèm với đó, việc xác minh hàng loạt trở nên đơn giản – một phép kiểm pairing cho chữ ký gộp thay vì hàng nghìn phép kiểm riêng lẻ, giúp tăng thông lượng của blockchain (Ethereum 2 nhờ BLS có thể hỗ trợ hàng nghìn validator cùng tham gia mỗi block một cách khả thi). BLS còn có tính chất deterministic – không cần số ngẫu nhiên khi ký, nên không lo vấn đề rò rỉ khóa do random xấu (một vấn đề của ECDSA nếu k sai). Mặt khác, BLS dựa trên bài toán logarit rời rạc tương tự ECDSA nên mức an toàn tương đương với cùng độ dài khóa, và cũng có nhược điểm lượng tử tương tự (không kháng lượng tử). Tuy nhiên, vì BLS dùng đường cong đặc biệt, hiện tại cũng có quan điểm cho rằng BLS có thể dễ dàng nâng cấp sang các biến thể hậu lượng tử khi cần (hoặc kết hợp với các sơ đồ kháng lượng tử trong quá trình gộp chữ ký).

**Nhược điểm:** Nhược điểm chính của BLS là hiệu suất tính toán: phép toán pairing (song ánh) dùng trong xác minh BLS tốn nhiều phép tính hơn so với phép nhân điểm như trong ECDSA, do đó xác minh một chữ ký BLS đơn lẻ thường chậm hơn xác minh ECDSA. Tuy nhiên, khi số lượng chữ ký lớn, BLS lại vượt trội nhờ khả năng gộp (1000 chữ ký BLS có thể xác minh bằng 1 phép pairing, trong khi 1000 chữ ký ECDSA phải 1000 lần kiểm). Một nhược điểm khác là vấn đề “khóa giả mạo” (rogue key) trong gộp chữ ký: nếu không có cơ chế đảm bảo, kẻ xấu có thể chọn khóa công khai đặc biệt sao cho chữ ký của mình triệt tiêu chữ ký người khác khi gộp. Biện pháp khắc phục là yêu cầu người tham gia chứng minh sở hữu khóa công khai của họ (thông qua proof of possession trước khi gộp). Các hệ thống thực tế như Ethereum 2 đã giải quyết điều này bằng cách yêu cầu bằng chứng kèm theo khi validator đăng ký khóa. Cuối cùng, BLS là công nghệ tương đối mới (so với ECDSA), việc triển khai cần cẩn thận để tránh lỗi. Và cũng như ECDSA, BLS sẽ mất an toàn trước máy tính lượng tử. Tóm lại, BLS đánh đổi một chút phức tạp và chi phí tính toán để lấy lợi ích vượt trội về gộp chữ ký và giảm tải mạng lưới.

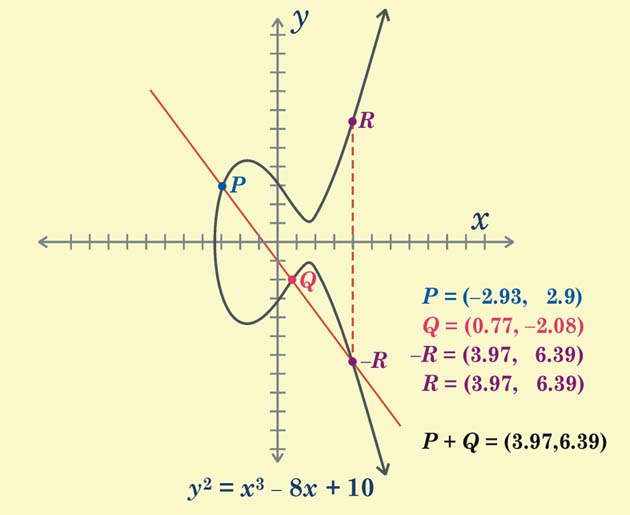
* 1. Hệ mật mã khóa công khai: Đường cong Elliptic (ECC)

**Định nghĩa:** Mật mã khóa công khai dựa trên đường cong elliptic (Elliptic Curve Cryptography - ECC) là nền tảng cho nhiều thuật toán quan trọng như ECDSA, ECDH và BLS. Đường cong elliptic trong ngữ cảnh này được mô tả bởi phương trình tổng quát có dạng: y bình phương bằng x mũ ba cộng ax cộng b, ký hiệu là: y² = x³ + ax + b, được xét trên một trường hữu hạn.

Tập hợp các điểm thỏa mãn phương trình này, cùng với điểm tại vô cực, tạo thành một nhóm Abel với phép toán cộng điểm được định nghĩa theo một quy tắc hình học đặc biệt. Cụ thể, để cộng hai điểm P và Q trên đường cong, ta kẻ một đường thẳng đi qua P và Q. Đường thẳng này sẽ cắt đường cong tại điểm thứ ba, gọi là -R. Khi phản xạ điểm -R qua trục hoành (đổi dấu tung độ), ta thu được điểm R chính là kết quả của phép cộng: *P + Q = R*. Nếu P và Q trùng nhau, đường thẳng trở thành tiếp tuyến tại điểm P, khi đó phép cộng là phép nhân đôi điểm.

Phép cộng này thỏa mãn các tính chất của nhóm: có phần tử đơn vị, phần tử nghịch đảo, tính kết hợp và tính giao hoán. Dựa vào phép cộng, ta định nghĩa phép nhân điểm như là phép cộng lặp lại nhiều lần: k nhân P (k là số nguyên) nghĩa là cộng P với chính nó k lần. Việc nhân điểm với số nguyên rất lớn có thể được thực hiện nhanh chóng bằng các thuật toán hiệu quả.

Tuy nhiên, bài toán ngược lại tức là tìm k khi biết P và *Q = kP* là cực kỳ khó nếu k đủ lớn. Đây chính là bài toán logarit rời rạc trên đường cong elliptic, nền tảng cho tính bảo mật của ECC. Cho tới nay, chưa có thuật toán hiệu quả nào để giải bài toán này trong thời gian khả thi, tương tự như việc phân tích một số nguyên lớn ra thừa số trong hệ mật mã RSA.Về mặt trực quan, có thể minh họa phép cộng điểm trên đồ thị của đường cong elliptic như sau: điểm P (màu xanh) và Q (màu đỏ) được nối bằng một đường thẳng (màu tím). Đường thẳng này cắt đường cong tại điểm thứ ba (ký hiệu -R, nằm phía dưới, màu hồng). Phản xạ -R qua trục hoành sẽ cho điểm R (màu tím, có hoành độ bằng -R nhưng tung độ đối dấu), chính là kết quả của phép cộng *P + Q*. Chính nhờ phép toán nhóm đặc biệt này, ECC cho phép thực hiện các thao tác mã hóa, ký và trao đổi khóa với hiệu suất cao và độ bảo mật mạnh.



**Khóa riêng & khóa công khai:** Trong hệ mật mã đường cong elliptic, khóa riêng thường là một số nguyên ngẫu nhiên d (thường có độ dài khoảng 256 bit đối với các đường cong phổ biến hiện nay). Khóa công khai tương ứng với khóa riêng này là một điểm *Q = d × G* trên đường cong elliptic, trong đó G là điểm sinh đã được thỏa thuận trước giữa các bên. Nhờ tính chất khó giải của bài toán logarit rời rạc trên đường cong elliptic, việc tính Q từ d là dễ dàng, nhưng ngược lại, việc tìm d từ Q là bất khả thi trong thời gian hữu hạn nếu thông số đủ lớn. Do đó, khóa công khai Q có thể được công khai cho mọi người, trong khi khóa riêng d phải được giữ bí mật tuyệt đối. Cơ chế này là nền tảng cho nhiều ứng dụng quan trọng của mật mã elliptic. Trong chữ ký số, điển hình là ECDSA, khóa riêng d được dùng để ký thông điệp, còn khóa công khai Q dùng để xác minh chữ ký. Trong trao đổi khóa ECDH, hai bên thực hiện trao đổi khóa công khai của nhau và nhân với khóa riêng của mình để cùng tính ra một khóa chung bí mật mà không cần truyền trực tiếp khóa đó qua mạng. Ngoài ra, mặc dù ít phổ biến trong lĩnh vực blockchain, thuật toán mã hóa sử dụng elliptic curve (ECC encryption), chẳng hạn như EC-ElGamal, vẫn tồn tại và được ứng dụng trong một số hệ thống cần bảo mật dữ liệu.

**Ứng dụng trong blockchain:** Hầu hết các blockchain tiền mã hóa hiện nay đều sử dụng ECC làm hệ mật mã cơ sở. Bitcoin từ những ngày đầu đã chọn đường cong secp256k1 cho việc sinh khóa và ECDSA chữ ký. Mỗi địa chỉ Bitcoin thực ra được tạo từ khóa công khai ECC (qua một bước băm RIPEMD-160/SHA-256 để rút gọn). Ethereum cũng sử dụng cùng đường cong secp256k1 cho ECDSA chữ ký giao dịch. Các coin ẩn danh như Monero sử dụng đường cong Ed25519 (một dạng ECC khác) cho chữ ký vòng. Việc quản lý ví và tài khoản trong blockchain thực chất là quản lý cặp khóa ECC: ai nắm được khóa riêng thì có toàn quyền tiêu xài số coin tương ứng trong địa chỉ khóa công khai. ECC còn ứng dụng trong các hệ thống đa chữ ký (multi-sig) và ví thông minh, nơi cần kết hợp nhiều khóa công khai/riêng. Một lợi ích khác của ECC trong blockchain là nhờ khóa ngắn, các giao dịch và block tiết kiệm được nhiều không gian – đây là lý do mà hầu như không có blockchain hiện đại nào sử dụng RSA (khóa dài hàng ngàn bit) vì sẽ quá cồng kềnh.

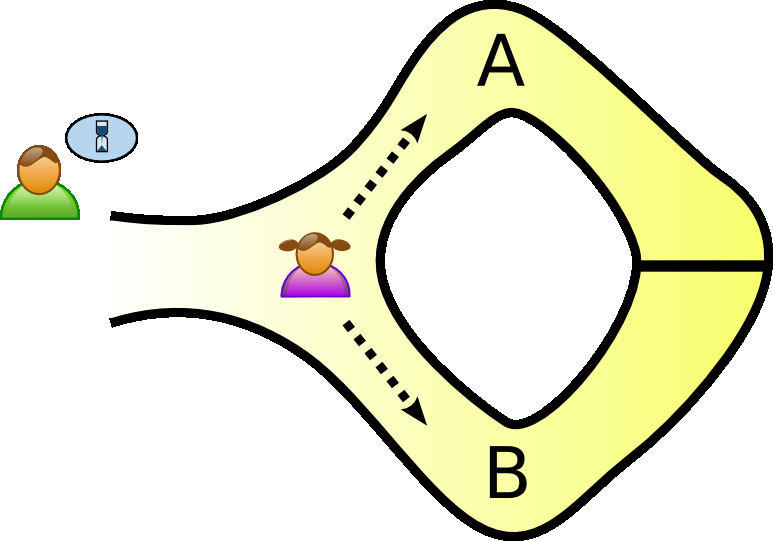
**Ưu điểm:** Như đã đề cập, lợi thế số một của mật mã elliptic so với các hệ mật mã khác (như RSA) là độ an toàn cao với độ dài khóa nhỏ. Điều này giúp cho việc truyền tải và lưu trữ khóa công khai/ chữ ký được gọn nhẹ, tiết kiệm chi phí – rất quan trọng trong môi trường blockchain giới hạn kích thước block và băng thông mạng. Các phép toán elliptic (cộng/ nhân điểm) cũng có thể được tăng tốc bởi phần cứng chuyên dụng, giúp hệ thống xử lý nhiều giao dịch hơn. ECC đã được nghiên cứu kỹ và sử dụng trong nhiều chuẩn an ninh (TLS, chính phủ, v.v.), nên độ tin cậy cao. Một ưu điểm nữa là ECC cho phép triển khai nhiều tính năng mật mã nâng cao như: chữ ký số an danh (ẩn danh người ký), chữ ký ngưỡng (n-of-m người ký), chia sẻ bí mật (Shamir’s secret sharing trên elliptic), v.v. – tạo nên các ứng dụng tài chính, ví đa chữ ký an toàn hơn trong blockchain. Tóm lại, ECC mang đến sự cân bằng tối ưu giữa bảo mật và hiệu quả, rất phù hợp với yêu cầu của sổ cái phân tán.

**Nhược điểm:** Mặt hạn chế của ECC chủ yếu nằm ở độ phức tạp toán học. Việc cài đặt ECC đòi hỏi lập trình viên hiểu rõ về số học modulo, đại số nhóm – nếu không, dễ xảy ra lỗi (ví dụ các lỗi về kiểm tra điểm trên đường cong, rò rỉ thông tin qua kênh bên...). Đã từng có những lỗ hổng do dùng tham số đường cong không an toàn (cố tình cài cửa hậu) hoặc do bộ sinh số giả mạo (như thuật toán Dual\_EC\_DRBG tai tiếng của NSA). Tuy nhiên, những vấn đề này có thể tránh được bằng cách dùng tham số tiêu chuẩn và thư viện ECC đã được kiểm định. Một nhược điểm khác là ECC cũng không miễn nhiễm với lượng tử: NSA thậm chí còn tuyên bố sẽ chuyển sang hệ hậu lượng tử vì lo ngại tấn công lượng tử lên ECC. Nếu máy tính lượng tử đủ mạnh ra đời, bài toán logarit rời rạc (nền tảng ECC) sẽ bị phá, tương tự RSA. Ngoài ra, do khóa ECC ngắn và “khó đoán hơn” RSA nên trong một số hệ thống cũ, việc quản lý khóa ECC có thể chưa đồng bộ (nhưng đây không phải vấn đề của blockchain, nơi ECC được thiết kế ngay từ đầu). Tóm lại, nhược điểm ECC chủ yếu là nguy cơ dài hạn (lượng tử) và yêu cầu cao về mặt triển khai an toàn.

* 1. Bằng chứng không tiết lộ (Zero-knowledge Proof – ZKP)

**Định nghĩa:** Bằng chứng không tiết lộ là một giao thức mật mã cho phép một bên (Người chứng minh) chứng tỏ cho bên khác (Người xác minh) rằng một tuyên bố nào đó là đúng, mà không cần tiết lộ bất kỳ thông tin chi tiết nào ngoài việc “nó đúng”. Nói cách khác, người chứng minh có kiến thức bí mật để xác nhận một sự thật, nhưng thay vì đưa ra kiến thức đó, họ chỉ cung cấp một bằng chứng mật mã cho thấy mình biết điều đó. Người xác minh sau khi kiểm tra bằng chứng sẽ tin tưởng vào tính đúng đắn của tuyên bố mà không học được gì thêm về bí mật.

Minh họa nguyên lý ZKP (câu chuyện hang Alibaba kinh điển): Giả sử Peggy (nhân vật áo tím) biết mật khẩu mở cửa ngăn giữa hai lối A và B trong hang động. Cô muốn chứng minh cho Victor (áo xanh) rằng mình biết mật khẩu, nhưng không muốn tiết lộ mật khẩu đó. Peggy thực hiện bằng chứng như sau: cô đi vào hang theo một lối bất kỳ (Victor không biết cô chọn A hay B). Sau đó Victor đứng ngoài gọi to yêu cầu Peggy đi ra bằng lối A. Nếu Peggy thực sự biết mật khẩu, cô sẽ mở được cửa bí mật và đi vòng qua để ra đúng lối A như yêu cầu. Victor thấy Peggy xuất hiện ở lối A thì tin rằng Peggy biết mật khẩu (vì nếu không biết, Peggy chỉ có xác suất 50% đoán đúng lối). Bằng cách lặp lại thử thách nhiều lần, xác suất Peggy gian lận sẽ rất nhỏ. Tuy nhiên, trong suốt quá trình, Peggy không hề tiết lộ mật khẩu cho Victor – đây chính là tinh thần của bằng chứng không tiết lộ: chứng minh mình biết bí mật mà không lộ bí mật.



Có hai loại ZKP: tương tác (interactive) và không tương tác (non-interactive). Ví dụ trên là dạng tương tác – người xác minh liên tục thử thách người chứng minh nhiều lần. Trong blockchain, để tiện dụng, người ta thường dùng dạng không tương tác (NIZK), nơi bằng chứng được tạo ra như một thông điệp độc lập mà bất kỳ ai cũng có thể kiểm mà không cần trao đổi qua lại nhiều bước. Những kỹ thuật ZKP không tương tác hiện đại thường được gọi là zk-SNARK, zk-STARK, Bulletproofs, v.v., mỗi loại có ưu nhược điểm khác nhau. Trong phạm vi câu hỏi, ta sẽ tập trung vào zk-SNARK và zk-STARK, hai kỹ thuật ZKP tiêu biểu trong blockchain ngày nay.

* 1. Vai trò của mật mã học trong Blockchain

Mật mã học giữ vai trò cốt lõi trong việc đảm bảo tính bảo mật và tin cậy của công nghệ Blockchain. Các kỹ thuật mật mã như hàm băm, chữ ký số, mật mã khóa công khai và bằng chứng không tiết lộ (zero-knowledge proofs) được ứng dụng để hiện thực hóa những thuộc tính quan trọng của blockchain, bao gồm: tính toàn vẹn dữ liệu, xác thực người dùng và giao dịch, tính bất biến và minh bạch của sổ cái, cũng như bảo vệ quyền riêng tư trong các hệ thống nâng cao.

* Hàm băm mật mã đảm bảo toàn vẹn dữ liệu và tính bất biến

Hàm băm mật mã (cryptographic hash function) tạo ra một “dấu vân tay” duy nhất cho dữ liệu, sao cho chỉ cần một thay đổi rất nhỏ trong dữ liệu đầu vào cũng sẽ làm giá trị băm đầu ra thay đổi hoàn toàn. Trong blockchain, mỗi khối (block) đều chứa mã băm của khối trước đó, tạo thành chuỗi liên kết mật mã chặt chẽ giữa các khối. Cấu trúc này đảm bảo rằng dữ liệu trong sổ cái không thể bị sửa đổi mà không bị phát hiện: nếu kẻ gian cố tình chỉnh sửa nội dung của một khối, mã băm của khối đó sẽ thay đổi, phá vỡ liên kết tới khối kế tiếp và ngay lập tức tiết lộ sự sai lệch. Nhờ cơ chế này, blockchain đạt được tính toàn vẹn cao và gần như bất biến – một khi khối mới đã được thêm vào chuỗi thì các khối cũ hơn trở nên cực kỳ khó bị chỉnh sửa hay xóa bỏ.

Ví dụ: Bitcoin sử dụng hàm băm SHA-256 để liên kết các khối và làm bằng chứng công việc (Proof-of-Work); điều này đảm bảo mọi thay đổi trái phép trong dữ liệu đều bị phát hiện và ngăn chặn, góp phần giữ cho sổ cái Bitcoin luôn đáng tin cậy và nhất quán

* Chữ ký số và mật mã khóa công khai đảm bảo xác thực giao dịch

Bên cạnh hàm băm, blockchain dựa vào mật mã khóa công khai (asymmetric cryptography) và chữ ký số để xác thực người dùng và giao dịch trên mạng một cách an toàn. Mỗi người dùng sở hữu một cặp khóa: khóa bí mật (private key) và khóa công khai (public key). Khóa bí mật được giữ kín, còn khóa công khai được công bố như địa chỉ nhận tiền của người dùng. Khi muốn thực hiện giao dịch, người dùng sẽ dùng khóa bí mật của mình để ký một chữ ký số lên dữ liệu giao dịch đó. Chữ ký số này đóng vai trò như bằng chứng mật mã xác nhận giao dịch thực sự được khởi tạo bởi đúng chủ sở hữu (người nắm khóa bí mật) và nội dung giao dịch không bị thay đổi kể từ lúc ký. Các nút trong mạng lưới sau đó dùng khóa công khai (địa chỉ) của người gửi để kiểm chứng chữ ký, qua đó độc lập xác thực tính hợp lệ của giao dịch mà không cần dựa vào bất kỳ bên trung gian nào. Nhờ chữ ký số, blockchain đảm bảo chỉ những người có quyền (nắm giữ khóa bí mật hợp lệ) mới có thể gửi giao dịch, và bất kỳ sự thay đổi bất hợp pháp nào lên dữ liệu giao dịch sẽ bị phát hiện vì chữ ký sẽ không còn khớp nếu dữ liệu bị chỉnh sửa.

Quan trọng hơn, chính cơ chế khóa công khai/chữ ký số mang lại danh tính số cho người dùng trên blockchain mà vẫn duy trì tính ẩn danh ở mức độ nhất định.

Ví dụ: trong Bitcoin và Ethereum, mỗi giao dịch đều được ký bằng khóa riêng của người gửi và mọi người có thể xác minh chữ ký đó bằng khóa công khai tương ứng. Địa chỉ ví (khóa công khai dạng rút gọn) thực chất được tạo ra từ khóa công khai đầy đủ của người dùng, và chỉ ai nắm khóa riêng tương ứng mới có thể chi tiêu tài sản ở địa chỉ đó. Nhờ vậy, hệ thống vừa nhận diện được danh tính mật mã của chủ sở hữu một cách duy nhất, vừa đảm bảo không ai có thể giả mạo giao dịch của người khác. Bởi tất cả giao dịch đều được ký và ghi vào sổ cái phân tán công khai, blockchain còn đạt được tính minh bạch – mọi người tham gia mạng đều có thể tự mình kiểm tra và xác thực toàn bộ lịch sử giao dịch, thay vì phải tin tưởng vào một cơ quan tập trung

* Bảo vệ quyền riêng tư với bằng chứng không tiết lộ (Zero-knowledge proof)

Mặc dù sổ cái blockchain công khai mang lại minh bạch cao, điều này cũng đồng nghĩa mọi thông tin giao dịch đều lộ rõ trên mạng, gây lo ngại về quyền riêng tư và bảo mật dữ liệu nhạy cảm. Để giải quyết mâu thuẫn giữa minh bạch và riêng tư, một số hệ thống blockchain nâng cao đã tích hợp các kỹ thuật mật mã tiên tiến, tiêu biểu là bằng chứng không tiết lộ (Zero-Knowledge Proof, viết tắt ZKP). Kỹ thuật ZKP cho phép một bên (người chứng minh) chứng tỏ cho bên khác (người kiểm chứng) rằng một mệnh đề nào đó là đúng (chẳng hạn “giao dịch này hợp lệ” hoặc “tôi có đủ số dư để thực hiện giao dịch”) mà không cần tiết lộ bất kỳ thông tin cụ thể nào về mệnh đề đóNói cách khác, ZKP giúp xác thực tính đúng đắn của giao dịch hoặc thông tin mà không phơi bày dữ liệu nhạy cảm ra công chúng.

Ví dụ: dựa trên giao thức Zerocash, đồng tiền mã hóa Zcash đã ứng dụng zk-SNARK (một dạng ZKP) cho phép người dùng gửi tiền mà không tiết lộ người gửi, người nhận hay số tiền trên blockchain – mọi thông tin chi tiết đều được ẩn, trong khi mạng lưới vẫn có thể xác minh giao dịch không vi phạm quy tắc (như tổng tiền không tự sinh ra thêm) thông qua bằng chứng mật mã. Tương tự, hệ sinh thái Ethereum gần đây cũng áp dụng ZKP trong một số giải pháp, chẳng hạn như zk-Rollups (gộp nhiều giao dịch off-chain và dùng ZKP để chứng minh tính hợp lệ khi đưa lên on-chain) và các hợp đồng thông minh tập trung vào quyền riêng tư. Những kỹ thuật này cho phép bảo vệ dữ liệu giao dịch nhạy cảm hoặc ẩn danh danh tính người dùng trên Ethereum, trong khi các nút mạng vẫn độc lập xác thực được tính hợp lệ của giao dịch nhờ bằng chứng không tiết lộ. Chính nhờ các giải pháp mật mã như ZKP, blockchain có thể vừa duy trì tính phi tập trung minh bạch vốn có, vừa bổ sung lớp bảo vệ quyền riêng tư cho người dùng một cách hiệu quả

Tóm lại, mật mã học là nền tảng đảm bảo cho blockchain các thuộc tính “tam bất” – bất tín nhiệm (không cần tin cậy bên thứ ba), bất biến (không thể sửa đổi lịch sử) và ẩn danh (pseudonymous) – đồng thời duy trì tính minh bạch và an toàn cho hệ thống. Thông qua việc kết hợp hàm băm để bảo vệ dữ liệu, chữ ký số và khóa công khai để xác thực giao dịch, cũng như các kỹ thuật tiên tiến như zero-knowledge proofs để bảo vệ riêng tư, blockchain đạt được một sổ cái phân tán nơi mọi giao dịch đều đáng tin cậy, minh bạch mà vẫn đảm bảo quyền riêng tư khi cần thiết. Các ví dụ từ Bitcoin, Ethereum cho thấy chính nhờ mật mã học, chúng ta có một hệ thống tài chính phi tập trung an toàn, nơi mọi người dùng đều có thể giao dịch trực tiếp với nhau một cách tin cậy và bảo mật.

1. Mô phỏng tấn công qua Lab thực hành
   1. Tấn công đồng thuận / chèn block giả
      1. Tấn công vào đồng thuận BLS-BFT
         1. Giới thiệu chung về bài thực hành

**Tên bài thực hành**: Tấn công giao thức đồng thuận BLS trong hệ thống BFT.

Giao thức BLS được dùng trong các hệ thống đồng thuận BFT để xác minh các thông điệp hợp lệ qua chữ ký tổng hợp. Tuy nhiên, nếu attacker có thể tạo hoặc giả mạo một block hoặc thông điệp không hợp lệ nhưng vẫn được xác thực (ví dụ: dùng chữ ký sai lệch, hoặc replay message), hệ thống có thể ra quyết định sai.

Những lỗ hổng này nếu không được xử lý triệt để sẽ làm suy giảm niềm tin vào hệ thống đồng thuận, tạo điều kiện cho các tấn công phá hoại hoặc chiếm quyền kiểm soát quá trình ra quyết định. Bài thực hành này giúp sinh viên hiểu cách các cuộc tấn công đó diễn ra từ đó phát triển cách phòng thủ.

**Mục đích**:

* Hiểu cơ chế hoạt động của một hệ thống đồng thuận BFT sử dụng chữ ký BLS.
* Hiểu và mô phỏng cách attacker có thể tấn công bằng cách giả mạo block, giả lập chữ ký, hoặc tái sử dụng thông điệp (replay).
* Quan sát hậu quả của các tấn công khi hệ thống không có biện pháp xác thực chặt chẽ.

**Yêu cầu đối với sinh viên**:

* Biết sử dụng terminal, câu lệnh Linux cơ bản.
* Đọc hiểu và sửa mã Python mô phỏng hệ thống validator và attacker.
* Biết kiểm tra log, thời gian chạy, cấu hình hệ thống.
* Trả lời các câu hỏi phân tích sau mỗi task.
  + - 1. Nội dung thực hành

Trong terminal ban đầu, gõ lệnh:

*imodule https://github.com/letienduong/labtainer/raw/refs/heads/main/imodule.tar*

để tải bài lab về.

Sinh viên khởi động bài lab:

*labtainer -r bls-bft*

*(Chú ý: Sinh viên sử dụng MÃ SINH VIÊN của mình để nhập thông tin người thực hiện bài lab khi có yêu cầu, để sử dụng khi chấm điểm.)*

Sau khi khởi dộng bài lab, 4 container sẽ xuất hiện tương ứng với validator1, validator2, validator3, attacker, sinh viên tiếp tục thực hiện các nhiệm vụ theo yêu cầu.

**Nhiệm vụ 1:** Kiểm tra kết nối giữa các node

Trong mỗi container ứng với các node, sinh viên sử dụng lệnh *ifconfig* để kiểm tra địa chỉ IP của từng container.

Tiếp theo, sinh viên thực hiện kiểm tra kết nối từ validator2 đến các container khác sử dụng lệnh:

*ping <Địa chỉ IP máy đích>*

Sau khi ping thành công, sinh viên trả lời câu hỏi: Tại sao việc kiểm tra kết nối lại quan trọng trong một hệ thống phân tán?

**Nhiệm vụ 2:** Giao tiếp giữa các node

Trong các hệ thống **đồng thuận BFT (Byzantine Fault Tolerant)**, các node (validator) cần liên tục **trao đổi thông tin block** với nhau để tiến tới đồng thuận. Một block có thể được tạo từ một node, sau đó gửi tới các node khác để yêu cầu xác thực, ký xác nhận, và tổng hợp thành một **chứng chỉ đồng thuận (QC – Quorum Certificate)**.

Việc một node lắng nghe block từ các node khác là một hoạt động thiết yếu trong quy trình đồng thuận, nhằm thực hiện:

* Xác minh chữ ký và nội dung block gửi đến.
* Kiểm tra thông tin vòng, leader, loại block.
* Tích lũy chữ ký từ nhiều node khác nhau, đủ quorum thì mới tiến hành tạo QC.

Trong bài thực hành này, validator2 thực hiện lắng nghe block từ nhiều nguồn để tích lũy thông tin và quyết định các bước tiếp theo.

Trước hết, trên terminal của node validator2, sinh viên kiểm tra các file hiện có:

*ls*

Sinh viên sẽ tìm thấy file *leader.py* trong danh sách file hiện có. File này sẽ mô phỏng hành vi của một leader trong hệ thống đồng thuận BFT, có nhiệm vụ:

* Lắng nghe các chữ ký gửi từ các validator khác.
* Kiểm tra tính hợp lệ của block.
* Tổng hợp chữ ký để tạo thành chứng chỉ đồng thuận (QC) bằng chữ ký BLS.

Tiếp theo, sinh viên chạy file *leader.py*  để mở một socket TCP lắng nghe kết nối từ các validator khác gửi chữ ký:

*python3 leader.py*

Mỗi block có một block\_id. Nếu leader thấy một block mới có cùng ID nhưng nội dung khác, nó coi đó là mạo danh. Đây là cách phát hiện tấn công kiểu gửi block giả mạo hoặc block bị sửa.

Sau khi chạy file leader ở validator2, sinh viên chuyển sang terminal của validator 1 và validator 3 để thực hiện gửi block và chữ ký tới leader (validator2). Sinh viên kiểm tra các file hiện có, thấy tồn tại file *node.py.* Sinh viên cần thực hiện chạy file này để có thể gửi chữ ký đến validator2 đang trong trạng thái lắng nghe:

*python3 node.py*

Sau khi thực thi xong file này, sinh viên quay trở lại terminal của validator2 và kiểm tra thông điệp vừa xuất hiện. Sinh viên có nhận xét gì?

Sau khi hoàn thành xong nhiệm vụ này, sinh viên hãy trả lời các câu hỏi sau:

* Điều gì sẽ xảy ra nếu validator1 và validator3 gửi cùng một block với nội dung giống hệt nhau?
* Leader tạo Quorum Certificate (QC) khi nào?
* Nếu chỉ có một validator gửi chữ ký đến leader, leader có thể tạo QC không? Vì sao?

**Nhiệm vụ 3:** Block giả mạo từ attacker

Trong nhiệm vụ này, sinh viên sẽ thực hiện phân tích và phát hiện một block giả mạo được gửi từ attacker đến validator2. Kẻ tấn công sẽ sử dụng khóa “validator1” để ký block, khiến hệ thống hiểu rằng block đến từ node hợp lệ.

Trong terminal attacker, sinh viên kiểm tra phát hiện file *attack.py,* file này chứa script thực hiện nhiệm vụ giả mạo validator1 gửi chữ ký đến validator2. Tuy nhiên, trong quá trình lập trình, kẻ tấn công đã không kiểm tra kỹ khiến trong file *attack.py* xuất hiện lỗi, từ đó không thể chạy được và gửi chữ ký đến validator2.

Nhiệm vụ của sinh viên trong phần này là kiểm tra lại file *attack.py* và áp dụng kiến thức từ những nhiệm vụ trước để sửa lại file *attack.py* để giúp kẻ tấn công có thể gửi chữ ký giả mạo đến validator2.

Sau khi sửa file *attack.py*, sinh viên chạy lại từ validator2 như nhiệm vụ 2 và lần lượt gửi chữ ký từ validator1 và attacker đến. Chạy lần lượt các lệnh sau ở 3 terminal:

*Validator2: python3 leader.py*

*Validator1: python3 node.py*

*Attacker: python3 attack.py*

Sinh viên quan sát kết quả xuất hiện ở validator2 và tiếp tục cho nhận xét? Sau đó trả lời các câu hỏi sau:

* Block ID mà attacker đã sử dụng là gì? Bạn tìm ra bằng cách nào?
* Nếu attacker sử dụng block ID trùng nhưng nội dung khác so với validator1, điều gì xảy ra?

**Nhiệm vụ 4:** Tìm hiểu về cách phòng chống kiểu tấn công này

Sinh viên hãy tìm hiểu một số kiến thức về chữ ký số BLS, xác thực danh tính trong hệ thống phân tán, reply attack và cách phòng chống, Quorum Certificate(QC) và cơ chế đồng thuận BFT để đề xuất các cách phòng chống.

**Kết thúc bài lab:**

Trên terminal đầu tiên sử dụng câu lệnh sau để kết thúc bài lab:

*stoplab*

Khi bài lab kết thúc, một tệp zip lưu kết quả được tạo và lưu vào một vị trí được hiển thị bên dưới lệnh stoplab.

**Kiểm tra kết quả:**

Sinh viên kiểm tra kết quả bài lab sử dụng lệnh:

*checkwork bls-bft*

**Khởi động lại bài lab:**

Trong quá trình làm bài sinh viên cần thực hiện lại bài lab, dùng câu lệnh:

*labtainer -r bls-bft*

* + 1. Tấn công Proof-of-Work giả (PoW-Fake)
       1. Proof of Work (PoW) là gì?

Proof of Work (PoW - Bằng chứng công việc) là cơ chế đồng thuận đầu tiên được tạo ra trên Blockchain và khá phổ biến trong thế giới tiền điện tử. Proof of Work được [Satoshi Nakamoto](https://coin98.net/satoshi-nakamoto-la-ai) áp dụng thành công cho [Bitcoin](https://coin98.net/bitcoin-btc-la-gi) vào năm 2009. Từ đó đến nay, PoW là một trong những cơ chế đồng thuận phổ biến nhất trong hệ sinh thái Cryptocurrency.

Proof of Work tập hợp các thợ đào (hay còn gọi là node) tham gia cạnh tranh xác thực các giao dịch, sau đó đưa giao dịch vào các block trong [Blockchain](https://coin98.net/blockchain-101) để nhận phần thưởng tùy theo mạng lưới.

Ví dụ: Các thợ đào của Ethereum sẽ xác nhận các giao dịch trên Ethereum, đưa vào block và nhận về ETH làm phần thưởng.

Bản chất của Proof of Work chính là xác nhận bằng chứng làm việc của ai đó là hợp lệ đến toàn bộ mạng lưới blockchain, thông qua việc tiêu tốn tài nguyên trong thế giới thực.

* + - 1. Giới thiệu chung về bài thực hành

Bài thực hành tấn công PoW-Fake giúp sinh viên tìm hiểu cách một attacker có thể gửi block giả vào hệ thống blockchain nếu nút mạng (node victim) kiểm tra thuật toán Proof of Work không đầy đủ. Cụ thể, attacker sẽ không thực hiện quá trình tính toán khai thác block như bình thường, mà dùng một block giả có hash “trông hợp lệ” (bắt đầu bằng 0000) để đánh lừa victim.

**Mục đích:**

Hiểu bản chất của **thuật toán PoW (Proof of Work)** trong blockchain.

Phân tích lỗ hổng khi node chỉ kiểm tra hash.startswith("0000").

Viết mã tấn công gửi block giả từ attacker → victim.

Thực hành kiểm tra nonce, kiểm thử block hợp lệ và đánh giá hệ thống.

**Yêu cầu đối với sinh viên:**

Hiểu cơ bản về thuật toán băm SHA256, nonce, thuật toán đồng thuận PoW.

Biết sử dụng Linux Terminal, chỉnh sửa file Python, sử dụng labedit, checkwork.

Biết cách xác định địa chỉ IP, giao tiếp socket giữa hai máy (attacker ↔ victim).

* + - 1. Nội dung thực hành

Sinh viên tải bài lab bằng lệnh:

*imodule* [*https://github.com/PoPo502/mmh2025/raw/main/imodule.tar*](https://github.com/PoPo502/mmh2025/raw/main/imodule.tar)

Sinh viên khởi động bài lab bằng lệnh:

*labtainer pow-fake*

(Chú ý: sinh viên sử dụng MÃ SINH VIÊN của mình để nhập thông tin người thực hiện bài lab khi có yêu cầu, để sử dụng khi chấm điểm)

Sau khi khởi động bài lab thành công sẽ hiện ra hai terminal một của attacker một của victim.

***TASK 1: Kiểm tra kết nối thành công giữa máy attacker và server***

Trên hai terminal thực hiện câu lệnh ***ifconfig***kiểm tra địa chỉ ip của hai máy

Từ terminal attacker thực hiện lệnh: ***ping <địa chỉ IP máy server>*** để kiểm tra kết nối giữa 2 máy.

***TASK 2: Tìm nonce hợp lệ***

Trên terminal attacker sinh viên kiểm tra file ***find\_nonce.py***và chạy file ***find\_nonce.py***  để tìm nonce hợp lệ:

***python3 find\_nonce.py***

***TASK 3: Sửa mã attacker và gửi block giả***

Sinh viên mở file fake\_block\_sender.py để sửa mã sao cho phần nonce là nonce hợp lệ vừa tìm được ở bên trên và sửa phần “host” là ip của victim

***nano fake\_block\_sender.py***

Trên terminal victim chạy file victim\_node.py để victim lắng nghe thông điệp

Trên terminal attacker chạy file fake\_block\_sender.py để gửi block giả sang máy victim.

Block sẽ được gửi, victim sẽ ghi lại log vào file fake\_pow.log:

***cat fake\_pow.log***

Nếu gửi thành công thì thông điệp sẽ là “Victim accepted fake block” nếu không thành công sẽ là “Victim rejected fake block”.

**Kết thúc lab:**

Trên terminal khởi động lab, sinh viên sử dụng lệnh:

***Stoplab***

Khi bài lab kết thúc, một tệp lưu kết quả được tạo và lưu vào một vị trí được hiển thị bên dưới stoplab. Sinh viên cần nộp file .lab để chấm điểm.

Để kiểm tra kết quả khi trong khi làm bài thực hành sử dụng lệnh:

***checkwork***

Sinh viên cần nộp file .lab để chấm điểm.

Kiểm tra kết quả trong quá trình làm bài:

***checkwork***

Khởi động lại bài lab: Trong quá trình làm bài sinh viên cần thực hiện lại bài lab, dùng câu lệnh:

***labtainer -r pow-fake***

* 1. Thám mã chữ ký số ECDSA
     1. ECDSA Signature Malleability
        1. Giới thiệu

Signature malleability (tự biến đổi chữ ký) là một kiểu tấn công trong đó kẻ tấn công không cần biết private key vẫn có thể tạo ra một chữ ký mới hợp lệ cho cùng một thông điệp đã được ký trước đó. Trong bên trong, chữ ký dòng (r, s) và (r, -s mod n) đều hợp lệ theo thuật toán ECDSA.

* + - 1. Cơ chế lý thuyết

Giả sử một thông điệp m đã được hash để tạo h:

* Khóa riêng: d
* Nonce ngẫu nhiên: k
* r = (k \* G).x mod n
* s = k^{-1} (h + r ⋅ d) mod n

Chữ ký hợp lệ là (r, s)

Nhưng ECDSA cho phép (r, -s mod n) cũng hợp lệ

* Vì với s' = -s mod n:
* Tính u1 = h ⋅ s'^{-1} = -h ⋅ s^{-1}
* u2 = r ⋅ s'^{-1} = -r ⋅ s^{-1}
* → điểm: R = u1G + u2Q = - (h ⋅ s^{-1} G + r ⋅ s^{-1} Q) = -R
* Do r = R.x = (-R).x ⇒ vẫn khớp r

Vậy: chữ ký (r, -s mod n) cũng được ECDSA đệm hợp lệ.

* + - 1. Hậu quả và nguy cơ

Trong các blockchain như Bitcoin và Ethereum, mã giao dịch txid = hash(to, value, signature).

Nếu attacker thay s bằng -s mod n, hash thay đổi, txid thay đổi.

Dẫn tới:

* Replay transaction
* Gây lỗi double-spending hoặc transaction dependency hông
  + - 1. Cách khắc phục
* Áp dụng quy tắc low-s: đồng bộ s sao cho s <= n/2
* Ethereum áp dụng từ EIP-2 (2016), Bitcoin từ BIP62
  + - 1. Bài lab mô phỏng

Sinh viên tải bài lab bằng lệnh imodule:

*imodule* [*https://github.com/NKTriS/labtainer/raw/main/imodule.tar*](https://github.com/NKTriS/labtainer/raw/main/imodule.tar)

Sinh viên khởi động bài lab bằng lệnh:

*labtainer ecdsa-malleability*

(Chú ý: sinh viên sử dụng MÃ SINH VIÊN của mình để nhập thông tin khi có yêu cầu, để sử dụng khi chấm điểm.) Sau khi khởi động thành công, màn hình sẽ mở ra ba terminal tương ứng với ba vai trò: signer, attacker, và victim.

**TASK 1: Sinh chữ ký gốc bằng signer**

Tại terminal signer, sinh viên thực hiện:

*python3 signer.py*

File signer.py để: tạo chữ ký gốc (r, s) cho thông điệp "authorize payment", ghi vào: /shared/original\_signature.json, ghi public key vào: /shared/public\_key.txt

Sử dụng lênh

*cat /shared/original\_signature.json*

để xem các thông tin được tạo

**TASK 2: Sinh chữ ký giả mạo bằng attacker**

Tại terminal attacker, thực hiện:

python3 attacker.py

File attacker.py để: đọc (r, s) từ chữ ký gốc, tính toán chữ ký mới (r, -s mod n), ghi vào: /shared/malleable\_signature.json

Sử dụng lênh

*cat /shared/malleable\_signature.json*

để xem các thông tin được tạo

**TASK 3: Kiểm tra chữ ký bằng victim**

Tại terminal victim, thực hiện:

*python3 verify.py*

File verify.py đọc public\_key.txt, original\_signature.json, malleable\_signature.json, kiểm tra chữ ký và ghi kết quả vào: /shared/verify\_result.txt.

Để xem kết quả kiểm tra:

*cat /shared/verify\_result.txt*

Nếu cả hai chữ ký đều hợp lệ → attack thành công.

**Kết thúc lab**

Sau khi thực hiện xong, tại terminal chính, sinh viên sử dụng lệnh:

*stoplab*

Sau khi dừng lab, một file .lab sẽ được tạo trong thư mục hiển thị. Sinh viên nộp file .lab này để chấm điểm.

Kiểm tra kết quả trong quá trình thực hiện

Tại terminal chính, có thể dùng lệnh:

*checkwork ecdsa-malleability*

để xem điểm tạm thời và xác nhận tiến độ bài lab.

* + 1. LAB: noncek-ecdsa
       1. Giới thiệu chung về bài thực hành

Tên bài thực hành: Khai thác lỗ hổng reused nonce trong thuật toán chữ ký số ECDSA để phục hồi khóa bí mật.

Thuật toán chữ ký số ECDSA (Elliptic Curve Digital Signature Algorithm) là một trong những thuật toán chữ ký số phổ biến nhất hiện nay, được sử dụng trong nhiều ứng dụng thực tế như xác thực giao dịch trong blockchain (Bitcoin, Ethereum), bảo mật giao tiếp qua TLS, và các hệ thống chứng thực chữ ký số.

Tuy nhiên, ECDSA có một điểm yếu nghiêm trọng nếu quá trình sinh nonce (số ngẫu nhiên duy nhất sử dụng một lần, ký hiệu là k) không đảm bảo tính ngẫu nhiên tuyệt đối hoặc bị lặp lại giữa các lần ký. Điều này tạo ra một lỗ hổng lớn, cho phép kẻ tấn công phục hồi khóa riêng d từ hai chữ ký sử dụng cùng k. Cụ thể:

Giả sử hai chữ ký của m1, m2 là:

S\_(1 )= k^(-1) (H(m\_1) + dr) ) mod n

S\_(2 )= k^(-1) (H(m\_2) + dr)) mod n

Lấy hiệu:

s\_1 - s\_2= k^(-1) (H(m\_1) - H(m\_2)) => k =□((H(m\_1) - H(m\_2))/(s\_1 - s\_2 ) mod n)

Sau khi có k, attacker có thể tìm ra khóa bí mật d:

d = (s\_1 k - H(m\_1))/r mod n

Mục đích:

Hiểu rõ cơ chế tạo chữ ký và xác minh chữ ký ECDSA.

Phân tích tác động của việc tái sử dụng hoặc sinh nonce không ngẫu nhiên.

Mô phỏng kịch bản tấn công phục hồi khóa bí mật từ hai chữ ký dùng chung.

Quan sát hậu quả khi attacker có thể giả mạo chữ ký đúng.

Yêu cầu đối với sinh viên:

Biết sử dụng terminal, câu lệnh Linux cơ bản.

Đọc hiểu và sửa mã Python mô phỏng hệ thống validator và attacker.

Biết kiểm tra log, thời gian chạy, cấu hình hệ thống.

Trả lời các câu hỏi phân tích sau mỗi task.

* + - 1. Nội dung thực hành

Trong terminal ban đầu, gõ lệnh:

*imodule <https://github.com/thanhdat317/Cryptoraphy/raw/main/noucek-ecdsa.tar>*

để tải bài lab về.

Sinh viên khởi động bài lab:

*labtainer noucek-ecdsa*

(Chú ý: Sinh viên sử dụng MÃ SINH VIÊN của mình để nhập thông tin người thực hiện bài lab khi có yêu cầu, để sử dụng khi chấm điểm.)

Sau khi khởi dộng bài lab, 4 container sẽ xuất hiện tương ứng với validator1, validator2, validator3, attacker, sinh viên tiếp tục thực hiện các nhiệm vụ theo yêu cầu.

Nhiệm vụ 1: Kiểm tra kết nối giữa các node

Trong mỗi container ứng với các node, sinh viên sử dụng lệnh ***ifconfig*** để kiểm tra địa chỉ IP của từng container.

Tiếp theo, sinh viên thực hiện kiểm tra kết nối từ validator1 đến các container khác sử dụng lệnh:

*ping <Địa chỉ IP máy đích>*

Sau khi ping thành công, sinh viên trả lời câu hỏi: Tại sao việc kiểm tra kết nối lại quan trọng trong một hệ thống phân tán?

Nhiệm vụ 2: Validator1 ký hai block khác nhau dùng chung nonce

Để thực hiện nhiệm vụ này, sinh viên truy cập terminal validator2 và validator3 sau đó kiểm tra các file hiện có:

*ls*

Sinh viên sẽ tìm thấy file *node.py* trong danh sách file hiện tại.

Tiếp theo, sinh viên tiến hành chạy file *node.py* trên từng lần lượt validator 2 và validator 3 để thực hiện việc lắng nghe chữ ký từ validator1:

*python3 node.py*

Sau khi thực thi file *node.py*, trên terminal của validator2 và validator3 sẽ hiện thông báo đang lắng nghe dữ liệu. Sau đó, sinh viên chuyển sang validator1 để thực hiện việc chạy script tạo block và gửi dữ liệu tới các node còn lại qua socket TCP.

Trong danh sách file của validator1, sinh viên tìm và chạy file leader.py để thực hiện việc ký block:

*python3 leader.py*

File leader.py khi được thực thi sẽ thực hiện việc ký và gửi dữ liệu đến validator2 và validator3 thông qua socket TCP. Sinh viên quan sát thông báo xuất hiện trên 2 terminal và cho nhận xét?

Sinh viên trả lời câu hỏi sau: Nếu validator không kiểm tra giá trị nonce hoặc block hash, điều gì có thể xảy ra?

Nhiệm vụ 3: Attacker phục hồi private key

File *leader.py* chứa script khi được thực thi sẽ tạo 2 block có chữ ký (r,s1) và (r,s2) và lưu vào file signatures.json. Sinh viên kiểm tra trên terminal validator1 để tìm vị trí của file signatures.json.

Nhiệm vụ của sinh viên là đọc file json này và tìm các tham số cần thiết để attacker có thể phục hồi private key.

*cat signatures.json*

Sau khi truy xuất được các tham số cần tìm, sinh viên quay trở lại terminal attacker và kiểm tra các file hiện tại bằng lệnh *ls*. Trong danh sách file có thể tìm được file *attack.py*.

Sinh viên thực hiện tải trình soạn thảo mousepad hoặc gedit để có thể chỉnh sửa file attack.py. Sinh viên thực hiện lệnh sau:

*sudo apt install mousepad*

Nếu sinh viên thực hiện bằng *gedit*, sinh viên thực hiện tương tự.

Sau khi sửa thành công file attack.py, lưu lại và chạy file này bằng câu lệnh:

*python3 attack.py*

Sau khi chạy, block giả sẽ được tạo và gửi tới validator2 và validator3. Sinh viên quan sát kết quả thông báo ở validator2 và validator3 và so sánh với kết quả nhiệm vụ 2, từ đó rút ra nhận xét.

Sinh viên trả lời câu hỏi sau: Việc sử dụng lại nonce k đã dẫn tới hậu quả gì?

Nhiệm vụ 4: Phân tích và phòng chống:

Câu hỏi phân tích:

Vì sao attacker lại giả mạo được?

Dấu hiệu nào để nhận diện block giả mạo?

Giải pháp đề xuất:

Sử dụng k được sinh từ RFC 6979 (deterministic).

Kiểm tra nguồn gốc block.

Ghi log, theo dõi hành vi bất thường.

Không dùng lại khóa k và đảm bảo sinh đủ ngẫu nhiên.

Kết thúc bài lab:

Trên terminal đầu tiên sử dụng câu lệnh sau để kết thúc bài lab:

*stoplab*

Khi bài lab kết thúc, một tệp zip lưu kết quả được tạo và lưu vào một vị trí được hiển thị bên dưới lệnh stoplab.

Kiểm tra kết quả:

Sinh viên kiểm tra kết quả bài lab sử dụng lệnh:

*checkwork noucek-ecdsa*

Khởi động lại bài lab:

Trong quá trình làm bài sinh viên cần thực hiện lại bài lab, dùng câu lệnh:

*labtainer -r noucek-ecdsa*

* 1. Lỗ hổng và tấn công trong Smart Contract
     1. Mục đích bài thực hành

Mục tiêu của bài tập này là để cung cấp cho sinh viên một trải nghiệm thực tế với smart contract (hợp đồng thông minh ) trong blockchain

Sinh viên sẽ được thực thi 4 dạng lỗ hổng kinh điển trong smart contract:

* Integer Overflow
* Reentrancy
* TOD (Transaction-Ordering Dependency)
* Selfdestruct (Destructible Contract Vulnerability)
  + 1. Yêu cầu đối với sinh viên

Sinh viên hiểu được 4 lỗ hổng cơ bản trong hợp đồng thông minh và các tấn công và các khắc phục các lỗ hổng đó trong hợp đồng thông minh

* + 1. Nội dung bài thực hành

Sinh viên tải bài lab bằng lệnh imodule:

*imodule https://github.com/huongcnc/MMHCS/raw/refs/heads/main/imodule.tar*

Sinh viên khởi động bài lab bằng lệnh :

*labtainer smartcontract*

*(Chú ý: sinh viên sử dụng MÃ SINH VIÊN của mình để nhập thông tin người thực hiện bài lab khi có yêu cầu, để sử dụng khi chấm điểm)*

Sau khi khởi động bài lab thành công sẽ hiện ra 2 terminal đều là của attacker

Trên terminal 1 sinh viên thực hiện cài đặt đầy đủ các plugin hỗ trợ phát triển và kiểm thử Smart Contract trong Hardhat bằng lệnh :

*npm install --save-dev @nomicfoundation/hardhat-toolbox*

Tiếp tục thực hiện cài đặt công cụ để làm việc với smartcontract

*npm install --save-dev hardhat*

Tiếp theo tạo ra một mạng Ethereum nội bộ chạy trên localhost:8545 trong đó có sẵn 20 tài khoản với 10000 ETH ảo mỗi tài khoản

*npx hardhat node*

**Task 1 : Tấn công vào lỗ hổng Integer Overflow**

Sau khi tạo thành công các tài khoản , sinh viên vào terminal 2 thực hiện chỉnh sửa file overflow.js để thực hiện tấn công sau đó sửa lại file OverflowToken.sol để không mắc phải lỗ hổng đó nữa

Quy trình tấn công:

* Ban đầu attacker , victim đều tham gia một dịch vụ sử dụng smart contract
* Ban đầu attacker , victim đều có 100 token trong tài khoản , tận dụng lỗ hổng về tràn số , attaker thực hiện một giao dịch cho victim , chuyển cho victim 101 token.
* Khi đó , đáng nhẽ tài khoản của attacker sẽ phải báo không đủ số dư . Nhưng tận dụng logic sai của hệ thống nên số dư của attacker sẽ trở nên vô cùng lớn , nạn nhân cũng không bị mất tiền mà còn nhận thêm 101 token , như vậy sẽ không ai kêu ca vì bị mất tiền cả

*npx hardhat run scripts/overflow.js --network localhost*

**Task 2 : Tấn công vào lỗ hổng Reentrancy**

Quy trình tấn công :

* Attacker tham gia một dịch vụ cũng sử dụng smart contract như bao người bình thường khác nhưng tham gia
* ExploitReentrancy là contract do attacker tự viết và tự deploy ra, nhằm tấn công contract nạn nhân (Reentrancy) bị lỗi.
* Giả sử : victim (nạn nhân) gửi 1 ETH vào trong smartcontract . Sau đó attacker cũng yêu cầu chuyển tiền vào trong đó 0.1 ETH và yêu cầu rút tiền . Nhưng khi gọi rút tiền attacker đã tận dụng việc chuyển tiền trước rồi mới trừ trong tài khoản của smart contract để tạo ra một vòng lặp rút tiền . Kết quả mặc dù số dư của attacker có thể không đủ nhưng nó có thể rút được toàn bộ tiền trong smartcontract ơ đây theo ví dụ trên là 1.1 ETH

Trong bài thực hành này : File reentrancy.js là scripts mô phỏng lại quá trình tấn công

* Vicim gửi 1 ETH vào ví chung
* Attacker gửi 0.1 ETH vào ví chung
* Attacker yêu cầu rút tiền ( đệ quy)

Thực hiện lệnh *:*

*npx hardhat run scripts/reentrancy.js –network localhost* để mô phỏng bài lab

Chụp kết quả và quy trình tấn công trên cả 2 terminal

**Task 3 : Tấn công vào lỗ hổng TOD (Transaction-Ordering Dependency)**

Trên nền tảng Ethereum, bất cứ giao dịch nào trước khi được xác minh bởi các Node đều được đẩy vào hàng đợi (pool). Hàng đợi này không có thứ tự rõ ràng, việc giao dịch nào được hệ thống xác minh trước phụ thuộc vào nhiều yếu tố. Việc giao dịch sẽ được ưu tiên theo phí giao dịch (gas)

Lợi dụng việc này, Hacker có thể tiến hành tấn công theo kịch bản như sau: Giả sử bạn tham gia một cuộc thi giải toán , nếu bạn đưa ra kết quả là 42 thì bạn sẽ nhận được giải thưởng. Vấn đề ở đây là trước khi kết quả lên hệ thống Hacker đã thực hiện phủ đầu bằng việc đưa ra kết quả với phí giao dịch (gas) cao hơn so với nạn nhận .

Yêu cầu sinh viên : tìm kiếm file tod.js , MathContest.sol để hiểu mô phỏng của bài lab

Thực hiện lệnh *:*

*npx hardhat run scripts/tod.js –network localhost* để mô phỏng bài lab

**Task 4 : Tấn công vào lỗ hổng Selfdestruct (Destructible Contract Vulnerability)**

Giả sử chúng ta có một Hợp đồng thực hiện nhiệm vụ nào không liên quan đến tiền. Quy định đặt ra là nếu bạn chuyển tiền vào hợp đồng, hợp đồng sẽ bị huỷ bỏ và các bên liên quan sẽ bị truy tố. Người thiết kế Hợp đồng thông minh sẽ kiểm soát việc này bằng cách: nếu ai đó chuyển tiền vào ví của hợp đồng, số tiền sẽ bị chuyển ngược lại hoặc cấm không cho chuyển tiền.

Cách dễ dàng nhất để huỷ hợp đồng này là chuyển tiền vào hợp đồng, nhưng khi chuyển tiền vào hợp đồng, hàm Fallback sẽ được gọi và huỷ lệnh chuyển tiền này. Tuy nhiên, Hacker đã phát hiện ra một cơ chế của hợp đồng thông minh đó là hàm SelfDestruct. Hàm này có tác dụng tự huỷ hợp đồng và chuyển toàn bộ số tiền của hợp đồng sang một địa chỉ ví nào đó. Như vậy Hacker có thể tự tạo ra một hợp đồng mới, chuyển tiền vào hợp đồng này, gọi hàm SelfDestruct để huỷ hợp đồng vừa tạo. Toàn bộ số tiền của hợp đồng mới sẽ chuyển sang hợp đồng trên mà không kích hoạt hàm Fallback

Thực hiện lệnh :

*npx hardhat run scripts/selfdestruct.js –network localhost* để thực hiện mô phỏng lab

Thực hiện lệnh : *checkwork* để kiểm tra các task

Kết thúc bài lab : stoplab

Kết luận

Mật mã học giữ vai trò cốt lõi trong việc đảm bảo tính an toàn, minh bạch, xác thực và toàn vẹn của hệ thống blockchain. Thông qua các cơ chế mật mã như hàm băm liên kết các khối, chữ ký số xác thực giao dịch và thuật toán đồng thuận (ví dụ: Proof of Work), blockchain duy trì được sự tin cậy và minh bạch mà không cần đến cơ quan trung gian. Nhờ nền tảng mật mã vững chắc này, mọi hành vi chỉnh sửa dữ liệu đều có thể bị phát hiện, quyền sở hữu và tính hợp lệ của giao dịch được bảo đảm, giúp mạng lưới blockchain vận hành một cách an toàn và toàn vẹn.

Trong khuôn khổ môn học, các lỗ hổng bảo mật tiêu biểu đã được nghiên cứu bao gồm lỗi tràn số nguyên (Integer Overflow), lỗ hổng Reentrancy (tái nhập), điểm yếu cho phép chỉnh sửa chữ ký số (Signature Malleability) và tấn công giả mạo bằng chứng công việc (PoW-Fake). Mỗi lỗ hổng trên minh họa một khía cạnh khác nhau của rủi ro bảo mật. Chẳng hạn, sai sót trong lập trình hợp đồng thông minh có thể dẫn đến hành vi ngoài ý muốn; kẽ hở trong thuật toán mật mã cho phép thay đổi dữ liệu giao dịch; hay việc kiểm tra đồng thuận không chặt chẽ khiến hệ thống có thể chấp nhận block không hợp lệ. Việc nhận diện và hiểu rõ những lỗ hổng này nhấn mạnh sự cần thiết của việc thiết kế hệ thống blockchain cẩn thận, tuân thủ nghiêm ngặt các nguyên tắc mật mã và lập trình an toàn nhằm ngăn chặn những cuộc tấn công tương tự.

Bên cạnh kiến thức lý thuyết, việc thực hành mô phỏng qua các bài lab có vai trò quan trọng trong quá trình học tập. Thông qua việc tái hiện các tình huống tấn công và phòng thủ trong môi trường thí nghiệm, sinh viên có cơ hội quan sát trực quan cơ chế tấn công, từ đó hiểu sâu hơn cách thức kẻ xấu khai thác lỗ hổng cũng như các biện pháp phòng chống thích hợp. Những trải nghiệm thực hành này không chỉ củng cố kiến thức đã học mà còn rèn luyện kỹ năng phân tích bảo mật, nâng cao tư duy phản biện và khả năng áp dụng các biện pháp bảo vệ vào thực tế.

Tóm lại, mật mã học là nền tảng không thể thiếu để đảm bảo an ninh cho các hệ thống blockchain. Qua việc tìm hiểu các lỗ hổng điển hình và thực hành với các bài lab mô phỏng, sinh viên ngành Công nghệ thông tin được trang bị cả kiến thức chuyên sâu lẫn kinh nghiệm thực tiễn, giúp họ sẵn sàng hơn trong việc phân tích và nâng cao độ an toàn cho các ứng dụng blockchain trong tương lai.

TÀI LIỆU THAM KHẢO

1. GeeksforGeeks (29/8/2024). *Blockchain Structures*. Được truy lục từ: <https://www.geeksforgeeks.org/blockchain-structure/>
2. Anastasiia Lastovetska (12/11/2021). *Blockchain Architecture Explained: How It Works & How to Build.* Được truy lục từ:

<https://mlsdev.com/blog/156-how-to-build-your-own-blockchain-architecture>

1. GeeksforGeeks (1/7/2024). *How does the Blockchain Work?*. Được truy lục từ:

<https://www.geeksforgeeks.org/how-does-the-blockchain-work/>

1. Vy Bùi (24/10/2024). *Node là gì? 5 Bước chạy Node trong Blockchain.* Được truy lục từ:

<https://coin98.net/node-la-gi>

1. Tạp chí Bitcoin (8/1/2019). *Tìm hiểu về ‘Block’ trong Bockchain*. Được truy lục từ: <https://tapchibitcoin.io/tat-tan-tat-ve-block-trong-blockchain.html>
2. Mikel Garcia (6/12/2023). *What is a Blockchain node: Insights and Applications*. Được truy lục từ: <https://www.diadata.org/blog/post/what-is-a-blockchain-node/>
3. Webisoft Blog (13/2/2024). *16 Disadvantages of Blockchain: Limitations and Challenges - Webisoft Blog.* Được truy lục từ:<https://webisoft.com/articles/disadvantages-of-blockchain/>
4. Dylan J. Yaga, Peter M. Mell, Nik Roby, Karen Scarfone (2018). *Blockchain Technology Overview.* Được truy lục từ: <https://nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8202.pdf>
5. Hrchannels (27/2/2023). *08 nền tảng Blockchain uy tín mà bạn nên biết.* Được truy lục từ:

<https://hrchannels.com/uptalent/08-nen-tang-blockchain-uy-tin.html>

1. pazz2211 (2022). *Hướng dẫn cài đặt HyperLedger Fabric*. Được truy lục từ:

<https://github.com/pazz2211/fabric/blob/main/README.md>

1. [Trần Đức Trung](https://vietnix.vn/author/tranductrung/) (25/6/2024). *Công nghệ Blockchain là gì? Blockchain có phải là tiền ảo không?.* Được truy lục từ: <https://vietnix.vn/blockchain-la-gi/>
2. [Washija Kazim](https://learn.g2.com/author/washija-kazim) (28/5/2024). *26 Top Blockchain Applications and Use Cases in 2024.* Được truy lục từ: [https://learn.g2.com/blockchain-applications#retail](https://learn.g2.com/blockchain-applications%23retail)
3. Nguyễn Tùng Anh (16/11/2022). *Hướng dẫn thao tác với mạng test-network*. Được truy lục từ: <https://youtu.be/DYyUMUREHEs?si=OOaPu9VMwfcTwvOV>
4. Wikipedia. *Hyperledger.* Được truy lục từ: <https://en.wikipedia.org/wiki/Hyperledger>
5. Wikipedia. *Bitcoin*. Được truy lục từ: <https://vi.wikipedia.org/wiki/Bitcoin>